DESCARTES

matemáticas interactivas

Documentación de Descartes JS

Alejandro Radillo Díaz

José Luis Abreu León

Joel Espinosa Longi

24 de junio de 2025

Índice general

1

Sobre la presente documentación

La presente documentación está destinada tanto a personas que no han usado Descartes como a personas que tienen cierta experiencia y desean mejorarla.

La documentación aborda tanto al Editor de *DescartesJS* como la funcionalidad general del mismo. No obstante, se da una importancia mayor a la funcionalidad. Adicionalmente, se trata la funcionalidad más común de Descartes, y no se abordan algunas particularidades del Editor y algunas de funcionalidad general debido a que no son de uso frecuente.

Con el objeto de que el lector pueda practicar a medida que avanza en la documentación, se proponen algunos ejercicios cuando se considera se ha acumulado suficiente conocimiento para ejecutarlos, y se incluyen los ejercicios resueltos en forma de interactivos de Descartes para que el lector pueda compararlos con los propios. Se encuentran en la carpeta *Ejercicios* dentro del archivo comprimido *DocumentacionDescartes.zip* en `https://descartes.matem.unam.mx/doc/DescartesJS/DocumentacionDescartesJS.zip`. Dentro de esta documentación se incluyen los mismos interactivos **en la red**, pero es importante recordar que están presentes en el zip proporcionado para poder revisarlos de manera local. Asimismo, es importante mencionar que los vínculos en el documento que redirigen al ejercicio en la red, redirigen a un archivo web que permite tanto visitar el interactivo terminado como la serie de instrucciones para lograrlo. Cabe mencionar que existen algunos comandos en las escenas de los interactivos que no vienen en las instrucciones particulares del interactivo y están ahí sólo para que el interactivo pueda ser visualizado correctamente en el contenedor en donde el usuario puede alternar entre el interactivo completo y el documento con las instrucciones.

Los ejercicios típicamente consisten en una serie de pasos guiando al usuario a crear un interactivo. No obstante, de forma intencional se busca que las instrucciones de los pasos no sean explícitas. Ello favorecerá que el lector pueda practicar, experimentar abordajes distintos, y revise, de ser necesario, la documentación para lograr una escena final. En este espíritu, es posible que un interactivo diseñado por el usuario difiera del proporcionado en la documentación. Dos estrategias diferentes pueden llevar a un mismo resultado. Adicionalmente, se proporcionan observaciones a cada paso. Normalmente, éstas observaciones mencionan el resultado esperado de cada paso, pero algunas ocasiones, éstas podrán traer sugerencias o información que ayude al usuario a lograr el paso correspondiente.

Los ejercicios en este documento van aumentando de dificultad. Poco a poco involucran más y más compleja funcionalidad. Por lo mismo, es buena idea que en la medida

de lo posible se aborden en orden, aunque existe la posibilidad de visitar la funcionalidad diversa de Descartes mediante los vínculos proporcionados.

Con esta documentación se espera que el usuario pueda, al concluirla, generar sus propios interactivos con *DescartesJS*. Y gracias a la versatilidad de esta herramienta de programación, es probable que el usuario termine generando escenas de mayor complejidad a las presentadas a lo largo de este documento.

Es preciso mencionar que esta documentación no es estática. Se pretende agregarle mejoras con el tiempo, por lo que se sugiere descargar nuevas versiones de la misma con regularidad. Esta versión de la documentación cubre hasta la versión 1.4 de *DescartesJS*.

2

¿Qué es DescartesJS?

2.1. Los inicios de Descartes

Descartes nace a fines del siglo XX como una herramienta de creación de interactivos que aprovecha el lenguaje de programación Java. Entonces consistió en un programa en Java que permitía generar archivos .html para ser visualizados como páginas web.

Los archivos html generados por Descartes suelen contener interactivos y son principalmente usados en la docencia de matemáticas y física en diversos niveles.

A pesar de que existe una gran variedad de programas interactivos con fines de docencia tales como GeoGebra, Cabri y otros, Descartes permite una gran versatilidad de interacciones. Adicionalmente, con Descartes es posible generar interacciones específicas diseñadas por el profesor, ejercicios aleatorios con restricciones particulares, además de que presenta muchas otras ventajas como se verá adelante.

En sus inicios, *Descartes* funcionaba directamente en Java para ser usados en computadora. No obstante, el advenimiento de dos tecnologías nuevas forzaron un cambio en Descartes, a saber:

- El advenimiento de dispositivos móviles
- El elemento canvas de HTML5

2.2. Modificaciones actuales

Se volvió entonces necesario que los interactivos corrieran en dispositivos móviles (en JavaScript). Descartes 5 nace de esta necesidad. Un nuevo editor fue creado con esto en mente. Aunque no hubieron cambios muy drásticos en el editor, fue necesario incluir una biblioteca nueva.

La versión 5 del editor de *Descartes* es la penúltima en la lista de versiones. También se encuentra programada en Java. No obstante, la versión JS de *Descartes* es la más actual. Dicho editor se encuentra programado en JavaScript y su funcionalidad es básicamente muy similar al de la versión Descartes 5, aunque tiene muchas ventajas sobre él. Todo esto se explica un poco más a fondo adelante.

También es importante mencionar que, a diferencia de Descartes 5, el nuevo editor *DescartesJS* tiene la virtud de que lo que se visualiza en el editor se representará idéntico en un navegador. Por ejemplo, en Descartes 5 existía el problema de que los tamaños de

fuente de texto mostrados en el editor no necesariamente coincidían con los mostrados en un navegador. Esto ya no ocurre en la versión *DescartesJS*. Aunque la edición de escenas se efectúa en un entorno propio, ajeno al navegador que después elija el usuario para ver e interactuar con dichas escenas, la funcionalidad y el aspecto es exactamente el mismo al pasar de un contexto al otro pues en ambos casos se está utilizando el mismo intérprete de Descartes.

3

Introducción a los componentes de Descartes JS

Descartes JS contiene dos elementos principales, a saber:

1. El editor
2. El intérprete

3.1. El editor

El editor consiste en el programa JavaScript (*DescartesJS.exe*). Este instalador puede bajarse de la red en este vínculo: https://descartes.matem.unam.mx, y se encuentra en constante mantenimiento incluyendo nuevas mejoras, de tal forma que es siempre una buena práctica reinstalarlo con cierta frecuencia (en mi experiencia, por lo menos una vez al mes).

Una vez instalado, se genera un acceso directo en el escritorio con el que se puede lanzar. De igual forma, se crea un acceso directo con el ícono de *Descartes* en la barra de tareas.

El programa del editor es una interfaz gráfica para el usuario mediante la cual podrá guardar archivos html con diversos tipos de interactivos. En la Figura 3.1 se muestra la vista del editor, y el uso del mismo se detalla más adelante. Este editor se puede buscar dentro de los programas en la computadora una vez que se haya instalado Descartes.

Recomendación: *Descartes* puede instalarse en cualquier directorio. De preferencia, se recomienda uno cercano a la raíz del sistema. Por ejemplo, en Windows podría guardarse en un directorio creado por el usuario *c:/DescartesJS*.

3.2. El intérprete

El intérprete es un archivo *js* que se puede bajar del siguiente vínculo:
https://arquimedes.matem.unam.mx/Descartes5/lib/descartes-min.js.
Sin embargo, cada vez que el editor se abre, permite realizar una actualización automática de la biblioteca, como se muestra en la Figura 3.2. El diálogo de actualización del intérprete sólo aparece si se detecta que éste está desactualizado. Adicionalmente, de encontrarse abajo el servidor donde está alojado el intérprete, se echa mano de un sitio alterno que aloja al intérprete para que éste siempre esté actualizado.

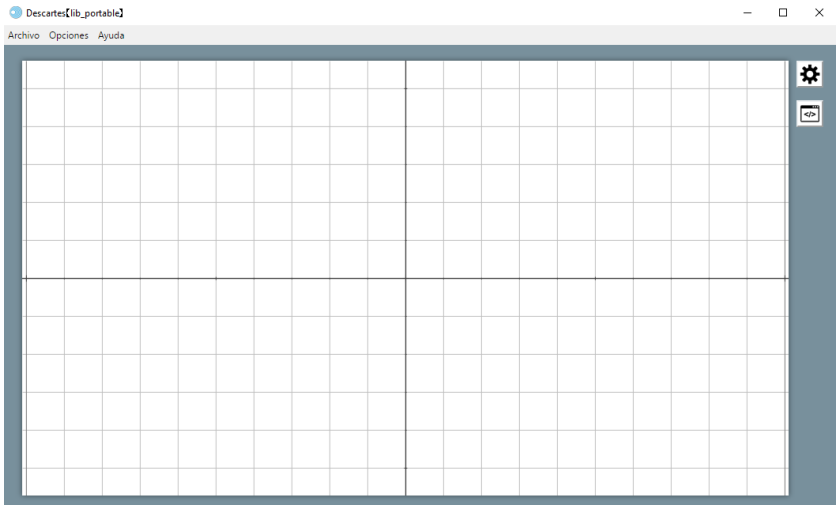


Figura 3.1: El editor de *Descartes JS* como se abre originalmente.

Es preciso notar que al aceptar las actualizaciones, pueden incluirse también cambios de funcionalidad en el Editor. Tanto el editor como las bibliotecas están en constante mejora, una razón más para aceptar las actualizaciones siempre que sean solicitadas.

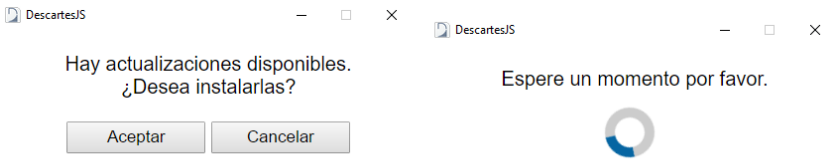


Figura 3.2: Ventanas de actualización del intérprete.

Se le conoce como intérprete debido a que es un archivo que permite a los navegadores entender el código de un archivo *html* creado por el Editor de *Descartes* para así desplegarlo correctamente. Se recomienda siempre aceptar la actualización para que los interactivos sean visualizados correctamente.

Ocasionalmente se le llega a llamar *biblioteca de Descartes*, aunque se prefiere no usar este término para no confundir con la biblioteca de Definiciones de Descartes.

4

Aprendiendo a usar el editor

4.1. La barra de menús del Editor

Al igual que cualquier programa, el Editor tiene una barra de menús. A continuación se detallan los dos primeros, que son los más comúnmente usados:

4.1.1. El menú *Archivo*

- **Nuevo**: Crea un documento de Descartes en blanco.
- **Nueva ventana**: Lanza una nueva instancia de Descartes cuando ya hay uno abierto. Evita tener que lanzarlo desde la barra de tareas. Cuando se tiene una ventana de Descartes abierta, y se quiere abrir un segundo archivo de Descartes existente, es preciso usar esta opción de menú. Al usarla, se abre una nueva ventana en la que se puede abrir la segunda escena.
- **Abrir**: Permite abrir un documento *html* creado en Descartes.
- **Abrir reciente**: Despliega un menú a su derecha con los archivos usados más recientes, incluyendo sus carpetas contenedoras, con el objeto de que el usuario pueda abrir archivos recientes con más facilidad y sin necesidad de buscarlos en el explorador de la ventana emergente. Incluye una opción *Vaciar menú* mediante la cual se puede limpiar la lista mostrada.
- **Recargar**: Recarga el interactivo con los cambios más recientes.
- **Guardar**: Si aún no había sido guardado el html, se muestra un diálogo para escoger el nombre y ruta de guardado. De lo contrario, se sobreescribe el archivo con los nuevos cambios. Guardar de forma periódica es una costumbre sabia para no perder el progreso en caso que la herramienta llegara a colgarse.
- **Guardar como**: abre un diálogo para escoger ruta y nombre de un nuevo archivo html en el que se guardarán la última versión del mismo.
- **Mostrar carpeta contenedora**: abre la carpeta en que se encuentra el archivo de escena de Descartes actualmente en edición.
- **Mostrar en el navegador**: abre la escena en el navegador predeterminado del usuario.
- **Exportar a**: contiene varias opciones, cada una de las cuales lanza un diálogo para guardar el archivo en cuestión:

- **macro de Descartes**: lanza un diálogo para guardar contenido parcial del documento de Descartes como un macro. El macro es un documento de texto (típicamente con una extensión *txt*) con los elementos de los selectores Definiciones, Programa y Gráficos, para poderlos eventualmente importar a otras escenas de Descartes. Se puede consultar los detalles de este funcionamiento más a fondo en el apartado sobre el gráfico Macro.
- **biblioteca de Descartes**: guarda el contenido del documento de Descartes como un archivo nuevo o biblioteca. También es por defecto un archivo de texto (con su respectiva extensión *txt*). Posteriormente, la biblioteca podrá ser utilizada dentro del selector Definiciones. Suele usarse cuando hay una gran cantidad de definiciones que no se desea estén visibles dado que no serán modificadas. De tal manera que dichas definiciones se mantienen "escondidas" en la biblioteca. En cierto sentido funciona parecido a un macro, con la salvedad de que en ella sólo se vierte el contenido del selector Definiciones. Se puede consultar más sobre el funcionamiento de la biblioteca en el apartado Biblioteca.
- **png**: permite guardar la visualización del documento de Descartes como una imagen .png.
- **jpg**: permite guardar la visualización del documento de Descartes como una imagen .jpg.
- **svg**: permite guardar la visualización del documento de Descartes como una imagen .svg. Por el momento se encuentra en fase experimental.
- **pdf**: permite guardar la visualización del documento de Descartes como un archivo .pdf. Por el momento se encuentra en fase experimental.

- **Cerrar escena**: Cierra el documento actual de Descartes. Si se realizaron cambios y se selecciona esta opción, se presenta un diálogo de confirmación que indica que si se decide continuar, se pierden los cambios.
- **Salir**: Cierra la aplicación *DescartesJS*. Al igual que para la opción de *Cerrar escena*, también se presenta un diálogo de confirmación si se hicieron cambios y estos no fueron guardados.
  Si se cierra *DescartesJS* cerrando la ventana completamente en lugar de mediante la opción *Salir*, se presenta también el diálogo de confirmación con la opción de guardar. En caso que se elija *Cancelar*, el programa no se cerrará y no se perderán los cambios.

Algunos elementos del menú *Archivo* cuentan con atajos de teclado para un acceso más ágil. Los atajos correspondientes aparecen en un texto en gris claro a la derecha del elemento del menú en cuestión. No se incluyen en la documentación dado que dependen del sistema operativo en que se ejecuta el editor. No obstante, el usuario puede desplegar el menú y ahí ver cuáles son los atajos.

Es posible también abrir un archivo *html* de *DescartesJS* arrastrándolo y soltando sobre el ícono de acceso directo de *DescartesJS* creado en el escritorio al instalar el programa.

Alternativamente, se puede abrir un *html* de *Descartes* mostrando el menú contextual del archivo y expandiendo el submenú *Abrir con*. En esta lista aparece una opción *nwjs* con el ícono de *Descartes*. Al pulsar en ella, el archivo será abierto automáticamente en *Descartes*.

4.1.2. El menú *Opciones*

Este menú contiene cuatro opciones. La primera está relacionada a la localización desde la cual se lee la biblioteca (el archivo descartes-min.js discutido antes). La segunda está relacionada con el lenguaje presentado en el editor. La tercera está relacionada con el tema de color del editor. La cuarta tiene que ver con una consola de Descartes.

- ***descartes-min.js***: al seleccionar este menú se lanza un submenú a su derecha con las siguientes opciones relacionadas con el intérprete de Descartes:

  - **en Internet**: Cuando se elige alguna opción de guardado del menú *Archivo*, se lee el archivo *descartes-min.js* de su ubicación en Internet. La ruta del archivo que se busca en Internet es `https://arquimedes.matem.unam.mx/Descartes5/lib/descartes-min.js`. Esta opción tiene la ventaja de que siempre se leerá el archivo más actual, pero la desventaja de que los archivos Descartes guardados bajo esta opción no funcionarán en ausencia de conexión a Internet.
  - **portable**: A la misma altura del archivo que se guarda, se genera una carpeta */lib* dentro de la cual se hace una copia del archivo *descartes-min.js*.
  - **de proyecto**: Se usa cuando varios archivos creados en Descartes, y que se encuentran al mismo nivel entre ellos, han de compartir el mismo intérprete. Esto ocurre frecuentemente en proyectos en que una misma unidad consta de varias páginas html de Descartes que van agrupadas. En este caso, la carpeta */lib* que contiene al intérprete *descartes-min.js* se ubica un nivel por arriba de donde se encuentran las escenas agrupadas. Por ejemplo, si los archivos html se guardan en *c:/Proyecto/escenas/*, la ubicación de la carpeta */lib* en este caso será en *c:/Proyecto/*.
  - **personalizada**: El usuario define, por medio de un diálogo, la carpeta en que se encontrará el intérprete. Es útil cuando en algún proyecto hay archivos en una variedad de niveles distintos, pero que todos hacen referencia a una misma localización del intérprete.

  Las tres últimas opciones permiten al usuario contar con una versión local del intérprete, de tal forma que sus escenas podrán funcionar aún si no tiene acceso a la red. Adicionalmente, cada vez que el usuario guarde la escena en el editor, el archivo de intérprete relacionado se sobre-escribirá con la última versión, siempre y cuando haya acceso a Internet al momento de guardar.

- **Agregar al HTML**: Al seleccionarlo, se muestra un submenú con tres opciones, y las cuales se encuentran marcadas por defecto.

  - **biblioteca**: Al estar marcado, se guarda el texto con el contenido de la biblioteca también en el archivo *html* como un *script*. Esto es, debajo del cuerpo del archivo html de *DescartesJS*, aparece un bloque `<script></script>` que incluye el nombre de la biblioteca. Si no está marcada la opción, el *html* echa mano del archivo de texto de la biblioteca para obtener el código. De estar marcada, se incluye adicionalmente la información en el bloque `<script></script>` en el *html*.
  - **macro**: De estar marcado, al guardar la escena de Descartes, esta incluye un bloque `<script></script>` con la información del macro, semejante a como lo hace la opción de biblioteca también.
  - **vector**: Hace lo mismo que las dos opciones anteriores pero para el caso de los vectores. Más adelante, en el apartado sobre vectores, se estudia el funcionamiento de este tipo de definiciones. Al estar marcada esta opción, la información sobre los vectores guardados en archivo se volca también en un bloque `<script></script>` en el archivo *html*, de donde puede leerse también la información aún si el archivo de texto no está presente.

- **Lenguaje**: Es un menú en el que se selecciona el lenguaje usado en selectores, controles, etc. Se manejan las siguientes opciones: *español*, *inglés*, *alemán*, *catalán*, *euskera*, *gallego*, *valenciano* y *portugués*.
- **Zoom**: Es un menú que cuenta con tres opciones que permiten controlar el acercamiento al área de trabajo en una escena de Descartes. Es particularmente útil en casos que el tamaño mostrado excede el tamaño de la pantalla y no es posible ver toda la escena completa. Modificar este tamaño es sólo para comodidad del programador en *Descartes*. Es decir, no afectará el tamaño real mostrado de la escena *html*. Las opciones con que cuenta son

  - **Aumentar**: aumenta ligeramente el tamaño.
  - **Disminuir**: disminuye ligeramente el tamaño.
  - **Inicial**: regresa al tamaño original que se tenía antes de cualquier modificación.

  Se puede echar mano de los atajos que se incluyen de algunas de las opciones de *Zoom*. No se mencionan en la presente documentación ya que son sensibles al sistema operativo en que se trabaja.
- **Tema de color**: Es un menú en el que se selecciona el color del fondo detrás del área de trabajo en el editor de *Descartes*. Las opciones que maneja son *clásico* (con el color verde grisáceo característico de *Descartes*), *obscuro*, *claro* y *azul*.
- **Mostrar consola**: al seleccionar esta opción se lanza una ventana llamada *Consola Descartes*. Su función es enlistar los errores de parseo que Descartes encuentra en el código. También sirve para imprimir valores que pueden servir para depurar

el código mediante la instrucción `_Print_()` o `_Trace_()` (son equivalentes), cuyo argumento es una variable de la cual se desea conocer el valor. Igualmente, el argumento puede consistir en una expresión. Por ejemplo, en un ciclo donde varía el valor de una variable *i*, la instrucción `_Print_(i+',',(i+1))` dentro del ciclo pintará en la consola el valor de *i* seguida del valor de *i* tras añadirle una unidad.

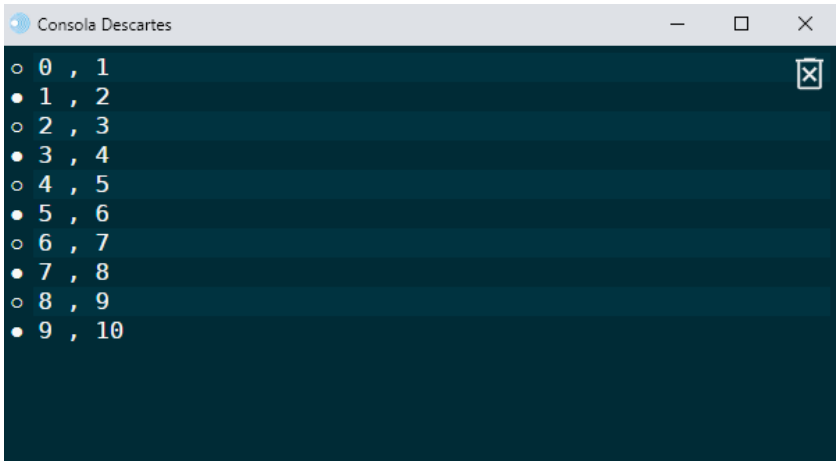


Figura 4.1: Visualización de la consola Descartes, con el ejemplo de imprimir `i` e `i+1` como se mencionó.

La consola incluye un botón, con forma de basurero, en la esquina superior derecha del área de impresión. Se puede usar este botón para borrar los datos impresos.

Note que la consola también permite visualizar errores internos a la programación (por ejemplo, divisiones por cero, raíces de argumentos negativos, y otros). Siempre es buena idea mantenerla abierta cuando se está haciendo programación complicada, para poder estar al tanto de los errores.

Cuando la cantidad de renglones en la consola supera el tamaño de la misma, aparece una barra de desplazamiento vertical para poderla navegar. Siempre que se agregan renglones al final la consola, la barra se desplaza al final de la misma para mostrar siempre la última línea agregada.

4.1.3. El menú *Ayuda*

Este menú incluye dos opciones relacionadas a la información de *Descartes JS*.

- **Documentación**: Lanza la presente documentación desde el sitio web en que se encuentra en el navegador predeterminado.
- **Notas de la versión**: Lanza una ventana en la que se indica el número de última versión y se enlistan los cambios / mejoras de la misma. Adicionalmente, cuenta con un hipervínculo *Versiones anteriores* donde se pueden consultar los cambios históricos de todas las versiones anteriores.

- **Acerca del editor Descartes**: Lanza una ventana que despliega el logo de *Descartes JS*, la versión del editor y de la biblioteca, así como los créditos de la herramienta.

5

El editor de configuraciones

El editor de configuraciones es una ventana nueva en la que propiamente se realiza la programación de un recurso. Consiste en una ventana que se lanza oprimiendo el botón con forma de engrane al extremo derecho del Editor de DescartesJS. Dependiendo del tamaño de la ventana, este botón puede también encontrarse en la parte inferior del editor. La Figura 5.1 muestra la ubicación de dicho botón cuando está a la derecha.

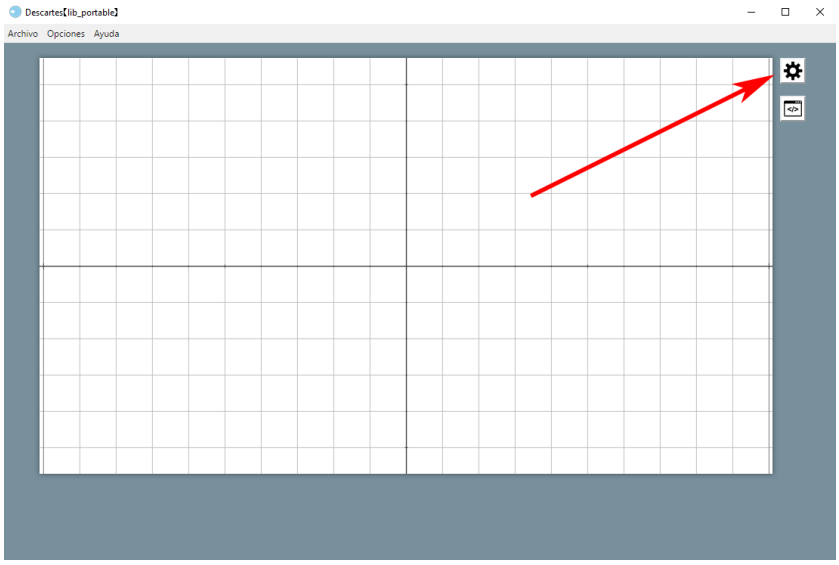


Figura 5.1: Botón para lanzar el editor de configuraciones.

El botón inferior con los corchetes lanza una pantalla en la que se puede editar directamente el código de configuración de la escena, como se muestra en la figura 5.2.

Como se observa, los elementos de la escena se encuentran enumerados y separados con mediante colores de fondo distintos, con el fin de poder leer mejor el código. Este código puede editarse manualmente y sus cambios se implementarán al pulsar *Aceptar*. No obstante, se debe tener cuidado pues un error de dedo en esta situación puede desconfigurar la escena. Una vez que se avance en esta documentación y se conozca más sobre los atributos de cada elemento, se podrá entender mejor el código.

En la figura 5.3 se muestra la ventana del editor de configuraciones. Como se observa, contiene en su parte superior un menú de selectores esenciales para su uso.

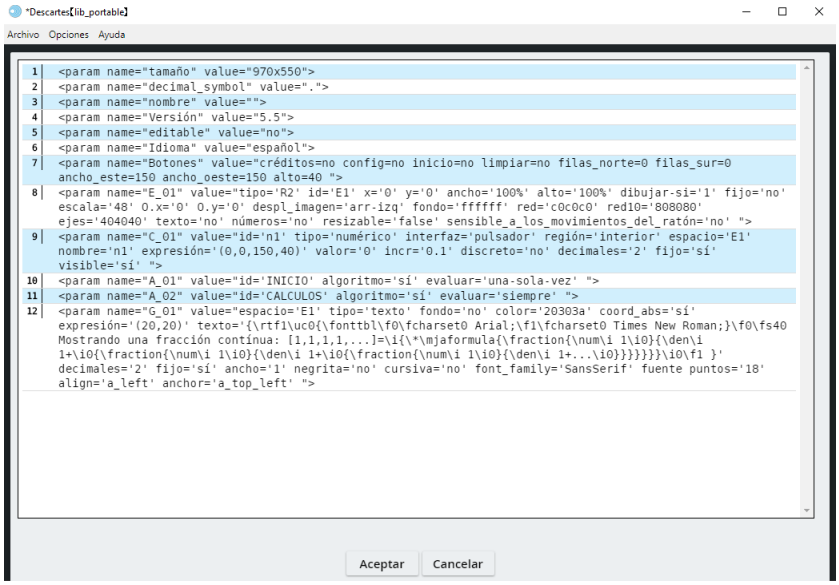


Figura 5.2: Pantalla de edición del código de configuración de escenas.

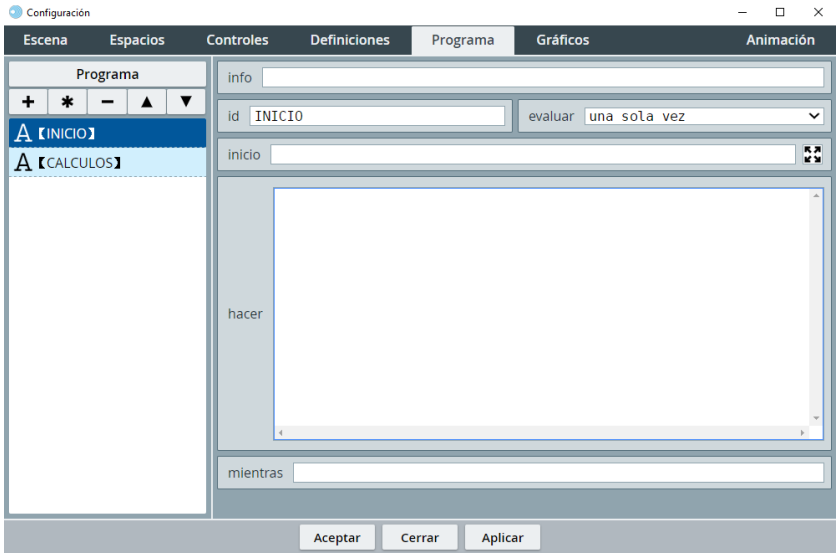


Figura 5.3: Ventana del editor de configuraciones.

5.1. Selectores

El contenido principal del editor de configuraciones consiste en siete selectores para editar diferentes partes del interactivo. A continuación se da una breve descripción de éstos, aunque posteriormente se abordan a profundidad.

El selector *Escena* permite hacer cambios de la interfaz general de la escena con el usuario.

El selector *Espacios* se usa para añadir, duplicar, eliminar o editar las propiedades de los espacios existentes en un interactivo. Los espacios son áreas en las que se muestra material gráfico al usuario.

El selector *Controles* permite añadir, duplicar, eliminar o editar las propiedades de los controles que se encuentran en el interactivo. Dichos controles son aquellos que el usuario puede directamente manipular.

El selector *Definiciones* permite añadir, duplicar, eliminar o editar las propiedades de los elementos que componen la programación como tal del interactivo. Esta parte contiene elementos dentro de los cuales se encuentra el código fuerte dentro de un interactivo.

El selector *Programa* contiene los algoritmos y eventos. Al igual que las definiciones, éstos contienen parte del código que permite el funcionamiento del interactivo. En particular, se refiere a la preparación inicial del interactivo para que esté listo para usarse, pero también a instrucciones que se repiten siempre o dadas ciertas condiciones.

El selector *Gráficos* permite añadir, duplicar, eliminar o editar las propiedades de los gráficos mostrados en los espacios.

El selector *Animación* permite editar las instrucciones y condiciones de una animación en caso de haberla.

Todos estos selectores se abordan a profundidad como capítulos individuales mostrados más adelante.

Cabe mencionar que hay disponible una explicación o información emergente (tipo *tooltips*) para los diferentes elementos dentro de los selectores. Por ejemplo, el panel *hacer* en la figura 5.3 mostrará su tooltip si se posa el mouse sobre el texto *hacer* y se deja ahí un breve momento. La información aparece como un panel pequeño con la información más importante respecto al panel, botón, etc. que se esté consultando. La información emergente desaparece al momento de quitar el mouse. Aunque la mayoría de los elementos de los selectores cuentan ya con información emergente, cabe mencionar que hay algunos que todavía no están disponibles. En la figura 5.4 se muestra el ejemplo de la ayuda emergente para el panel *hacer* del algoritmo *INICIO* que se encuentra en el selector *Programa*. Actualmente, los tooltips vienen con soporte en 10 idiomas distintos: Español, Inglés, Alemán, Catalán, Euskera, Francés, Gallego, Italiano, Valenciano y Portugués.

5.2. Botones del editor de configuraciones

Adicionalmente, el editor de configuraciones cuenta con tres botones en la parte inferior, los cuales se detallan a continuación:

- *Aceptar*: Siempre que se hace un cambio en el editor de configuraciones, éste no se muestra de forma inmediata en el Editor de DescartesJS. Al presionar el botón en cuestión, el editor de configuraciones se cierra y el interactivo se recarga con la última información aplicada.

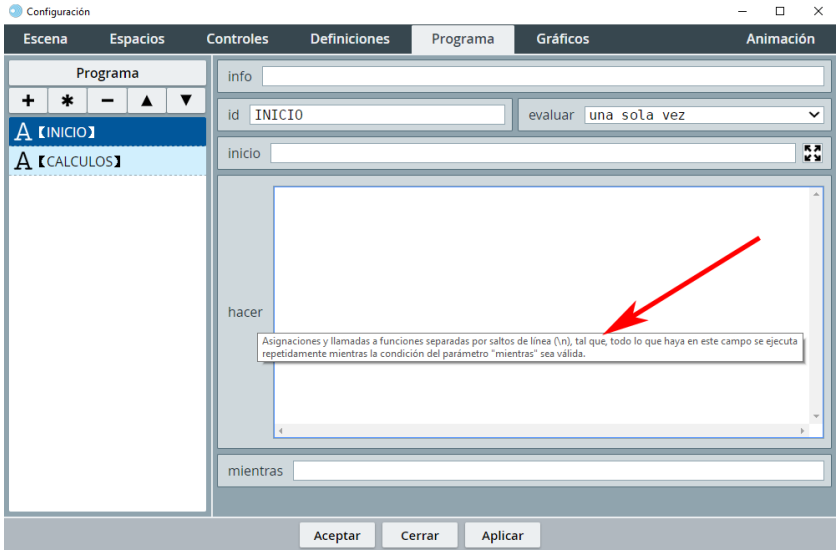


Figura 5.4: Explicación emergente de elementos del editor de configuraciones.

- *Cerrar*: Este botón cierra el editor de configuraciones pero sin aplicar los cambios. Es decir, cualquier cambio hecho no aparecerá en el interactivo tras cerrar el editor de configuraciones. Si se cierra el editor de configuraciones y se vuelve a abrir, los cambios hechos habrán desaparecido.
- *Aplicar*: Este botón aplica los cambios hechos en el editor de configuraciones de tal forma que se muestren en el Editor de DescartesJS. El interactivo se recargará con los últimos cambios hechos. Es decir, hace lo mismo que el botón *Aceptar*, con la salvedad de que el editor de configuraciones se mantiene abierto.

Cuando se usa cualquiera de estos tres botones en el editor de configuraciones, se despliega la ventana del editor principal de DescartesJS asociado a dicho editor de configuraciones. Esto permite que, cuando se están manejando múltiples editores de Descartes simultáneamente, cada uno con su editor de configuraciones, al implementar un cambio se muestra el editor de Descartes al que se le hizo el cambio.

Es importante notar que, cuando se manejen animaciones automáticas (que inician desde el principio de la escena, y no hasta que el usuario lo decide), muchas veces es más conveniente usar el botón *Aceptar* en lugar de *Aplicar*, debido a que el inicio automático de algunas animaciones en el Editor de Descartes JS sólo se podrá ver usando dicho botón (es decir, si se usa *Aplicar* la animación no empezará automáticamente, aún cuando se decida que empiece automáticamente).

También es importante tener en mente que presionar *Aceptar* o *Aplicar* no implica que la escena se guarde. Es decir, se pueden hacer cambios en el Editor de configuraciones y, por ejemplo, presionar *Aceptar*, con lo que el interactivo se recargará con los cambios hechos. Sin embargo, el archivo html no tendrá estos cambios hasta que se utilice la opción de *Guardar* o *Guardar como* en el menú *Archivo*.

6

El selector *Escena*

Escena permite controlar las propiedades más generales de la interfaz. A continuación se muestra una descripción de los parámetros en este selector. En la Figura 6.1

- **título de la página**: Es un campo de texto en que se introduce el título del archivo *html*. Es decir, el contenido de la etiqueta `<TITLE>...</TITLE>` del archivo *html*.
- **ancho**: Es un campo de texto en que se introduce el ancho que tendrá el interactivo en pixeles (en adelante podrá usarse la abreviatura px en lugar de pixeles).
- **alto**: Es un campo de texto en que se introduce el alto que tendrá el interactivo en px.
- **botón créditos**: Es un checkbox (o casilla de verificación) que al estar marcado incluye en el interactivo un botón llamado *créditos* que, al ser oprimido, despliega en una ventana emergente los créditos y licencia de la herramienta de autor DescartesJS.
- **botón config**: Es un checkbox que al estar marcado incluye en el interactivo un botón llamado *config* que, al ser oprimido, despliega una ventana emergente con el código de toda la escena del interactivo. Este código puede embeberse en una página web, por ejemplo, si se desea que dicha página incluya el interactivo de DescartesJS.
- **botón inicio**: Es un checkbox que al estar marcado incluye en el interactivo un botón llamado *inicio* que, al ser oprimido recarga la escena desde el inicio, como si recién hubiera sido abierta, y quitando cualquier modificación que el usuario pudiera haber hecho desde que fue abierta.
- **botón limpiar**: Es un checkbox que al estar marcado incluye en el interactivo un botón llamado *limpiar* que, al ser oprimido borra cualquier rastro gráfico que pudiera haber quedado por interacción del usuario. Los rastros se detallan más adelante.
- **filas al norte**: Es un campo de texto que determina cuántas filas se añadirán al norte del interactivo (en la parte superior) de color gris. Esta área al norte se puede usar para alojar controles. Cuando los botones *créditos* y/o *config* se encuentran presentes, ellos se colocan en una fila al norte independientemente de si está definida en el campo *filas al norte*.
- **filas al sur**: Es un campo de texto que determina cuántas filas se añadirán al sur del interactivo (en la parte inferior) de color gris y que delimitan un área que se puede usar para alojar controles. Cuando los botones *inicio* y/o *limpiar* se encuentran presentes, ellos se colocan en una fila al sur independientemente de si está definida en el campo *filas al sur*.

- **ancho oeste**: Es un campo de texto que determina el ancho de la columna al oeste (izquierda) del interactivo. Esta columna sólo se muestra si efectivamente aloja controles. Los controles y su localización se discute en el apartado Controles.

- **ancho este**: Es un campo de texto con la misma función que el *ancho oeste*, sólo que para el margen al este (derecho) del interactivo.

- **alto filas**: Es un campo de texto mediante el cual la altura en px de cada fila (agregada por *filas al norte* o *filas al sur*).

- **signo decimal**: Es un menú que permite seleccionar si el símbolo decimal será el punto o la coma.

- **mostrar región exterior**: Es un checkbox que, cuando se encuentra marcado, permite lanzar el espacio exterior con un clic derecho del mouse. El espacio exterior consiste en una ventana emergente que aloja controles también. Dicha ventana siempre aloja a los botones *créditos*, *config*, *inicio* y *limpiar*, independientemente de si están visibles en el interactivo principal. Si hay otros controles cuya ubicación está definida como en *exterior*, se mostrarán también en esta ventana emergente. Esto suele usarse para controles con que el programador depura el programa.

- **expandir escena**: Es un menú. Su funcionamiento sólo es evidente cuando la escena se carga en un navegador, dado que en el editor siempre se mantiene un espacio definido por los parámetros *ancho* y *alto*.

  La opción *cubrir* garantiza que una escena dada, cuando se carga, cubre el área del Editor de DescartesJS. Si el área en el navegador es más grande, se considerará un espacio más grande para cubrirla completamente. No se reescala o estira el espacio, sino que se hace más grande. El espacio mantiene la misma escala que el original.

  La opción *escalar* asegura que si el espacio en el navegador es más grande que el de la escena original, se reescale todo el espacio hasta ajustarse al tamaño en el navegador.

- **imagen del cargador**: Es un campo de texto en donde se indica la ruta, relativa a la carpeta que contiene al archivo html, a una imagen que se muestra cuando se carga la escena en un navegador. Si este campo de texto se deja vacío, se deja el logo de DescartesJS como la imagen a mostrar. La imagen escogida por el usuario no se visualiza en el editor. Esta funcionalidad está pensada para verse directamente en un navegador.

Figura 6.1: Parámetros del selector *Escena*.

Los botones *config*, *inicio* y *limpiar* pueden llegar a ser útiles en el proceso de programación, pero no suelen dejarse en interactivos en su etapa final. Adicionalmente, dado que estos botones se encuentran dos en un renglón superior y los otros dos en el renglón inferior, su presencia resta espacio al área propiamente del interactivo.

Los parámetros *filas al norte*, *filas al sur*, *ancho oeste* y *ancho este* pueden pensarse como márgenes que flanquean al interactivo. Ellos pueden contener controles y los parámetros mencionados se encargan precisamente de definir las dimensiones de dichos márgenes.

7

El selector *Espacio*

Espacios permite controlar todos los espacios (inicialmente mostrados como planos cartesianos) de la escena. Es sobre estos espacios que se trazan los textos y gráficos mostrados al usuario.

Por defecto, siempre se pinta un espacio bidimensional inicialmente al ejecutar el Editor de Descartes. Este espacio se visualiza en el selector *Espacio* normalmente como *E1* en el panel con la lista de espacios. En la Figura 7.1 se puede observar el selector *Espacio*.

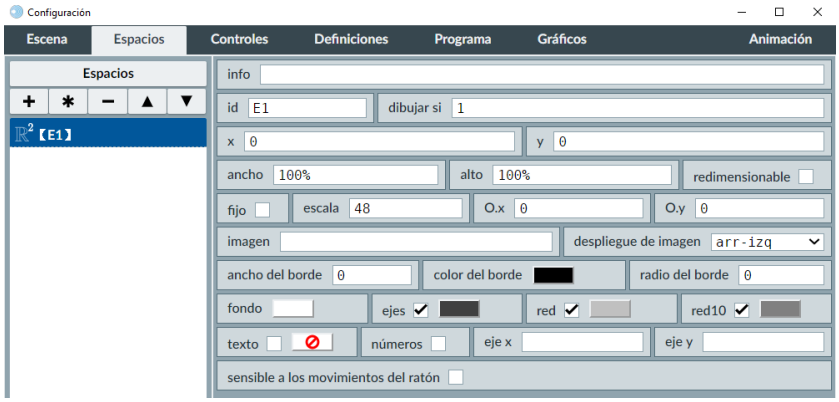


Figura 7.1: Visualización del selector *Espacio*.

El espacio mostrado por defecto tiene un identificador *E1*, que se observa en el campo de texto '*id*'. La mayoría de los otros controles son campos de texto que se encuentran vacíos. Ello se debe a que éste es el espacio principal y, si no se determina su posición y tamaño, por defecto abarcará el tamaño de la escena especificado en el selector *Escena*.

7.1. Espacio R2 o bidimensional

A continuación se muestra una descripción de los controles de espacios bidimensionales del selector *Espacio*:

- **info**: información sobre el espacio. Permite introducir una breve descripción del programador sobre el espacio en cuestión. No será mostrada al usuario, sino que es sólo un recordatorio al programador. Adicionalmente, si este campo contiene una descripción, ésta será la que se mostrará en el panel de elementos del selector en la izquierda del editor de configuración, con el objeto de poder ubicar más fácilmente

un elemento determinado cuando se cuenta con una gran cantidad de ellos. El campo de texto *info* está presente en todos los elementos del editor de configuraciones de DescartesJS y su función siempre es la de recordar qué es o para qué se usa tal elemento. Por lo mismo, en adelante ya no se hará una descripción específica de este campo de texto cuando vuelva a estar presente.

- **id**: el identificador del espacio. Es el nombre con el que el programa identifica al espacio en cuestión. Los identificadores, de forma general, suelen empezar con letras. El primer carácter de un identificador no debe ser un número.
- **dibujar si**: indica una condición booleana que se evalúa para determinar si el espacio se muestra o no. Muchos distintos elementos de Descartes tienen este control, lo cual lo hace muy versátil para mostrar u ocultar elementos. Por ejemplo, si uno introduce una expresión como 2 == 1, el programa compara 2 con 1. Como no son iguales, considera que la expresión es falsa y vale 0. Por ello, el espacio no sería trazado. Si, por el contrario, se utiliza 1 == 1, al ser verdadera la expresión se le otorga un valor de 1. En este caso, el espacio sí sería trazado. Igualmente, se puede introducir directamente 1 ó 0 en este control para trazarlo o no trazarlo, respectivamente. El abordaje de las condiciones booleanas se profundiza en el apartado condicionales y operadores booleanos.
- **x**: la coordenada horizontal, en pixeles, de la esquina superior izquierda del espacio. Se comienza a medir desde la esquina superior izquierda del Editor para contar. Así pues, la esquina superior izquierda del espacio estará a x pixeles a la derecha de la esquina superior izquierda del espacio de trabajo del Editor.
- **y**: la coordenada vertical, en pixeles, de la esquina superior izquierda del espacio. Se comienza a medir hacia abajo desde el borde superior del espacio de trabajo del Editor. Es decir, el espacio estará a y pixeles por debajo del margen superior del espacio de trabajo del Editor.
- **ancho**: el ancho, en pixeles, del espacio. Si se agrega el sufijo %, se tomará el número indicado como el porcentaje de ancho del tamaño de la escena en lugar de considerarlo como el número de pixeles.
- **alto**: el alto, en pixeles, del espacio. Al igual que para el ancho, si se agrega el sufijo %, se tomará el número indicado como el porcentaje de altura del tamaño de la escena en lugar de considerarlo como el número de pixeles.
- **redimensionable**: es un checkbox que permite, de estar marcado, un comportamiento más flexible de los espacios. Cuando está activado, el ancho y alto del espacio puede definirse a través de variables. Es también posible asignarle un tamaño mayor al espacio que aquél que tiene la escena.
- **fijo**: es un checkbox que, de estar marcado, fija el espacio cartesiano de tal manera que el usuario no puede mover ni su posición ni su escala con el mouse o pantalla táctil. Si el espacio no está fijo, el usuario puede mover el espacio cartesiano arrastrándolo y también puede hacer un acercamiento o alejamiento arrastrando el

mouse mientras oprime el botón derecho del mismo. Estas funciones se inhabilitan si el checkbox en cuestión está activado. Es importante mencionar que aún con este checkbox marcado, es posible controlar la escala y posición de un espacio cambiando dentro del programa los valores de sus variables intrínsecas de escala y localización. Para más detallas al respecto se puede consultar el apartado sobre variables de espacio.

- **escala**: indica el número de pixeles que componen una unidad en el plano cartesiano. Por defecto (si el campo de texto está vacío) el valor es 48. Es decir, una unidad del plano cartesiano estará compuesta por 48 pixeles. Disminuir este valor corresponde a hacer un alejamiento, mientras que aumentarlo corresponde a un acercamiento.
- **O.x**: es el desplazamiento en pixeles, sobre la horizontal, del origen del plano cartesiano hacia la derecha. Por ejemplo, un valor de 100 en este campo de texto implica que el plano cartesiano no estará situado en el centro del editor, sino 100 pixeles a la derecha.
- **O.y**: es el desplazamiento en pixeles, sobre la vertical, del origen del plano cartesiano hacia abajo. Por ejemplo, un valor de -100 en este campo de texto implica que el plano cartesiano no estará situado en el centro del editor, sino 100 pixeles hacia arriba del centro.
- **imagen**: corresponde a la ruta relativa de un archivo de imagen que puede presentarse como fondo del espacio. Descartes maneja sin problema imágenes png y jpg. Por ejemplo, si un archivo *portada.png* está en una carpeta */images* que está en el mismo nivel que el archivo de Descartes, entonces se introduciría *images/portada.png* en el campo de texto para desplegar dicha portada como fondo.
- **despliegue de imagen**: es un menú que determina cómo desplegar una imagen que es de menores dimensiones que el espacio que la contiene.

  -**arr-izq**: la imagen se despliega pegada a la esquina superior izquierda del espacio.

  -**expandir**: la imagen se estira horizontal y verticalmente hasta cubrir el espacio.

  -**mosaico**: la imagen se repite en forma de mosaico hasta cubrir todo el espacio.

  -**centrada**: la imagen se despliega en el centro del espacio.
- **ancho del borde**: es un campo de texto en el que se introduce el grosor en pixeles de un marco al espacio en cuestión. Por defecto vale 0, valor que implica que no se trazará un marco.

  Es preciso notar que, si el espacio tiene iguales dimensiones que la escena (establecidas mediante los parámetros *ancho* y *alto* del selector Escena), el borde no se notará en el margen derecho y el inferior. Ello se debe a que los bordes no modifican las dimensiones del espacio, sólo su posición, lo que recorre el espacio a la derecha y hacia abajo un número de pixeles correspondiente al ancho del borde. Por ejemplo,

para tener un espacio con borde de ancho 2 y que quede justo contenido en el área de la escena, el usuario debe indicar un ancho y alto de espacio 4 px menor que el ancho y alto de la escena. Es decir, para el ancho se reservan 2 px para el margen izquierdo y 2 para el derecho; y lo mismo ocurren en el alto para el margen superior e inferior.

- **color del borde**: es un botón que lanza la Herramienta del control de colores. El color elegido se aplicará al borde del espacio cuyo grosor se define en *ancho del borde* recién mencionado.
- **radio del borde**: es un campo de texto en el que se indica, en pixeles, el radio de un arco circular que se trazará en las esquinas del espacio. Por defecto vale 0, que implica esquinas en ángulo recto. Para valores mayores que 0, las esquinas serán redondeadas y el arco circular será del tamaño definido en este parámetro.
- **fondo**: es un botón que lanza una ventana emergente para controlar el color del fondo del espacio. Para mayor información sobre esta ventana puede consultar el apartado Herramienta del control de colores.
- Conjunto de checkbox que determinan si se muestran los ejes, redes, texto y números del espacio cartesiano. Cada checkbox se acompaña de un botón que lanza la Herramienta del control de colores para asignar el color al elemento en cuestión. Los checkbox son:

  -**ejes**: los ejes cartesianos.

  -**red**: la red del plano cartesiano.

  -**red10**: una red de trazos más gruesos que se muestra cada 10 unidades.

  -**texto**: muestra las coordenadas del punto en que se hace clic en el espacio. Este checkbox no tiene control de color.

  -**números**: muestra algunas marcas de valores sobre los ejes. Este checkbox no tiene control de color.
- **eje-x**: muestra un título para las abscisas al lado de su eje.
- **eje-y**: muestra un título para las ordenadas al lado de su eje.
- **sensible a los movimientos del mouse**: forza a que se recalculen las variables del mouse cada vez que el mouse se desplaza, independientemente de si su botón se pulsa o no. Se sugiere no abusar de esta funcionalidad, ya que la cantidad de cálculos internos que se tienen que hacer puede crecer al estar implementada.
  Esta funcionalidad también se encuentra disponible para los espacios tridimensionales.

7.2. Espacio R3 o tridimensional

Hasta ahora se ha lidiado con espacios bidimensionales. También existe la posibilidad de usar espacios tridimensionales. Éstos son un tipo de espacio en que se pueden colocar

objetos tridimensionales. Como se muestra en la Figura 7.2, una vez que se ha añadido uno de estos espacios y se ha pulsado el botón aplicar, aparece hasta arriba del editor de configuraciones un nuevo selector llamado *Gráficos 3D*, en el que se pueden incluir gráficos de tipo tridimensional. Para mayor información se puede consultar el apartado sobre los gráficos 3D.

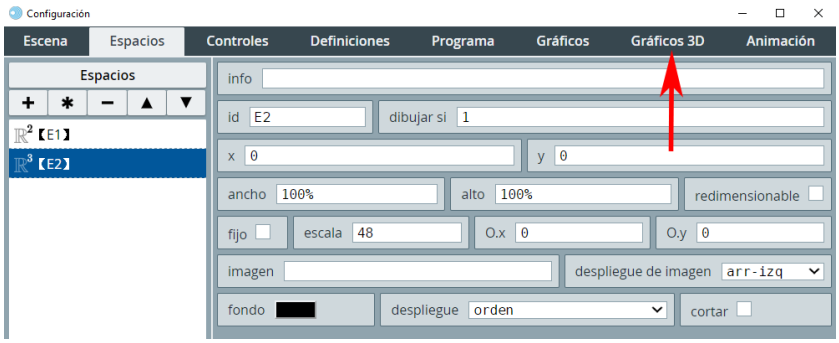


Figura 7.2: Visualización de la localización de los *Gráficos 3D*.

La mayoría de los controles de estos espacios son iguales a los de espacios bidimensionales, con algunas excepciones, a saber:

- **despliegue**: al usar tres dimensiones, los objetos colocados en el espacio pueden estar unos frente a otros. Para determinar cuáles quedan frente a cuáles hay 3 métodos distintos:
  - **orden**: pinta los objetos de atrás hacia adelante. Involucra menos cálculos y es el más rápido, pero suele fallar para objetos muy grandes.
  - **pintor**: dibuja primero los objetos que son tapados por otros. Involucra más cálculos y es más lento, pero es más preciso que el método *orden*.
- **cortar**: es un checkbox que por defecto se encuentra inactivo. Es útil cuando cuando superficies u objetos tridimensionales se cortan, pues permite un correcto despliegue de los objetos. Si los objetos desplegados no se cortan, no es necesario marcar esta opción.

Los espacios 3D se manipulan pulsando y arrastrando. De esta forma se producen giros para poder visualizar los objetos desde diferentes perspectivas. Si se pulsa el botón derecho del mouse y se arrastra hacia arriba mientras éste está pulsado, se realizará un acercamiento. Si el arrastre es hacia abajo, se realizará un alejamiento.

7.3. Espacio HTMLIFrame

Es un espacio en el que se puede mostrar alguna página web con formato html. Este tipo de espacio contiene un campo *archivo* que los demás no tienen. En él se teclea la dirección web o local al html que se desea abrir en dicho espacio.

7.4. Panel de espacios en el selector

Adicional a los controles del selector *Espacio*, existe un panel a la izquierda que permite crear nuevos espacios, seleccionar los espacios a editar, duplicar espacios, eliminarlos y alterar el orden de los mismos, como se muestra en la Figura 7.3.

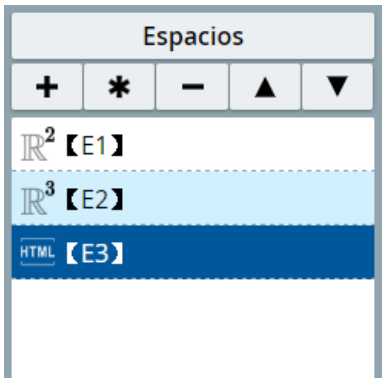


Figura 7.3: Panel izquierdo del selector *Espacio*.

- : *Espacio*: Un botón gris claro con la palabra *Espacio* dentro. Lanza una ventana de introducción de texto donde se puede realizar la edición manual de todos los espacios del interactivo. En algunos casos, cuando se desea hacer cambios similares en varios espacios, puede resultar engorroso tener que moverse de espacio en espacio para realizar el mismo cambio. En este caso, la ventana mostrada permite realizar los cambios en el código directamente para ahorrar tiempo. La desventaja es que hay que ser muy cuidadoso al usar esta ventana puesto que la eliminación o adición accidental de caracteres especiales puede hacer al interactivo inservible. Por lo mismo, se recomienda no usar esta opción hasta tener cierto grado de conocimiento del funcionamiento de Descartes. Esta funcionalidad de edición simultanea de varios espacios se encuentras presente en los demás selectores, excepto *Escena* y *Animación*. Y siempre consiste en un un botón claro con el nombre del selector en cuestión.
- +: Un botón que lanza una ventana en la cual se puede introducir el nombre de un nuevo espacio, el tipo de espacio, e incluye un botón *aceptar* para crearlo. Los espacios creados de esta forma aparecen con la misma configuración de un espacio por defecto. Cabe mencionar que se pueden agregar también espacios *3D*. A diferencia de los *2D*, éstos permiten representar objetos tridimensionales. Para añadir un espacio tridimensional, en el diálogo *agregar* hay que seleccionar *R3* en lugar de *R2*, asignar un nombre al espacio y aceptar.
- *: Un botón que lanza una ventana en la cual se puede introducir el nombre de un espacio que, más allá del nombre, será un copia idéntica del espacio previamente seleccionado. Este botón es útil cuando varios espacios han de compartir mucha información.

- -: Un botón que lanza una ventana de confirmación para eliminar el espacio seleccionado.
- pulsador de orden: Dos botones: uno consiste en una flecha hacia arriba y el otro en una flecha hacia abajo. Si se cuenta con una lista de espacios, en ocasiones es necesario alterar el orden en que aparecen. Una vez seleccionado un espacio, que se muestra con un fondo azul oscuro como en el ejemplo de la Figura 7.3, presionar la flecha hacia arriba resultará en subirlo un nivel, mientras que presionar la flecha hacia abajo resultará en bajarlo. El orden en que se muestran los espacios en el panel es importante dado que los espacios se pintan en el orden en que aparecen. En el ejemplo de la Figura 7.3, primero se trazará el espacio *E1* y sobre éste se trazará el *E2*.

Cabe hacer notar que cuando se selecciona un espacio en la lista del panel a la izquierda del selector, la edición tendrá lugar sobre el espacio seleccionado.

Hagamos un ejercicio para practicar los espacios. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Espacio. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*. Se recomienda aplicar los cambios tras cada paso y guardar frecuentemente al hacer los ejercicios.

8

El selector *Gráficos*

Nos adelantamos al último de los selectores, el selector *Gráficos* pues los gráficos constituyen el siguiente paso natural después de los espacios. Los gráficos consisten de una variedad de figuras y textos que permiten enriquecer el interactivo.

Muchos de estos gráficos comparten funcionalidad entre ellos. Así pues, esta funcionalidad se describe independientemente de cada gráfico que la contiene al final del presente tema y se omite en la descripción de controles. Es decir, la descripción de controles de un gráfico en particular sólo involucra los controles únicos a ese gráfico. No obstante, cuando se hace referencia a alguna funcionalidad común a todos los gráficos, se incluyen hipervínculos a la explicación de la misma.

Todos los gráficos tienen en común un panel a la izquierda semejante al del selector *Espacio*. En este panel se muestra una lista de los gráficos existentes. Al igual que en el caso del selector *Espacio*, cuenta con un botón para añadir un gráfico nuevo, uno para duplicar el gráfico seleccionado, uno control para mover el gráfico seleccionado arriba o debajo de la lista y un botón para eliminar el gráfico seleccionado. En la Figura 8.1 se observa el selector *Gráficos*.

Figura 8.1: Visualización del selector *Gráficos*.

Los gráficos sólo pueden existir dentro de un espacio. De tal forma que existe un menú en la parte superior del panel izquierdo que despliega los espacios existentes. Este menú funciona como un filtro. Por defecto, muestra un símbolo de asterisco (*), que quiere decir que se muestran todos los gráficos de todos los espacios, que corresponde a cuando el filtro está desactivado. Si se tienen varios espacios y el menú se coloca a alguno de éstos, sólo se mostrarán los gráficos en el espacio seleccionado en el menú, lo que permite una más fácil identificación de los gráficos en caso de haber muchos.

Cuando hay más de un gráfico en un espacio y los gráficos se intersecan, se pintarán en el orden en que aparecen en el panel izquierdo. Es decir, el de hasta arriba será el pri-

mero en pintarse y el de hasta abajo, que será el último en pintarse, cubriría a los que se encuentran arriba de él en la lista.

También al igual que en el selector *Espacio* hay un botón con el texto *Gráficos* que lanza una ventana con el código de los gráficos en forma de texto. Es importante que, aunque el filtro de algún espacio en particular esté activo, este botón mostrará el texto de todos los gráficos de todos los espacios.

8.1. Gráficos 2D

Éstos constituyen los gráficos más comúnmente usados, que son bidimensionales. Se pintan sólo en espacios de dos dimensiones.

8.1.1. Gráfico *Ecuación*

Este gráfico consiste en una curva correspondiente a una ecuación. En la figura 8.2 se muestra un gráfico tipo *Ecuación*, mientras que en la figura 8.3 se observa cómo deben ajustarse los controles del gráfico para que quede así trazado.

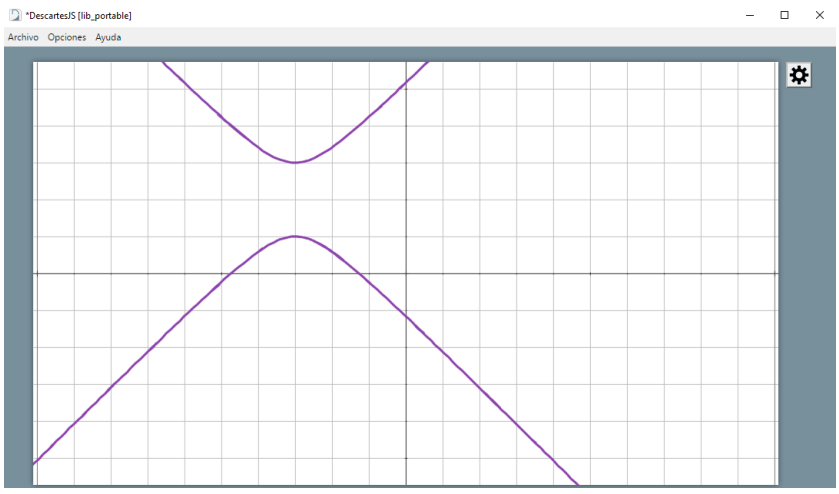


Figura 8.2: Ejemplo del gráfico *Ecuación*.

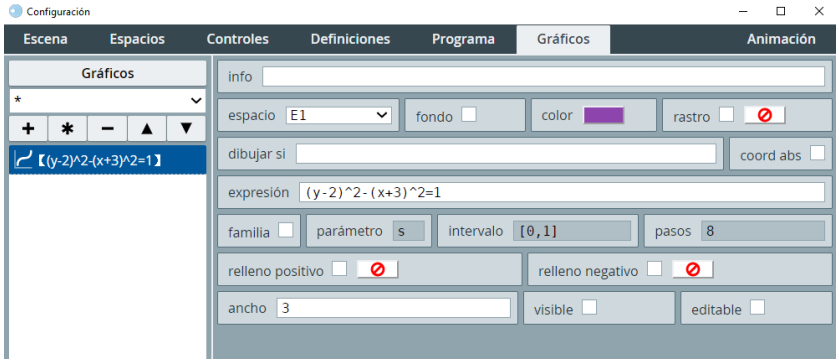


Figura 8.3: Configuración del ejemplo del gráfico *Ecuación*. El color usado es 8E44AD.

- **expresión**: es un campo de texto donde se introduce la ecuación que desea graficarse. Ocupa las variables y y x para referirse a la ordenadas y abscisas respectivamente en el plano. Es realmente una ecuación la que se puede trazar. Por ejemplo, $y = x$ es la que aparece por defecto, pero bien podría graficarse $x^2 + \frac{y^2}{2} = 1$ para una elipse, o incluso $x = 3$. Es decir, la ecuación no tiene que forzosamente ser una función de y en x.
- **relleno positivo**: es un checkbox que, de estar marcado, permite que se rellene el área entre el eje y la gráfica con el color determinado por el control de color al lado del checkbox. Por ejemplo, para la expresión $y = x^3$, se rellenará la parte entre el eje x y la gráfica que queda por encima del eje con el color seleccionado. Si la gráfica es de la forma $x = y^3$, se rellenará el área entre el eje y y la gráfica a la derecha de este eje. El botón de color sólo será accesible si el checkbox está marcado.
- **relleno negativo**: es un checkbox que, de estar marcado, permite que se rellene el área entre el eje y la gráfica con el color determinado por el control de color al lado del checkbox. Por ejemplo, para la expresión $y = x^3$, se rellenará la parte entre el eje x y la gráfica que queda por debajo del eje con el color seleccionado. Si la gráfica es de la forma $x = y^3$, se rellenará el área entre el eje y y la gráfica a la izquierda de este eje.
- **visible**: es un checkbox que permite visualizar en el interactivo la ecuación que está graficada si se encuentra marcado. La ecuación se visualiza en una barra en la parte inferior del interactivo.
- **editable**: es un checkbox que funciona sólo si el control *visible* está marcado. Permite, además de ver la ecuación en el interactivo, que ésta sea modificada, resultando en que el gráfico cambie dinámicamente.
- Los controles **fondo**, **dibujar-si**, **coord abs**, **rastro**, **familia**, **parámetro**, **intervalo**, **pasos**, **ancho**, **info** y **estilo de línea** se discuten al final del presente tema. Los botones de controles del color del gráfico, y el de los controles *relleno positivo* y *relleno negativo* son los genéricos explicados en la herramienta de control de colores.

Hay un detalle importante a considerar cuando se usa el operador potencia en un gráfico tipo ecuación. Funciones del tipo `y=x∧p`, con p no entero, sólo están definidas para $x>0$. Así, por ejemplo, será necesario introducir `y=(x∧3)∧(1/5)` si se desea obtener la gráfica completa incluidos los valores negativos del argumento x. Si se introduce sólo `y=x∧(3/5)`, la gráfica presentada aparecerá sólo donde $x>0$. Esto tiene que ver con la forma en que se interpretan las raíces cuando se usa el operador ∧, así que se explica con mayor detalle en el apartado sobre el operador potencia.

8.1.2. Gráfico *Curva*

Este gráfico consiste en varios trazos que unen puntos determinados por un parámetro. El parámetro es una variable que adopta valores en un intervalo en un número dado de pasos. De tal forma que se puede ver como una serie de segmentos uno tras otro unidos por sus extremos. En la Figura 8.4 se muestra un ejemplo de una curva, y en la Figura 8.5 se muestra cómo se configura dicho ejemplo.

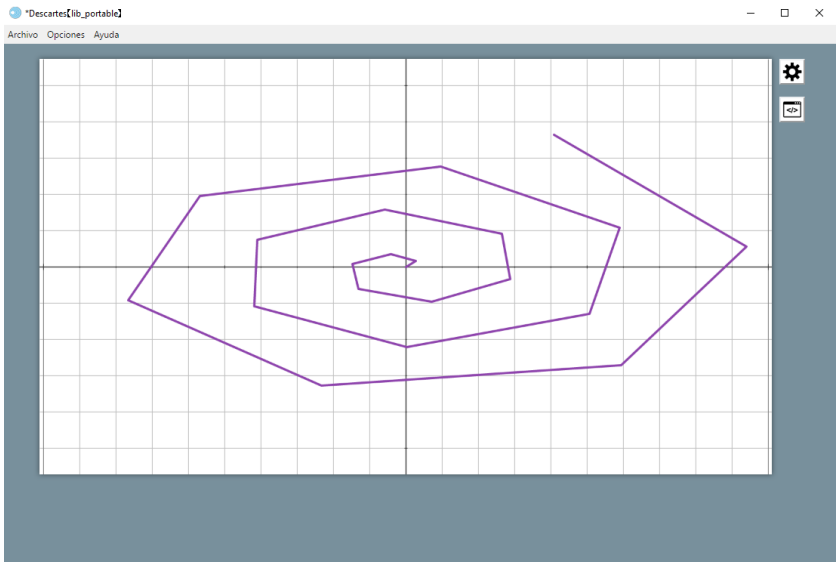


Figura 8.4: Ejemplo del gráfico *Curva*.

Figura 8.5: Configuración del ejemplo del gráfico *Curva*. El color usado es 8E44AD.

- **expresión**: es un campo de texto donde se escribe la sucesión de puntos que representan el par ordenado de los extremos de los segmentos que componen la curva. Para el gráfico *curva*, la expresión es del tipo (t, t). Esto es, es un par ordenado de funciones de la variable t, que es el parámetro del que depende la curva, y que se detalla abajo.
- **parámetro**: es un campo de texto donde se introduce el nombre de la variable de la cual depende la curva. La variable por defecto es t. Es importante hacer la diferencia entre el parámetro de una familia (cuya variable por defecto es s) y el parámetro de la curva, puesto que puede prestarse a confusión.
- **visible**: es un checkbox que permite visualizar en el interactivo la expresión de la curva graficada si se encuentra marcado. Esta expresión se visualiza en una barra en la parte inferior del interactivo.
- **editable**: es un checkbox que funciona sólo si el control *visible* está marcado. Permite, además de ver la expresión de la curva en el interactivo, que ésta sea modificada, resultando en que el gráfico cambie dinámicamente.

Ahora que conocemos ya dos gráficos distintos, aprovechemos para hacer un ejercicio breve. Se pueden aprovechar los gráficos vistos para observar cómo un polígono de muchos lados puede aproximar una circunferencia.

El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en GráficosCurva. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

8.1.3. Gráfico *Punto*

Este gráfico consiste en un único punto cuyas coordenadas se dan de forma explícita. En la Figura 8.6 se muestra un ejemplo de este gráfico. En la Figura 8.7 se muestra la con-

figuración para lograr dicho ejemplo. Recuerde que puede arrastrar el plano cartesiano para moverlo.

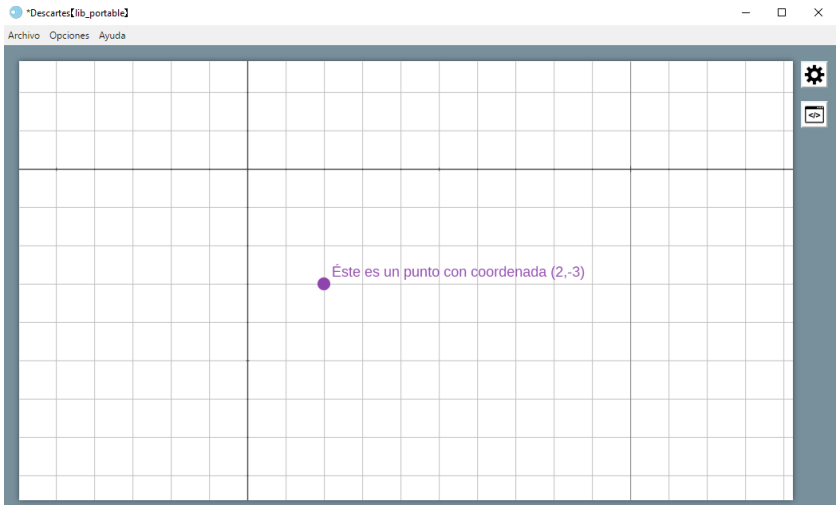


Figura 8.6: Ejemplo del gráfico *Punto*.

Figura 8.7: Configuración del ejemplo del gráfico *Punto*. El color usado es 8E44AD.

- **expresión**: es un campo de texto en donde se colocan las coordenadas del único punto.
- **decimales**: es un campo de texto donde se introduce un número entero correspondiente al número de decimales que habrán de mostrarse en caso que el texto contenga alguna variable cuyo valor se quiera visualizar.
- **fijo**: es un checkbox que determina si siempre se muestran el número de decimales elegido en el control *decimales*, aún cuando no sean significativos. Siempre se mostrarán los decimales elegidos si este checkbox está marcado. De lo contrario, sólo se mostrarán si son significativos.

Hagamos un breve ejercicio para ver el funcionamiento de los puntos. Aprovecharemos esta oportunidad para revisar las familias de gráficos (en este caso, sólo del punto en cuestión), un poco sobre los textos y algo sobre coordenadas relativas y absolutas. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Gráficos Punto. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

8.1.4. Gráfico *Segmento*

Este gráfico consiste en un segmento determinado por las coordenadas de sus extremos. En la Figura 8.8 se muestra un ejemplo de este gráfico. En la Figura 8.9 se muestra la configuración necesaria para lograr este ejemplo.

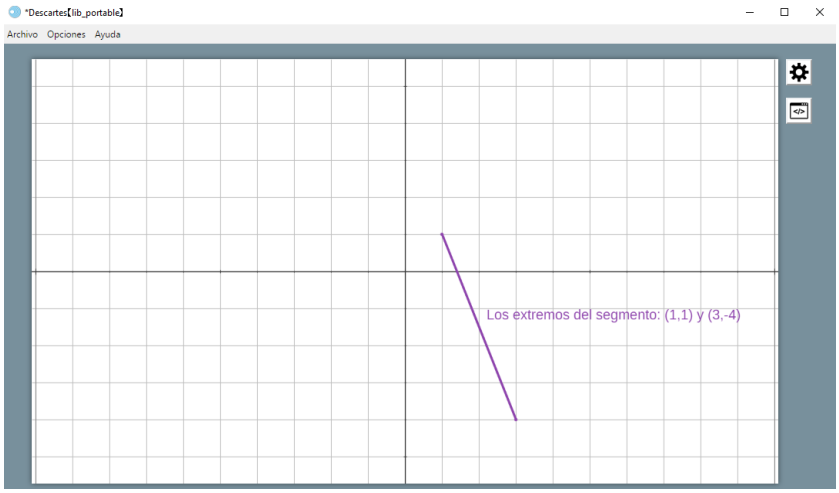


Figura 8.8: Ejemplo del gráfico *Segmento*.

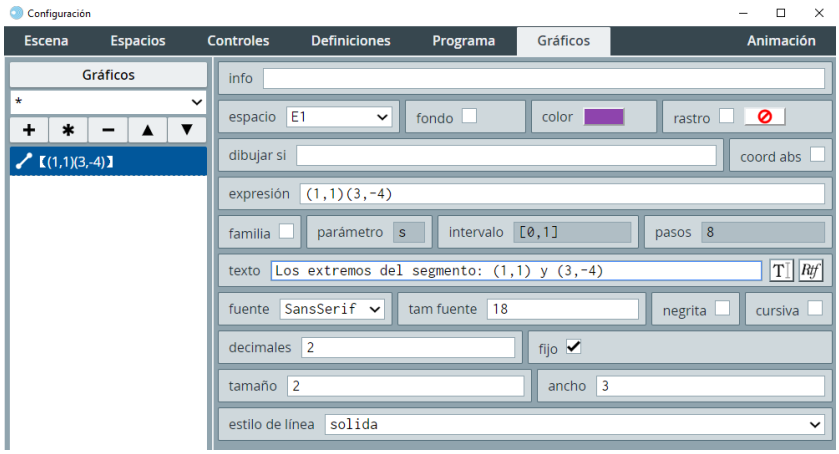


Figura 8.9: Configuración para el ejemplo del gráfico *Segmento*. El color usado es 8E44AD.

- **expresión**: es un campo de texto en el cual se introducen un par de coordenadas correspondientes a los extremos del segmento.

- **ancho**: es un campo de texto donde se introduce un número entero correspondiente al ancho en pixeles del segmento mismo.

Hagamos un breve ejercicio para practicar el uso de segmentos. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Gráficos Segmento. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En el ejercicio anterior se pudo notar algo interesante. Los gráficos en coordenadas absolutas, dado que se encuentran definidos con respecto a la esquina superior izquierda del interactivo, no son susceptibles a movimientos del plano cartesiano del espacio, mientras que los que están definidos en coordenadas relativas sí. Esto representa una ventaja de usar coordenadas absolutas para aquellos que se desee permanezcan estáticos.

8.1.5. Gráfico *Polígono*

Este gráfico es trazado a partir de coordenadas de puntos individuales que representan los vértices de un polígono. Las coordenadas de los vértices en secuencia determinan los segmentos que formarán los lados del polígono. En la Figura 8.10 se muestra un ejemplo de este tipo de gráfico. En la Figura 8.11 se muestran la forma de configurar dicho gráfico.

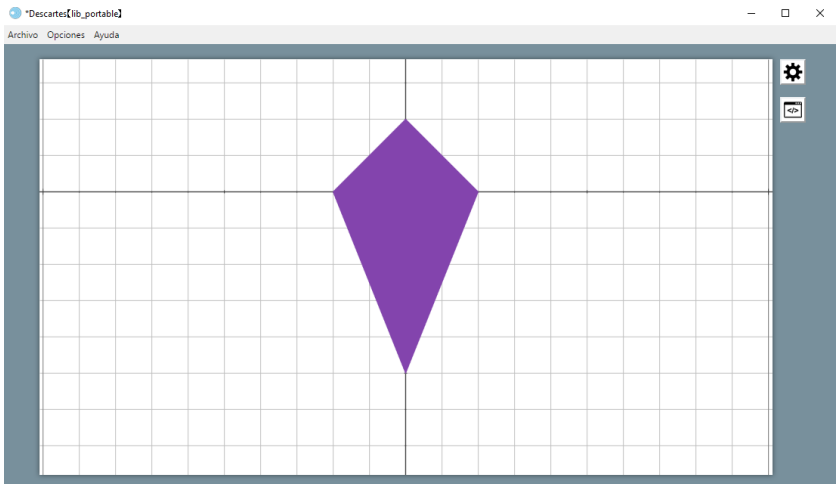


Figura 8.10: Ejemplo del gráfico *Polígono*.

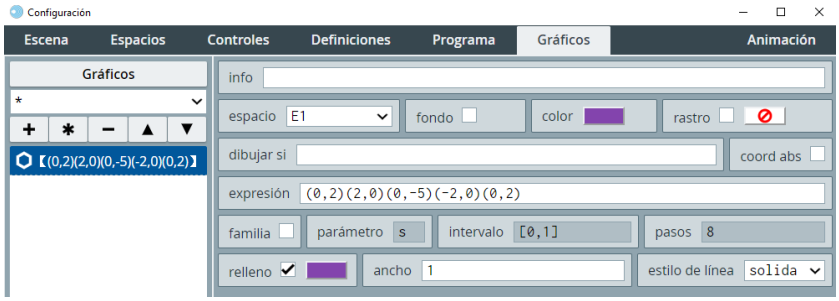


Figura 8.11: Configuración del ejemplo del gráfico *Polígono*. El color usado es 8E44AD

- **expresión**: es un campo de texto en el cual se introducen las coordenadas de los vértices que componen el polígono. Los lados del mismo se trazarán del primer vértices introducido al segundo, del segundo al tercero y así sucesivamente. A pesar del nombre *polígono*, bien puede consistir en una figura abierta si el primer vértice y el último no coinciden.

- **relleno**: es un checkbox que de estar marcado hará que el interior del polígono tenga un relleno con color. El color se determina a partir del botón de selección de color al lado izquierdo del checkbox, cuya funcionalidad se detalla en la herramienta de control de colores.

Aprovechemos para construir un espacio en el que se podría incluir un texto explicativo en el interior (algo así como una ventana emergente). Es conveniente que dicha ventana tenga un marco, y el marco será el polígono mismo. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Gráficos Polígono. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio tuvimos la oportunidad de ver cómo algunas variables internas de Descartes (en este caso el ancho y alto de un espacio) pueden ser útiles. Una descripción más detallada de estas variables se encuentra en el apartado sobre variables de espacio. Observamos nuevamente la utilidad de las coordenadas absolutas respecto a las relativas.

8.1.6. Gráfico *Rectángulo*

Este gráfico constituye una forma abreviada de trazar un rectángulo sin necesidad de usar el gráfico *Polígono* para ello. En la Figura 8.12 se muestra un ejemplo de este gráfico y en la Figura 8.13 se muestra cómo debe configurarse dicho ejemplo.

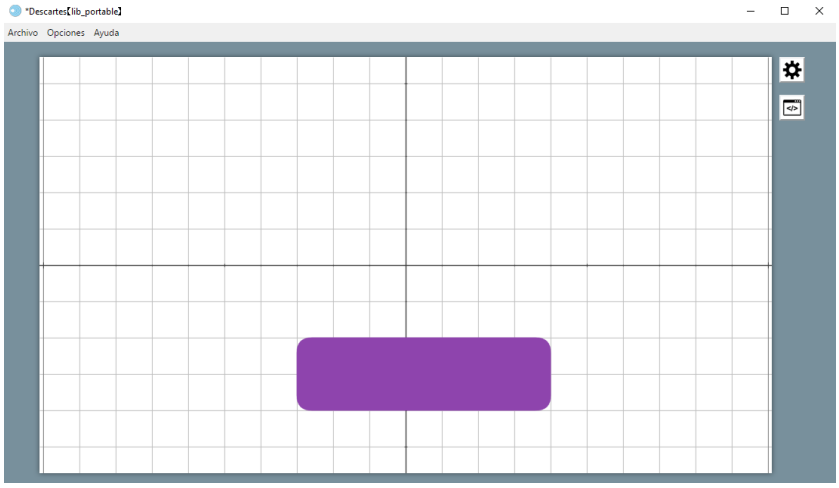


Figura 8.12: Ejemplo del gráfico *Rectángulo*.

Figura 8.13: Configuración del ejemplo del gráfico *Rectángulo*. El color usado es 8E44AD.

- **expresión**: es un campo de texto en el que se introducen 4 coordenadas flanqueadas por paréntesis. Las primeras dos entradas corresponden a un vértice del rectángulo. La tercera corresponde a las unidades (o pixeles) a la derecha en que quedará el vértice opuesto al primero. La cuarta corresponde a las unidades (o pixeles) hacia arriba (o hacia abajo en caso de usar coordenadas absolutas) en que quedará el vértice opuesto. Cabe mencionar que es posible usar valores negativos para estas dos últimas coordenadas.
- **radio del borde**: es un campo de texto en el que se introduce el número de pixeles del radio del borde que se usa cuando se desea que las esquinas del rectángulo se encuentren redondeadas. Mientras mayor sea el número de pixeles, más grande será la porción del rectángulo redondeada.

Hagamos un ejercicio para implementar el rectángulo nuevamente como el borde de un espacio bidimensional en una escena, al igual que se hizo en el ejercicio correspondiente al gráfico *Polígono*. El interactivo de este ejercicio, junto con las instrucciones para

lograrlo, se encuentran en Gráficos Rectángulo. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Podemos observar de este ejercicio que el definir un rectángulo es mucho más eficiente usando el gráfico *Rectángulo* en vez del *Polígono*.

8.1.7. Gráfico *Flecha*

Este gráfico es casi idéntico al *segmento*, con la salvedad de que en la última coordenada indicada en su campo de texto *expresión* se pinta una punta de flecha. En la Figura 8.14 se muestran un ejemplo de este gráfico. En la Figura 8.15 se muestra la forma de configurar este ejemplo.

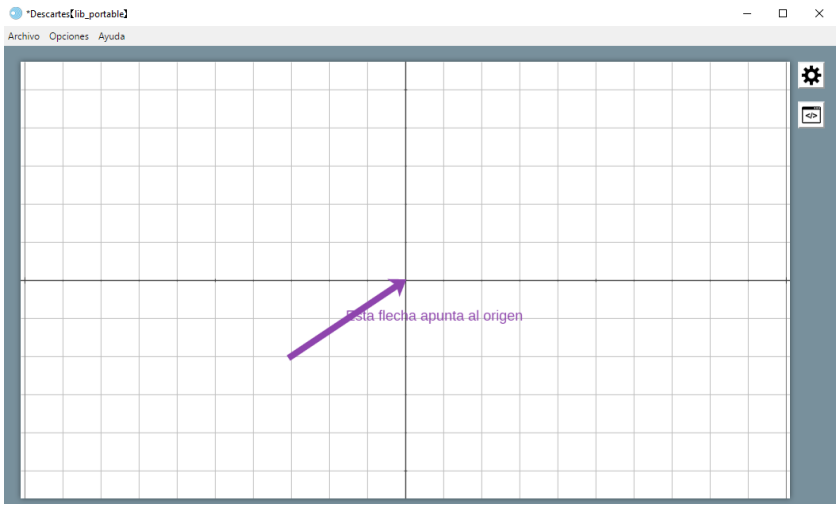


Figura 8.14: Ejemplo del gráfico *Flecha*.

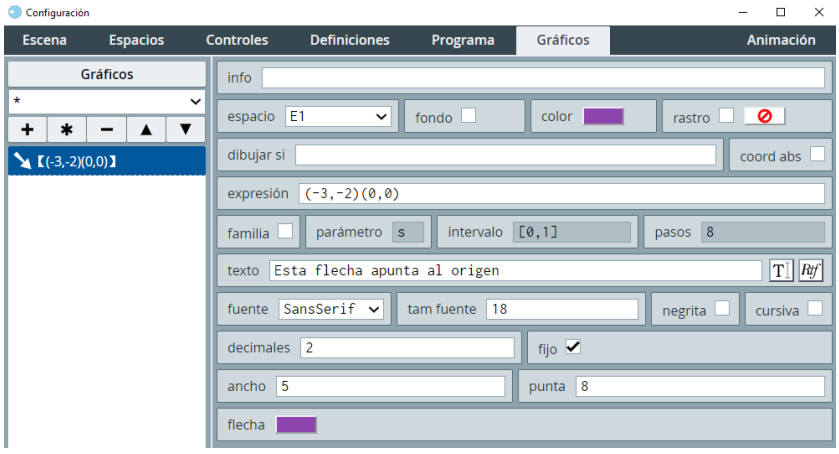


Figura 8.15: Configuración del ejemplo del gráfico *Flecha*. El color utilizado es 8e44ad.

- **expresión**: es un campo de texto en donde se indica el par de coordenadas de los extremos de la flecha. Como se mencionó anteriormente, la última coordenada corresponde al extremo donde está la punta de la flecha.
- **ancho**: es un campo de texto que determina el grosor en pixeles del ancho de la flecha. En versiones actuales existen dos parámetros *ancho* para la flecha. Sólo el segundo es funcional.
- **punta**: es un campo de texto que determina el tamaño de la punta de la flecha en pixeles.
- **flecha**: es un botón mediante el cual se accede a la herramienta de control de colores para seleccionar un color para la flecha. A diferencia del control *color* del mismo tipo de gráfico, éste controla el interior de la flecha, mientras que el otro controla el borde de la misma.

Debido a la similitud entre el segmento y la flecha, no se hará un ejercicio particular para la flecha.

8.1.8. Gráfico *Arco*

Este gráfico consiste en una parte de una circunferencia. Aunque se puede hacer lo mismo con un gráfico tipo *ecuación*, este nuevo gráfico permite una introducción más cómoda en que se proporciona simplemente el centro, el radio y el ángulo que habrá de barrer el arco. En la Figura 8.16 se muestra un ejemplo con este gráfico. En la Figura 8.17 se muestra la configuración para obtener dicho ejemplo.

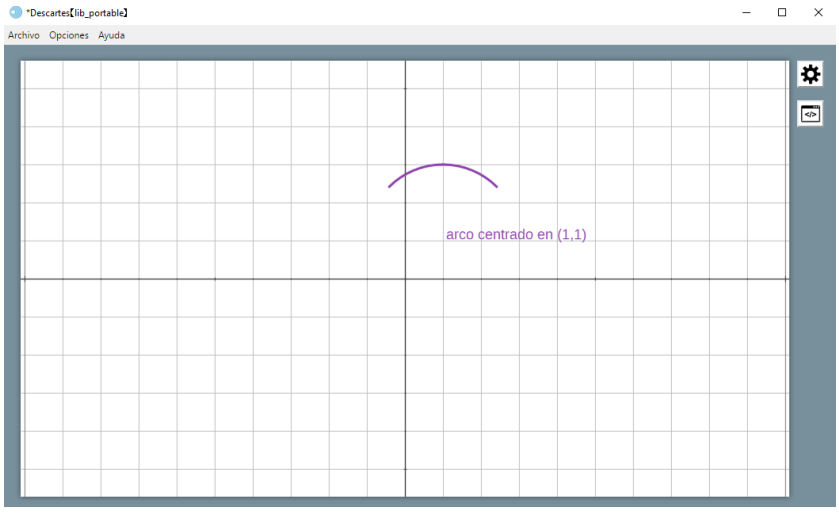


Figura 8.16: Ejemplo del gráfico *Arco*.

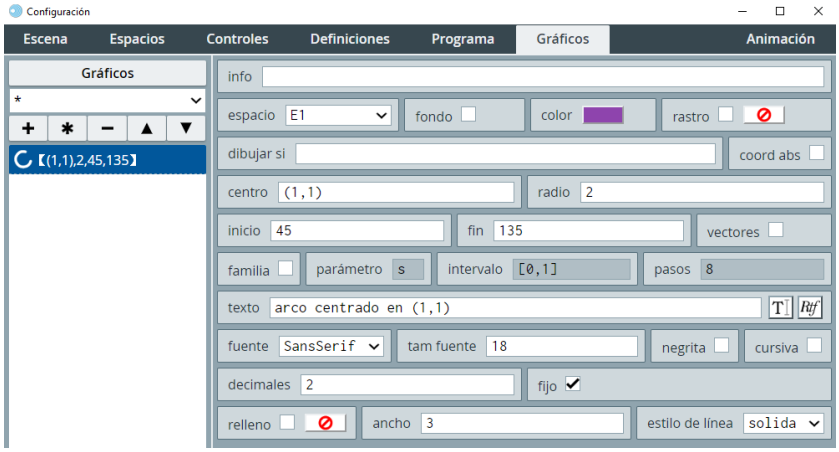


Figura 8.17: Configuración del ejemplo del gráfico *Arco*.

- **centro**: es un campo de texto donde se indican las coordenadas del centro del arco. Por ejemplo (3,2) implica que el centro del arco estará 3 unidades desplazado a la derecha del origen y 2 unidades hacia arriba.
- **radio**: es un campo de texto donde se indica el radio que habrá de tener el arco.
- **inicio**: es un campo de texto donde se introduce el ángulo inicial en grados a partir del cual se empieza a trazar el arco.
- **fin**: es un campo de texto donde se introduce el ángulo final en grados hasta donde se trazará el arco.
- **vectores**: es un checkbox que cambia la forma de introducir el *inicio* y *fin*. Si no está marcado, el inicio y fin representan los grados de inicio y fin del arco. Si se encuentra marcado, *inicio* y *fin* reciben una coordenada de un vector que representa un lado de inicio del arco y el otro el lado del fin. Por ejemplo, se puede introducir (3,2) como *inicio*. Estos vectores siguen teniendo como origen el centro del arco.
- **relleno**: es un checkbox que determina si el arco habrá de estar rellenado. Incluye un control de color para elegir el color a usar. El funcionamiento de este control de color se detalla en la herramienta de control de colores. Como se observará en el ejercicio, el relleno para el caso del arco es el área flanqueada entre los radiovectores que lo definen (dados por los ángulos o por los vectores como tal) y el arco formado por el radio dado.

Hagamos un ejercicio para practicar la funcionalidad de este tipo de gráfico. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Graficos Arco. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Se pudo observar en este interactivo que los arcos son una forma más inmediata de trazar una parte de una circunferencia cuando se cuenta con el centro y el radio de la misma.

En ocasiones, puede resultar útil considerar a esta parte de la circunferencia como flanqueada por vectores (por ejemplo, cuando se trazan ángulos flanqueados por segmentos). De ahí que sea posible introducir un arco mediante la modalidad *vectores*. Igualmente, si se cuenta con el centro y radio de una circunferencia, es más fácil introducirla como un arco de 360°que mediante una ecuación.

8.1.9. Gráfico *Texto*

Este gráfico consiste en texto que se puede introducir con el fin de explicar algo particular de una lección. Por ejemplo, con este texto se pueden incluir instrucciones, explicaciones de fenómenos, preguntas, etc. En ocasiones se puede inclusive usar con el objeto de mostrar datos temporalmente al programador para depurar una escena.

Anteriormente, este gráfico no contaba con coordenadas relativas. Cambios recientes en el editor ahora permiten usar este tipo de coordenadas. Gran parte del funcionamiento está detallado en el apartado sobre la herramienta de introducción de textos. En la Figura 8.18 se muestra un ejemplo de un texto con una fracción continua. En la Figura 8.19 se muestran cómo es la configuración para dicho ejemplo. Se presenta también una imagen de la ventana de introducción de texto enriquecido usada para el ejemplo en la Figura 8.20.

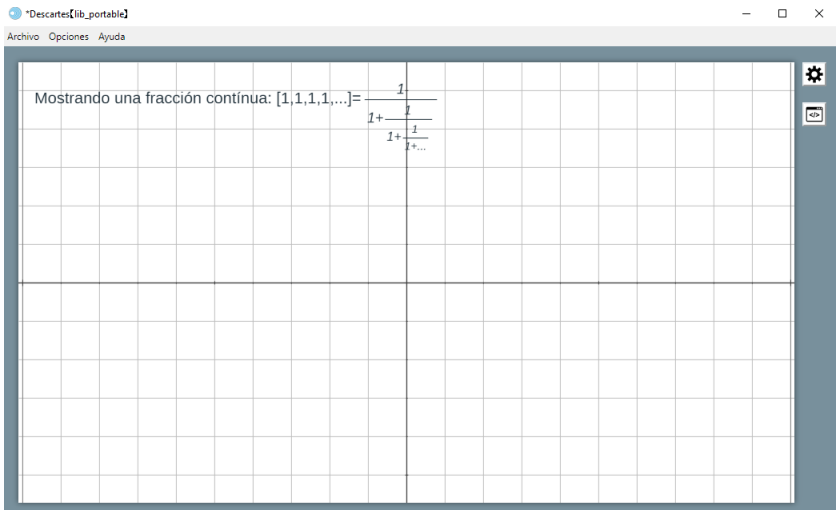


Figura 8.18: Ejemplo del gráfico *Texto*.

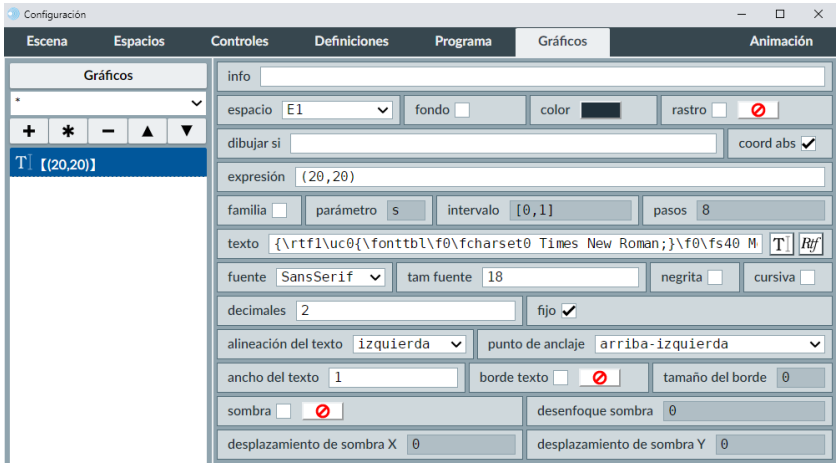


Figura 8.19: Configuración del ejemplo del gráfico *Texto*. Note que se usa texto enriquecido.

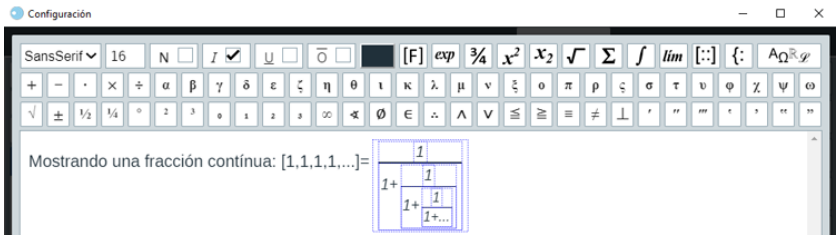


Figura 8.20: Ventana de texto enriquecido para el ejemplo del gráfico *Texto*.

- **expresión**: es un campo de texto donde se introduce el par de coordenadas de anclaje del texto entre corchetes [y], o bien, paréntesis (y). Se puede escoger el punto de anclaje del texto de tal manera que esté en el centro del texto, o en su esquina inferior derecha, etc. Ello se logra con el menú *punto de anclaje* que se describe en breve.
- **alineación del texto**: es un menú que sólo sirve si el texto es simple, no enriquecido. Cuando un cierto texto largo rebasa el ancho en px especificado en el parámetro *ancho*, se genera una o más líneas. El menú en cuestión determina si la alineación de estas líneas es a la izquierda, si van centradas, o si se alinean a la derecha.
- **punto de anclaje**: Al igual que para *alineación del texto*, cuando hay saltos de línea dependiendo del *ancho del texto*, se puede pensar que el texto se encuentra contenido en una caja invisible. El menú en cuestión determina la relación entre las coordenadas de la *expresión* del texto y la caja. Por ejemplo, la opción *arriba-derecha* hará que la esquina superior derecha de la caja sea la que se ancle a las coordenadas indicadas en la *expresión* del texto.
- **ancho del texto**: es un campo de texto donde se introduce el ancho en pixeles del texto. Cuando se usa texto simple, es posible tener líneas muy largas. En este caso

el ancho determinará hasta qué límite se permitirá que aparezca el texto antes de introducir una nueva línea. Esta funcionalidad sólo es válida para texto simple, no enriquecido. En caso de introducir texto enriquecido, el cambio de línea deberá hacerse de forma manual en la ventana de introducción de texto, y el parámetro *ancho* deberá tener un valor de 1.

- **borde texto**: es un checkbox que traza un borde al texto del color elegido en el control de color a su lado. Para más detalles sobre el control de color se puede consultar el apartado de la herramienta del control de colores.

  Cuando el parámetro *borde texto* se encuentra desmarcado, se inactiva su parámetro asociado *tamaño del borde*.

- **tamaño del borde**: es un campo de texto en el que se introduce el grosor, en pixeles, del borde. Si se deja su valor por defecto de cero, pero el checkbox *borde texto* se encuentra marcado, el grosor del mismo será definido automáticamente por Descartes.

- **sombra**: es un checkbox que cuando está marcado permite trazar una sombra del texto. Viene acompañado de un control para definir el color de la sombra. Para más detalles sobre este control, consulte el apartado de la herramienta del control de colores. Cuando la sombra está activa, cobran sentido y se activan los siguientes parámetros:

  - **desenfoque sombra**: es un campo de texto en el que se introduce el factor de desenfoque de la sombra. Un valor de cero trazará una sombra con fronteras muy definidas. Al aumentar este valor, se pierde la nitidez de la sombra.
  - **desplazamiento de sombra X**: es un campo de texto donde se indica el número de pixeles de distancia a la cual la sombra aparecerá a la derecha del texto.
  - **desplazamiento de sombra Y**: es un campo de texto donde se indica el número de pixeles de distancia a la cual la sombra aparecerá por debajo del texto.

A continuación haremos un ejercicio que involucra el gráfico tipo *texto* para aclarar posible dudas. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Graficos Texto. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

8.1.10. Gráfico *Imagen*

Este gráfico consiste en una imagen tipo *jpg, png, gif, svg* (imagen tipo vectorial) o *webp* que puede insertarse en el interactivo. La imagen debe estar en la misma carpeta que el interactivo o en una subcarpeta. Cabe mencionar que las imágenes *webp* funcionan en todos los navegadores modernos; y en Safari sólo funciona a partir de la versión *Big Sur* de *macOS*. En la Figura 8.21 se muestra un ejemplo de un gráfico tipo imagen. En la Figura 8.22 se muestra la configuración necesaria para lograr dicho ejemplo. Note que la imagen

usada en este ejemplo se guarda en una carpeta *images* a la altura del archivo html del interactivo, y su nombre es *1.png*.

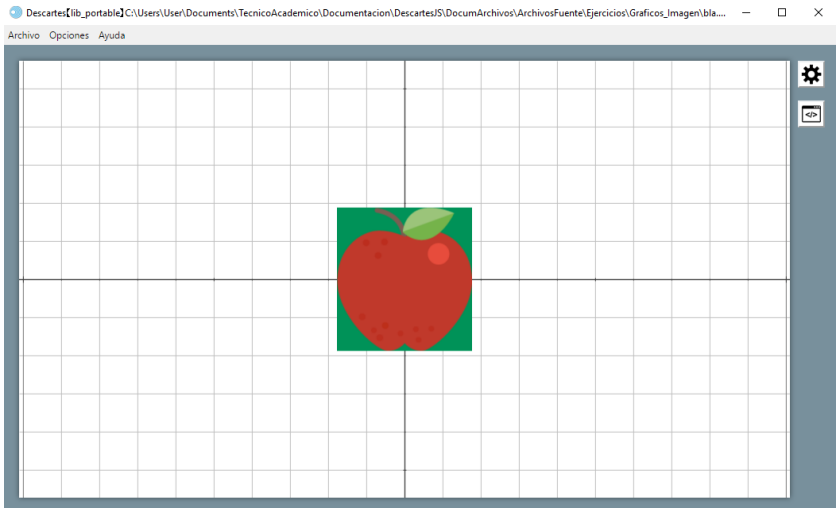


Figura 8.21: Ejemplo del gráfico *Imagen*.

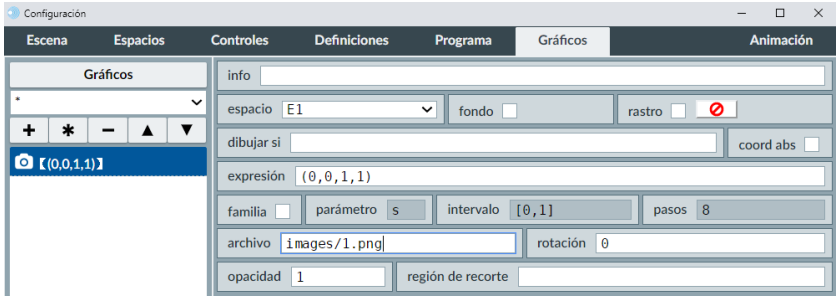


Figura 8.22: Configuración del ejemplo del gráfico *Imagen*.

- **expresión**: es un campo de texto en el que se introducen las coordenadas donde habrá de mostrarse la imagen en el interactivo. Por defecto son dos coordenadas y corresponden a dónde estará situada la esquina superior izquierda de la imagen. No obstante, se pueden introducir cuatro entradas en los paréntesis, en donde la tercera y cuarta entradas corresponden al factor de escala del ancho y alto de la imagen. En caso de definir las cuatro entradas, las dos primeras ya no marcan la esquina superior izquierda de la imagen, sino su centro. La tercera y cuarta entradas (las escalas horizontal y vertical) pueden adoptar valores negativos, en cuyo caso la imagen se invertirá horizontal y/o verticalmente.
- **archivo**: es un campo donde se indica la ruta al archivo de imagen. En caso de incluirse en una subcarpeta, se usa la diagonal normal (/) para separar los niveles.

Es posible agregar una imagen que no esté explícitamente disponible como un archivo como tal. Esto se logra agregando un *script* dentro del archivo html de la escena. La etiqueta de apertura del *script* es:
`<script type="descartes/imagen" id="nombre_de_la_imagen" >`.
Así, después del *id* se introduce entre comillas el nombre a usar. Este mismo nombre es el que se especifica en el parámetro *archivo* dentro del gráfico tipo *imagen* en el editor de configuraciones de *DescartesJS*. El nombre de la imagen debe tener asociada la extensión del archivo mediante el que se generó el código en el script. El código en el *script* debe ser de imagen codificada en *base64*. Para conseguir el código correspondiente a partir de un archivo de imagen, se puede usar la página `https://www.base64-image.de/`, en la cual se alimenta la imagen y la página devuelve el código que se ha de copiar dentro de las etiquetas *script* de apertura y cierre.
Este abordaje es útil si se quiere que la imagen se encuentre dentro del mismo código html del archivo de la escena.

- **rotación**: es un campo de texto donde se introduce el ángulo en grados en que aparece rotada la imagen.
- **opacidad**: es un campo de texto donde se introduce un valor o expresión entre 0 y 1. El valor de 0 corresponde a una opacidad mínima, por lo que la imagen será totalmente transparente. El valor de 1 corresponde a una opacidad máxima, es decir, una transparencia nula.
- **región del recorte**: es un campo de texto donde se introduce una expresión del tipo `(x,y,ancho,alto)`. Mediante esta expresión, el usuario puede hacer que se vea sólo un recorte de una imagen, en lugar de la imagen completa. *x* e *y* corresponden a las coordenadas horizontal y vertical de la esquina superior izquierda donde comenzará el recorte de la imagen. *ancho* y *alto* corresponden a el ancho y alto del recorte, medidos respecto a la esquina superior izquierda definida.
  Así, por ejemplo, si se cuenta con una imagen de 300px por 150px, y se requiriera sólo mostrar la región central de la misma que corresponde horizontalmente al tercio medio y verticalmente también al tercio medio, en el parámetro *región de recorte* se tendría que introducir `(100,50,100,50)`. Con esta expresión, horizontalmente se toma del pixel 100 al 200 (el tercio horizontal), y verticalmente se toma del pixel 50 al 100 (el tercio vertical).

Hagamos un ejercicio para practicar el uso de este tipo de gráfico. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Graficos Imagen. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio se observó que es posible introducir imágenes en los interactivos. A éstas se les puede controlar su tamaño y proporción, su colocación y su rotación. Recuerde que aunque todos los parámetros en el ejercicio son números, también pueden usarse

variables que permiten hacer estos cambios de forma interactiva. No obstante, esto se verá cuando se aborden ejercicios más avanzados.

Es importante tener en cuenta que es necesario guardar el archivo html (si éste ha sido recién creado) y recargarlo antes de que las imágenes se visualicen. Ello se deba a que es sólo hasta que se ha guardado el archivo que Descartes puede determinar a partir de qué ubicación abordar la ruta relativa donde se encuentra la imagen. Así pues, si sólo se aplican los cambios sin haber antes guardado la escena, ésta no será mostrada.

Imagen de un espacio: Es posible generar una imagen a partir de un espacio si como nombre del archivo de la imagen se usa el identificador del espacio y como extensión se usa `image`. Suponga, por ejemplo, que cuenta con un espacio con identificador *E1* y, sobre él, otro espacio con identificador *E2* que lo cubre. *E1* cuenta con ciertos elementos gráficos como textos, curvas, etc. Si ahora se crea una imagen en *E2* con el texto `E1.image` en su parámetro *archivo*, se pintará la imagen del espacio *E1* en el espacio *E2* respetando las reglas ya mencionadas para las imágenes. Por ejemplo, si su *expresión* cuenta con cuatro entradas, se centrará sobre el punto determinado por las dos primeras y se escalará según las últimas dos. A la imagen se le puede, además, definir su opacidad, rotarla, etc. Esto resulta particularmente útil cuando se requiere una imagen dinámica (que responde a controles manipulados por el usuario), compuesta por varios gráficos, y que por ejemplo se desea repetir como una familia. Es importante tener en mente que, para este propósito, el espacio cuya imagen se desea proyectar debe quedar debajo de aquél en el que se proyecta en la lista del panel izquierdo de espacios en el selector *Espacio*. Y es preciso notar que sólo los gráficos contenidos en el espacio a proyectar se manifiestan en la imagen. Por ejemplo, el color de fondo o el borde que pudiera tener este espacio no se pintarán como parte de la imagen del mismo.

8.1.11. Gráfico *Macro*

Un macro es un archivo de texto que incluye partes de una escena de Descartes correspondientes a elementos en los selectores *Definiciones*, *Programa* y *Gráficos*. Para generar dicho archivo de texto es necesario tener una escena de Descartes guardada en alguna ubicación y que tenga los elementos a mostrar en los selectores antes mencionados.

En la Figura 8.23 se muestra un ejemplo de un gráfico tipo macro. En la Figura 8.24 se muestra la configuración necesaria para lograr dicho ejemplo. Note que el macro usado en este ejemplo se lee de una carpeta *macros* a la altura del archivo del interactivo, y su nombre es *macr.txt*. La flecha mostrada en el editor, de hecho, no está presente en panel del selector *Gráficos*, así que realmente es leída del macro.

Cabe notar dos cosas importantes. Aunque el macro viene incluido como un gráfico, realmente va más allá de solamente manipular gráficos. Recordemos que involucra también los elementos de *Definiciones* y *Programa*. Por otra parte, existe un macro en el selector *Gráficos* y otro en el selector *Gráficos 3D*. Los dos funcionan de manera muy similar, aunque hay diferencias leves en los parámetros de cada uno. Sólo se abordarán los macros

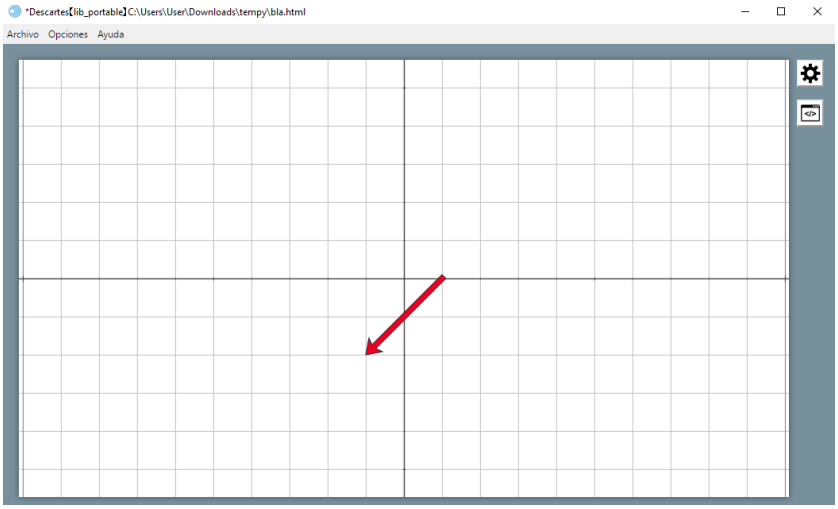


Figura 8.23: Ejemplo del gráfico *Macro*.

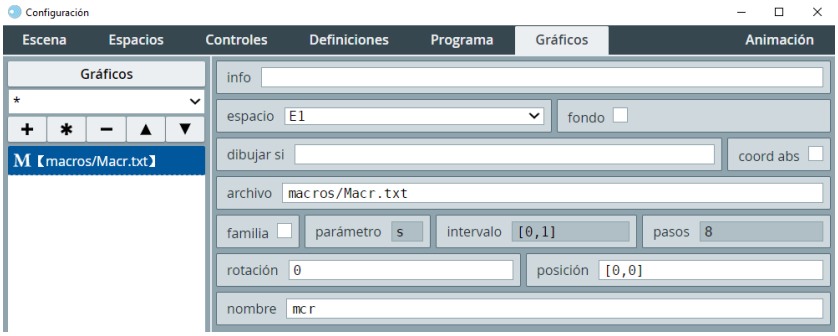


Figura 8.24: Configuración del ejemplo del gráfico *Macro*.

en dos dimensiones en la presente documentación, dado que las diferencias entre ambos se explican solas.

- **expresión**: es un campo de texto en el que se introduce la ruta al macro en cuestión, el cual se puede encontrar a la misma altura del interactivo que lo contiene o en una subcarpeta. Se recuerda usar la diagonal sencilla para bajar niveles hacia las subcarpetas en caso que sea necesario.
- **rotini**: es un campo de texto en el que se introduce una expresión para la rotación en grados de los gráficos que pudiera incluir el macro. Note que para el macro en tres dimensiones, también se incluye *rotfin*.
- **posini**: es un campo de texto en el que se introduce una coordenada de desplazamiento de los elementos gráficos del macro en caso de haberlos. La coordenada ha de escribirse flanqueada por corchetes cuadrados. Note que para el macro en tres dimensiones, también se incluye *rotfin*. Los parámetros de rotaciones y posiciones iniciales y finales para gráficos tridimensionales se pueden abordar en este vínculo.

- **nombre**: es un campo de texto donde se introduce el nombre que se le dará al macro. Cuando un elemento no gráfico (por ejemplo, algún elemento del selector *Definiciones* o *Programa* contenido en un macro ha de ser llamado, se debe incluir como prefijo el nombre del macro seguido de un punto y luego la función en cuestión. Por ejemplo una función *f1()* en un macro llamado *mcr* debe ser llamada como *mcr.f1()*.

En el menú principal *Archivo* del editor de Descartes existe la opción *Exportar a*, dentro de la cual hay una subopción *macro de Descartes*. Cuando se selecciona dicha opción, se lanza una ventana emergente en la que se puede seleccionar la ubicación del macro en cuestión. Se recomienda guardar el macro ya sea a la misma altura del html del interactivo o en una subcarpeta.

La función del macro es simplificar el diseño de escenas subsecuentes que compartan ciertas características con entre sí. Por ejemplo, considere que se quiere colocar un marco que flanquee las escenas en varios interactivos, cada uno guardado como un archivo html diferente. El marco compuesto en el ejemplo de gráficos, será el mismo para todos y sería necesario repetir el código en cada interactivo. Si el marco está compuesto de muchos elementos, ello implicaría colocar una gran cantidad de elementos gráficos en cada interactivo. Si son muchos interactivos, tendríamos una gran cantidad de código repetido. Así pues, resulta cómodo usar un mismo macro que sea leído por los diferentes interactivos para no tener que repetir mucho código en cada uno de ellos.

Adicionalmente, es posible pasar información de la escena principal a las variables del macro. Ello se logra haciendo asignaciones, en la escena principal, a la variable <*nombre del macro*>.<*nombre de la variable en el macro*>. Por ejemplo, suponga que un macro con nombre *macr* echa mano de una variable *ancho*. En la escena principal (la que llama al macro), la asignación `macr.ancho=2` le asignará el valor de 2 a la variable *ancho* dentro del macro llamado *macr*.

Hagamos un ejercicio para practicar el uso de este elemento. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Graficos Macro. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio pudimos notar que un macro nos permite tener una especie de plantilla gráfica para una gran multitud de otras escenas sin necesitar el código explícito en cada una de ellas para definir el entorno gráfico. En este caso, sólo se tuvieron cuatro triángulos, pero bien podrían haber sido 20 gráficos distintos que han de repetirse en cada escena. Tener 20 gráficos sólo con el propósito de tener un entorno gráfico puede dificultar la navegación de objetos en dicho selector. Mientras que teniendo un solo macro que los represente hace la navegación entre elementos más ágil.

También es importante recordar que la escena que hereda elementos gráficos de un macro debe tener ya un espacio del mismo nombre de aquél en que fueron creados dichos objetos en la escena de la cual se exporta el macro.

Note también que fue posible controlar alguna variable del macro desde la escena principal (aquella que llama al macro). Ello resulta particularmente útil cuando se requiere modificar el tamaño o posición de algún elemento gráfico del macro dependiendo de condiciones en la escena principal.

En ocasiones puede resultar útil hacer modificaciones manuales a un macro (fuera de Descartes) con un editor de textos. Esto puede conllevar un problema, que es el tipo de codificación que el editor use. Es preciso asegurarse que el editor está guardando el archivo en codificación *UTF-8* (de preferencia *UTF-8 sin BOM*, en caso que el editor permita dicha codificación) siempre que se haga cambio en un macro de forma manual. Esto se puede consultar un poco más a detalle también en el apartado sobre la definición tipo biblioteca.

8.1.12. Gráfico *Sucesión*

Este gráfico consiste en una serie de puntos que representan una sucesión matemática. Una sucesión matemática en Descartes requiere de un número parámetro que se encuentra en los naturales. A partir de este parámetro se toma una expresión para las coordenadas de las abscisas y las ordenadas. Las coordenadas de los puntos de la sucesión se manejan como pares ordenados. En la Figura 8.25 se muestra un ejemplo de este gráfico y en la figura 8.26 se muestra cómo debe configurarse dicho ejemplo. Obsérvese que hay que recorrer el plano cartesiano a la izquierda (se puede arrastrar con el mouse) para ver toda la sucesión.

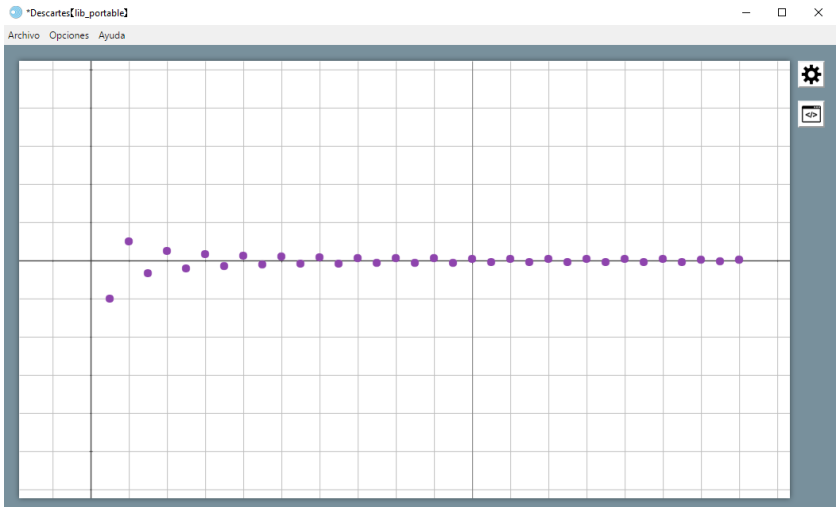


Figura 8.25: Ejemplo del gráfico *Sucesión*.

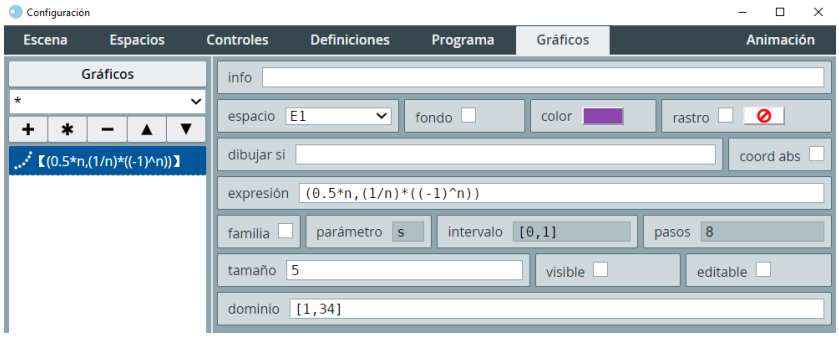


Figura 8.26: Configuración del ejemplo del gráfico *Sucesión*. El color usado es 8E44AD.

- **expresión**: es un campo de texto donde se introduce la expresión de la sucesión como un par ordenado. Por defecto trae la sucesión $(n, 1/n)$. Es decir, el primer punto será el $(1, 1/1)$ o $(1, 1)$; el segundo punto será el $(2, 1/2)$ o $(2, 0.5)$; y así sucesivamente. En este sentido, n es el parámetro que va sobre los naturales, y la expresión para las abscisas y ordenadas dependen de dicho parámetro.
- **visible**: es un checkbox que cuando está marcado permite visualizar la regla de correspondencia de la sucesión dentro del interactivo como una barra en la parte inferior de éste.
- **editable**: es un checkbox que cuando está marcado permite que el usuario modifique el texto visible de la regla de correspondencia de la sucesión directamente en el interactivo, permitiendo así que los puntos que representan la sucesión cambien de forma dinámica.
- **dominio**: es un campo de texto donde se introduce el dominio de la sucesión. Por defecto tiene el texto $[1, 100]$, que quiere decir que sólo incluirá los puntos para los cuales la n adopta valores enteros de 1 y hasta 100.

Hagamos un breve ejercicio para ver cómo funcionan las sucesiones. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Gráficos Sucesión. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

8.1.13. Gráfico *Relleno*

Este gráfico consiste en un relleno de un color seleccionado para el usuario. Los límites de dicho relleno se determinan por los bordes de otros gráficos.

Aunque varios gráficos cuentan con su propio relleno, a veces resulta útil colorear porciones del espacio que no están delimitadas por un solo gráfico, sino por una multitud de ellos. Es en estos casos que conviene usar el gráfico *relleno*. En la Figura 8.27 se muestra un ejemplo de este tipo de gráfico. En la Figura 8.28 se muestra la forma de configurar dicho ejemplo. Note que la figura que se habrá de rellenar es un gráfico tipo ecuación de una

hipérbola: $y^2 - x^2 = 1$. El punto de relleno es el origen (0,0), por lo que se rellena el área alrededor de dicho punto.

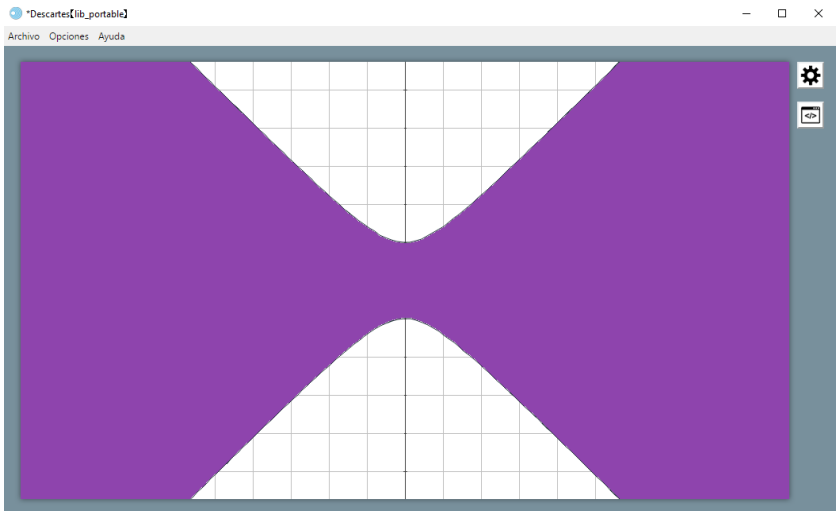


Figura 8.27: Ejemplo del gráfico *Relleno*.

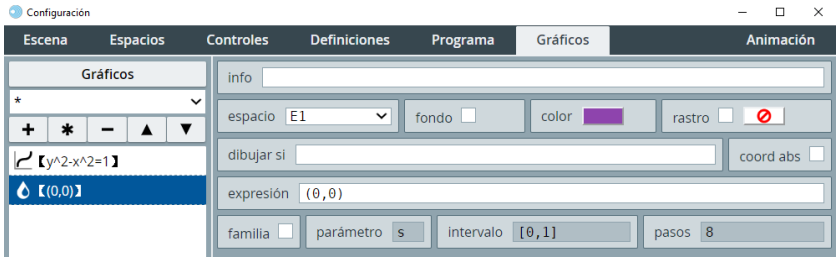


Figura 8.28: Configuración del ejemplo del gráfico *Relleno*. El color utilizado es 8e44ad.

- **expresión**: es un campo de texto en el que se introduce la coordenada de un punto que habrá de estar dentro del área delimitada que se quiere rellenar con color.

A continuación haremos un ejercicio que involucra varias figuras geométricas que se intersecan entre sí, y cómo el gráfico *relleno* nos puede ayudar a colorear el área delimitada entre dichas figuras. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Graficos Relleno. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Se observó en este ejercicio la utilidad del gráfico tipo *relleno*, que puede usarse, por ejemplo, en ejercicios de cálculo de áreas compuestas por varias figuras. Es importante notar que el relleno busca fronteras que lo limiten. Si no encuentra una, continuará pintando. Por lo mismo, es preciso asegurarse que las áreas estén delimitadas correctamente. También es importante tener en mente que los objetos que habrán de delimitar al relleno

deben encontrarse **antes** de éste en la lista de gráficos, y deben encontrarse trazados (en su parámetro *dibujar si*).

8.2. Gráficos 3D

Estos gráficos son de tipo tridimensional. Por lo mismo, sólo se pintan en espacios tridimensionales o 3D. Se recuerda al usuario que para agregar un espacio 3D en el selector *Espacios* para poder trabajar con los objetos que a continuación se describen.

8.2.1. Gráfico *Segmento*

Es igual al gráfico segmento en dos dimensiones, con la salvedad de que las coordenadas de inicio y fin del segmento en el campo *expresión* de éste conllevan tres entradas (una por cada eje en tres dimensiones).

En la Figura 8.29 se muestra un ejemplo del gráfico tridimensional *segmento*. En la Figura 8.30 se muestra la configuración para lograr este ejemplo. En el editor de configuraciones sólo se ve el último de los segmentos (a lo largo del eje z). No obstante, note que hay 2 segmentos más. Cada uno tiene longitud 2, y están orientados respectivamente a lo largo de los ejes coordenados. Esto facilita ubicar dónde se encuentran los gráficos tridimensionales en el espacio 3D. De tal forma que este ejemplo se usará subsecuentemente en los siguientes gráficos.

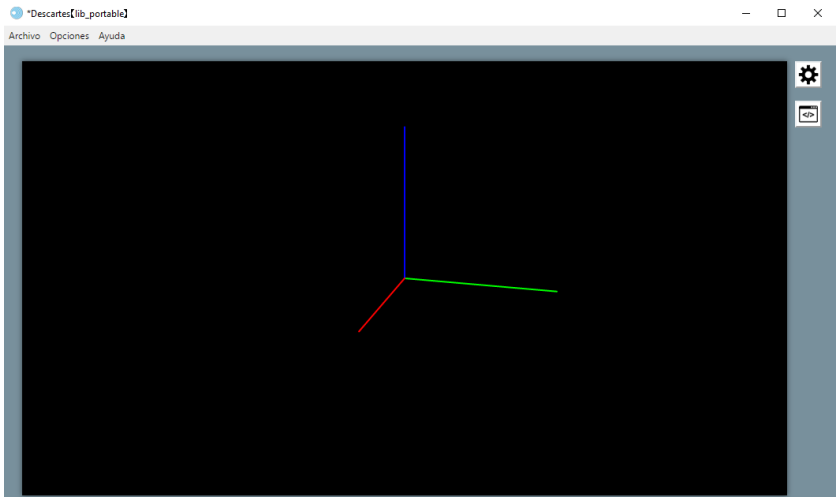


Figura 8.29: Ejemplo del gráfico tridimensional *Segmento*.

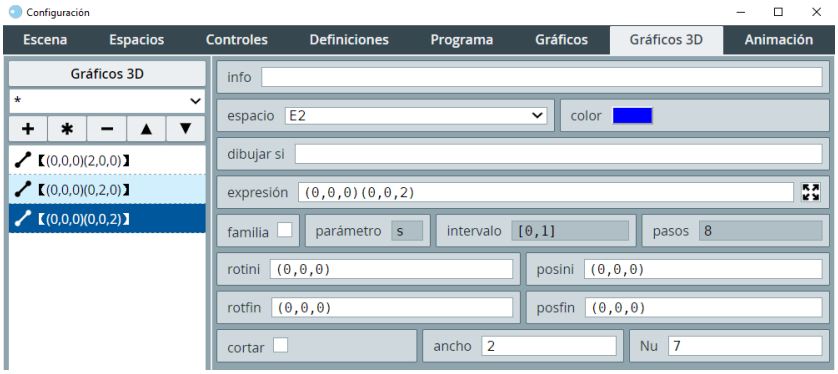


Figura 8.30: Configuración del ejemplo del gráfico tridimensional *Segmento*.

8.2.2. Gráfico *Punto*

Es un gráfico que se inserta sólo en un espacio tridimensional. Es igual al gráfico punto en dos dimensiones, con la salvedad de que el campo *expresión* de éste conlleva tres coordenadas (la primera para el eje x, la segunda para el eje y y la tercera para el eje z).

En la Figura 8.31 se muestra un ejemplo de este tipo de gráfico. En la Figura 8.32 se muestra la configuración necesaria para lograr este ejemplo. Los colores usados para los segmentos son C0392B para el rojo, 27AE60 para el verde y 2980B9 para el azul. Consulte la herramienta de control de colores para mayor información.

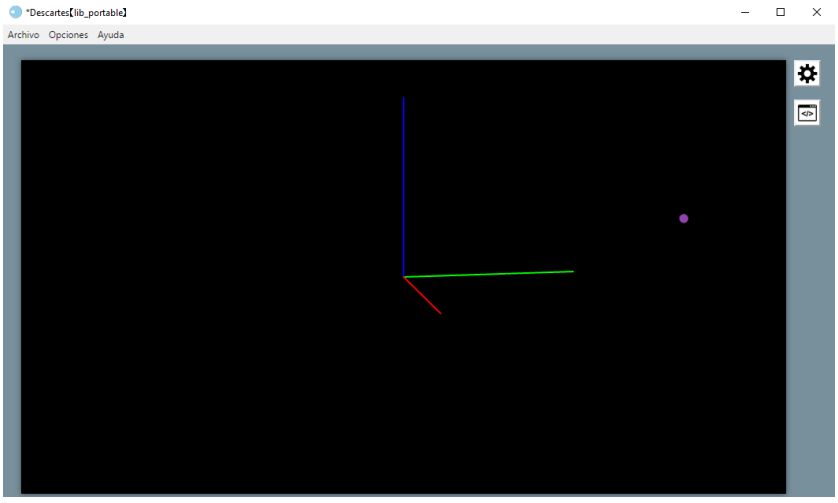


Figura 8.31: Ejemplo del gráfico tridimensional *Punto*.

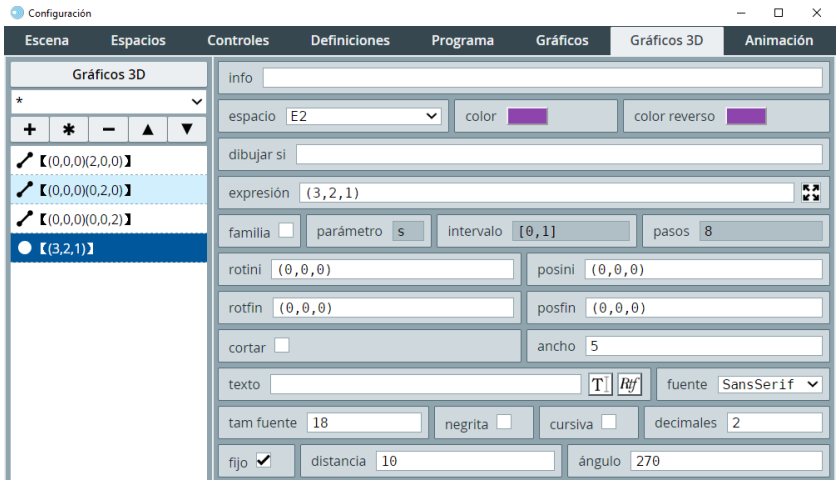


Figura 8.32: Configuración del ejemplo del gráfico tridimensional *Punto*. El color usado para el punto es 8e44ad.

- **distancia**: Es un campo de texto que afecta al texto asociado al punto. Representa una distancia en px entre el punto y el texto que conlleva.

- **ángulo**: Es un campo de texto que afecta al texto asociado al punto. Representa el ángulo de dirección (medido en grados), respecto a las coordenadas del punto, en que se muestra el centro de su texto. Es decir, tomando las coordenadas del punto como base, el texto se mostrará a una distancia en px correspondiente al parámetro *distancia* y a un ángulo correspondiente al parámetro *ángulo*.

8.2.3. Gráfico *Polígono*

Es parecido al gráfico polígono en dos dimensiones, con la salvedad de que las coordenadas de los vértices en el campo *expresión* de éste conllevan tres entradas cada una (una entrada por cada eje en tres dimensiones).

En la Figura 8.33 se muestra un ejemplo de este tipo de gráfico. En la Figura 8.34 se muestra la configuración necesaria para lograr este ejemplo. Note que en este ejemplo, los vértices del polígono pueden no necesariamente pertenecer a un mismo plano.

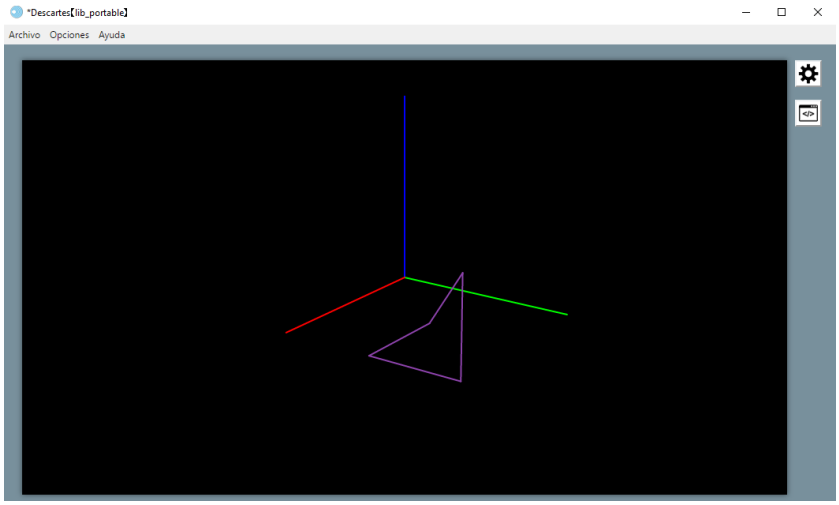


Figura 8.33: Ejemplo del gráfico tridimensional *Poligono*.

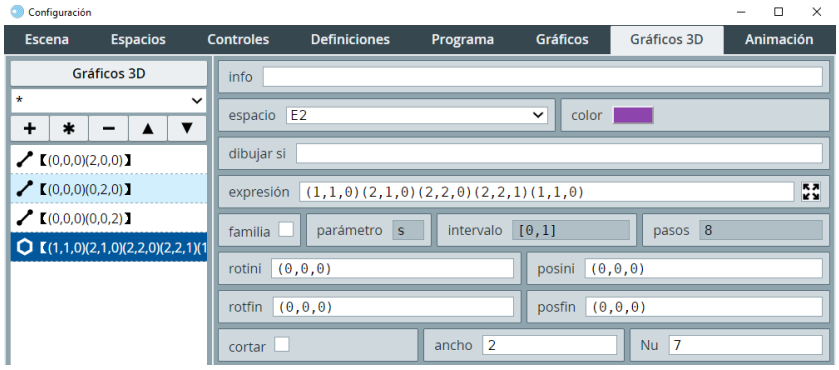


Figura 8.34: Configuración del ejemplo del gráfico tridimensional *Poligono* de color 8e44ad.

8.2.4. Gráfico *Curva*

Es parecido al gráfico curva en dos dimensiones, con la salvedad de que el campo *expresión* de éste lleva una expresión para x, una para y y una para z cada una separada de la otra por un espacio. Las expresiones dependen de un parámetro u que adopta valores continuos entre 0 y 1.

En la Figura 8.35 se muestra un ejemplo de el gráfico tridimensional curva. En la Figura 8.36 se muestra la configuración para lograr dicho ejemplo.

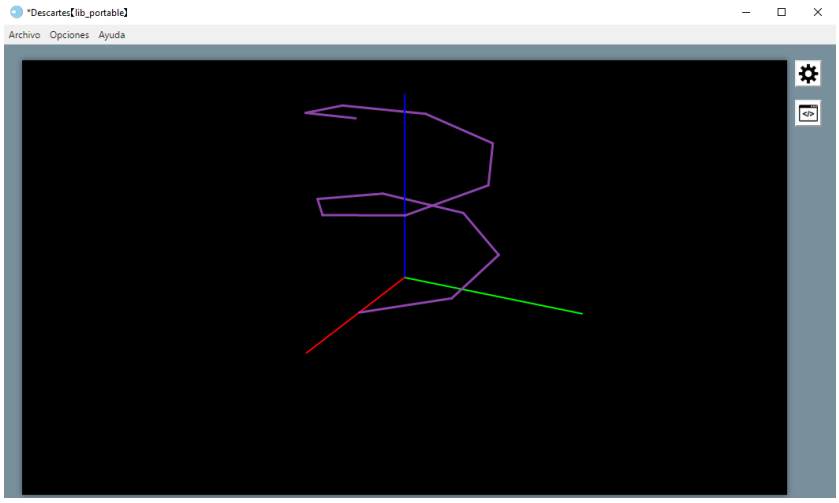


Figura 8.35: Ejemplo del gráfico tridimensional *Curva*.

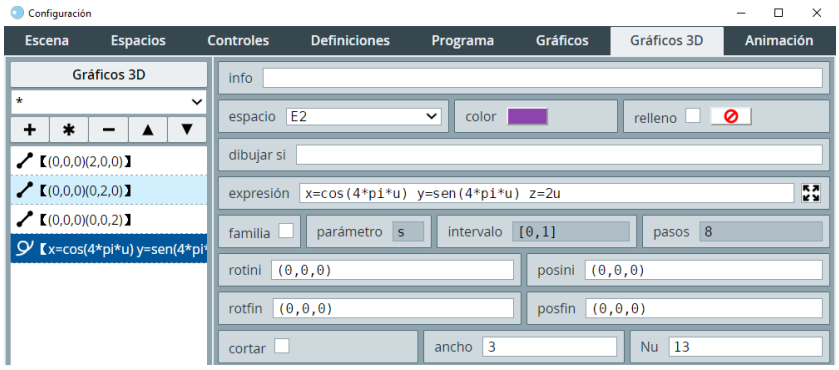


Figura 8.36: Configuración del ejemplo del gráfico tridimensional *Curva* con color 8e44ad.

Incluye un campo *Nu* en el que se introduce el número de partes en que se subdivide el intervalo entre 0 y 1 que define los valores que adopta el parámetro *u*. Mientras más grande sea este valor, más fina será la curva, pero hacer éste valor demasiado grande puede resultar en ralentizar el interactivo.

8.2.5. Gráfico *Triángulo*

Es básicamente un polígono de tres lados solamente. Puede referirse al gráfico polígono tridimensional para mayor información.

En la Figura 8.37 se muestra un ejemplo de este gráfico. En la Figura 8.38 se muestra la configuración para conseguir este ejemplo.

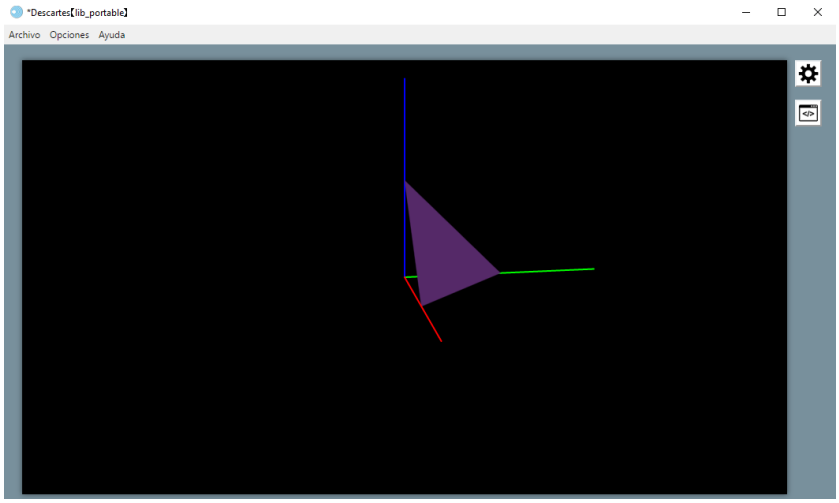


Figura 8.37: Ejemplo del gráfico *Triángulo*.

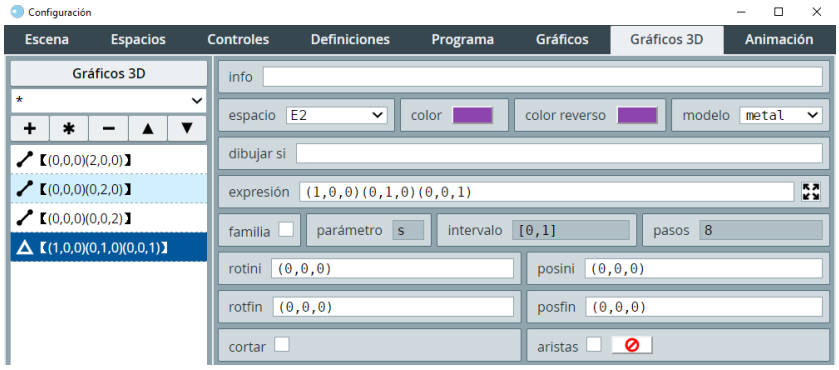


Figura 8.38: Configuración del ejemplo del gráfico *Triángulo* de color 8e44ad.

8.2.6. Gráfico *Cara*

Permite definir en su campo *expresión* los vértices de una cara en dos dimensiones (cada vértice se asocia a un par ordenado, no a tres coordenadas). Posteriormente es posible definir la inclinación tridimensional de la cara creada usando los campos *rotini*, *posini*, *rotfin* y *posfin*. Para mayor información sobre estos campos consulte el apartado sobre los controles comunes a los gráficos 3D.

Es conveniente notar que es posible introducir los vértices de la cara en el espacio (es decir, con el formato `(x,y,z)`. Esto conlleva la ventaja de no tener que manipular los parámetros *posini*, *rotini*, *posfin* y *rotfin* para orientar la cara, que puede resultar más complicado. No obstante, para que esto funcione **es preciso que los vértices de la cara sean coplanares**. De lo contrario, la cara no se dibujará correctamente, puesto que no se trataría realmente de una cara.

En la Figura 8.39 se muestra un ejemplo de este gráfico. En la Figura 8.40 se muestra la configuración que ha de implementarse para reproducir este ejemplo. Note que la cara en este ejemplo está originalmente definida en el plano *XY* ya que sólo se usan pares ordenados y no triadas. No obstante, echando mano de una rotación en el parámetro *rotini*, esta cara puede mostrarse con una inclinación.

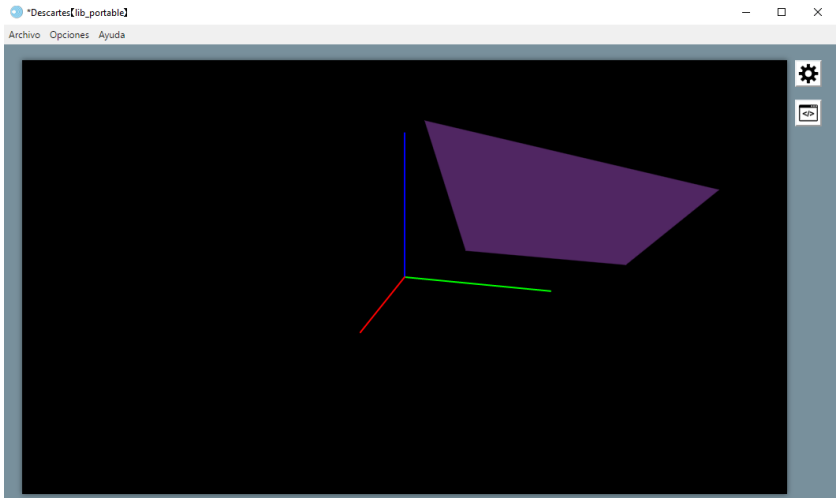


Figura 8.39: Ejemplo del gráfico *Cara*.

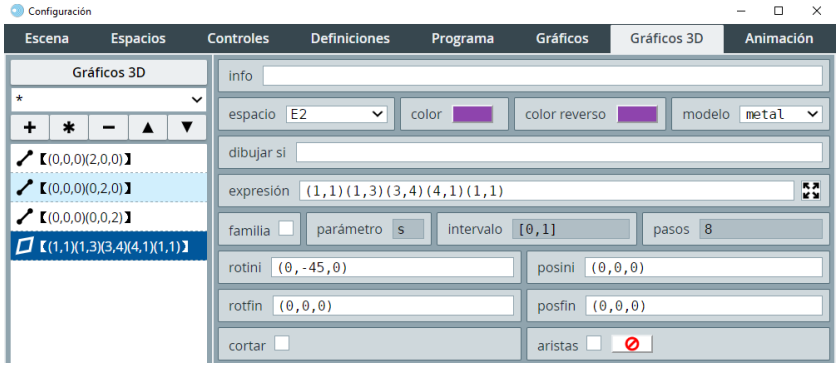


Figura 8.40: Configuración del ejemplo del gráfico *Cara* de color 8e44ad.

8.2.7. Gráfico *Polígono regular*

Es un gráfico que permite definir un polígono regular usando un parámetro *Nu* que determina el número de caras del polígono. Permite rotaciones y traslaciones iniciales y finales como se describe en el apartado sobre controles comunes a los gráficos 3D.

En la Figura 8.41 se muestra un ejemplo de este gráfico. En la Figura 8.42 se muestra la configuración para lograr este ejemplo. Note que en este ejemplo el número de lados del

polígono regular está determinado por *Nu*. De forma natural, el polígono yacería sobre el plano *XY*, pero gracias a los cambios en los parámetros *rotini* y *posini*, el polígono se encuentra rotado y parece que se recarga en el eje *Z*.

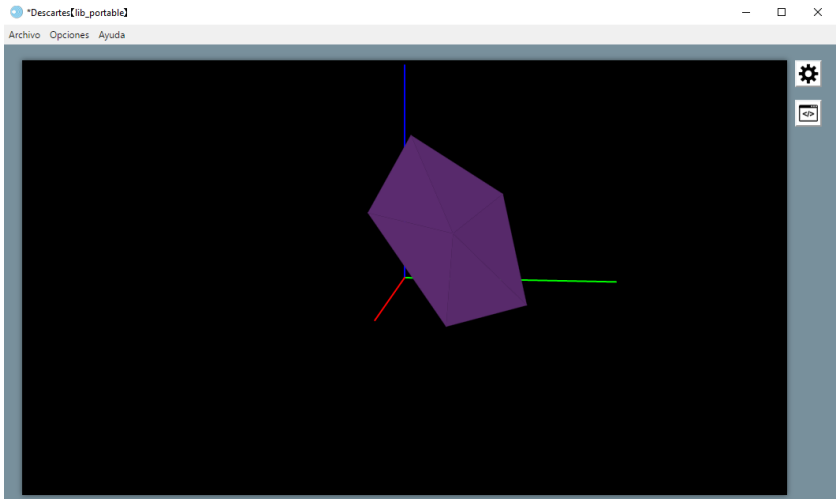


Figura 8.41: Ejemplo del gráfico *Polireg*.

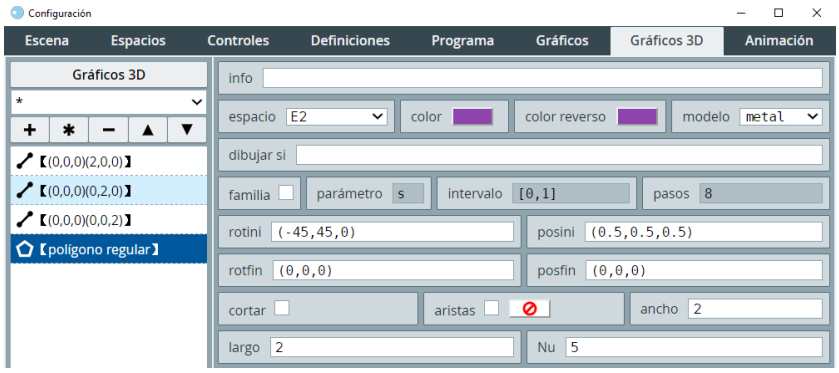


Figura 8.42: Configuración del ejemplo del gráfico *Polígono regular* de color 8e44ad.

8.2.8. Gráfico *Superficie*

Es un gráfico que permite definir una superficie a partir de dos parámetros que adoptan valores continuos en el intervalo entre 0 y 1. Los parámetros son u y v. El número de particiones de estos parámetros en su intervalo están dados por los campos *Nu* y *Nv*, respectivamente.

Es necesario llenar el campo *expresión* del gráfico con una expresión para x (en función de u y v), una para y y una para z separadas por espacios.

En la Figura 8.43 se muestra un ejemplo de este gráfico. En la Figura 8.44 se muestra la configuración para lograr este ejemplo.

Las asignaciones dadas en este ejemplo en el parámetro *expresión* son las siguientes: $x = 3cos(2\pi u)sen(\frac{pi}{2}v)$, $y = 3sen(2\pi u)sen(\frac{pi}{2}v)$, y $z = cos(3\sqrt{x^2+y^2})$. Note que las coordenadas en este caso son cilíndricas y con eso se logra que el borde de la superficie sea circular. La raíz en la expresión para z involucra la distancia del eje Z, y es el argumento de un coseno, lo que da la sensación de onda en el gráfico.

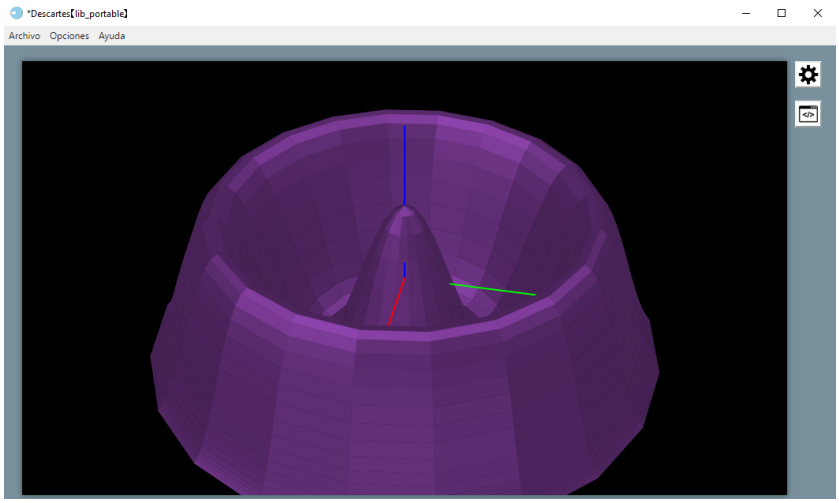


Figura 8.43: Ejemplo del gráfico *Superficie*.

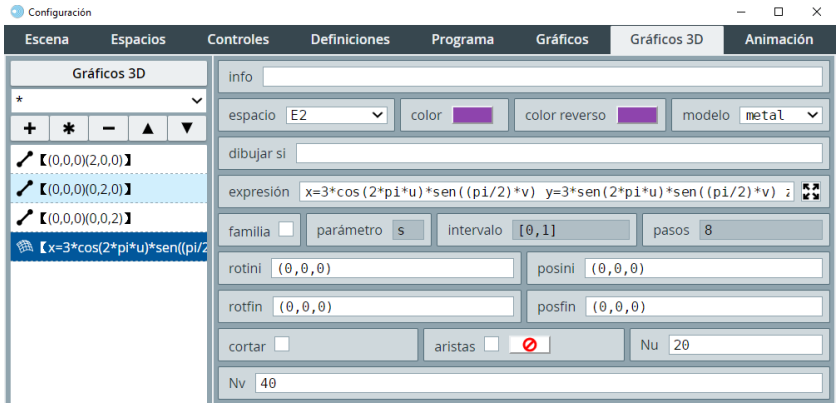


Figura 8.44: Configuración del ejemplo del gráfico *Superficie* de color 8e44ad.

8.2.9. Gráfico *Texto*

Es igual al gráfico *texto* para espacios en dos dimensiones, con la excepción de que en 3D, el texto no permite generar rastros y sólo se puede usar en coordenadas absolutas. Por lo mismo, las casillas de verificación *rastro* y *coord abs* se encuentran ausentes en textos en espacios 3D.

8.2.10. Gráfico *Macro*

Funciona igual que el gráfico macro para espacios en dos dimensiones. La única diferencia es que la ruta al archivo para el macro 3D se inserta en un campo de texto llamado *expresión* en lugar de *archivo*.

Es preciso notar que para usar macros en tres dimensiones, la escena (o archivo *html*) que hereda al macro debe ya contar con un espacio 3D del mismo nombre que aquél en la escena original en que se crearon los elementos del mismo. Por ejemplo, suponga que se crean objetos gráficos en una escena original y éstos han de ser parte del macro, y están en un espacio tridimensional de nombre *E2*. Posteriormente se exportan como un macro de *Descartes*. La escena que herede dicho macro ya deberá contar con el espacio tridimensional *E2* para poder importar los elementos correctamente.

8.2.11. Gráfico *Cubo*

Es un gráfico al que se le define un campo *ancho* que corresponde a la longitud de la diagonal del cubo. El lado del cubo se puede calcular dividiendo la longitud de la diagonal por $\sqrt{3}$. Se puede reorientar y trasladar usando los campos *rotini*, *posini*, *rotfin* y *posfin* descritos en el apartado sobre controles comunes a los gráficos 3D.

En la Figura 8.45 se muestra un ejemplo de este gráfico. En la Figura 8.46 se muestra la configuración para lograr este ejemplo. Note que en este ejemplo el cubo aparece rotado debido a la expresión introducida en el parámetro *rotini*.

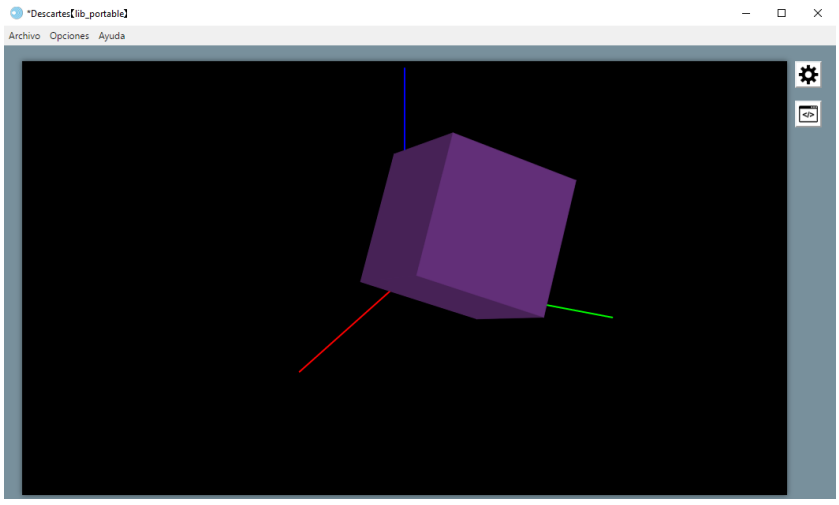


Figura 8.45: Ejemplo del gráfico *Cubo*.

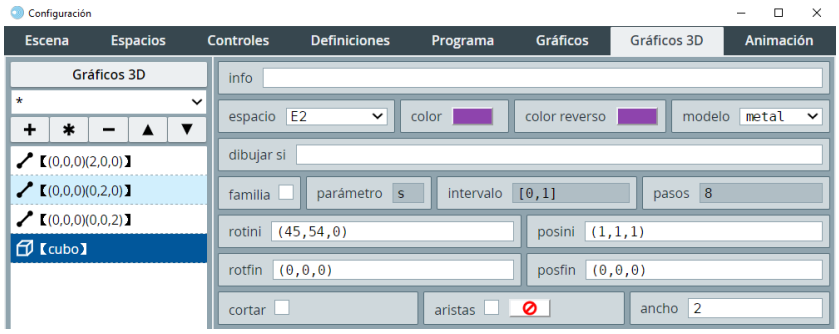


Figura 8.46: Configuración del ejemplo del gráfico *Cubo* de color 8e44ad.

8.2.12. Gráfico *Paralelepípedo*

Es un gráfico parecido al cubo, salvo que a éste se le define un ancho, largo y un alto.

En la Figura 8.47 se muestra un ejemplo de este gráfico. En la Figura 8.48 se muestra la configuración para lograr este ejemplo. Note que en este ejemplo el paralelepípedo aparece rotado debido a la expresión introducida en el parámetro *rotini*.

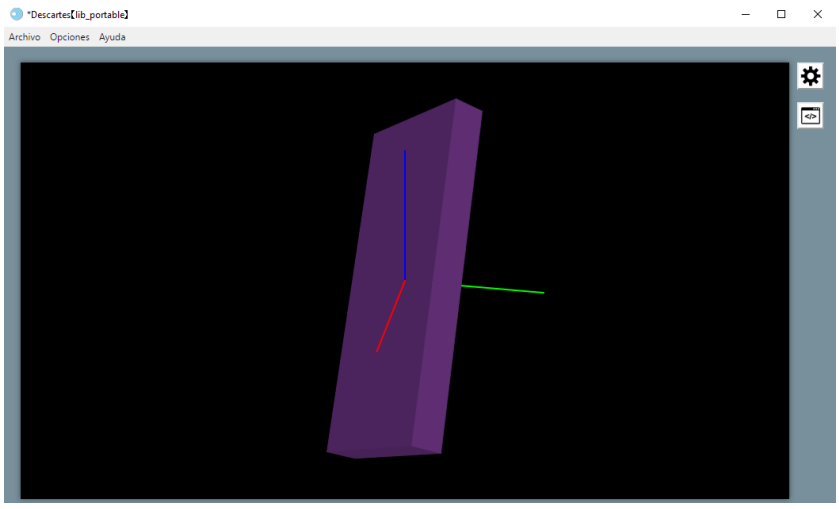


Figura 8.47: Ejemplo del gráfico *Paralelepípedo*.

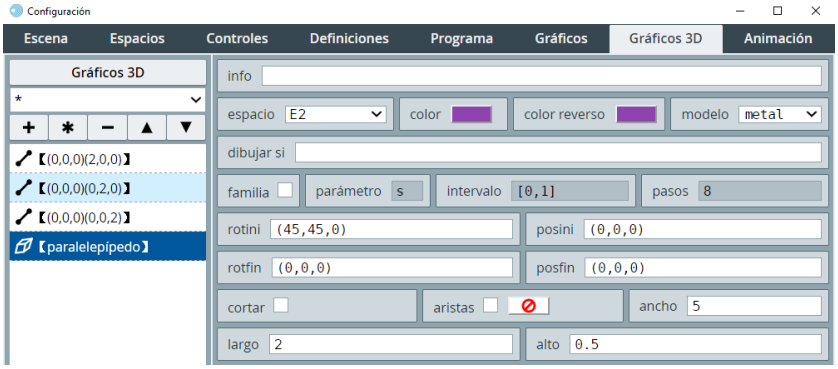


Figura 8.48: Configuración del ejemplo del gráfico *Paralelepípedo* de color 8e44ad.

8.2.13. Gráficos *Tetraedro, Octaedro, Dodecaedro* e *Icosaedro*

Son gráficos prediseñados a los que se les asigna un valor en su campo *ancho* que determina sus tamaños. Se pueden reorientar y trasladar usando los campos *rotini, posini, rotfin* y *posfin*descritos en el apartado sobre controles comunes a los gráficos 3D.

En la Figura 8.49 se muestra un ejemplo de uno de estos gráficos: el icosaedro. En la Figura 8.50 se muestra la configuración para lograr este ejemplo. Note que en este ejemplo el icosaedro no aparece centrado en el origen debido a que la expresión introducida en el parámetro *rotini* desplaza el centro del icosaedro a esa posición.

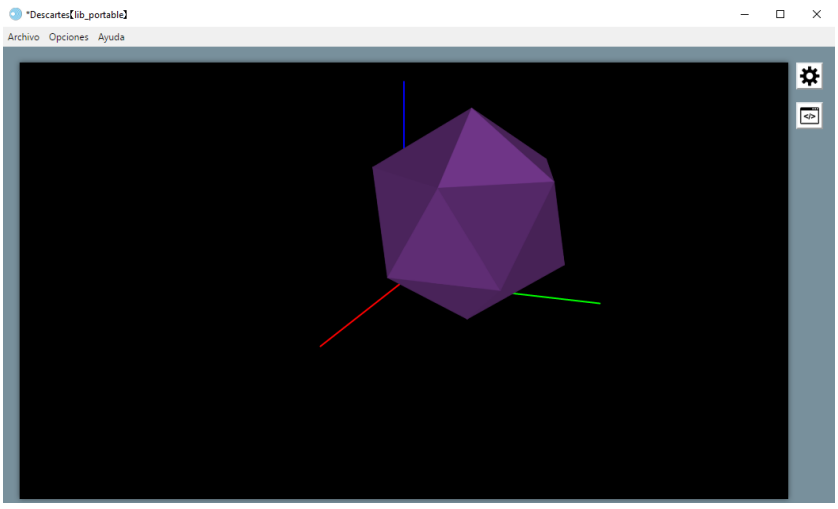


Figura 8.49: Ejemplo del gráfico *Icosaedro*.

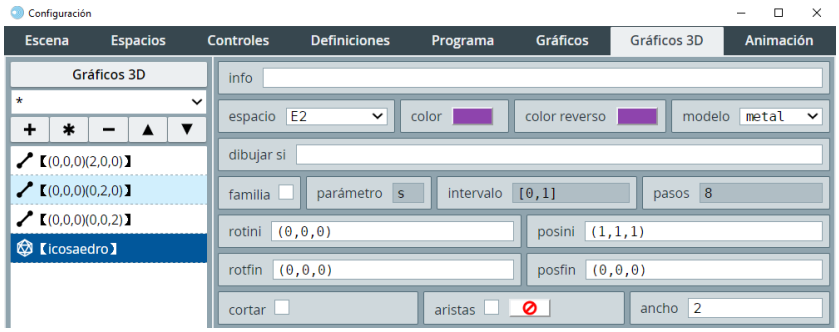


Figura 8.50: Configuración del ejemplo del gráfico *Icosaedro* de color 8e44ad.

8.2.14. Gráfico *Esfera*

Es un gráfico al que se le asigna un valor para un parámetro *ancho* que define su diámetro, así como valores para su campo *Nu* (que define el número de paralelos para la división de la esfera) y su campo *Nv* (que define el número de meridianos para la división de la esfera). Mientras mayores sean *Nu* y *Nv*, el gráfico se verá más parecido a una esfera, pero exagerar el valor de estos parámetros puede resultar en un interactivo de lenta actualización.

En la Figura 8.51 se muestra un ejemplo de este gráfico. En la Figura 8.52 se muestra la configuración para lograr este ejemplo.

Note que en este ejemplo los paralelos y meridianos de la esfera se marcan de un tono que resalta. Ello es porque el checkbox *aristas* se encuentra marcado. Observe que hay muchos paralelos y pocos meridianos. Ello corresponde a que el parámetro *Nu* vale 15 y el *Nv* vale 7. También vea que los paralelos de la esfera no son horizontales respecto al plano *XY* (aquél marcado por el eje rojo y el verde) del espacio. Ello responde a que hay una expresión, usando ángulos de Euler, introducida en el parámetro *rotini* que se encarga de rotar la esfera.

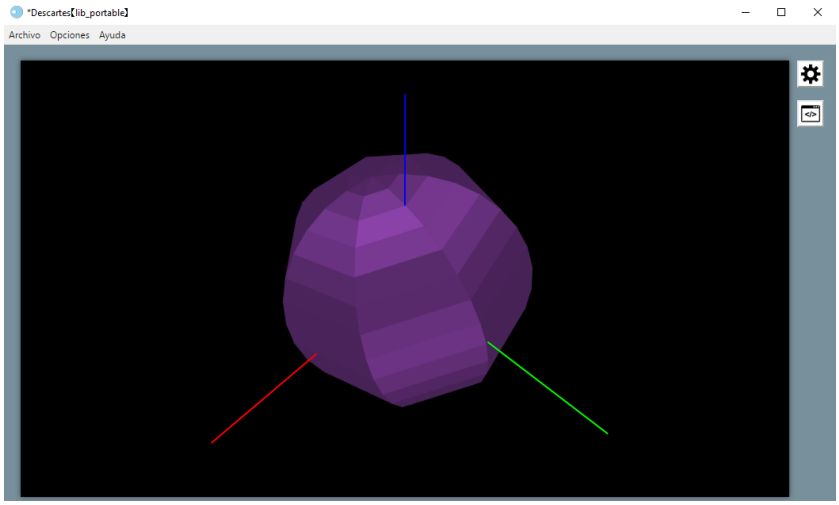


Figura 8.51: Ejemplo del gráfico *Esfera*.

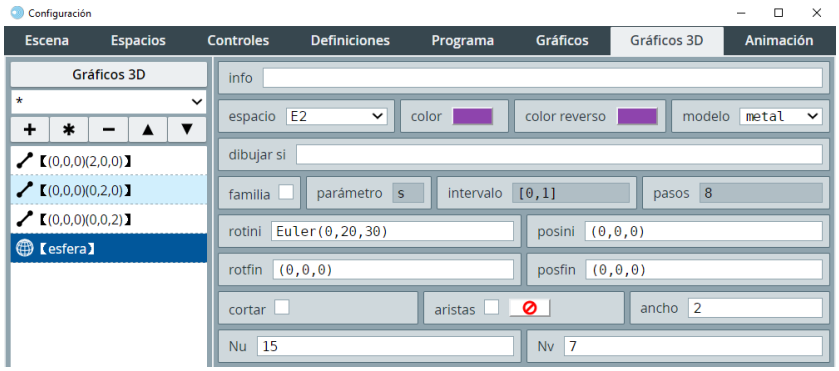


Figura 8.52: Configuración del ejemplo del gráfico *Esfera* de color 8e44ad.

8.2.15. Gráfico *Elipsoide*

Es un gráfico al que se le asigna un valor para los campos *ancho*, *largo* y *alto*, que corresponden a las longitudes de los ejes en *x*, *y* y *z* del elipsoide, respectivamente. También involucra un parámetro *Nu* y uno *Nv* con las que se determina el número de secciones en paralelos y en meridianos, respectivamente.

En la Figura 8.53 se muestra un ejemplo de este tipo de figura. En la Figura 8.54 se muestra la configuración necesaria para lograr este ejemplo.

Note que el elipsoide tiene cierta transparencia para poder ver los segmentos que representan los ejes detrás de él. Las aristas se encuentran marcadas para que se vea el efecto de los parámetros *Nu* y *Nv*, y el color de las mismas es el color *gris* de las opciones de color predeterminadas en Descartes.

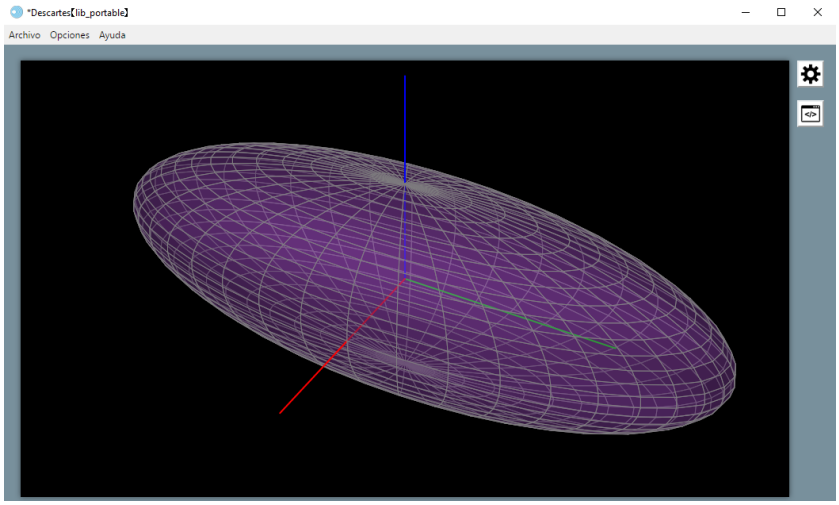


Figura 8.53: Ejemplo del gráfico *Elipsoide*.

Figura 8.54: Configuración del ejemplo del gráfico *Elipsoide* de color 8e44ad.

8.2.16. Gráfico *Cono*

Es un gráfico de forma cónica cuya base puede ser una elipse. A este gráfico se le define un parámetro *ancho*, que corresponde al eje en x de la elipse que es la base; uno *largo*, que define el eje en y de la elipse de base; y uno *alto* que define la altura en el eje z del cono. También cuenta con un parámetro *Nu* que determina el número de vértices que definen la curva que forma la elipse de la base. Este número es, así pues, algo así como los meridianos que definen el gráfico (semejantes a los meridianos de una esfera). Cuenta también con un parámetro *Nv* que, aunque no brinda un cambio en la resolución del gráfico como en otras figuras tridimensionales, define el número de paralelos (también semejantes a los de una esfera) del gráfico.

En la Figura 8.55 se muestra un ejemplo de este tipo de gráfico. En la Figura 8.56 se muestra la configuración necesaria para lograr este ejemplo.

Note que las aristas se encuentran marcadas y del color *gris* de los colores predeterminados de Descartes con el objeto de que se vea el efecto de los parámetros *Nu* y *Nv*. Observe que el número de paralelos del gráfico (curvas paralelas al plano *XY*) es 10, como lo define el parámetro *Nv*. También note que el gráfico se encuentra rotado 180°respecto al eje *X* debido a la expresión en el parámetro *rotini*. Es decir, de forma natural, el vértice del cono aparece hacia abajo, pero aquí se rotó para que la base aparezca abajo. Asimismo, el centro del gráfico se encuentra desplazado del origen debido a la expresión en el parámetro *posini*.

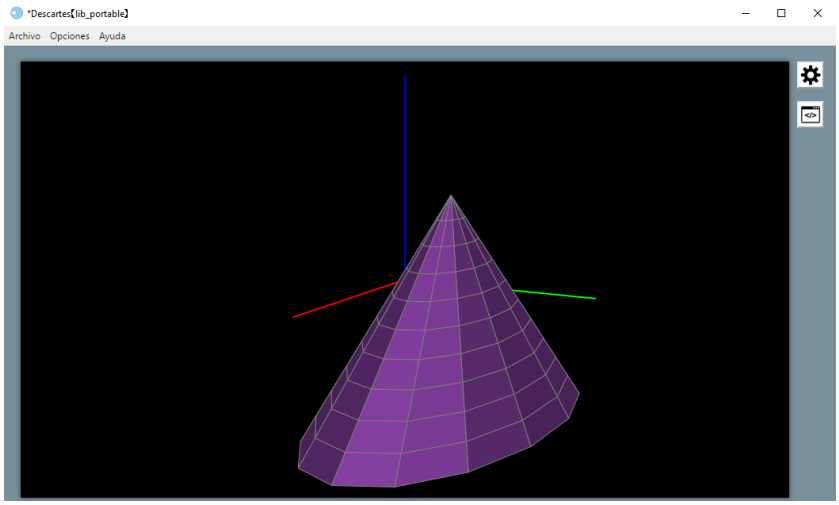


Figura 8.55: Ejemplo del gráfico *Cono*.

Figura 8.56: Configuración del ejemplo del gráfico *Cono* de color 8e44ad.

8.2.17. Gráfico *Cilindro*

Es un gráfico de forma cilíndrica cuya base puede ser una elipse. Cuenta con un parámetro *ancho*, que corresponde al eje en *x* de la elipse que es la base; uno *largo*, que

define el eje en *y* de la elipse de base; y uno *alto* que define la altura del cilindro en el eje *z*. También cuenta con un parámetro *Nu* que determina el número de vértices que definen la curva que forma la elipse de la base. Este número es, así pues, algo así como los meridianos que definen el gráfico (semejantes a los de una esfera). Cuenta también con un parámetro *Nv* que, aunque no brinda un cambio en la resolución del gráfico como en otras figuras tridimensionales, define el número de paralelos (también semejantes a los de una esfera) del gráfico.

En la Figura 8.57 se muestra un ejemplo de este tipo de gráfico. En la Figura 8.58 se muestra la configuración necesaria para lograr este ejemplo.

Note que las aristas se encuentran marcadas con el color *gris* de los colores predeterminados de Descartes para que se vea el efecto de los parámetros *Nu* y *Nv*. También note que el gráfico se encuentra rotado 45°respecto al eje *X* y 45°respecto al *Y* (observe la expresión en su parámetro *rotini*), por lo que no aparece vertical o parado. El color del cilindro tiene cierta transparencia para poder mejor visualizar los ejes.

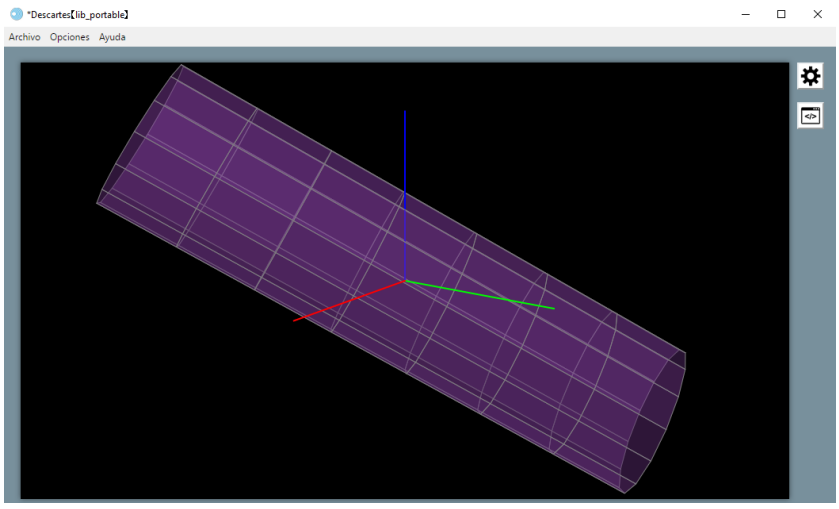


Figura 8.57: Ejemplo del gráfico *Cilindro*.

Figura 8.58: Configuración del ejemplo del gráfico *Cilindro* de color 8e44ad.

8.2.18. Gráfico *Toroide*

También conocido como *toro*, es un gráfico que consiste en un sólido de revolución generado al rotar una circunferencia alrededor de un eje contenido en el mismo plano que la circunferencia. Su parámetro *R* es un campo de texto en el que se determina el radio de giro (la distancia del eje de giro al centro de la circunferencia a revolucionar), mientras que *r* es el radio de la circunferencia que al girar traza el toro. Para el caso particular del toroide, en el campo de texto *Nu* se introduce el número de de pasos usados para trazar la circunferencia que al girar pinta el toroide, mientras que en *Nv* va el número de pasos que se usan para trazar la circunferencia en que se divide el giro de la circunferencia. Como de costumbre, mayores valores de *R* y *r* resultan en una mejor resolución del toroide, pero también en mayor lentitud al moverlo.

En la Figura 8.59 se muestra un ejemplo de este tipo de gráfico. En la Figura 8.60 se muestra la configuración necesaria para lograr este ejemplo.

Note que las aristas se encuentran marcadas en el color *gris* predeterminado de Descartes para visualizar mejor el efecto de los parámetros *Nu* y *Nv*. Igualmente, se le dio algo de transparencia al gráfico.

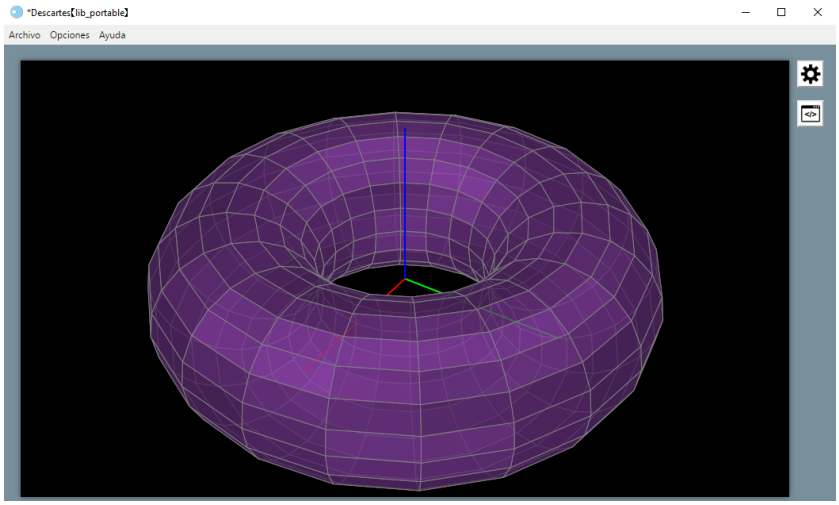


Figura 8.59: Ejemplo del gráfico *Toroide*.

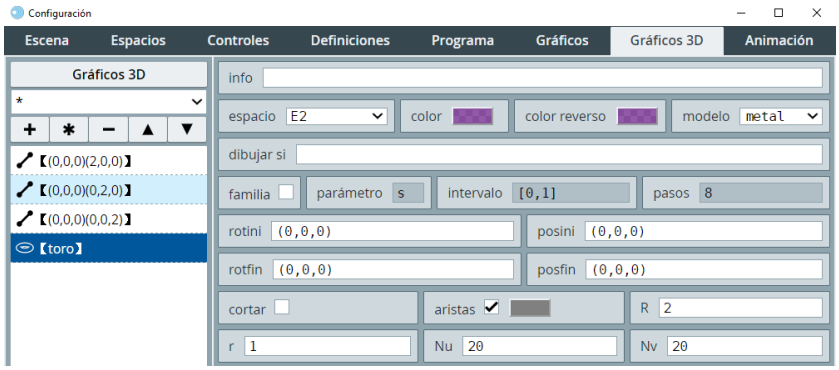


Figura 8.60: Configuración del ejemplo del gráfico *Toroide* de color 8e44ad.

8.2.19. Ejercicio general de gráficos 3D

Realicemos un ejercicio general sobre gráficos 3D. Elegimos uno representativo, que es el elipsoide, dado que su funcionamiento permite entender el de otros gráficos 3D. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Gráficos3D. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Este ejercicio sirve como una muestra de la funcionalidad general de los gráficos tridimensionales. En particular, el uso de las rotaciones y posiciones iniciales y finales permite reducir la cantidad de cálculos para lograr un gráfico con una determinada disposición. Por ejemplo, sería posible incluir un polígono inclinado y desplazado dándolo sólo sus vértices. No obstante, esto involucraría muchos cálculos que se pueden ahorrar trazando el polígono en el origen y luego sólo indicando sendas rotaciones y traslaciones. Es muy

importante tomar en cuenta el orden en que se hacen las rotaciones y traslaciones para lograr una posición e inclinación particular de un gráfico. En particular para el ejemplo usado, se pudo observar que el orden es muy importante pues la segunda rotación se hace sobre el objeto (donde sea que se encuentre) pero respecto al origen.

Es importante enfatizar el uso de segmentos para no perder de vista en qué dirección apuntan los ejes del espacio tridimensional. Estos segmentos suelen sólo ser útiles al programador mientras coloca sus gráficos, por lo que pueden finalmente ser ocultados.

Se sugiere que el usuario experimente un poco más agregando algunas otras figuras y pruebe el funcionamiento del checkbox *cortar* cuando éstas se intersecan. Igualmente, conviene que vea los cambios que el menú *modelo* genera en los gráficos.

Para este ejercicio se cambiaron los colores de los ejes. Puede consultar el apartado sobre la herramienta de control de colores para más información.

8.3. Controles comunes a los gráficos

- **menú de espacios**: es un menú que muestra los espacios existentes en el interactivo. A diferencia del menú de espacios que se encuentra sobre el panel a la izquierda, este menú selecciona adónde mover el gráfico seleccionado, en vez de filtrar los gráficos de un espacio como hace el otro menú. Se recomienda tener en mente esta diferencia pues se pueden hacer errores si se usa un menú en lugar del otro.
- **fondo**: es un checkbox que si está seleccionado hace que el gráfico en cuestión se dibuje una sola vez al cargar el interactivo. Si se encuentra marcado, los cambios subsecuentes en variables que controlen a dicho gráfico (su color, forma, etc.) no se reflejarán en el mismo. Es decir, el gráfico se encuentra como un fondo estático (no interactivo) de la escena. Este checkbox suele marcarse cuando no se planea que cambie el gráfico en cuestión. Esto permite que el interactivo cargue más rápidamente y no se perderá tiempo refrescando el gráfico constantemente.
  NOTA: es posible que el gráfico se refresque, aún estando marcada la opción *fondo* si se cambia la escala del espacio o se desplaza el mismo. En este sentido, *fondo* sólo ignora el recálculo del gráfico mediante controles, pero lo hará si se hacen cambios sobre la escala o posición de los ejes coordenados de los espacios presentes.
- **dibujar si**: es un campo de texto en el cual se introduce alguna condición booleana que determina si el gráfico ha de pintarse o no. Una condición booleana puede ser verdadera (a la que se le otorga un valor de 1) o falsa (a la que se le otorga un valor de 0). Por ejemplo, si se tuviera una variable a con valor de 3 y en el campo de texto se introduce la condición $a == 2$, se compara el valor de a con 2 y se ve que no son iguales, por lo que la condición vale 0 y el gráfico no se pintará. Si a valiera 2, entonces sí se pintaría. Más adelante se verán más ejemplos de condiciones booleanas. Pueden consultarse también en el apartado sobre condicionales y operadores booleanos. Es importante tener en cuenta que el comportamiento de este campo de

texto no sólo aplica a los gráficos, sino que también puede ser usado en controles, como se verá después.

- **coordenadas absolutas**: es un checkbox que determina si el gráfico se pinta en coordenadas absolutas o en relativas. En coordenadas absolutas, el origen está en la esquina superior izquierda del espacio, las unidades se miden en pixeles y los valores de las ordenadas crecen hacia abajo. En coordenadas relativas, el origen es el origen del plano cartesiano del espacio, las unidades son las determinadas por la escala del espacio, y el valor de las ordenadas crece hacia arriba.
- **texto**: es un campo de texto, acompañado de dos botones *T* y *Rtf* (para lanzar ventanas de introducción de texto simple y enriquecido, respectivamente) mediante los cuales se puede introducir un texto que se muestra cercano al gráfico en cuestión. La funcionalidad del texto es la misma que la detallada en el apartado de la herramienta de edición de textos.
- **decimales**: es un campo de texto que usualmente acompaña al campo *texto* en varios gráficos, y cuya función es indicar el número de decimales a mostrar de ser necesario.
- **fijo**: es un checkbox que también suele acompañar a los campos *texto* y *decimales* y que, de estar marcado, indica que el número de decimales especificado siempre ha de mostrarse, aún si no son significativos.
- **fuente**: es un menú con el cual se selecciona la fuente deseada para el texto del gráfico. Sus opciones son *SansSerif*, *Serif* y *Monospaced*. Sólo aplica para gráficos que involucran texto.
- **tam fuente**: es un campo de texto donde se introduce un número entero correspondiente al tamaño en pixeles del texto correspondiente asociado a un gráfico. Sólo aplica para gráficos que involucran texto.
- **negrita**: es un checkbox que, de estar seleccionado, hace que la fuente del texto asociado al gráfico en cuestión aparezca en negritas. Sólo aplica para gráficos que involucran texto.
- **cursiva**: es un checkbox que, de estar seleccionado, hace que la fuente del texto asociado al gráfico en cuestión aparezca en cursivas. Sólo aplica para gráficos que involucran texto.
- **rastro**: es un checkbox acompañado de un control de color. Cuando el gráfico en cuestión se modifica dinámicamente (por ejemplo, cambia su forma o posición), éste simplemente se borra y se traza en su nueva forma o posición. Pero si este checkbox está marcado, dejará rastros de sus posiciones y formas anteriores conforme cambia. El control de color permite elegir qué color tiene el rastro dejado por el gráfico.
- **familia**: es un checkbox que determina si el gráfico ha de pintarse de forma única, o si ha de repetirse su trazo varias veces. El número de veces y las posiciones de trazo

de una familia están determinadas por los controles *parámetro*, *intervalo* y *pasos* descritos a continuación.

- **parámetro**: es un campo de texto en el que se introduce una variable o letra cuyo valor varía de acuerdo al número de pasos que caben en un intervalo dado. El número de pasos e intervalo se describen a continuación.
- **intervalo**: es un campo de texto en el que se introduce un intervalo entre corchetes cuadrados. El intervalo consiste en dos números separados por una coma. El primer número es el primer valor que tendrá la variable dada en *parámetro*, mientras que el segundo número será el último valor que tiene. Este intervalo se divide en $n-1$ partes iguales (donde n es el número de pasos elegido) de tal forma que adopta valores equidistantes uno del siguiente dentro del intervalo. El número de pasos usado es el que se detalla a continuación.
- **pasos**: es un campo de texto donde se indica el número de pasos que da la variable usada en *parámetro* de tal forma que, empezando en el valor izquierdo del intervalo, salta de forma equidistante ése número de pasos y termina en el valor derecho del intervalo.
- **ancho**: es un campo de texto en el que se indica el número de pixeles de grosor del gráfico en cuestión.
- **info**: es un campo de texto donde suele introducirse una descripción del gráfico. Es una buena práctica introducir información relevante en este control para fácilmente ubicar los gráficos según su función, sobre todo cuando hay un gran número de gráficos. Este control no tiene efecto en el interactivo, y sólo sirve para identificar más fácilmente un gráfico dado. Si el campo *info* se deja en blanco, la descripción del gráfico en el panel de gráficos a la izquierda puede no ser muy clara (por ejemplo, para el gráfico *ecuación* muestra la ecuación solamente). Pero si se llena con una breve descripción, será precisamente esta descripción la que se mostrará en el panel para poder localizar un determinado gráfico de forma más ágil.
- **estilo de línea**: es un menú con tres opciones. La primera es *sólida*, con la cual el contorno del gráfico en cuestión se trazará de forma sólida. La segunda opción es de forma punteada, y la tercera permite trazarlo mediante líneas discontinuas (con guiones).

8.4. Controles comunes a los gráficos 3D

A continuación se presenta una descripción de los controles comunes a estos gráficos. No se presenta una descripción de todos; sólo de aquellos particulares a los gráficos 3D y que no están incluidos en los controles comunes a los gráficos de dos dimensiones.

- **color reverso**: es un botón que lanza la herramienta de control de colores. Para ciertos gráficos tridimensionales, además del color común que se define también para

los gráficos bidimensionales, hay un color reverso que es el de la cara posterior del gráfico. Es éste el que se define mediante el control en cuestión. Para más información sobre la funcionalidad de la herramienta que se lanza con este botón, consulte el apartado sobre la herramienta de control de colores.

- **rotini**: es un campo de texto en el que se introduce la rotación inicial, en forma de un vector de tres entradas. Una vez que se define un gráfico mediante el campo *expresión* en el selector *gráficos 3D*, éste se puede rotar de forma inicial con el campo en cuestión. La primera entrada del vector corresponde a una rotación del gráfico, en grados, alrededor del eje *x*. La segunda entrada corresponde a una rotación respecto al eje *y* y la tercera respecto al eje *z*.
  En ocasiones puede usarse una rotación con ángulos de Euler, en la cual la primera entrada corresponde a una rotación en grados respecto al eje *z*, con lo que el eje *x* ya no es el mismo. La segunda entrada corresponde a una rotación en grados respecto al nuevo eje *x*, con lo que el eje *z* habrá cambiado. Y la tercera entrada corresponde a una rotación en grados respecto al nuevo eje *z*. Para usar este tipo de ángulos es necesario incluir la palabra *Euler* inmediatamente antes del vector.

- **posini**: es un campo de texto en el que se introduce un vector de tres entradas que corresponde a la traslación inicial del gráfico. La primera entrada corresponde a la traslación en el eje *x*, la segunda a aquella en el eje *y* y la tercera a aquella en el eje *z*. Esta traslación se aplica después de haber hecho la rotación inicial.

- **rotfin**: es un campo de texto que funciona igual que el *rotini*, pero que se aplica después de haber hecho la traslación inicial definida por el campo *posini*. Esta rotación se hace también a partir del origen.

- **posfin**: es un campo de texto que funciona igual que el *posini*, pero que se aplica después de haber hecho la rotación final definida por el campo *rotfin*.

- **cortar**: es un checkbox que cuando está marcado permite que los gráficos que se intersecan en el espacio sean pintados considerando estas intersecciones. De esta forma se hace notoria la intersección entre gráficos, que de lo contrario puede no pintarse correctamente. Dado que esta opción puede ralentizar al interactivo, sólo debe activarse cuando no hay muchos gráficos que se intersecan entre ellos.

- **aristas**: es un checkbox que cuando está marcado hace que se pinten las aristas del gráfico en cuestión (que pueden ser caras, polígonos y superficies). Viene acompañado de un botón para lanzar la herramienta de control de color, mediante la cual se puede elegir el color deseado para las aristas.

- **modelo**: es un menú con cuatro opciones:

  color: se usa para pintar el gráfico con un color fijo.

  luz: se usa para dar sensación de iluminación. De tal forma que el gráfico puede ser más o menos brillantes en algunas partes dependiendo de su orientación.

metal: se usa para dar sensación de iluminación igual que la opción *luz*, pero aumenta el contraste de los brillos, con lo que la superficie aparenta ser metálica.

alambre: se usa para sólo trazar las aristas del gráfico con el color seleccionado. En este caso, el color de las aristas no será gris, sino que tendrá el color especificado para el objeto.

- **Nu**: es un campo de texto en donde se introduce un número entero que determina, para los gráficos que dependen de un parámetro *u*, en cuántas partes se dividirá el intervalo [0,1] que da sus valores a *u*.
- **Nv**: es un campo de texto que funciona igual que el *Nu*. Aplica a los gráficos que requieren dos parámetros para ser trazados, como el gráfico *superficie*.
- **ancho**: es un campo de texto que aplica a algunos gráficos como el *elipsoide* que indica el ancho del mismo.
- **largo**: es un campo de texto que aplica a algunos gráficos como *elipsoide* que indica el largo del mismo.

9

El selector *Controles*

Ya que hemos analizado el selector *Gráficos*, nos conviene regresar al de controles. En algunos ejercicios ya se ha visto un poco sobre los controles numéricos. En este apartado se abordarán todos los controles de forma profunda.

Los controles son objetos con los que el usuario, valga la redundancia, controla el interactivo. Éstos permiten que el usuario haga modificaciones a los valores de las variables, lance animaciones, ejecute funciones, así como un gran número de otras acciones. Muchas de estas herramientas de interacción son seguramente familiares para el usuario, como lo son los pulsadores, las barras horizontales, los menús y los botones.

Al igual que en el caso de los gráficos ya abordados, hay muchos elementos en común entre los diferentes controles. Se estudiarán sólo los elementos particulares a cada control, y al final de este apartado se estudiarán los elementos en común a todos los controles.

El selector *Controles* se escoge haciendo clic en la opción *Controles* hasta arriba del editor de configuraciones. En la Figura 9.1 se muestra el editor de configuraciones cuando está seleccionado el selector *Controles* justo después de que se añade un control tipo pulsador nuevo.

Al igual que con los gráficos, el selector de controles tiene un panel a la izquierda dentro del cual se enlistan los controles existentes en todo el interactivo. Arriba de este panel se encuentran los botones para crear, duplicar, desplazar dentro de la lista y eliminar algún control. Su funcionamiento es igual que en los gráficos, cuya descripción se encuentra en el apartado Gráficos. También cuenta con el menú que permite filtrar los controles de un determinado espacio. Por defecto, dicho menú se encuentra en la opción de asterisco (*), en la cual se muestran todos los controles en la escena. Como ejemplo, en la Figura 9.1 se muestra el selector *Controles* como se muestra por defecto tras añadir un control tipo pulsador.

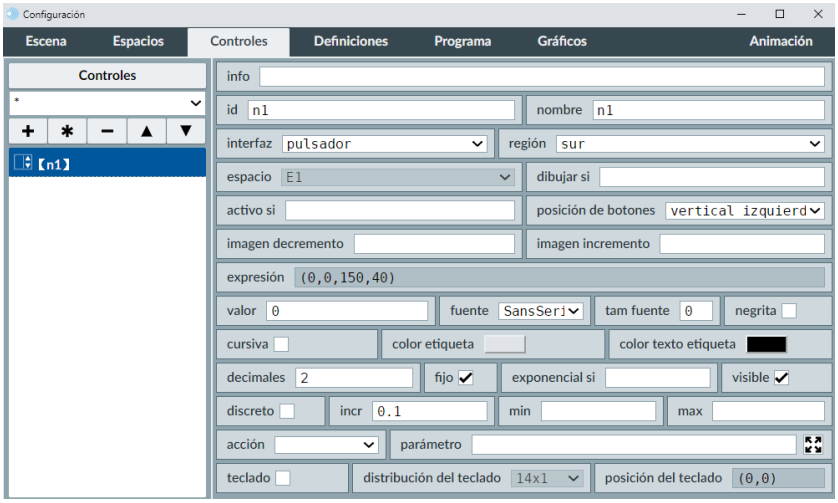


Figura 9.1: Ejemplo del selector *Controles*.

Hay varios tipos de controles: los llamados *numéricos* que involucran el pulsador, campo de texto, menú, barra, botón y casilla de verificación; además de los *gráficos*, el *texto*, *video* y *audio*. Al añadir un nuevo control, aparece una ventana donde se ha de seleccionar en un menú el tipo de control deseado.

9.1. Control numérico tipo *Pulsador*

Este tipo de control se asocia con una variable que tiene el mismo nombre del identificador del pulsador. El pulsador permite aumentar y disminuir los valores de la variable en cuestión mediante un par de botones que incluye el control. Adicionalmente, también puede involucrar un campo de texto en el cual se le puede teclear un valor o expresión para modificar su valor. La mayoría de los elementos del pulsador se pueden consultar dentro de los elementos comunes a los controles. Los particulares al pulsador son:

- **posición de botones**: es un menú mediante el cual se determina cómo se disponen los botones de decremento / incremento del pulsador respecto al campo de texto del mismo. Sus opciones son:
  - **vertical izquierda**: (Opción por defecto) Con esta opción, los botones de decremento / incremento del pulsador estarán a la izquierda del campo de texto del pulsador y el de decremento abajo del de incremento (es decir, dispuestos verticalmente).
  - **vertical derecha**: Con esta opción, los botones estarán dispuestos en forma vertical, pero a la derecha del campo de texto del pulsador.
  - **horizontal izquierda**: Los botones están dispuestos horizontalmente (el de decremento a la izquierda del de incremento), y ambos a la izquierda del campo de texto del pulsador.

  - **horizontal derecha**: Los botones están dispuestos horizontalmente, y ambos a la derecha del campo de texto del pulsador.
  - **horizontal extremos**: Los botones están dispuestos horizontalmente, pero en medio de ambos yace el campo de texto del pulsador.

- **imagen decremento**: es un campo de texto, por defecto vacío (en cuyo caso no se asociará una imagen) en el que se puede introducir la ruta de un archivo de imagen relativa a la carpeta en que se encuentra el archivo *html* de la escena. Esta imagen se asocia al botón del pulsador encargado de incrementar su valor. Por ejemplo, `../images/baja.png` asociará al botón de decremento la imagen *baja.png* ubicada en la carpeta *images* que se encuentra en una ubicación por encima de donde está alojado el *html* de la escena. Cabe decir que la imagen sólo será visible una vez que el archivo haya sido guardado y luego cargado, pues de lo contrario no se puede saber respecto a qué carpeta se sigue la ruta. La imagen se escala en caso que el tamaño de la imagen no coincida con el del pulsador.
- **imagen incremento**: actúa igual que el *imagen decremento* recién mencionado, pero la imagen indicada en este se asocia al botón de incremento del pulsador.

En la Figura 9.2 se muestra un ejemplo de un control numérico tipo pulsador en una escena. En la Figura 9.3 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo.

Note que el pulsador en este ejemplo se supone se puede usar para controlar la escala del espacio *E1*, como se indica en el parámetro de sus cálculos. Observe que la ubicación del control en este ejemplo se usaron variables de espacio, que pueden consultarse en el apartado Variables de Espacio

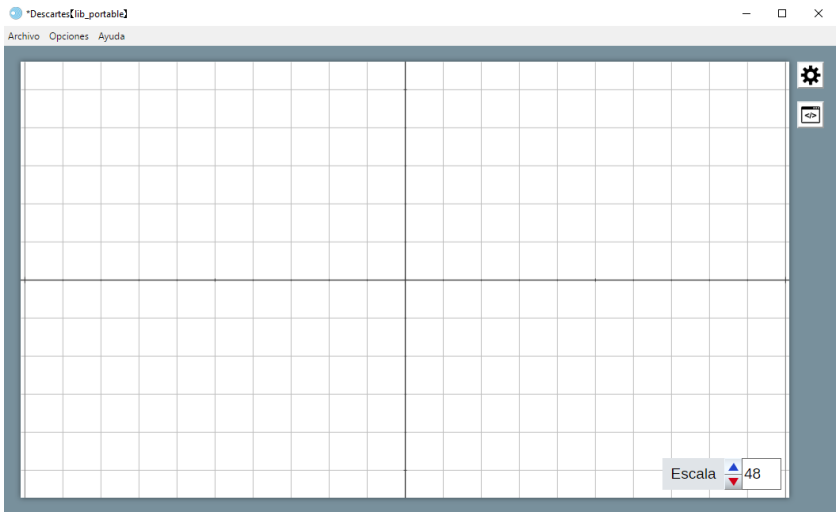


Figura 9.2: Ejemplo del control numérico *Pulsador*.

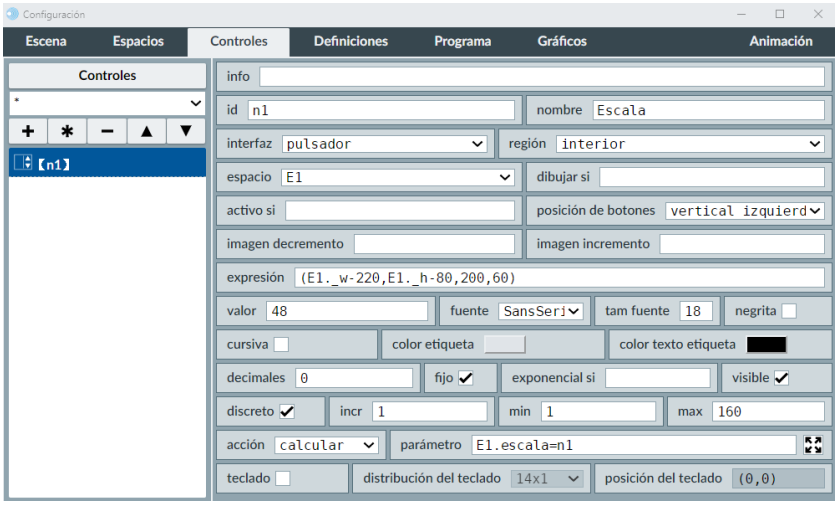


Figura 9.3: Configuración del ejemplo del control numérico *Pulsador*.

Cabe mencionar que un pulsador que tiene su casilla de verificación *visible* marcada muestra un campo de texto en el que el usuario puede indicarle el valor a adoptar. Esta casilla, además de aceptar valores numéricos explícitos, puede aceptar también expresiones que evalúa mediante una calculadora interna. Por ejemplo, si se introduce `2+1` y se pulsa *INTRO* para aceptar, el valor del pulsador cambia automáticamente a `3`. O bien, si se introduce `pi/2` devolverá 1.5707... (dependiendo de los decimales indicados en el pulsador, y considerando que `pi` corresponde al valor de π). Este comportamiento también lo tiene el control tipo Campo de texto.

Hagamos un breve ejercicio con dos pulsadores que controlan el ancho y alto de un triángulo rectángulo. El interactivo mostrará la longitud de la hipotenusa y el área del triángulo. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Controles Pulsador. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio se pudo observar la importancia del menú *acción* de los controles, así como la forma de introducir el texto en el parámetro del mismo. También se observa la importancia de incluir cálculos iniciales, como los usados en *INICIO* para que el interactivo quede listo con los datos correctos desde el inicio.

9.2. Control numérico tipo *Campo de texto*

Este tipo de control numérico consiste en un campo donde el usuario puede introducir texto o números. Resulta principalmente útil en fines de evaluación. Sus elementos principales se encuentran descritos en el apartado Elementos comunes a todos los controles.

En la Figura 9.4 se muestra un ejemplo de un control numérico tipo campo de texto en una escena. En la Figura 9.5 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo.

Note que para este ejemplo el campo de texto se supone se usa para recibir el nombre del usuario. El campo es sólo de texto y aparece vacío de inicio (por eso su valor es un par de comillas sencillas, que es vacío). En la imagen, el usuario ya introdujo su nombre en él. El texto a la izquierda del campo de texto se introdujo mediante la Herramienta de introducción de textos y no tiene que ver directamente con el campo de texto.

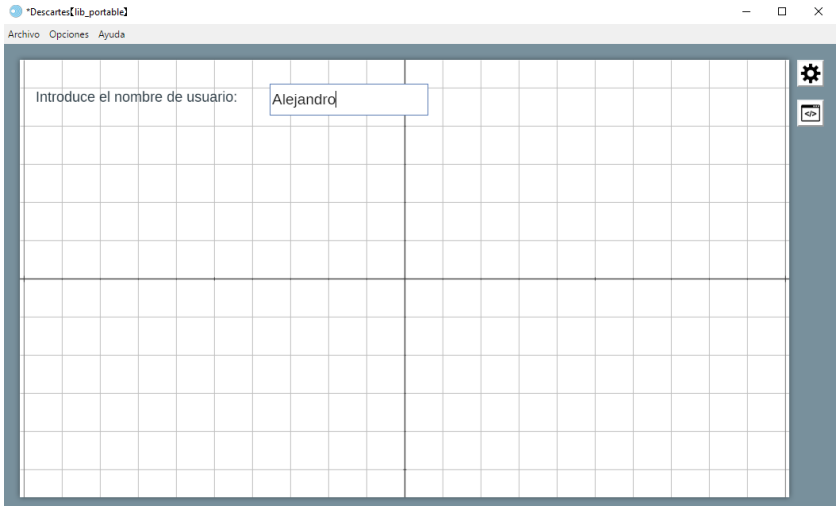


Figura 9.4: Ejemplo del control numérico *Campo de texto*.

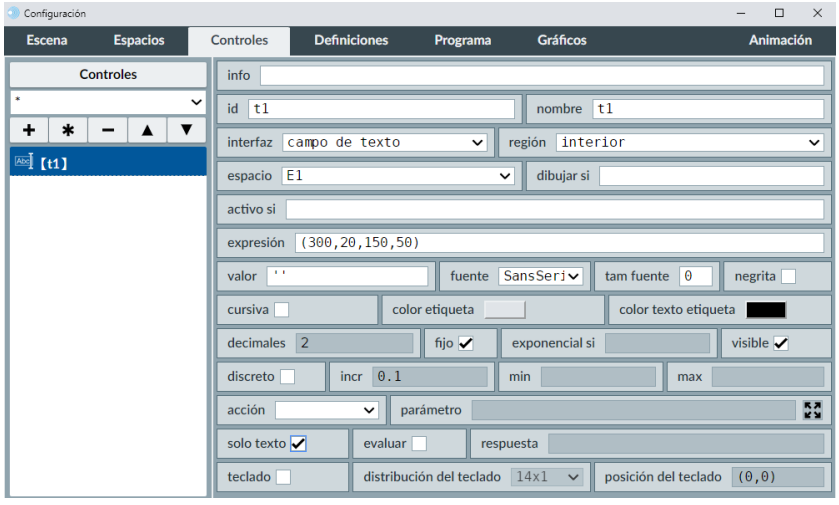


Figura 9.5: Configuración del ejemplo del control numérico *Campo de texto*.

- **solo texto**: es un checkbox que indica que el contenido del campo de texto se leerá preferentemente como una cadena de texto y no como un número. El comportamiento del campo de texto depende mucho de si este checkbox está marcado o no. Por ejemplo, si está marcado y el *valor* del campo de texto es 123, el hacer la operación de suma de este campo de texto con 3 hará que este último 3 se vea como cadena y concatenará ambos para generar *1233*. Pero si no está marcado el checkbox, el resultado de la operación será la suma de 123 + 3 y el resultado será 126. No obstante, Descartes tiene un cierto grado de reconocimiento de errores. Por ejemplo, si el campo es de tipo *solo texto*, y se hace la operación de su contenido 123 pero ahora multiplicando por 3, aunque sea de *solo texto*, Descartes no generará un error, sino que entenderá que el contenido es el número 123 y el resultado que dará es 369.

  Cuando este checkbox está marcado, se inactivan los parámetros *decimales*, *fijo*, *exponencial si*, *discreto*, *incr*, *min* y *max* del campo de texto, debido a que sólo tienen sentido cuando el contenido del campo es distinto de texto puro.

Los campos de texto cuentan con una evaluación automática de expresiones, siempre y cuando su casilla *sólo texto* **no** se encuentre marcada (un comportamiento que comparten con el control tipo Pulsador, cuando el valor de pulsador es visible). Por ejemplo, si se introduce `2+1` en un campo de texto así dispuesto y se pulsa *INTRO*, el campo de texto mostrará 3. Igualmente, si se introduce `cos(pi)` y se pulsa *INTRO*, el campo cambiará su valor a -1 (pues `pi` es interpretado como el valor de π). Sin embargo, si un campo de texto tiene activada su casilla *sólo texto*, entonces al pulsar *INTRO* no se usa la función calculadora y el campo mantiene el texto indicado. En ocasiones conviene que el campo no evalúe mediante su calculadora interna una expresión. En dicho caso, se puede echar mano de la función `_Num_()`, como se describe en el apartado sobre Funciones del lenguaje de Descartes.

Hagamos un breve ejercicio con un campo de texto donde se muestra cómo se puede manipular lo que el usuario ingresa en él. Se verá la diferencia entre un campo de texto numérico y uno que maneja sólo texto. Aprovecharemos para practicar un poco con el gráfico tipo *texto*. Para información detallada sobre los textos se puede consultar la herramienta de introducción de textos.

El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Controles Campo Texto. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *Documentacion-DescartesJS.zip*.

En este ejercicio tuvimos la oportunidad de ver muchas cosas importantes. En particular se vio la diferencia entre variable que llevan texto y las que llevan valores numéricos. El error NaN, además de aparecer cuando se intenta dividir por cero, o cuando se intenta extraer la raíz cuadrada de un número negativo, típicamente también aparece cuando se intenta operar algo que no es un número. Además es importante recordar que porque se le asigne un número a un campo de texto, no necesariamente será considerado como un número, sino que puede tomarse como un caracter o cadena de caracteres.

9.3. Control numérico tipo *Menú*

Este tipo de control numérico es un menú típico. Resulta principalmente útil para permitirle al usuario tener acceso a alguna opción dentro del interactivo. Muchos de los parámetros del menú son comunes a todos los controles y se pueden consultar dentro de los elementos comunes a los controles.

En la Figura 9.6 se muestra un ejemplo de un control numérico tipo menú en una escena. En la Figura 9.7 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo.

Note que para este ejemplo el menú se supone se usa para recibir una respuesta a una pregunta. El campo tiene 4 opciones, aunque sólo 3 de ellas corresponden a una posible respuesta (las últimas 3). El texto a la izquierda del menú se introdujo usando la Herramienta de introducción de textos y no tiene que ver directamente con el menú.

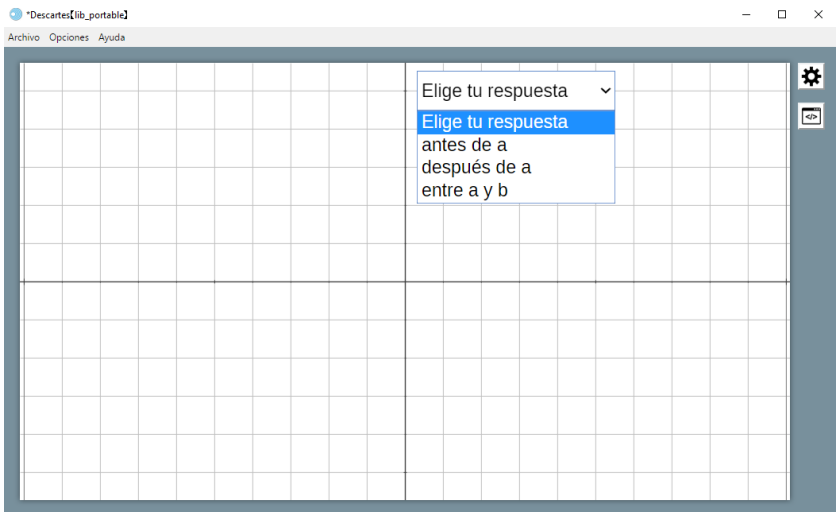


Figura 9.6: Ejemplo del control numérico *Menú*.

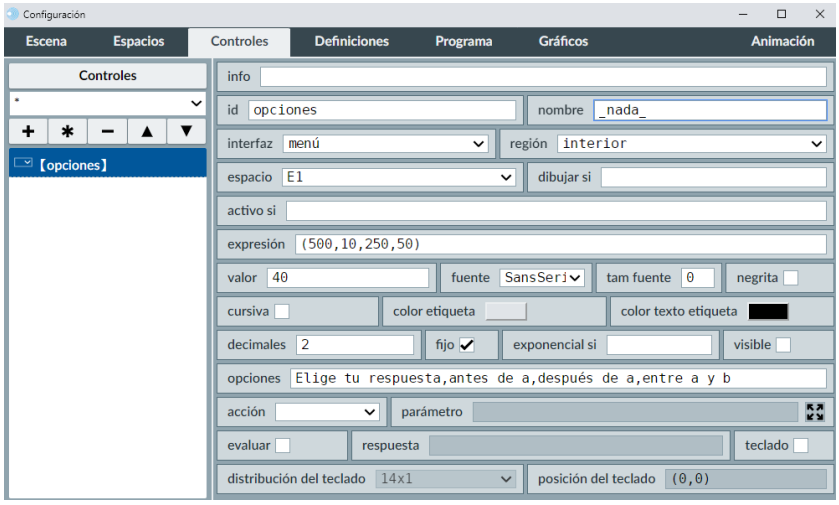


Figura 9.7: Configuración del ejemplo del control numérico *Menú*.

- **valor**: es un campo de texto que indica el valor original que adopta un menú. Para el caso de los menús, que tienen un cierto número de opciones (descritas a continuación), el *valor* debe ser un número entero cero o mayor que indica cuál de las opciones estará seleccionada al iniciar el interactivo. Por ejemplo, un menú con las opciones *sumar*, *restar*, *multiplicar* y *dividir* presentará como opción seleccionada al inicio *multiplicar* si su *valor* esta en 2. Ello se debe a que la primera opción conlleva un valor 0, la segunda un valor 1, la tercera un valor 2 y la cuarta un valor 3.
- **opciones**: es un campo de texto en el que se introducen las opciones del menú. Las opciones se separan mediante comas (,). Si se desea que la primera opción esté vacía, se puede colocar un espacio, luego una coma, y luego el resto de las opciones. Esto se hace cuando se quiere que el menú no tenga una opción particular seleccionada al iniciar el interactivo, sino que aparezca vacío. Las opciones del menú determinan el valor del mismo. La primera opción hace que éste valga 0, la segunda opción hace que valga 1 y así sucesivamente.

Ahora abordaremos un ejercicio que consiste en mostrar distintas figuras y texto dependiendo de la opción del menú seleccionado. Aprovecharemos este ejercicio para practicar, además del menú, algunas otras funcionalidades como las condicionales, algunos gráficos y el uso de espacios para texto. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Controles Menú. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Este ejercicio es un poco más profesional respecto a los que se habían hecho anteriormente. Por un lado, ya se está tomando en consideración la estética de la escena al centrar el menú y al no mostrar el plano cartesiano que para nuestros propósitos no es necesario. Adicionalmente, el identificador del menú es ahora un nombre escogido por el usuario

que permite, valga la redundancia, identificar al control y lo que hace de manera más inmediata. Los gráficos ahora tienen una descripción en su parámetro *info*, lo que permite también identificarlos con más facilidad.

9.4. Control numérico tipo *Barra*

Este tipo de control numérico consiste en una barra de desplazamiento (o scroll) horizontal que, al moverla, cambia su valor. Puede servir para moverse de un valor a otro alejado de forma rápida, aunque también permite cambios de valores más finos. Todos los parámetros de este control se encuentran entre los elementos comunes a los controles, por lo que no se detalla ninguno en este apartado.

En la Figura 9.8 se muestra un ejemplo de un control numérico tipo barra en una escena. En la Figura 9.9 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo.

Note que para este ejemplo la barra se supone también se usa para controlar la escala del espacio *E1*, como se ve en los parámetros de su cálculo. Esto no es funcionalidad propia de la barra, pero puede consultarla en el apartado Variables de Espacio.

Si el alto de la barra es mayor que el ancho, el control se dispondrá con orientación vertical en lugar de horizontal.

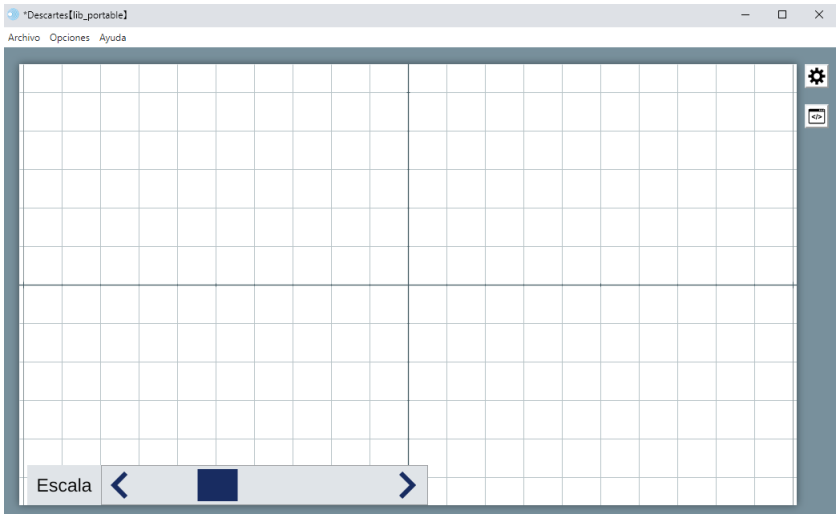


Figura 9.8: Ejemplo del control numérico *Barra*.

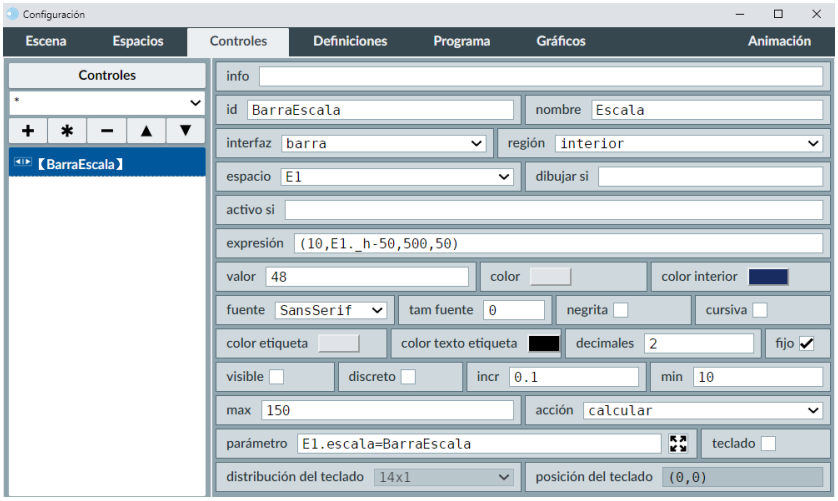


Figura 9.9: Configuración del ejemplo del control numérico *Barra*.

- **color**: es un botón que lanza la herramienta de control de colores para especificar el color de *fondo* de la barra.
- **color interior**: es un botón que lanza la herramienta de control de colores para especificar el color del manipulador de la barra y las flechas de los extremos. Si el color asignado es mediante la pestaña *Gradiente* de la herramienta de colores, los extremos se pintan del color correspondiente a los extremos del gradiente; y el manipulador cambia de color al aproximarse a cada extremo de la barra.

Hagamos un ejercicio que consiste en mostrar un espacio que contiene la gráfica de una ecuación. La idea es que la barra nos sirva como un elemento que controla la escala de este espacio para poder ver particularidades de la gráfica. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Controles Barra. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio pudimos practicar el control de la escala mediante un control numérico. Se abordó también la realización de un cálculo cada vez que se modifica el valor del control numérico. El ejercicio muestra también la utilidad de poder controlar la escala. En este caso, se puede amplificar una gráfica aparentemente tradicional para notar que al acercarse al origen, ella realmente oscila una infinidad de veces. Es importante considerar un posible error, que es usar un valor de cero para la escala. La escala debe ser un valor mayor que cero, razón por la cual el valor mínimo de la barra que la controla es precisamente 10.

Adicionalmente, es útil saber que la barra puede ser también vertical. Esto se logra colocando la barra en el interior de un espacio y asignándoles una anchura menor que su altura.

9.5. Control numérico tipo *Botón*

Este tipo de control numérico es el más simple. Consiste en un botón que al ser oprimido hace alguna acción. Muchos de los parámetros del botón son comunes a todos los controles y se pueden consultar dentro de los elementos comunes a los controles.

En la Figura 9.10 se muestra un ejemplo de un control numérico tipo botón en una escena. En la Figura 9.11 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo.

Note que para este ejemplo el botón se supone se usa para lanzar un cálculo que determinará si la respuesta del usuario es correcta. La respuesta está colocada en otro control numérico (*CamResp*) tipo campo de texto. Observe que, en los cálculos de acción del botón hay una variable *correcto* que de ser correcta la respuesta valdrá 1 y de lo contrario valdrá 2. Puede consultar más a fondo toda esta funcionalidad en el apartado condicionales y operadores booleanos.

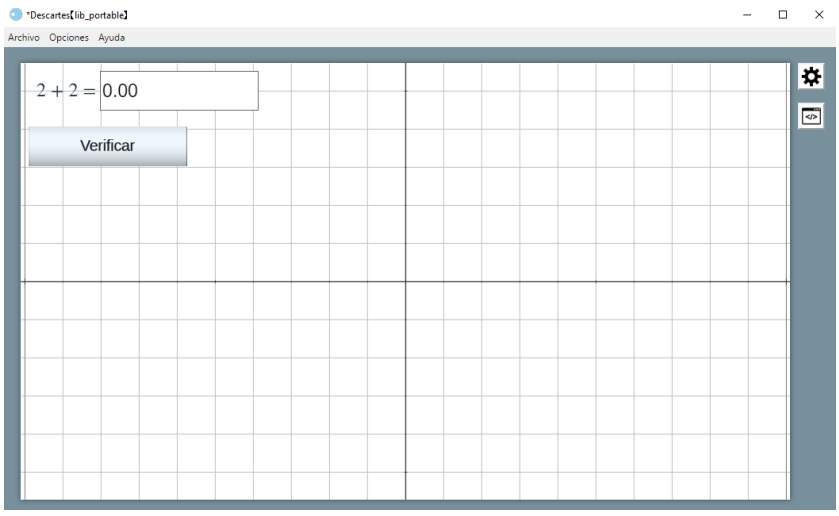


Figura 9.10: Ejemplo del control numérico *Botón*.

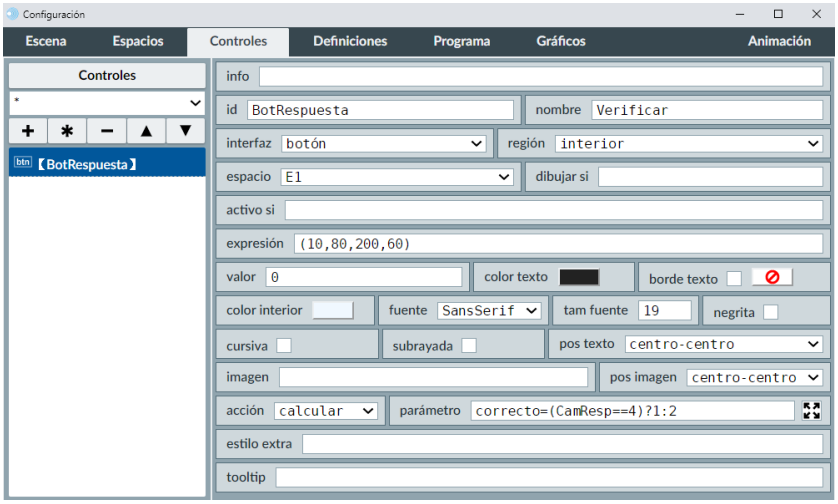


Figura 9.11: Configuración del ejemplo del control numérico *Botón*.

- **color texto**: es el botón que por defecto aparece de color oscuro. Al usarlo, se lanza la herramienta de control de colores, donde se puede seleccionar el color de la fuente del texto del botón.
- **borde texto**: es un checkbox que, de marcarse, habilita un botón a su derecha para controlar el color del borde del texto. Dicho control se hace también a través de la herramienta de control de colores.
- **color interior**: es el botón que aparece por defecto con un color claro. Al oprimirlo lanza la herramienta de control de colores, donde se puede seleccionar el color del fondo del botón. Como se describe a continuación, es posible asignar una imagen jpg o png al fondo del botón con el campo de texto *imagen*. Si esta imagen tiene transparencia, se recomienda que la transparencia del fondo del botón se ajuste al máximo mediante el color interior del botón.
- **fuente**: es un menú del cual se puede seleccionar el tipo de fuente del texto del botón. Las opciones disponibles son *SansSerif*, *Serif* y *Monospaced*.
- **tam fuente**: es un campo de texto en el que se introduce el número de puntos que controla el tamaño de la fuente.
- **negrita**: es un checkbox que debe estar marcado si se desea que el texto del botón aparezca en negritas.
- **cursiva**: es un checkbox que debe estar marcado si se desea que el texto del botón aparezca en letra cursiva.
- **subrayada**: es un checkbox que debe estar marcado si se desea que el texto del botón aparezca subrayado.
- **pos texto**: es un menú que controla la posición del texto del botón respecto a su forma rectangular. Sus opciones son *arriba-izquierda*, *arriba-centro*, *arriba-derecha*,

centro-izquierda, *centro-centro*, *centro-derecha*, *abajo-izquierda*, *abajo-centro* y *abajo-derecha*.

- **imagen**: es un campo de texto donde se introduce una ruta relativa a la carpeta donde se encuentra el interactivo, y que apunta hacia a una imagen png o jpg que se ha de usar como fondo del botón. Si la imagen se encuentra en una carpeta, es necesario usar la diagonal sencilla (/) en la ruta. Por ejemplo, si se quisiera usar una imagen *bot.png* en la carpeta *images* que está en la misma carpeta que el interactivo, tendría que escribirse en este campo *images/bot.png*. Dado un archivo de imagen en una ruta proporcionada, se pueden asociar otro par de archivos con el mismo nombre y unos sufijos particulares para mostrar cuando el puntero se pasea sobre el botón sin oprimirlo, o bien cuando el botón es oprimido:
  - *_down*: si un archivo de imagen ha sido proporcionado (por ejemplo, *bot.png*) en una ruta determinada, y se incluye en la misma carpeta un archivo con la misma extensión y sufijo *_down* (por ejemplo, *bot_down.png*), esta imagen se muestra cada vez que el botón está siendo oprimido.
  - *_over*: similar al archivo con sufijo *_down*. Dado el archivo de imagen, si se incluye otro archivo con igual nombre y extensión *_over* (en nuestro ejemplo *bot_over.png*), este archivo se muestra como imagen del botón cuando el puntero se pasea sobre el botón sin oprimirlo.
- **pos imagen**: es un menú que controla la posición de la imagen del botón respecto a su forma rectangular. Aplica si la imagen en cuestión no cubre completamente el área del botón. Sus opciones son *arriba-izquierda*, *arriba-centro*, *arriba-derecha*, *centro-izquierda*, *centro-centro*, *centro-derecha*, *abajo-izquierda*, *abajo-centro* y *abajo-derecha*.
- **tooltip**: es un campo de texto mediante el cual se introduce el texto de la ayuda contextual a mostrar cuando el usuario posa el puntero del mouse sobre el botón por más de 1.5 segundos. De esta forma es posible dar indicaciones más precisas al usuario cuando el nombre del botón resulta insuficiente.
- **estilo extra**: es un campo de texto mediante el cual se le puede definir estilo adicional a los botones. Este estilo es independiente a cualquier imagen que pudiera asociarse al botón. Por ejemplo, a un botón se le podría indicar la siguiente cadena (que involucra todos los tipos de edición al mismo) en este parámetro:
  `border=3|borderRadius=10|borderColor=ff0000|overColor=e0a12b|downColor=0000ff|inactiveColor=c0c0c0|font=Serif|shadowTextColor=a6620e|shadowBoxColor=808080|shadowInsetBoxColor=b46100|flat=1`
  Todo este texto se pone de corrido. Cada indicación de la edición se separa de la siguiente con el símbolo | (conocido como *pipe*, y que suele encontrarse en el teclado a la izquierda del número 1). Después del nombre del parámetro a cambiar se coloca un signo de igual seguido de su valor. De no incluirse alguno de los parámetros, el botón no incluirá dicho cambio y se usará la configuración por defecto. Se describen los parámetros de edición:

- **border**: se le asocia un número que corresponde al número de pixeles que tendrá el marco que rodea al botón.
- **borderRadius**: se le asocia el número de pixeles correspondiente al radio del arco circular que hará las veces de las esquinas del botón. De no indicarse, o dársele un valor de cero, las esquinas serán en ángulo recto.
- **borderColor**: se le asocia el código de color en hexadecimal del borde del botón. Para más información sobre el control de color, se puede revisar el apartado sobre la herramienta de control de color.
- **overColor**: se le asocia el código de color en hexadecimal que adopta el relleno interior del botón cuando se pasea el mouse sobre el botón.
- **downColor**: se le asocia el código de color en hexadecimal que adopta el relleno interior del botón cuando está siendo oprimido el botón del mouse sobre el botón.
- **inactiveColor**: se le asocia el código de color en hexadecimal que adopta el relleno interior del botón čuando el botón está inactivo. En este sentido, dicho color de relleno aparecerá sólo cuando la condición dentro del parámetro *activo si* del botón sea verdadera.
- **font**: permite cambiar por este medio la fuente de texto del botón. Sus opciones son las mismas que en el parámetro *fuente* del botón, a saber: *SansSerif*, *Serif* y *Monospaced*.
- **shadowTextColor**: se le asocia el código de color en hexadecimal que lleva la sombra que produce el texto. De no darse esta indicación, el texto no producirá sombra alguna.
- **shadowBoxColor**: se le asocia el código de color en hexadecimal de la sombra que produce la caja del botón (por fuera del borde de la misma). De no darse esta indicación, la caja no producirá sombra alguna.
- **shadowInsetBoxColor**: se le asocia el código de color en hexadecimal de la sombra que produce el botón como tal (por dentro del borde de la misma). De no darse esta indicación, no se producirá sombra alguna.
- **flat**: se le asocia el valor de 0 si se desea que el botón tenga degradado de color en su interior (como sucede por defecto), o 1 cuando se desea que el color interior del botón sea plano. Cuando hay degradado, éste se aprecia verticalmente en el botón, siendo más claro cerca del centro del mismo y oscureciéndose hacia arriba y hacia abajo.

Ahora haremos un ejercicio que involucra un botón cuya función es calcular el siguiente valor de una sucesión numérica dados dos números iniciales. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Controles Botón. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio practicamos el uso del control numérico tipo *botón* como un medio para realizar cálculos sucesivos. El valor de la variable *suma* en cada paso resulta seguir

una sucesión llamada *Sucesión de Fibonacci*, cuyos primeros valores son 1, 1, 2, 3, 5, 8, 13, 21, 33, ... Como se puede observar, el valor en alguna posición corresponde a la suma de los dos valores anteriores. Se trata de una sucesión recursiva puesto que el valor de alguna posición sólo puede conocerse calculando todos los valores anteriores hasta él. Así pues, el botón realiza el cálculo del valor de la sucesión en la posición en que se encuentra. Algo a notar en este ejercicio es la importancia de inicializar variables en algunas ocasiones. De no inicializar las variables con el valor 1 en este ejercicio, los valores de *suma* nunca cambian. Las variables que no se inicializan siempre adoptan el valor de cero por defecto.

9.6. Control tipo *Casilla de verificación*

Las casillas de verificación, también conocidos como *checkbox*, son controles que pueden estar marcados por una paloma o bien desmarcados. Son especialmente útiles para reactivos de tipo *opción múltiple*.

Existen dos variantes de la funcionalidad de la casilla de verificación. La primera, conocida simplemente como *casilla de verificación*, permite tener varias casillas que pueden estar marcadas simultáneamente. Imagine, por ejemplo, un ejercicio en el que uno ha de seleccionar los géneros de películas que le gustan. En este caso, uno puede escoger tanto *Terror* como *Comedia*. Es decir, no son mutuamente exclusivos de tal forma que si uno elige uno, debe encontrarse el otro desmarcado. No obstante, en algunas situaciones es necesario elegir sólo una opción de las posibles. En ese caso, al marcar una opción, las demás se desmarcan automáticamente. En este caso, a las casillas se les conoce como *radio botones*.

En la Figura 9.12 se muestra un ejemplo de un control casilla de verificación en una escena. En la Figura 9.13 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo.

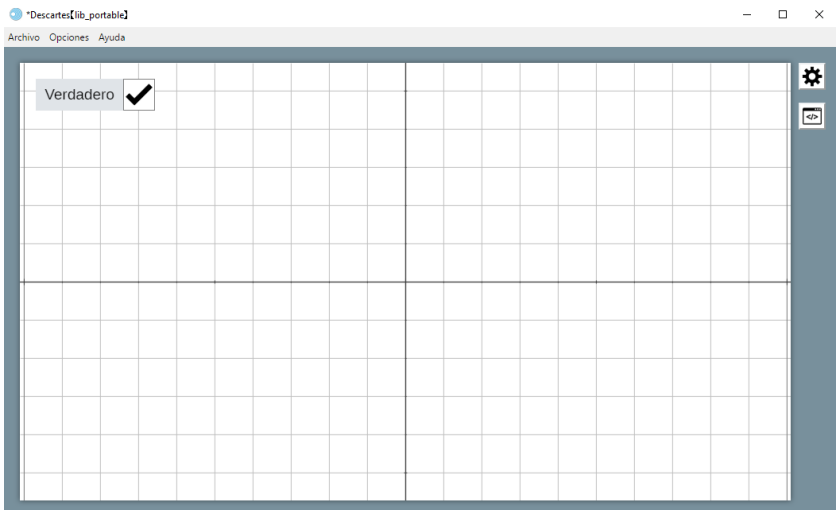


Figura 9.12: Ejemplo del control tipo *Casilla de verificación*.

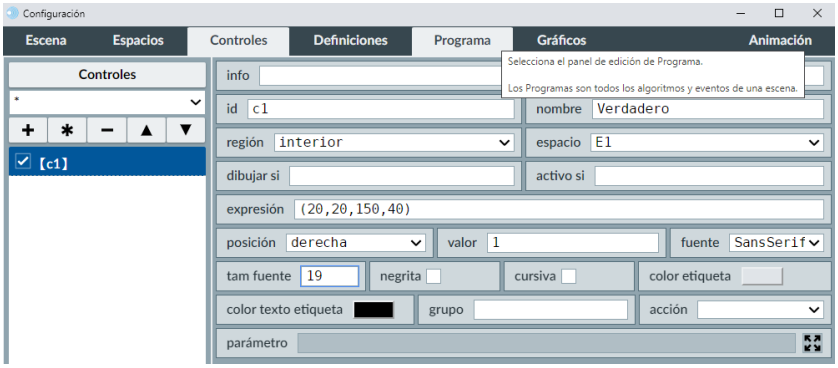


Figura 9.13: Configuración del ejemplo del control tipo *Casilla de verificación*.

- **id**: es el identificador de la casilla, que es el nombre de la variable interna que lleva el valor de la misma.
- **valor**: es el valor que de inicio tiene el identificador de la casilla. En el caso de las casillas de verificación, el valor puede ser 0 (descmarcada) o 1 (marcada).
- **grupo**: es un campo de texto en el que se introduce el nombre del grupo al que pertenecen varias casillas que han de formar parte de un radio botón. En este caso, el marcar una casilla del grupo al que pertenecen otras resultará en desmarcar todas las otras.
- **posición**: es un menú para elegir si la casilla de verificación (el recuadro a marcar o descmarcar) queda a la *derecha* o a la *izquierda* del nombre del control en cuestión.

Hagamos un ejercicio con el objeto de entender mejor las dos funcionalidades de las casillas de verificación. Supongamos que se requiere encontrar cuál de las unidades físicas de una serie de opciones es la que no encaja con las demás. En este caso requeriremos de un radio botón, puesto que sólo debe haber una respuesta que es dispar respecto a las demás.

El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Controles Casilla. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Es importante notar en este ejercicio la diferencia entre usar una casilla de verificación y un radio botón. Todos los controles que son mutuamente excluyentes en un radio botón deben llevar la misma letra en el campo de texto *grupo*.

Por otra parte, en este ejercicio se usó una condición booleana para determinar si la respuesta dada por el usuario es la correcta. Puede consultar más al respecto en el apartado condicionales y operadores booleanos.

9.7. Control tipo *Deslizador*

Los deslizadores (o *sliders* en inglés) son controles con comportamiento similar a la barra, pero donde el objeto deslizable es un círculo que se desliza a lo largo de un eje. El identificador del deslizador es una variable que adopta un valor según la posición del deslizador en el eje, y los extremos determinados para el mismo, mediante los parámetros *min* y *max* del deslizador.

En la Figura 9.14 se muestra un ejemplo de un control casilla de verificación en una escena. En la Figura 9.15 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo.

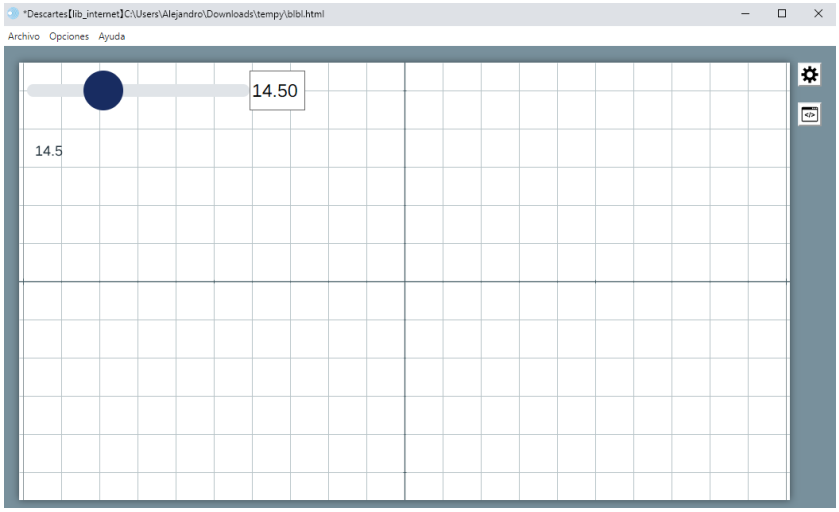


Figura 9.14: Ejemplo del control tipo *Deslizador*.

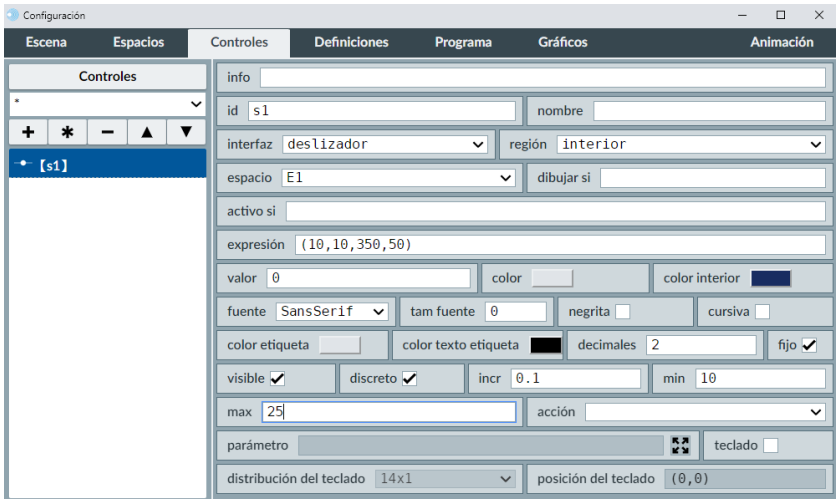


Figura 9.15: Configuración del ejemplo del control tipo *Deslizador*.

Si el ancho del deslizador es menor que el alto del mismo, el control se dispondrá con orientación vertical.

9.8. Control tipo *Gráfico*

Los controles gráficos, a diferencia de los numéricos, consisten en puntos que se pueden manipular directamente con el ratón o, en dispositivos móviles, arrastrarlos en la pantalla táctil. Para agregar un control gráfico se oprime el botón + en el selector *Controles*, con lo que se abre una ventana dentro de la cual se puede seleccionar la interfaz a *gráfico*. Los identificadores de los controles gráficos empiezan con la letra *g* por defecto.

En la Figura 9.16 se muestra un ejemplo de un control gráfico en una escena. En la Figura 9.17 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo. Note que se traza una ecuación en este ejemplo. Ello se logra mediante un gráfico tipo *ecuación*, y la ecuación a trazar es la misma que la constricción para el control gráfico: $y^2 - x^2 = 1$. El propósito de trazar dicha gráfica es para mostrar que el control gráfico sigue esa misma curva cuando se le indica en su parámetro *constricción*.

Note que para este ejemplo el control gráfico se usa como un punto que está constreñido a una determinada curva, que se encuentra trazada aparte como un gráfico tipo ecuación. Se muestra la curva sólo con fines de que el usuario pueda ver que el punto sigue dicha trayectoria, aunque no es indispensable mostrarla. Para mayor información sobre el gráfico usado en este caso, consulte el apartado Gráfico ecuación.

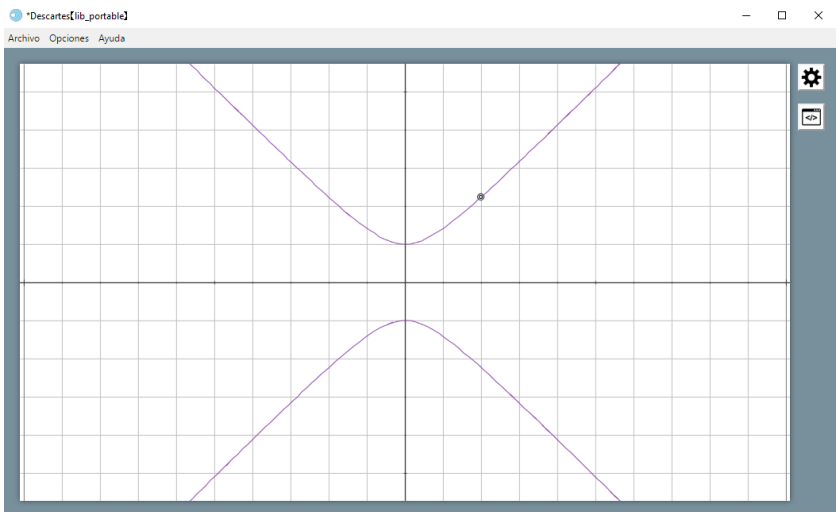


Figura 9.16: Ejemplo del control tipo *Gráfico*.

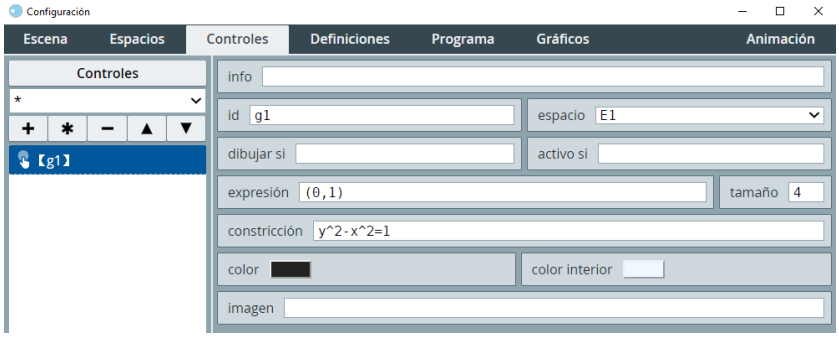


Figura 9.17: Configuración del ejemplo del control tipo *Gráfico*.

- **id**: es un campo de texto donde se agrega el identificador. Este identificador permite, como se verá en breve, extraer propiedades del control como las coordenadas de su posición y si está siendo usado o no. En este sentido no es una variable que guarda un valor como sucede para los identificadores de controles numéricos.
- **espacio**: es un menú donde se escoge en qué espacio aparecerá el control gráfico como el punto interactivo móvil.
- **expresión**: es un campo de texto donde se introduce la coordenada inicial del control gráfico. Por defecto se encuentra en el origen.
- **tamaño**: es un campo de texto en el cual se introduce el radio en pixeles que tendrá el círculo que representa al control gráfico.
- **constricción**: es un campo de texto en el que se introduce una ecuación. Algunas veces es deseable que el control gráfico no pueda moverse a lo largo de todo el espacio que lo contiene, sino que se mueva dentro de una curva permitida. En dicho caso, la ecuación introducida en este espacio restringirá la posición del control. Por ejemplo, una constricción $x^2 + y^2 = 4$ restringirá al control a una circunferencia centrada en el origen y de radio 2. Si la constricción fuera $x^2 + y^2 <= 4$, el control gráfico habitaría en el interior de la circunferencia, pero no podría salirse de ella. Note que si la coordenada indicada en *pos* no es parte de la constricción dada al control gráfico, cuando se aplican los cambios, el editor de Descartes puede no mostrar el control gráfico en la posición elegida, pero al intentar arrastrarlo éste se ajusta inmediatamente a dicha constricción.
- **color**: es un botón que abre la herramienta de control de colores para seleccionar el color del contorno del control gráfico. Como se mencionó, el control gráfico se muestra como un punto o círculo de tamaño ajustable. Es precisamente el color del contorno de este círculo lo que se modifica con este parámetro.
- **color interior**: es otro botón que también abre la herramienta de control de colores para seleccionar el color interior del control gráfico, que es el centro del círculo con que se representa.
- **imagen**: es un campo de texto para la ruta relativa a un archivo de imagen jpg o png

que acompañará al control gráfico. Cuando se usa una imagen, la imagen misma puede arrastrarse como si fuera el control gráfico.

En ocasiones es necesario manipular de forma más precisa y/o conocer el valor de la coordenada horizontal y vertical del control gráfico. Hay dos variables asociadas a los controles gráficos que corresponden a estas coordenadas (que resultan ser las coordenadas relativas al plano del control gráfico). Son <*identificador del control*>.*x* y <*identificador del control*>.*y*. Por ejemplo, la coordenada vertical de un control gráfico con identificador *grf* sería *grf.y*. Estas variables se pueden usar tanto para conocer o imprimir el valor del gráfico, pero también es posible asignarles valores para colocar al gráfico en una posición particular deseada. Más aún, se pueden generar controles numéricos tipo *pulsador* con los nombres de estas variables para poder restringir más finamente el movimiento del control. Por ejemplo, se puede hacer que un control no pueda desplazarse en todo un continuo de valores, sino cada 0.5 unidades, etc.

Hagamos un ejercicio para que todo esto quede más claro. Como motivo de este ejercicio, supongamos que queremos mostrar un histograma del número de personas con edad de 26 y 27 años. El histograma se debe poder construir con columnas cuya altura es ajustable arrastrando un control gráfico. Aprovecharemos el ejercicio para calcular el promedio de las edades.

El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Controles Gráfico 1. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *Documentacion-DescartesJS.zip*.

Pudimos notar en este interactivo la utilidad de poder controlar cosas mediante arrastre de objetos. Esto se puede extender a la manipulación de figuras geométricas, imágenes, etc. Y son precisamente los controles gráficos los que permiten esta funcionalidad. También observamos la utilidad de colocar instrucciones o asignaciones en el algoritmo *CÁLCULOS*. No obstante, como se menciona en el apartado de dicho algoritmo, hay que usarlo con ciertas reservas ya que puede acabar innecesariamente gastando recursos de cálculos en repetir cálculos innecesarios. Finalmente, pudimos notar que en un ejercicio tan sencillo como calcular el promedio de algún dato se presentó un error. Aunque se puede argumentar que el no tener individuos no tiene sentido para un promedio, el interactivo se podría pulir un poco para que no generase errores en la ventana de comandos, y para que el texto mostrado al usuario fuera un poco más explicativo. Algo así como *No se puede buscar un promedio si no se tiene elementos*.

Hagamos otro interactivo con controles gráficos que permite además una herramienta para el programador muy útil. Ya se sabe que hay dos tipos de coordenadas: las absolutas y las relativas. Algunos gráficos permiten el uso de ambas; y todos los controles manejan únicamente las absolutas. Por otra parte, los controles gráficos se manejan en coordenadas relativas. En ocasiones se requiere conocer las coordenadas absolutas de un punto en un espacio para, por ejemplo, saber qué coordenadas darle a un control o un texto. Aunque el control gráfico maneja coordenadas relativas, podemos convertirlas a absolutas y

así, de manera ágil, moverlo a alguna posición y nos dirá inmediatamente tanto las coordenadas relativas como las absolutas de la misma. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Controles Gráfico 2. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

El resultado de este ejercicio es un control gráfico que puede desplazarse a cualquier parte del espacio y nos indica las coordenadas relativas así como las absolutas. Si al programador le gusta un punto particular para colocar digamos un botón, puede colocar el control gráfico ahí y conocerá las coordenadas absolutas que debe usar para el botón.

Este tipo de herramientas son útiles para el programador. Una vez colocados los objetos en las posiciones deseadas, el control gráfico y el punto con su texto se vuelven innecesarios y es conveniente que no se muestren. No es necesario eliminarlos completamente, por lo que se pueden solo ocultar aprovechando los parámetro *dibujar si* tanto del control como del punto. De hecho, si eventualmente ha de cambiarse algo en el interactivo, es posible que se requiera usarlos otra vez, razón por la cual conviene no eliminarlos.

A lo largo de este ejercicio se usaron variables del tipo *E1._w*, *E1._h*, *E1.Ox*, *E1.Oy* y *E1.escala*. Todas estas variables son intrínsecas a Descartes. Si se desea estudiarlas más a fondo puede consultar el apartado sobre variables de espacio.

9.9. Control tipo *Texto*

Los controles tipo *texto* consisten en bloques de texto enriquecido que contienen un marco que los delimita. Es posible editar el texto dentro del recuadro en el interactivo, y no necesariamente dentro del editor de configuraciones. Adicionalmente, se puede mostrar un par de textos tipo pregunta y respuesta mediante un botón que se incluye en la esquina inferior derecha dentro del recuadro de texto.

En la Figura 9.18 se muestra un ejemplo de un control tipo texto en una escena. En la Figura 9.19 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo.

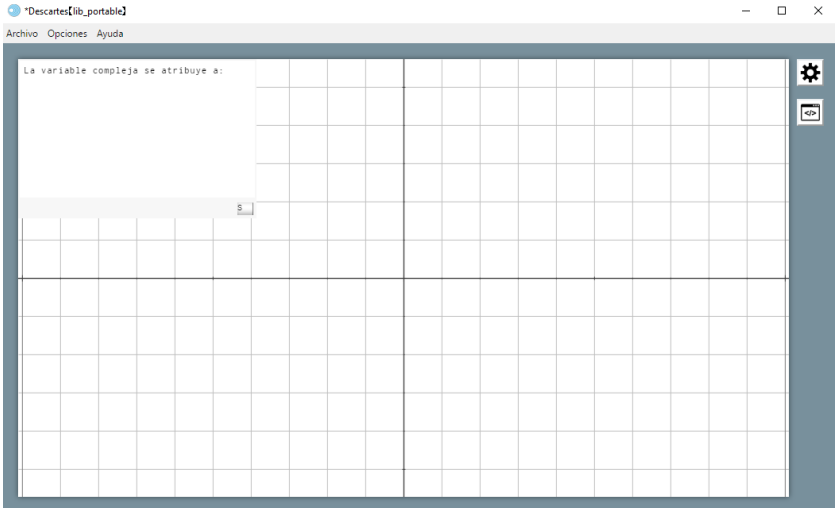


Figura 9.18: Ejemplo del control numérico *Texto*.

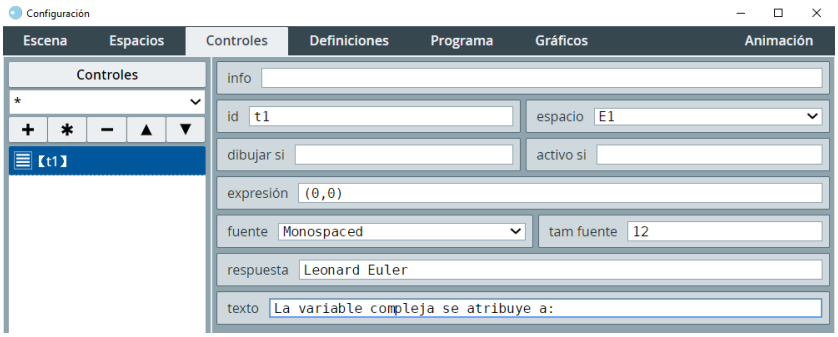


Figura 9.19: Configuración del ejemplo del control tipo *Texto*.

- **fuente**: es un menú para elegir el tipo de fuente usada para el texto.
- **tam fuente**: es un campo de texto en el que se elige el número de puntos para el tamaño de la fuente del texto.
- **respuesta**: es un campo de texto. Consiste en la respuesta a ser mostrada como correcta cuando el usuario quiera corroborar la suya. Note que esta respuesta sólo es mostrada. No se hace una comparación para determinar si lo que el usuario introdujo es correcto o no. Un ejemplo de la respuesta podría ser *El año en que vivimos es 2016*.
- **texto**: es un campo de texto. La idea de uso de este texto es que el programador incluya una pregunta aquí que el usuario ha de responder. Por ejemplo: *Indica en qué año vivimos:*

El interactivo mostrará el texto introducido en *texto*. El usuario podrá poner en ese mismo recuadro su respuesta a la pregunta. Si se incluyó algo en el campo *respuesta* del

control, aparece en la esquina inferior de la caja de texto en el interactivo un botón con la letra *S*. Si el usuario oprime dicho botón, se muestra lo que se haya introducido en el campo *respuesta* y aparece ahora un botón *T* con el que se regresa a la pregunta. La funcionalidad del control tipo texto prácticamente no se usa dado que no permite una comparación de respuesta; sólo muestra la respuesta correcta.

9.10. Control tipo *Audio*

Este control consiste en un reproductor de audio que puede ser activado en el interactivo. Se puede colocar sólo en el interior de un espacio. Reproduce archivos mp3 y wav.

En la Figura 9.20 se muestra un ejemplo de un control tipo audio en una escena. En la Figura 9.21 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo.

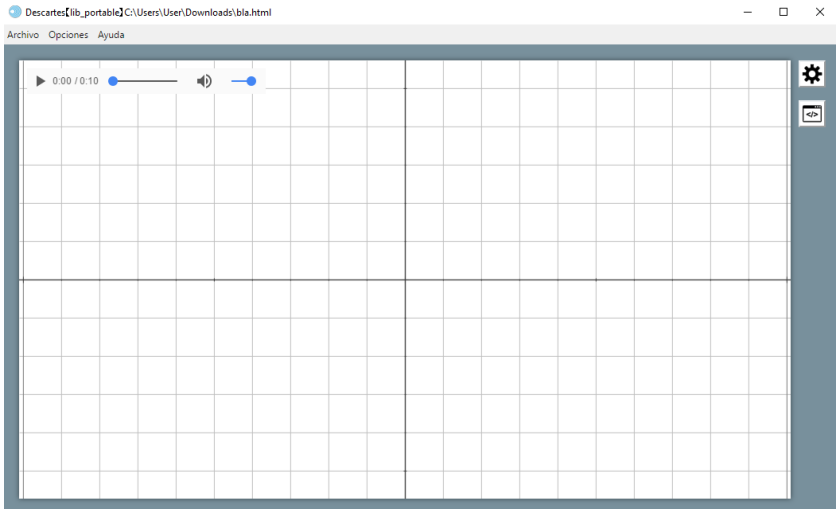


Figura 9.20: Ejemplo del control tipo *Audio*.

Figura 9.21: Configuración del ejemplo del control tipo *Audio*.

- **archivo**: es un campo de texto en el que se introduce una ruta relativa a donde se encuentra el interactivo y que apunta al archivo de audio a reproducir. Recuerde que

si el archivo se encuentra en una subcarpeta, es necesario usar la diagonal sencilla para separar carpetas. Por ejemplo, *audio/ruido.mp3*.

Existen algunas funciones y variables intrínsecas que comienzan con el nombre del identificador del control de audio (para información más detallada se puede consultar el apartado sobre funciones de controles de audio y video y variables de controles de audio y video). Las funciones llevan un paréntesis de apertura y cierre. Cuando la función no maneja argumentos, el interior del paréntesis está vacío. Por ejemplo, <*identificador del control de audio*>.*play*() con la que se puede controlar la reproducción del archivo. La única variable de audio y video no lleva paréntesis.

Hagamos un breve ejercicio en donde se reproducen un par de frecuencias cuando se eligen a partir de un menú. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Controles Audio. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*. Los archivos de audio para este ejercicio se encuentran en 221.mp3 y 371.mp3, y también están dentro del archivo *DocumentacionDescartesJS.zip*.

Algo importante a notar de este ejercicio es que el reproductor que se visualiza en el navegador es nativo al navegador en cuestión y no a Descartes. Puede no verse igual en Chrome que en Firefox, debido a cambios que se hacen en las versiones de dichos programas. Pero los botones básicos del control suelen ser los mismos.

Adicionalmente, fue necesario usar una animación sólo para que el tiempo de reproducción de uno de los archivos se despliegue. Si no se quisiera desplegar, no sería necesario tener la animación. Si se requiere más detalles sobre las animaciones, se puede consultar el apartado Animación.

9.11. Control tipo *Video*

Este control consiste en un reproductor de video que puede ser activado en el interactivo. Su funcionamiento es muy similar al control de audio. Se utiliza para reproducir archivos con formato *mp4*, *WebM* y *ogv*.

En la Figura 9.22 se muestra un ejemplo de un control tipo video en una escena. En la Figura 9.23 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo. Como se observa en la Figura 9.23, este ejemplo es necesario, al igual que para los controles de audio, tener una carpeta *video* a la altura del interactivo dentro de la cual se encuentra el archivo *Todo.mp4* a reproducir.

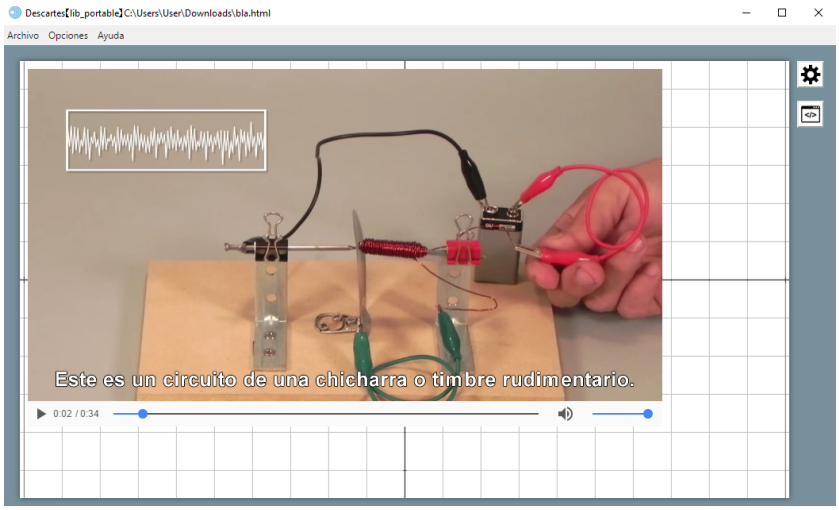


Figura 9.22: Ejemplo del control tipo *Video*.

Figura 9.23: Configuración del ejemplo del control tipo *Video*.

- **archivo**: es un campo de texto en el que se introduce una ruta relativa a donde se encuentra el interactivo y que apunta al archivo de video a reproducir. Recuerde que si el archivo se encuentra en una subcarpeta, es necesario usar la diagonal sencilla para separar carpetas. Por ejemplo, *video/peli.mp4*.

Los controles de video en Descartes cuentan con las funciones intrínsecas: <identificador del control>.play(), <identificador del control>.stop(), <identificador del control>.pause(), <identificador del control>.currentTime(<segundo de inicio del video>). Adicionalmente, también cuentan con la variable intrínseca <identificador del control>.currentTime que guarda el valor del tiempo en que se encuentra el video en segundos. Como se observa, las funciones y variables intrínsecas son exactamente las mismas que las descritas para controles de audio, que se pueden consultar más detalladamente en el apartado sobre funciones de controles de audio y video y variables de controles de audio y video.

Al igual que para los controles de audio, el control de video que aparece en un navegador puede diferir de otro en funcionalidad. Algunos pueden desplegar, por ejemplo, la opción de extender el video a pantalla completa, otros no. Todo esto depende del reproductor de video del navegador en que se muestran los interactivos, y no de Descartes.

Se recomienda que cuando se use un control de video, el parámetro *expresión* incluya las cuatro entradas de posición en la horizontal, en la vertical, ancho y alto. Conviene que el ancho y alto del control correspondan al ancho y alto del video, o mínimo que respeten la misma proporción.

Si se llegara a requerir una portada para un video (una imagen estática que se muestra al inicio del video), esto se logra guardando en la misma carpeta que el video una imagen png cuyo nombre es el mismo que el del video y cuyas dimensiones también son iguales a las del video.

Debido a la similitud entre la funcionalidad del control de audio y video, no se incluye un ejercicio para este apartado. Se recomienda, sin embargo, que el usuario haga experimentos con algún video.

9.12. Elementos comunes a múltiples controles

A continuación se muestran los elementos comunes a los controles.

- **info**: es un campo de texto en que se introduce una breve descripción del control en cuestión para poder ubicarlo mejor posteriormente. Al igual que el *info* de los elementos en el selector *Gráficos*, lo que se introduzca en este campo será lo que se muestra para el control en cuestión en la lista del panel de controles, lo que permite fácilmente localizar un control.
- **id**: es un campo de texto conocido como *identificador*. Es el nombre que se le asigna al control dentro de Descartes, y no el que se observa dentro del interactivo. Cuando se quiere hacer referencia al valor del pulsador como una variable, éste es el nombre de la variable que lo representa. Como finalmente es una variable, cabe mencionar que ningún identificador puede empezar con un número. El identificador se puede definir en la ventana que aparece al pulsar el botón de agregar un control nuevo (arriba del panel de la lista de controles y que tiene un signo + en su interior). No obstante, posteriormente siempre se puede cambiar en el campo de texto *id*.
- **nombre**: es un campo de texto en el cual se introduce el nombre del pulsador como habrá de verse en el interactivo. Sólo aplica para controles de tipo numérico. A diferencia del *identificador*, este nombre no es la variable con la que se relaciona al control, sino simplemente el nombre que el usuario final verá en el interactivo. Aunque por defecto el nombre del control se copia del identificador, es posible, y muchas veces deseable, que el nombre se cambie por otro. Por ejemplo, un control numérico de tipo pulsador cuyo objeto es ajustar una distancia en un interactivo puede tener como identificador *distPl*. Así, el programador rápidamente lo puede localizar como el pulsador que controla la distancia. Sin embargo, es preferible que este identificador no sea el nombre final del pulsador en el interactivo. En su lugar, el nombre final podría ser algo como *control de distancia*. En caso que no se desee mostrar explícitamente el nombre del pulsador, se puede introducir el texto *_nada_* en este campo o, en su defecto, simplemente dejarlo vacío.

- **interfaz**: es un menú que por defecto aparece en la opción *pulsador*. Determina el tipo de control que se desea. Sus opciones son *pulsador*, *campo de texto*, *menú*, *barra* y *botón*. Los detalles de cada uno de estos tipos de control se detallan a lo largo del apartado de Controles.
- **región**: es un menú que por defecto aparece en la opción *sur*. Determina donde ha de colocarse el control.

  sur: hace que el control se coloque en una barra horizontal insertada en la parte inferior del interactivo, fuera de los espacios. Pueden incluirse varios controles con esta opción, y todos los que la tengan aparecerán en dicha barra.

  norte: hace que el control se coloque en una barra horizontal insertada en la parte superior del interactivo, fuera de los espacios. Pueden incluirse varios controles con esta opción, y todos los que la tengan aparecerán en dicha barra.

  este: hace que el control se coloque en una barra vertical insertada en la parte derecha del interactivo, fuera de los espacios. Pueden incluirse varios controles con esta opción, y todos los que la tengan aparecerán en dicha barra.

  oeste: hace que el control se coloque en una barra vertical insertada en la parte izquierda del interactivo, fuera de los espacios. Pueden incluirse varios controles con esta opción, y todos los que la tengan aparecerán en dicha barra.

  exterior: hace que el control esté fuera del interactivo, a diferencia de las anteriores cuatro opciones. Un control en el *exterior* se puede mostrar en una ventana que aparece al hacer clic derecho sobre el interactivo, **siempre y cuando la opción mostrar región exterior se encuentre marcada en el selector *Escena***. Los controles colocados en el exterior suelen usarse con propósitos internos para el programados. Es decir, no suelen ser parte del interactivo final.

  interior: hace que el control esté dentro del interactivo, y dentro de un espacio en particular. En este caso, el campo de texto *expresión* explicado en breve determina las coordenadas y tamaño del control en pixeles dentro del espacio que lo contiene.
- **espacio**: es un menú dentro del cual se determina cuál de los espacios existentes alojará al control en cuestión cuando la *región* del control está marcada como *interior*. Por lo mismo, se encuentra desactivada a menos que la *región* se encuentre en *interior*.
- **dibujar si**: es un campo de texto que admite una expresión booleana. De valer 1 dicha expresión, el control en cuestión estará visible, mientras que de valer 0 no lo estará.
- **activo si**: es un campo de texto que admite una expresión booleana. De valer 1 dicha expresión, el control en cuestión estará activo, mientras que de valer 0 no lo estará. A diferencia de *dibujar si*, *activo* determina si el control puede usarse o no. El control puede estar visible, pero si está inactivo se ve de un color más opaco y el usuario no podrá interactuar con él.

- **expresión**: es un campo de texto que determina las coordenadas de la esquina superior izquierda del control (primeras dos entradas de la expresión) y su ancho y alto (tercera y cuarta entradas de la expresión) cuando se encuentra alojado dentro de un espacio. A diferencia de los gráficos, los controles siempre están determinados en coordenadas absolutas. Así pues, si *expresión* tiene *(150,30,200,50)*, la esquina superior izquierda del control estará 150 pixeles a la derecha de la esquina superior izquierda del interactivo y 30 hacia abajo, tendrá un ancho de 200 px y un alto de 50.
- **valor**: es un campo de texto que indica el valor inicial que tendrá el control en el interactivo. Así pues, es un número en la mayoría de las ocasiones, aunque puede ser también un texto en algunos casos cuando el valor está dado por una variable. Su funcionalidad es un poco distinta para el caso de los menús, y se detalla en la descripción de dicho tipo de control.
- **fuente**: es un menú del cual se puede elegir el tipo de fuente de la etiqueta (aquella que imprime el nombre dado al control) de entre las opciones SansSerif, Serif y Monospaced. Este elemento se encuentra presente para todos los controles numéricos.
- **tam fuente**: es un campo de texto en el que se puede especificar el número de puntos del tamaño de fuente usada para imprimir la etiqueta (el nombre) del control. Este elemento se encuentra presente para todos los controles numéricos.
- **negrita**: es una casilla de verificación que cuando está marcada hace que el texto de la etiqueta del control se muestre resaltada en negritas. Este elemento se encuentra presente para todos los controles numéricos.
- **cursiva**: es una casilla de verificación que cuando está marcada hace que el texto de la etiqueta del control se muestre en itálicas. Este elemento se encuentra presente para todos los controles numéricos.
- **color etiqueta**: consiste en un control de color mediante el cual se selecciona el color de la etiqueta en la que se encuentra el texto del nombre del control. El uso de la herramienta asociada se puede consultar en el apartado sobre la herramienta de control de colores. Este elemento se encuentra presente para todos los controles numéricos.
- **color texto etiqueta**: un control de color mediante el cual se selecciona el color del texto de la etiqueta. Es decir, el nombre elegido para el control. El uso de la herramienta asociada se puede consultar en el apartado sobre la herramienta de control de colores. Este elemento se encuentra presente para todos los controles numéricos.
- **decimales**: es un campo de texto en el que se indica cuántos decimales se han de mostrar para el parámetro. Tras evaluar el parámetro, se redondeará el valor a ese número de decimales y ése es el valor que se mostrará.
- **fijo**: es un checkbox que determina si el número de decimales elegido ha de mostrarse siempre (aún cuando los decimales en cuestión no sean significativos) cuando está marcado, o bien si se ha de usar la notación ajustada en la que sólo se muestran

el número de decimales indicado si son significativos cuando dicho checkbox está desmarcado.

- **exponencial si**: es un campo de texto que admite una expresión booleana (es decir, cuyo valor puede ser 0 ó 1). Si el valor de la expresión es 0, nunca se usará la notación exponencial, pero si es 1, es posible que ésta se muestre si el interactivo lo determina pertinente.
- **visible**: es un checkbox que cuando está marcado hace visible el valor del parámetro y cuando no lo mantiene invisible.
- **discreto**: es un checkbox que cuando está marcado obliga al parámetro a adoptar valores que difieren de su valor inicial sólo en múltiplos del incremento indicado. Por ejemplo, un pulsador con valor inicial 0.05, valor mínimo de 0, incremento de 0.1 y no considerado discreto variará su valor de 0.05 a 0.00 cuando se hace clic para disminuir su valor un paso. Sin embargo, uno igual pero con el checkbox *discreto* marcado se mantendrá en 0.05 pues 0.00 no difiere en un múltiplo del incremento del valor inicial indicado.
- **incr**: es un campo de texto común a varios controles que indica el tamaño del incremento de un control numérico. Aplica particularmente para el pulsador y la barra. Por ejemplo, si se desea que un pulsador aumente de 0.5 en 0.5 unidades, se colocaría 0.5 en este campo de texto. En el caso del campo de texto, este parámetro es pertinente cuando se encuentra activa la casilla *discreto*, en cuyo caso el campo sólo podrá adoptar valores que difieren, respecto al valor inicial, en un múltiplo exacto del incremento.
- **min**: es un campo de texto que indica el valor mínimo que el control puede adoptar. Si se deja vacío quiere decir que no hay un límite inferior para el parámetro.
- **max**: es un campo de texto que indica el valor máximo que el control puede adoptar. Si se deja vacío quiere decir que no hay un límite superior para el parámetro.
- **acción**: es un menú que determina qué hará el interactivo cada vez que es usado. Las opciones disponibles en dicho menú son:

  calcular: hará los cálculos indicados en el campo *parámetro*, que se explica adelante. Pueden consistir en asignaciones de valor a variables y llamados a funciones. No permite hacer asignaciones a vectores o matrices. En caso que sea necesario hacer eso, es preciso que la *acción* mande llamar a una función que hará las asignaciones pertinentes a los vectores o matrices.

  inicio: carga el interactivo como si fuera la primera vez que se abre, ignorando cambios que el usuario haya hecho sobre el mismo posteriormente. Su función es la misma que la del botón *inicio* que aparece en la esquina inferior izquierda en un interactivo cuando el checkbox *inicio* en el selector *Escena* se encuentra marcado.

  limpiar: cuando hay gráficos que estén definidos de tal forma que dejen rastro, esta opción permite que cada vez que el control en cuestión es usado, dichos

rastros se limpien. Su función es la misma que la del botón *limpiar* que aparece en la esquina inferior derecha en un interactivo cuando el checkbox *limpiar* en el selector *Escena* se encuentra marcado.

animar: lanza la animación, cuyos detalles se especifican en el selector Animación.

abrir URL: permite abrir una dirección URL al hacer uso del control. La dirección URL destino se especifica en *parámetro* como se explica más adelante.

abrir escena: permite abrir una escena que se encuentra en la misma carpeta que el interactivo. La escena se especifíca en *parámetro*, cuya descripción se aborda adelante.

reproducir: permite reproducir un archivo de audio tipo *.mp3*. Si el archivo se encuentra en la misma carpeta que el interactivo basta introducir su nombre en el campo de texto *parámetro*. También es posible incluir en *parámetro* la ruta a una subcarpeta (cada carpeta separada por el símbolo /) en caso que el archivo de audio se encuentre en ella. Cada vez que el valor del control es modificado, el archivo de audio se reproducirá o pausará alternadamente.

Cuando el parámetro *acción* de un control se encuentra sin acción asociada, o bien con acción *inicio*, *limpiar*, o *animar*, el campo *parámetro* asociado a la acción se inactiva, debido a que dichas acciones no involucran parámetros.

- **parámetro**: Es un campo de texto acompañado de un botón. El botón abre una ventana en la cual el parámetro en cuestión se puede introducir cómodamente como texto en varios renglones. El campo de texto tiene la misma función, pero en una sola línea. En caso que se requiera más de un renglón (como en el caso de asignaciones a más de una variable), en el campo de texto se separa el contenido de cada renglón por punto y coma (;).

  Si la acción del control es *calcular*, el parámetro consiste en asignaciones y/o llamados a funciones.

  Si la acción es *abrir URL*, el parámetro puede llevar una dirección relativa a la carpeta en que se encuentra el interactivo o una dirección absoluta comenzando por ejemplo con *https://*. Es posible también agregar, después de la dirección, una indicación `target=_self`, mediante la cual la página que se abre no se abre en una nueva pestaña de un navegador (como lo hace por defecto), sino en la misma del interactivo. Por ejemplo, si la escena original lleva `./tst.html target=_self` como parámetro de la acción *abrir URL*, entonces la escena *tst.html*, ubicada en la misma carpeta que la escena original, se abre en la misma pestaña del navegador de la escena original, en lugar de en otra pestaña.

  Si la acción es *abrir Escena*, el parámetro ha de llevar el nombre de la escena a abrir con todo y su extensión *.html* que se encuentra en la misma carpeta o en una subcarpeta respecto a donde se encuentra el interactivo que la llama. Al igual que para la acción *abrir URL*, también es posible incluir `target=_self` para indicar que

la escena en cuestión ha de abrirse en la misma pestaña que el interactivo principal, y no en una nueva como sucede por defecto.

En general, se puede establecer el *objetivo* para las opciones *abrir URL* como *abrir Escena*. Por *objetivo* se entiende dónde habrá de abrirse el URL o escena destino, y se establece con el comando `target=` seguido de alguna de las siguientes opciones:

- `_self`: abre el destino en la misma página de la escena.
- `_blank`: abre el destino en una pestaña nueva.
- `_parent`: abre el destino en la página padre. Sólo sirve si la escena está embebida en una escena padre mediante la funcionalidad detallada en el apartado sobre Espacio HTMLIFrame.
- `_top`: abre el destino en el cuerpo completo de la ventana.

Se recuerda que este comando va en el campo *parámetro* asociado a la acción del control; y va siguiendo a la dirección del destino, separada por un espacio.

Si la acción es *reproducir*, el parámetro lleva el nombre del archivo de audio mp3, con todo y su extensión, que se ha de reproducir al activar el control. Si el archivo está en una subcarpeta, es necesario incluir la ruta al mismo.

- **evaluar**: es un checkbox común a campos de texto y menús que, de estar marcado, permite la evaluación automática del control, comportamiento que se aprovecha en la elaboración de reactivos.

  Si dicho checkbox está desmarcado, el parámetro *respuesta* se inactiva, dado que sólo tiene sentido de lo contrario.

- **respuesta**: es un campo de texto común a menús y campos de texto que incluye los elementos a comparar como patrones de respuesta correcta. Sólo tiene sentido usarlo si el checkbox *evaluar* está marcado. Puede consistir en uno o varios campos de texto separados por cada uno de los cuales se compara con lo escrito o seleccionado por el usuario. Si hay una coincidencia se considera que la respuesta es correcta, o errónea de lo contrario.

  - Cuando la respuesta es numérica, el rango de respuestas válidas es un intervalo. Por ejemplo: [a,b], (a,b), (a,b] o [a,b).
  - Cuando el control es un campo de texto del tipo *sólo texto*, entonces la comparación se hace letra por letra.
  - El asterisco * funciona como comodín: si al final de una opción de respuesta se escribe un asterisco *, entonces sólo se busca si la respuesta de usuario comienza con la propuesta. Si hay un asterisco al principio entonces se busca si la respuesta propuesta aparece al final de la del usuario. Si hay un asterisco al principio y uno al final, por ejemplo *respuesta*, sólo se busca que la propuesta aparezca dentro de la del usuario.

- Si se quiere que el programa ignore las diferencias entre mayúsculas y minúsculas, hay que escribir la propuesta entre comillas simples, por ejemplo: 'respuesta'.
- Si se quiere que el programa ignore los acentos y la diferencia entre n y ñ, entonces hay que escribir la respuesta entre un acento grave y uno agudo. Por ejemplo, `respuesta´. `'respuesta'´ ignoraría tanto mayúsculas y minúsculas como acentos.
- La interrogación funciona como comodín de letras que quieren ignorarse. En lugar de la letra a ignorar debe escribirse una interrogación ?
- Si el control es un campo de texto y el usuario lo deja vacío, se considera que no contestó.
- El sistema de evaluación definido por el administrador de las evaluaciones es quien decide cómo se interpretan las respuestas correctas, las incorrectas y las vacías.

- **teclado**: Es una casilla de verificación que implementa el uso de un teclado virtual. Se encuentra presente sólo para los controles que involucran un campo de texto, tales como pulsador, campo de texto, menú y barra.

  Si la casilla está marcada, se activan los parámetros *distribución del teclado* y *posición del teclado*, descritos en breve.

  La virtud de los teclados virtuales es más evidente en dispositivos móviles. De forma natural, al pulsar un campo de texto en un móvil, se despliega el teclado nativo del mismo. Éste suele ser de gran tamaño, y opaca completamente la escena detrás. Para evitar esto, el uso del teclado virtual de *DescartesJS* da la opción al usuario de elegir el tamaño, complejidad y posición del teclado.
- **distribución del teclado**: Es un menú que permite elegir entre las siguientes opciones: 14×1, 7×2, 10×2, 4×4, 5×4, $10\times4\_alfa$, $10\times4\_num$, 11×3 y 11×4. El número antes del símbolo × indica la cantidad de teclas en una fila, mientras que aquél después indica el número de teclas en una columna. El sufijo *_alfa* implica un teclado alfabético, mientras que el prefijo *_num* implica uno numérico. En la Figura 15.8, dentro del apartado sobre la herramienta de teclado virtual, se puede consultar la disposición de los diversos teclados en el orden mencionado anteriormente.
- **posición del teclado**: Es un campo de texto donde va un par de coordenadas (por ejemplo `(100,50)`), con las que se indica la posición absoluta, dentro de la escena completa (no dentro del espacio), de la esquina superior izquierda del teclado a mostrar.

  Para mayor información sobre el teclado, consulte el apartado sobre el teclado virtual.

Cabe mencionar que las opciones del menú *acción* se ejecutan cuando el control es usado de alguna manera. Esto no sólo implica hacer uso directo del control. Por ejemplo,

un pulsador puede modificarse con las flechas para subir o bajar su valor. No obstante, también se puede introducir texto en el campo de texto que por defecto acompaña a los pulsadores y oprimir la tecla INTRO. Esto también lanzará la acción asignada a ese control.

Abordemos un ejercicio para practicar un poco lo aprendido en esta parte. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en ComunesControles. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio se observaron algunas de las funciones más típicas de los parámetros comunes a la mayoría de los controles. En este caso, el control *grosBr* se mantuvo fuera del interactivo puesto que el grosor es algo que quedará fijo en el interactivo. Es decir, el programador querrá intentar varios grosores y ver cuál queda mejor al final del interactivo, pero el control como tal no deberá quedar disponible para el usuario final.

10

El selector *Programa*

El selector *Programa* incluye ya parte de la programación dura que se hace con Descartes. Consta básicamente de dos tipos de algoritmos presentes por defecto en cualquier escena (*INICIO* y *CÁLCULOS*), y de un elemento conocido como evento.

El selector *Programa* contiene un panel a la izquierda igual al de los otros selectores. Es decir, hay un botón + con el cual se puede agregar un elemento nuevo. El único elemento nuevo que puede agregarse es, de hecho, el elemento tipo evento. Los algoritmos *INICIO* y *CÁLCULOS* se encuentran presentes por defecto. Los botones para duplicar, eliminar y mover el orden de los elementos del panel también están presentes. El panel muestra la lista de elementos del selector *Programa*. El botón *Programa* muestra la lista de elementos en forma de texto (para una edición más ágil) al igual que sucede en otros selectores. A continuación se describen los componentes del selector *Programa* con más detalle:

10.1. INICIO

El algoritmo *INICIO* permite hacer los cálculos iniciales que dejan listo al interactivo para ser usado. Estos cálculos pueden incluir asignaciones, condicionales y llamados a funciones. En la Figura 10.1 se muestran los componentes del algoritmo *Inicio*.

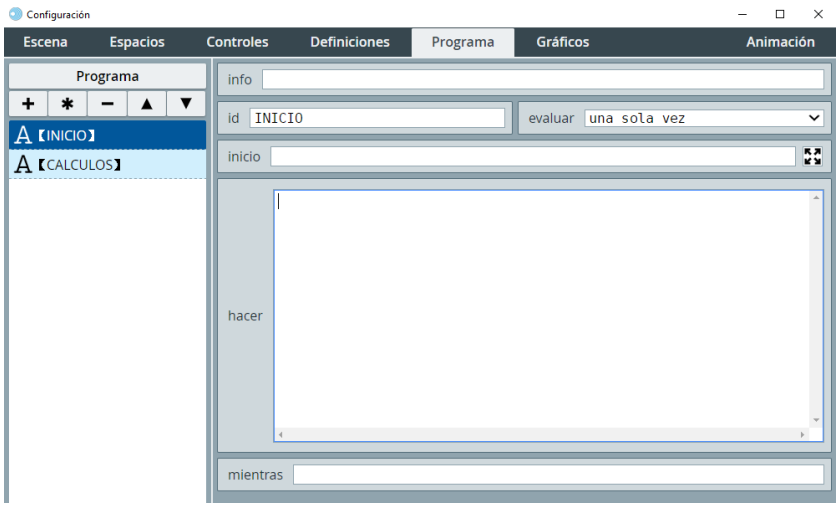


Figura 10.1: Visualización del algoritmo *Inicio*.

- **id**: es un campo de texto para el nombre del algoritmo. Por defecto trae el nombre *INICIO*, y es recomendable no cambiarlo.
- **evaluar**: es un menú mediante el cual se indica cuándo han de evaluarse las instrucciones del algoritmo:

  una sola vez: Sólo se evalúa el algoritmo en cuestión una sola vez al inicio de la escena. Como en este caso se trata del algoritmo *INICIO*, este menú se encuentra por defecto en *una-sola-vez*. Si un algoritmo sólo se ha de realizar una vez, es conveniente dejar esta opción marcada ya que reduce el número de cálculos y evita que el interactivo se ralentice.

  siempre: El algoritmo se evalúa siempre que el usuario haga un cambio en el interactivo, tal como modificar un control. Esta opción se usa para algoritmos que es necesario efectuar constantemente. Es preciso tener cuidado de hacer estos algoritmos cuan pequeños sea posible ya que, al repetirse constantemente, pueden ralentizar el interactivo.
- **inicio**: es un campo de texto donde se pueden incluir también asignaciones, llamados a funciones, etc. que se hacen al inicio del algoritmo. Es posible que un algoritmo requiera repetirse un cierto número de veces y que en dichas repeticiones algunas variable tomen distintos valores. En este campo de texto se incluyen las asignaciones, etc. que se hacen antes de realizar las repeticiones. De ahí que su nombre sea *inicio*. Cuando se hacen diversas asignaciones en este campo de texto, se debe usar el punto y coma (;) para separar cada una.
- **hacer**: es un panel en que se puede introducir las diversas instrucciones en varias líneas. Si el algoritmo involucra hacer instrucciones de forma cíclica, las instrucciones que se ciclarán deben ir precisamente en este panel. Si el algoritmo no involucra instrucciones de forma cíclica, da lo mismo si las instrucciones se introducen en este panel o si se introducen en el campo de texto *inicio*.
- **mientras**: es un campo de texto en el cual se introduce una condicional que, de ser cierta, obligará que se repitan las instrucciones en el panel *hacer* hasta que la condicional deje de ser cierta. Si no se incluye texto en este campo, Descartes considera que las instrucciones en *hacer* sólo se harán una vez.

  Es importante tener en mente que existe una limitante respecto al número de ciclos que se pueden realizar. El número máximo de ciclos es 100,000. Aunque en el campo *mientras* se coloque una condición que pudiera implicar un número mayor a 100,000 iteraciones, éstas se detendrán cuando se alcance dicho número. Esta restricción es intencional y su propósito es evitar que el usuario potencial de *DescartesJS*, que en muchas ocasiones consiste de personas con experiencia limitada de programación, incurra en errores que involucren, por ejemplo, la ejecución de una infinidad de ciclos.

IMPORTANTE: De forma general, en cualquier parte o ventana de *Descartes* en que se incluya código, siempre es posible agregar comentarios (líneas que no se leerán al inter-

pretar el código) comenzándolas con doble diagonal sencilla (//). Esto aplica no solo para partes de introducción de código en el selector *Programa*, sino también para el selector Controles (en los parámetros de cálculo de un botón, por ejemplo), Animación y Definiciones.

10.2. CÁLCULOS

El algoritmo Cálculos puede incluir instrucciones, asignaciones, condicionales y llamados a funciones. Dicho algoritmo se llama cada vez que el usuario hace un cambio en el interactivo. Su disposición en el Editor de configuraciones es idéntica al del algoritmo INICIO, con la salvedad de que el menú *evaluar* se encuentra por defecto en *siempre*. Aunque es posible cambiar dicho menú, se recomienda que se mantenga en *una sola vez* para el algoritmo *INICIO* y en *siempre* para el algoritmo *CÁLCULOS*.

Abordemos un ejercicio para practicar los algoritmos INICIO y CÁLCULOS. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Algoritmos Varios. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*. El algoritmo *INICIO* en el ejemplo provisto para este ejercicio incluye algunas asignaciones adicionales que no se abordan en las instrucciones. Ellas sirven el propósito de vincular la escena con el contenedor que la muestra y pueden ser ignoradas.

En este ejercicio se puede notar que el algoritmo *INICIO* se usa para asignaciones o instrucciones que no se deben actualizar todo el tiempo. Si se desea que se actualicen todo el tiempo, conviene usar el algoritmo *CÁLCULOS*. Pero hay que evitar saturar dicho algoritmo con cosas que no necesariamente deben calcularse de forma constante puesto que puede ralentizar el interactivo.

Note además que las asignaciones en este ejercicio se hacen en el panel *hacer* de los algoritmos *INICIO* y *CÁLCULOS*. Bien se pueden colocar en el panel *inicio*, dado que no se están usando los algoritmos como ciclos (el campo *mientras* está vacío, que implica que lo que se incluya en *hacer* sólo se hará una vez y no se repetirá cíclicamente).

10.3. Eventos

Un evento es una acción, o conjunto de acciones que se realizan cuando una cierta condición se cumple. Es posible determinar la frecuencia con que se implementarán las acciones, como se describe a continuación. Las acciones disponibles son las mismas que se ejecutan mediante el uso de controles. En la Figura 10.2 se muestran los componentes del algoritmo *evento*.

- **id**: es un campo de texto en el cual se introduce el identificador del evento. Este identificador suele servir sólo para que el programador localice el evento en cuestión. No suele hacerse referencia al mismo en otras partes del programa.

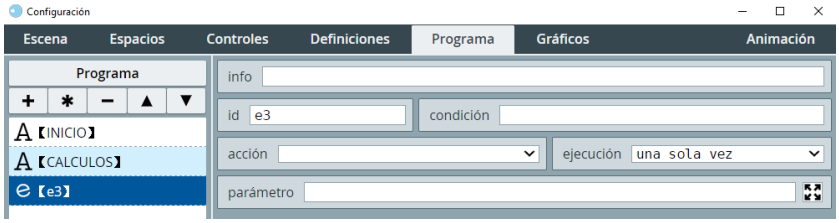


Figura 10.2: Visualización del algoritmo *Evento*.

- **condición**: es un campo de texto en el cual se introduce la condición que ha de cumplirse para que el evento sea ejecutado. Si la condición es verdadera, el evento se lanzará.
- **acción**: es un menú en el que se elige la acción que lanzará el evento en caso de cumplirse su condición. Este menú es el mismo que el menú *acción* en los controles y sus opciones se detallan en dicha sección, de tal forma que cualquiera de las acciones posibles por los controles también será posible mediante un evento.
- **ejecución**: es un menú mediante el cual se determina, dependiendo de la condición del evento, exactamente cuándo ha de ejecutarse la acción. Sus opciones son:

  una sola vez: hace que la acción se ejecute una única vez, que es la primera vez que la condición pasa de ser falsa a verdadera.

  alternar: hace que la acción se ejecute cada vez que la condición pasa de ser falsa a verdadera.

  siempre: hace que la acción se ejecute siempre que la condición sea verdadera, y no sólo cuando pasa de ser falsa a verdadera, como en el caso de *alternar*.
- **parámetro**: es un campo de texto en el que se incluyen las instrucciones que se han de ejecutar en caso que la acción del evento sea *calcular*; o la dirección URL en caso que la acción sea *abrir URL*; o la dirección de la escena si la acción es *abrir Escena*; o la ruta al archivo a reproducir si la acción es *reproducir*. Funciona igual que para las acciones de los controles numéricos. Incluye un botón que lanza una ventana de edición de texto para una más cómoda introducción del parámetro.

Hagamos un ejercicio para practicar los eventos y, en particular, las diferencias en sus modos de ejecución. Se busca un interactivo que haga que una imagen se pegue al mouse siempre que éste esté oprimido. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Algoritmos Evento. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio se puede ver que una condición por sí sola no siempre especifica qué debe hacer un evento. El parámetro *ejecución* del mismo determina de forma más precisa su comportamiento.

En este interactivo se usaron variables intrínsecas de Descartes relacionadas a acciones del mouse. Éstas se pueden consultar en el apartado sobre variables del mouse. Igualmente, se puede consultar el apartado sobre condicionales y operadores booleanos dado que también se usaron condicionales en el evento mismo.

Los eventos resultan particularmente útiles cuando se desea que un botón controle una animación además de ejecutar otras instrucciones que no se quiera incluir dentro de la animación misma. En ese caso, la variable que controla la animación se puede asociar a un evento cuya acción es lanzar o detener la animación.

11

El selector *Definiciones*

Las definiciones también incluyen gran parte de la programación dura que se hace en Descartes. Éstas consisten en *variables*, que pueden aceptar un valor o una expresión; *vectores*, que pueden verse como una variable que puede tener simultáneamente una hilera de valores dentro de ella; *matrices*, que es una extensión de un vector más allá de una hilera de valores; *funciones*, que suelen ser un conjunto de instrucciones que se puede elegir cuándo se lanzan y además tienen la capacidad de realizarse cíclicamente; y *bibliotecas*, que consisten en agrupaciones de varias de las otras definiciones y cuyo propósito es poder mantener un orden más claro del código de este selector.

En adelante veremos con más detalle en qué consiste y cómo funciona cada una de las definiciones. También podremos ver cómo éstas permiten manejar los datos de una forma más cómoda, simplificada y ágil y, a su vez, permiten reducir el código de un interactivo en gran medida.

Por cierto, anteriormente el selector *Definiciones* no contaba con la funcionalidad de filtro. Ahora, debajo del botón *Definiciones* (aquél que enlista las definiciones disponibles) en el panel izquierdo del selector, se muestra un filtro en el que se puede elegir mostrar todas las definiciones, aquellas definiciones de la escena (las que no fueron leídas de bibliotecas) y las definiciones por biblioteca. Este filtro se muestra enmarcado en rojo en la figura 11.1.

Figura 11.1: Ejemplo del tipo de definición *Vector*.

11.1. Vector

En ocasiones se desea utilizar una gran cantidad de variables que guardarán un tipo similar de información. En estos casos, puede resultar engorroso crear muchas variables

distintas, y conviene en su lugar definir una especie de “mueble con muchos cajones”. Aunque el nombre del mueble es el mismo, cada cajón tiene un número o índice que lo identifica.

Cuando se agrega un vector, sólo es necesario dar su nombre. Por ejemplo, el vector *v1*. Una vez agregado, siempre se hará referencia al mismo indicando el número de cajón en cuestión. Por ejemplo, *v1[0]* representa el primer cajón, *v1[1]* representa el segundo cajón, y así sucesivamente. Como se puede notar, el índice de los cajones empieza siempre en cero. Cada cajón puede guardar un valor o cadena, como si fuera una variable común y corriente.

En la Figura 11.2 se observan los componentes de una definición tipo *vector*.

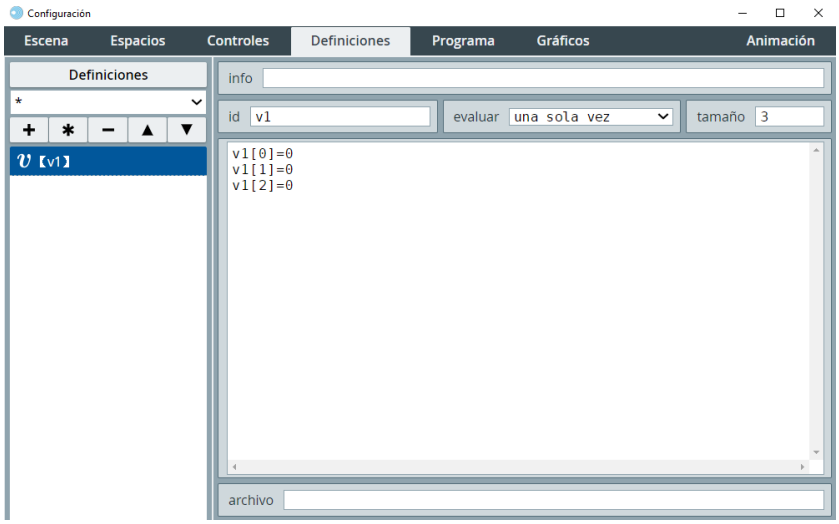


Figura 11.2: Ejemplo del tipo de definición *Vector*.

- **id**: es un campo de texto en el que se introduce el identificador del vector. Éste será el nombre con el que se refiere a dicho vector, y cuando se quiera introducir alguna entrada (que sería el cajón) en el programa, ésta se pone flanqueada por corchetes cuadrados después del identificador. Por ejemplo, *Vc[2]* corresponde a la tercera entrada del vector *Vc*.
- **evaluar**: es un menú con dos posibles opciones. Son las mismas opciones presentadas para los algoritmos INICIO y CALCULOS: *una sola vez* y *siempre*. Si se selecciona *siempre*, las asignaciones en el panel de texto central del vector se hacen siempre que el usuario modifique un control. De lo contrario, sólo se hacen cuando se carga la escena. Dado que las actualizaciones a los valores de los vectores suelen hacerse mediante funciones (como se verá más adelante), conviene dejar esta opción en *una sola vez* la mayoría de las veces.
  De momento, sólo la opción *una sola vez* se encuentra disponible, aunque es posible en el futuro se habilite la función *siempre*. No obstante, para fines prácticos no

es necesario contar con esta última opción, como se verá en los diversos ejercicios que ocupan vectores.

- **tamaño**: es un campo de texto en el que se introduce el valor del número de entradas (o cajones) del vector.
- **panel de asignaciones del vector**: es un panel para introducir el texto de las asignaciones del vector. Por defecto aparecen las 3 primeras entradas del vector inicializadas en cero. Pero también puede incluir asignaciones a otras variables que no sean el vector. Si el vector ha de modificarse constantemente mediante una función y no requiere de valores iniciales particulares, entonces conviene dejar dicho panel vacío.
- **archivo**: es un campo de texto en el que se introduce la ruta relativa a un archivo del cual se obtendrá la información para llenar las entradas del vector. Ello resulta útil cuando el número de entradas de un vector es tan grande que resulta prohibitivo o indeseable ingresarlos manualmente en el editor.
  El archivo en el que se introducen los datos deberá ser un archivo de datos preferiblemente con codificación UTF-8. Las entradas del vector se ingresan separadas por saltos de línea, o *ENTER*s.
  Es preciso notar que dicha funcionalidad es inmediatamente disponible tanto en el editor como en navegadores como Firefox. En Chrome, dicha funcionalidad puede no ser inicialmente visible. El archivo bien puede no ser leído dependiendo de la seguridad web implementada en el navegador. Sin embargo, puede desactivarse en Windows usando la bandera `-disable-web-security` al lanzarlo desde la ventana de comandos.

  Para que un *html* generado en *DescartesJS* efectivamente lea un vector desde archivo es preciso que ya esté dicho *html* guardado en la ubicación tal que la ruta al archivo *txt* sea válida. Así, *DescartesJS* no encontrará (y no podrá leer) el archivo si no se ha guardado el *html* en la ubicación correcta.

  Por otra parte, note que es posible guardar los datos del archivo de texto como un *script* dentro del *html* de la escena de *Descartes* una vez que ésta se guarda, como se ve en la Figura 11.3. Como se observa en la figura, el vector lee el archivo *arch.txt* y los datos se incluyen como parte del *html* dentro de la etiqueta *script*. De hecho, es posible seguir manejando el archivo *html* con el *script* del vector aún cuando el archivo *txt* ya no se encuentre en la ubicación (los datos del vector se leerán entonces directo del *script*), aunque se recomienda conservar el archivo en su ubicación. La decisión de si guardar los datos del vector en el *html* o no se determina en el menú *Opciones*, como se indica en el apartado sobre agregar al HTML.

Ahora que ya conocemos las partes del vector, hagamos un ejercicio que tira un par de dados y el número de veces que sale un número (entre el 2 y el 12) se guarda en las entradas de un vector. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Definiciones Vector. El documento del interactivo como tal se

```
      <param name="A_01" value="id='v1' vector='si' evaluar='una-sola-vez' tamaño='10' archivo='./arch.txt' tipo='vector'
      <param name="A_02" value="id='INICIO' algoritmo='si' evaluar='una-sola-vez' ">
      <param name="A_03" value="id='CALCULOS' algoritmo='si' evaluar='siempre' ">
    </ajs>
  </div>

<script type="descartes/vectorFile" id="./arch.txt">
1.5
2
3
3.7
2.89
6.21
</script>

</body>
</html>
```

Figura 11.3: *Script* en el archivo *html* correspondiente a un vector leído desde archivo.

encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Este ejercicio puede usarse para mostrar cómo en un tiro de un par de dados es más probable tirar, por ejemplo, 7 que 2. Ello se debe a que un tiro de 2 sólo se logra si cada dado tiene un valor de 1 (una posible configuración), mientras que el tiro de 7 se logra con 1 y 6, 2 y 5, 3 y 4, 4 y 3, 5 y 2, y 6 y 1 (6 distintas configuraciones). Una posible extensión a este ejercicio que se deja al usuario es representar las frecuencias de los tiros usando una gráfica de barras. Ello se puede lograr usando una familia de polígonos o rectángulos.

En este ejercicio podemos notar varias cosas importantes. Por un lado, que un vector reduce el trabajo de programación pues evita la necesidad de agregar, en este caso, 11 distintas variables. Adicionalmente, si se hubieran usado 11 variables en lugar del vector, sería necesario un código muy largo para indicar a qué variable se le debe sumar la unidad cada vez que sale un determinado tiro.

El uso de las funciones es generalmente indispensable cuando se manejan vectores. La documentación contiene un apartado sobre funciones poco adelante. También se utiliza en este ejercicio el uso de una variable de Descartes: *rnd*. Para mayor información al respecto, puede consultar el apartado sobre variables generales de Descartes.

Otra cosa interesante es el orden en que debe estar el vector en el panel *Definiciones*. Se debe encontrar de preferencia sobre la función que lo llama. De otra forma, la función buscará un vector aún no definido y un error ocurrirá. Es por ello que siempre es conveniente colocar tanto vectores como matrices hasta arriba de la lista del panel izquierdo de *Definiciones*.

Por último, notamos la estrecha relación que se forma entre el uso de vectores (y matrices también) con las funciones. Las funciones son muy útiles para asignar valores a los vectores ya que ellas pueden manejar una variable que funciona como índice, y este índice puede usarse directamente para modificar el valor de una entrada de un vector.

11.2. Matriz

Se observó previamente lo poderoso que puede ser un vector para manejar grandes cantidades de información. No obstante, a veces es necesario manejar información no sólo en una hilera de datos, sino como celdas en una tabla. Es entonces cuando ocupamos las matrices.

Las matrices (a veces también llamadas *arreglos*) consisten en una especie de tabla con diversas celdas. Puede ejemplificarse con una cajonera que tiene hileras de cajones y columnas, y cada cajón guarda un determinado valor. En este caso, a diferencia de los vectores, se debe especificar la coordenada de la entrada de la matriz con un par de índices separados por una coma. Por ejemplo para una matriz *M*, *M[3,7]* correspondería a la entrada de la cuarta columna (recuerde que los índices se empiezan a contar desde el valor cero) y octava fila. Es decir, la notación es de la forma *M*[*columna*,*fila*]. En la Figura 11.4 se observan los componentes de una definición tipo *matriz*.

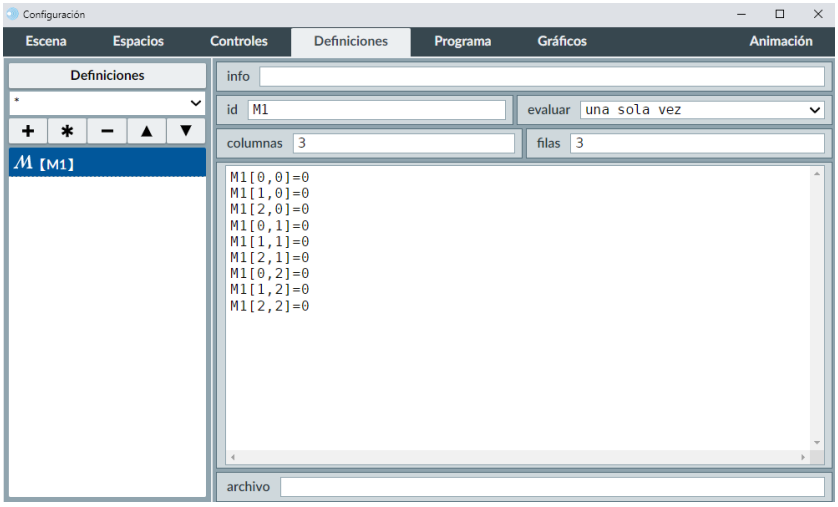


Figura 11.4: Ejemplo del tipo de definición *Matriz*.

- **id**: es un campo de texto en el que se introduce el identificador de la matriz. Cuando se hace referencia a alguna entrada de la matriz, se usa dicho identificador seguido del par de coordenadas de la entrada de la misma flanqueado por corchetes cuadrados. Por ejemplo, para una matriz *M*, una entrada de la misma sería *M[1,2]*.
- **evaluar**: es un menú con el mismo par de opciones que para los algoritmos *INICIO* y *CALCULOS*, y de hecho también se encuentra para los vectores. La opción *siempre* se usa cuando se desea que las instrucciones del panel de la matriz se realicen cada vez que el usuario interactúa con algún control del interactivo. La opción *una sola vez* se usa para sólo hacer dichas instrucciones al cargarse el interactivo. Nuevamente, ésta opción es la más utilizada ya que la modificación del contenido de la matriz suele hacerse mediante funciones. Por el momento, sólo se encuentra disponible la opción *una sola vez*.

- **columnas**: es un campo de texto donde se introduce el número de columnas que tendrá la tabla. Recuerde que la tabla no tiene un *tamaño* como el vector, pues no es de una sola dimensión, sino que es de dos dimensiones. Es por ello que se requiere definir tanto las *columnas* como las *filas*.
- **filas**: es un campo de texto donde se introduce el número de filas que tendrá la tabla.
- **panel de asignaciones de la matriz**: es un panel de texto en el que se introducen las asignaciones o instrucciones para inicializar la matriz. Por defecto vienen algunas de las entradas (o cajones) de la matriz inicializadas en cero. No obstante, recuerde que, al igual que los vectores, las matrices pueden inicializarse mediante una función.
- **archivo**: es un campo de texto en el que, similar al caso de un vector, se puede introducir la ruta (relativa al archivo de la escena), de un archivo de texto que contiene los datos para llenar la matriz.

  Los datos en dicho archivo deben ser separados por comas. Son, así pues, archivos CSV (comma-separated values), como los típicamente exportados por programas como Excel.

  Los valores en el archivo de texto son dependientes del signo decimal elegido en la escena (mediante el selector Escena). Si el signo decimal elegido es el punto decimal, el separador entre valores en el archivo será la coma (,). Por ejemplo:

```
1,2,3,4,5
'a',b,c,d,'e'
10.1,20.2,30.3,40.4,50.5
```

  Pero si el signo decimal es la coma, entonces el separador entre valores en el archivo será el punto y coma (;). Por ejemplo:

```
1;2;3;4;5
'a';b;c;d;'e'
10,1;20,2;30,3;40,4;50,5
```

  Las hojas de cálculo en ocasiones exportan las cadenas de texto con comillas dobles. Sin embargo, es posible tener cadenas de texto en el archivo de texto flanqueadas por comillas sencillas o dobles, ya que internamente el programa convierte las dobles en sencillas.

Hagamos ahora un ejercicio que muestra un ejemplo de cuándo son útiles las matrices. En este ejercicio se desea mostrar un gran número de partículas que podrían ser de aire en un espacio. Algunas serán rojas y otras azules. El color y la posición de las partículas están dadas al azar. Cada partícula tiene dos coordenadas (horizontal y vertical) en el espacio. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Definiciones Matriz. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

El presente ejercicio muestra cómo las matrices facilitan la manipulación de una gran cantidad de información. De haber usado sólo vectores, hubieran sido necesarios como mínimo tres vectores diferentes (uno que guardara la coordenada horizontal, otro que guardara la vertical, y un tercero que guardara el color). ¡Y en algunos otros ejemplos es posible que ser requirieran aún más! Así pues, las matrices pueden verse como un *vector de vectores*, que permite una manipulación más ágil y compacta de la información.

En este ejercicio notamos que el tipo de dato que guarda una matriz puede ser distinto. En algunas entradas (o cajones) se guardaron números (las coordenadas), mientras que en otras se guardó una cadena de texto (*azul* o *rojo*).

Observe también que fue nuevamente una función la que se encargó de asignar las posiciones a las partículas, ignorando la asignación de valores inicial en la matriz misma.

Se utilizaron condicionales para la asignación del color de las partículas. Se recuerda que dicho tema se puede estudiar en el apartado sobre condicionales y operadores booleanos. También se hicieron cambios en los colores de los gráficos. El uso de la herramienta asociada se puede consultar en el apartado sobre la herramienta de control de colores.

11.3. Función

Las funciones involucran una o un conjunto de instrucciones que se pueden activar sólo en ciertas ocasiones. Tienen la virtud de que se pueden agrupar ciertas instrucciones que se repiten muchas veces a lo largo de un programa en un solo bloque de código. Es probablemente el núcleo sobre el cual gira la programación en general, por lo que se le dará un énfasis particular en esta documentación. En la Figura 11.5 se muestran los elementos de este tipo de definición.

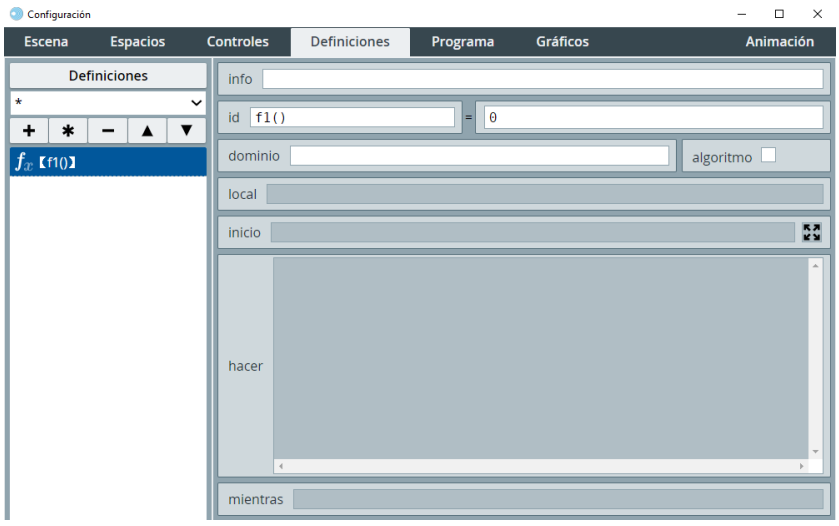


Figura 11.5: Ejemplo del tipo de definición *Función*.

- **identificador de la función**: es un campo de texto en el que se introduce el nombre por el cual se ha de llamar la función. Las funciones pueden o no llevar argumentos cuando son llamadas. Estos argumentos se incluyen entre paréntesis al final del nombre de la función, y si se trata de más de uno, éstos se separan por comas. En caso que la función no involucre argumentos, el paréntesis sigue siendo necesario, aún cuando se deje vacío.
- **valor o expresión de retorno de la función**: es un campo de texto que incluye un valor o expresión que la función ha de devolver cuando es llamada. Se encuentra a la derecha del símbolo = a la derecha del identificador de la función. Cuando se llama una función, muchas veces se asigna ésta a alguna variable. El valor de la expresión de retorno es precisamente el que se asignará a la variable asociada. Si no se desea que la función devuelva un valor a la variable, sino que simplemente se ejecute alguna acción, este campo puede dejarse en blanco.
- **dominio**: es un campo de texto donde se introduce una condición. La función sólo se evaluará en los puntos donde la condición es verdadera. Por ejemplo, si una función depende de x y en su dominio tiene la condición $(x > 1)\&(x < -1)$, dicha función excluirá el intervalo $[-1,1]$ al ser evaluada.
- **algoritmo**: es un checkbox que cuando está marcado permite que la función no se calcule sólo mediante una única instrucción, sino como un grupo de instrucciones. De estar marcado, habilita los campos *Local*, *inicio*, *hacer* y *mientras* de la función, cuya funcionalidad es igual a la de los algoritmos *INICIO* y *CALCULOS*.
- **local**: es un campo donde se introducen los nombres de las variables que se han de considerar locales. Una variable local aquí definida puede cambiar su valor sólo dentro de la función. Por ejemplo, si hay una variable i dentro de la función en cuestión, y otra variable i en alguna otra parte del programa, aunque se llamen igual, si está declarada como local, aunque se le cambie su valor en la función no se afectará el valor de la otra variable que está en la otra parte del programa. Así, colocar variables en local *protege* al código de dañar otras variables que pudieran llamarse igual. Cabe mencionar que cuando la función involucra argumentos, éstos son tratados como variables locales.
  Si una función con una variable declarada como local manda llamar una función subordinada dentro de ella, en esta función subordinada la variable conserva el valor que heredó de la función principal. Es inclusive posible cambiar el valor de la variable en la función subordinada, y se reflejará dicho cambio en la principal. Esta herencia de valor ocurre siempre y cuando la misma variable no esté declarada como local también en la función subordinada.
- **inicio**, **hacer** y **mientras**: son campos de texto que se habilitan si el checkbox *algoritmo* está marcado. Su funcionalidad es igual a la del algoritmo *INICIO*, por lo que se puede consultar en el apartado INICIO.

Hagamos primero un ejercicio para practicar las funciones de una sola instrucción. El objeto de este ejercicio es hacer un interactivo que indique la longitud de los tres la-

dos de un triángulo cuyos vértices son controles gráficos. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Definiciones Función 1. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio usamos una función que sólo involucra una instrucción y que, además, devuelve un valor. Para que devuelva el valor, es necesario que la expresión que devuelve el valor esté tras el signo = de la función. En este caso, la expresión devuelta es la raíz cuadrada de la suma del cuadrado de los catetos del triángulo rectángulo cuya hipotenusa está flanqueada por los puntos (*ax*,*ay*) y (*bx*,*by*). Observamos que dicho campo puede tener un valor, una variable, o inclusive (como es el caso presente) una expresión a devolver. La función devuelve un valor que se asigna a la variable en cuestión (en este caso *distg1g2* y las otras dos distancias).

En este caso sólo calculamos las distancias entre tres puntos, pero si tuviéramos veinte puntos, la misma función se puede usar para las veinte distancias. Podemos, así pues, notar que la función sirve como una instrucción que tiene el potencial de ser usada muchas veces, aunque en el código sólo se introduce una vez.

En este ejercicio se usaron variables del tipo *g1.x* y funciones del tipo *sqrt()*. Todas estas son variables y funciones intrínsecas de Descartes y se pueden estudiar en los apartados sobre variables de los controles gráficos y funciones comunes a diversos lenguajes de programación.

Ahora hagamos un ejercicio que involucre una función que no devuelve explícitamente un valor a una variable, pero que sí calcula varias cosas a la vez. Queremos que el interactivo de este ejercicio nos indique no sólo la distancia entre un par de puntos asociados a dos controles gráficos, sino que también nos diga la componente horizontal y vertical de la hipotenusa definida entre ambos. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Definiciones Función 2. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio pudimos ver que en ocasiones es necesario que una función no sólo calcule y devuelva un valor en particular. En este caso queríamos tres valores distintos. En lugar de hacer tres funciones, una para cada cálculo, se pudieron agrupar en una misma. La función bien pudo haber involucrado argumentos, pero no es necesario, ya que las variables *g1.x*,*g1.y*, *g2.x* y *g2.y* pueden ser usadas directamente en la función. Es preciso notar que esta función es útil para este caso en que sólo queremos conocer tres distancias de dos controles gráficos, pero que si hubiera de aplicarse a una gran cantidad de controles gráficos, mejor convendría usar una función general que involucre argumentos.

Es válido preguntarse por qué las instrucciones dentro de la función no se colocaron directamente en el algoritmo *CALCULOS*. Esto hubiera servido igual. Sin embargo, con el propósito de tener un orden un poco más claro en el código, se prefiere muchas veces hacer un código modular, donde es más fácil llamar a la función, que es el módulo que contiene las instrucciones específicas para realizar algo.

Para este ejercicio se usaron variables del tipo *g2.x* y funciones como *sqrt()* y *abs()*. Las variables de dicho tipo son intrínsecas de Descartes y se pueden estudiar en el apartado sobre las variables de los controles gráficos y las funciones, también intrínsecas, se pueden estudiar en el apartado sobre las funciones comunes a diversos lenguajes de programación.

Hagamos un último ejercicio con funciones que involucre el cálculo del máximo común divisor de dos enteros distintos de cero usando el algoritmo de Euclides. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Definiciones Función 3. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio pudimos ver cómo funciona una función iterativa o recursiva. Es decir, una función que tiene un algoritmo que ha de repetirse hasta que se alcanza una condición. El conocer el máximo común divisor se puede usar, por ejemplo, para reducir una fracción a su expresión más simple dividiendo el numerador y el denominador de la misma por el máximo común divisor encontrado.

Para este ejercicio se utilizó una condición booleana en el campo *mientras* de la función. Los operadores booleanos se pueden consultar en el apartado sobre condicionales y operadores booleanos. Igualmente, las funciones tales como *ent()* se pueden consultar en el apartado sobre las funciones comunes a diversos lenguajes de programación.

11.4. Variable

La definición *variable* permite usar una variable a la que se le asigna un valor o expresión desde el inicio en que se carga el interactivo. En sí, cuando recibe una expresión se comporta de forma muy similar a la definición tipo función (que se puede consultar en el apartado Función), por lo que ya no se utiliza mucho.

En la Figura 11.6 se muestra un ejemplo del uso de una definición tipo variable en una escena. En la Figura 11.7 se muestra la configuración del editor de configuraciones para lograr dicho ejemplo.

Note que la asignación a la variable no necesita ser un valor como tal, sino una expresión que en este caso da la distancia de un punto al origen. En la Figura 11.6 se muestra un texto en el que se imprime el valor de dicha variable. Claro que previamente debieron definirse los valores de x e y (por ejemplo, en el algoritmo *INICIO* para que el cálculo genere el resultado correcto.

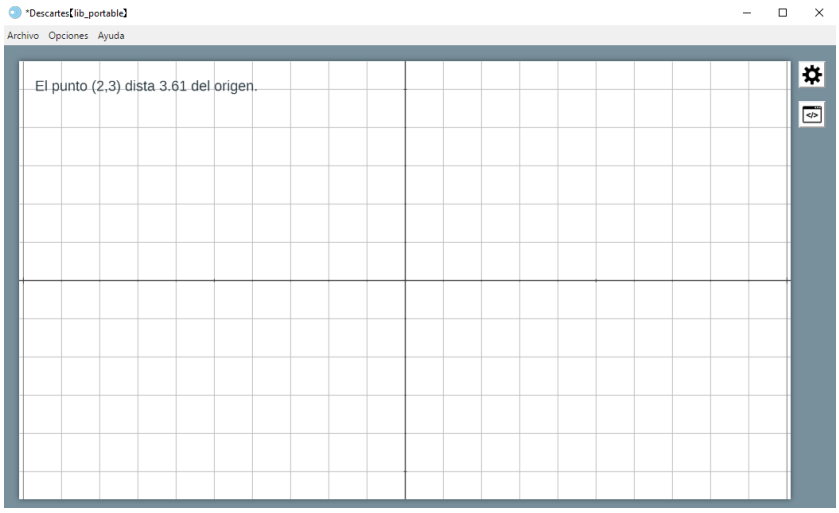


Figura 11.6: Ejemplo del la definición tipo *Variable*.

Figura 11.7: Configuración del ejemplo de la definición tipo *Variable*.

- **identificador**: es un campo de texto en el que se introduce el nombre del identificador asociado a la variable en cuestión. Este identificador es el que se usa cada vez que se necesite usar el valor de la misma.
- **valor de la variable**: es un campo de texto que se encuentra a la derecha del signo igual de la variable y en el que se introduce el valor o la expresión para la misma. Así, puede llevar un número explícitamente o hacer referencia, mediante su identificador, a otra variable.

Hagamos un muy breve ejercicio para practicar este tipo de definición. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Definiciones Variable. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Notamos en este ejercicio que las variables pueden recibir una expresión, en cuyo caso se realizarán los cálculos de la expresión y el valor se asignará a la variable. De hecho, este ejercicio repite lo mismo que otro en el que el cálculo de la distancia se hacía en los algoritmos del selector *Programa*. Sólo que aquí es más directo hacer el cálculo aprovechando la variable. No obstante, dado que las funciones permiten también las asignaciones de expresiones y muchas funcionalidad adicional, las variables ya casi no se usan.

11.5. Biblioteca

En ocasiones, un programa en Descartes puede llegar a contener una gran cantidad de elementos en el selector *Definiciones*. Esta gran cantidad de elementos puede hacer incómoda la búsqueda de elementos en particular. Y aunque todos ellos sean fundamentales para la escena de Descartes, es posible que muchos de ellos no requieran cambios subsecuentes, sino que sólo estorban la visibilidad de aquellos que sí requieren edición.

Así pues, resulta útil poder extraer de *Definiciones* a todos los elementos que habrán de quedarse fijos, de tal forma de que no permanezcan visibles en dicho selector. Los elementos extraídos se pueden guardar en un bloque de texto llamado *biblioteca*.

En la Figura 11.8 se muestra la configuración de una biblioteca a la que se le asigno un nombre de *info*: *motor*, que es el que aparece como identificador de la biblioteca en el panel izquierdo que enlista las diferentes definiciones presentes.

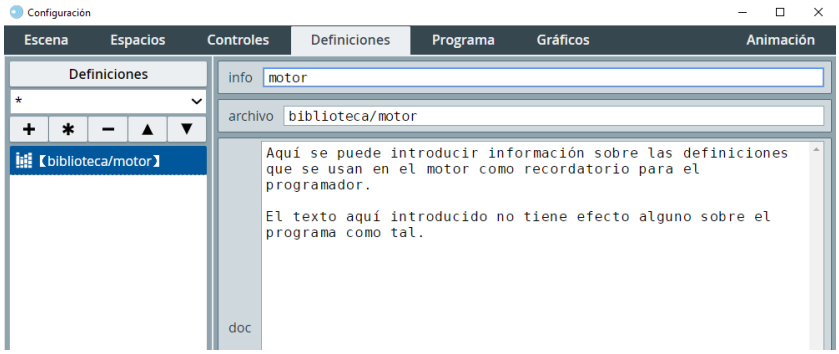


Figura 11.8: Ejemplo del la definición tipo *Biblioteca*.

- **archivo**: es un campo de texto donde se introduce la ruta al archivo que contendrá las definiciones que se desea no estén explícitamente presentes en el archivo de Descartes. Al igual que siempre, la ruta será relativa a la carpeta en que se guarda el archivo html que corresponde a la escena de Descartes.
- **doc**: es un panel de introducción de texto. El contenido de dicho panel no está sujeto a interpretación de Descartes. Es decir, aunque se hicieran asignaciones condicionales, o se introdujera cualquier texto a ser evaluado, dichas acciones no serán implementadas pues el texto aquí introducido se considera como de tipo *solo texto*. Su propósito es incluir recordatorios para el programador referentes al material que queda escondido en la biblioteca como tal.

Como se verá en el ejemplo a continuación, la biblioteca agrupa un conjunto de elementos en *Definiciones* en un archivo de texto ajeno a la escena de Descartes. Una vez que se tiene guardada una biblioteca, puede cargarse y guardarse también en el archivo html de la escena misma. Entonces es también posible editar los elementos de la biblioteca directamente desde el editor de configuraciones de Descartes.

Hagamos un pequeño ejercicio en el que se verá cómo funciona la biblioteca. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Definiciones Biblioteca. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Este ejercicio nos permite ver varios aspectos interesantes. Por un lado, notamos que si se han de usar funciones que ya no van a cambiar (como *CalcProm()* y *CalcDsvSt()*, éstas pueden guardarse en una biblioteca de tal forma que no estorben en el interactivo. Aquí sólo fueron dos funciones dado que es un ejemplo, pero imagine que el interactivo cuente con 30 funciones. El ubicar una función entre un total de arriba de 30 funciones sería difícil, aún cuando cada función tenga su parámetro *info* con datos descriptivos de la naturaleza de cada una.

El motor siempre puede eliminarse si se desea extrayendo su contenido y agregándolo a la ventana emergente que muestra el código fuente de las *Definiciones*, con lo que las funciones serán visibles de nuevo en el editor. Cuando se haga esto, se recomienda quitar manualmente el bloque *script* observado cuando se abre el html como archivo de texto, así como quitar la definición tipo *biblioteca* en *Definiciones*.

Es preciso poner especial atención a la codificación del archivo usado como biblioteca. Muchos editores de texto guardan sus archivos por defecto en codificación ANSI. El usar un archivo codificado en ANSI como biblioteca puede provocar que, al ser embebido en el html, sea embebido con errores (caracteres no bien definidos), y que por lo mismo no funcione correctamente. Si el archivo es codificado en *UTF-8* (de preferencia *UTF-8 sin BOM*), será embebido correctamente en el html y el interactivo funcionará bien.

Aunque no compete directamente el objetivo del interactivo de este ejercicio, conviene notar otro aspecto. La variable *i* usada como contador se usa en 3 distintas funciones (*AsignaValores()*, *CalcProm()* y *CalcDsvSt*). Esto no ocasiona problemas pues siempre que una función se ejecuta, no se está ejecutando alguna de las otras. Si dentro de una función se manda llamar a otra, y ambas tienen el mismo contador, las cosas no funcionarán correctamente. Lo mismo va para la variable *suma* que se encuentra presente en múltiples funciones. Cuando se quiere que una variable se respete dentro de una función a pesar de llamarse igual que otra variable fuera de ella, se puede agregar en el parámetro *local* de la función. En este vínculo puede obtener más información al respecto.

Otro detalle de importancia que se puede observar en este ejercicio es que es preciso que la función *CalcDsvSt()* se ejecute después de *CalcProm()*, ya que echa mano de la variable *promedio* calculada en la última. El alterar el orden en que se ejecutan resultará en un cálculo incorrecto de la desviación estándar.

Por último, note que se puede acceder a la biblioteca *motor* en este ejemplo mediante el menú en el panel izquierdo del selector *Definiciones* una vez que se ha guardado la escena ya con la biblioteca incluida. Una vez seleccionada, las funciones que dicha biblioteca contienen aparecen en el listado en el panel izquierdo de *Definiciones*, y se pueden revisar

y editar. Cualquier cambio en una de ellas tras aplicar y guardar la escena quedará guardado como parte del *script* de la escena misma (el archivo *html*), así como en el archivo de biblioteca (en este caso, *motor*). Esto permite un manejo más ágil de las bibliotecas que sustituye a la edición directa de los archivos externos de las mismas.

Observe que el filtro de definiciones puede usarse para mostrar las definiciones correspondientes a una cierta biblioteca. Es decir, si una biblioteca se importó como un elemento en el selector *Definiciones*, ella aparecerá como opción de filtrado y al seleccionarla se filtrarán las definiciones correspondientes a esa biblioteca de entre todas las definiciones presentes en la escena. Esto permite agrupar definiciones de un tipo similar en bibliotecas para tener un mejor orden del código. Recordamos que la opción *escena* en dicho filtro mostrará las definiciones naturales a la escena y que no se obtuvieron a partir de bibliotecas, mientras que la opción * muestra todas las definiciones.

Si en una escena hay dos definiciones del mismo nombre (por ejemplo, dos funciones que comparten un mismo nombre), una importada de una biblioteca y la otra como elemento directo de las definiciones, pero que tienen contenido distinto, la que tendrá precedencia es aquella que no fue importada en una biblioteca. Sin embargo, se recomienda evitar estos casos y siempre contar con una versión de una misma función para evitar confusiones, o si se trata de dos funciones que hacen cosas distintas, asignarles nombres diferentes.

De forma general, es importante notar que, para el caso de los vectores que se leen por archivo, así como para las bibliotecas, la información de ellos (que se leen desde archivos de texto) puede incluirse como parte del *html* de la escena de *Descartes* en forma de scripts. Pero para ello es preciso que se encuentren marcadas las opciones *biblioteca* y *vector* dentro de la opción Agregar al HTML del menú *Opciones* en la barra principal de *DescartesJS*. Todas las opciones vienen marcadas por defecto, razón por la cual en los ejercicios no fue necesario marcarlas para obtener el bloque *script* correspondiente en el archivo *html*.

12

El selector *Animación*

Las animaciones se usan para visualizar cambios del interactivo en el tiempo, en lugar de pasar directamente de un estado del interactivo a otro. Permiten al usuario *ver cómo cambia* el interactivo conforme el tiempo transcurre. Por lo mismo, es necesario especificar al programa qué tan rápido debe moverse el tiempo en el interactivo, además de qué es lo que cambiará en cada paso de tiempo.

Adicionalmente, las animaciones pueden ser cíclicas (cuando llega al final de la animación, ésta vuelve a empezar), pueden iniciar desde que se carga la escena o hasta que el usuario modifique algún control.

Las animaciones funcionan de la misma manera que los algoritmos *INICIO* y *CALCULOS*, y las funciones. Es decir, cuentan con un campo para asignaciones iniciales, un campo para asignaciones recurrentes o cíclicas, y un campo en donde se da la condición para detener la animación.

En la Figura 12.1 se muestra un ejemplo de cómo configurar una animación en el selector en cuestión.

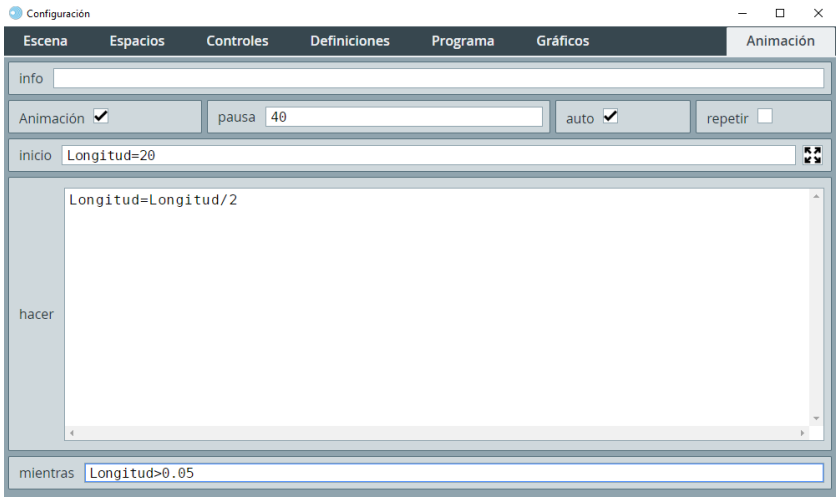


Figura 12.1: Visualización del ejemplo en el selector *Animación*.

- **Animación**: es un checkbox que habilita todos los parámetros de de la animación descritos a continuación. Si no se encuentra marcado, es imposible editar los demás parámetros. Así pues, si no se planea tener una animación, es conveniente tener este checkbox desmarcado.

- **pausa**: es un campo de texto en el que se introduce un valor o una expresión que determina el número de milisegundos de la pausa entre cada paso de la animación. Es preciso notar que si la animación involucra una gran cantidad de cálculos complejos que excedan el tiempo indicado en *pausa*, dicho tiempo no se respetará.
- **auto**: es un checkbox que si se deja marcado hace que el interactivo lance la animación desde el momento en que es cargado, sin previa instrucción del usuario. Es importante notar que esta funcionalidad en el editor de Descartes sólo funciona cuando se oprime el botón *Aceptar* en lugar de *Aplicar*. Si sólo se pulsa *Aplicar*, el interactivo se carga pero la animación no se lanza. Para ello es necesario presionar *Aceptar*. Todo esto con el objeto de que no se lancen animaciones mientras se está trabajando en el editor. En el navegador, las animaciones inician automáticamente desde el principio si este checkbox está marcado.
- **repetir**: es un checkbox que si está marcado hace que el ciclo *hacer* que define la animación se repita indefinidamente ignorando cualquier condición que se encuentre en el campo *mientras* de la misma. Básicamente, es lo mismo usar *repetir* que marcar 1 en el campo *mientras* de la animación, ya que hacer esto último garantiza que la animación se repetirá indefinidamente.
- **inicio**, **hacer** y **mientras**: son campos de texto cuya funcionalidad es igual a la del algoritmo *INICIO*, *CÁLCULOS* y las funciones, por lo que se puede consultar en el apartado INICIO.

Si hay una animación en curso y se presiona el botón de engrane para lanzar el editor de configuraciones, la animación se pausará, con el objeto de no gastar cálculos cuando la atención no esté sobre la escena misma.

Hagamos ahora un ejercicio que consiste en hacer un interactivo que el movimiento de las manecillas (segundero y minutero) de un reloj. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Animación 1. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Este ejercicio abordó un ejemplo muy sencillo de animación, donde sólo se modifica el valor de un contador en cada paso de la animación. Hay animaciones que pueden ser muy complicadas e involucrar incluso funciones complicadas dentro de la animación misma. Pero convino primero abordar este ejemplo para familiarizarnos con las animaciones. Es importante notar que aquí el 1000 en *pausa* corresponde casi exactamente a 1000 milisegundos (un segundo). Ello se debe a que la instrucción que se hace en la animación es una asignación muy sencilla que no le toma tiempo al procesador. Cuando se usan instrucciones muy complicadas, el tiempo entre un paso y el siguiente de la animación puede bien no corresponder al tiempo en *pausa*, sino ser más grande puesto que además de la pausa se incluye el tiempo de procesamiento.

En este ejercicio se usaron funciones como el seno y el coseno. Note que los ángulos se miden en radianes. Pueden consultarse las funciones en cuestión en el apartado sobre las funciones comunes a diversos lenguajes de programación. Igualmente, se usó una

condición en la animación. Así pues, se puede consultar el apartado sobre condicionales y operadores booleanos. Finalmente, también se modificaron los colores de las manecillas. El control de colores se puede consultar en el apartado sobre la herramienta de control de colores.

Hagamos otro ejercicio que involucra animaciones un poco más complicadas. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Animación 2. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Para no empezar desde cero, comenzamos usando el interactivo que resultó del ejercicio sobre matrices (Definiciones Matriz). Puede usar este vínculo para referirse a dicho ejercicio y revisarlo antes de comenzar el nuevo.

La idea de este interactivo es que las partículas ahora no sólo se muestren al inicio, sino que cada una se mueve simultáneamente como en un movimiento browniano (que vibren como si fueran partículas de polvo en el aire).

En este ejercicio se pudo notar que la animación puede incluir varias instrucciones en su interior, y algunas de estas pueden inclusive ser funciones cíclicas. Se observó que el incluir las instrucciones iniciales de la función, que mueven a las partículas una por una, pueden ser incrustadas en una función llamada por la animación para lograr que todas se muevan simultáneamente. La función, en cada paso de la animación, mueve todas y cada una de las 500 partículas.

Conviene también notar nuevamente el papel del campo *local* en una función, el cual permite que una variable con igual nombre a otra fuera de la función se comporte como si no fuera la misma. Un cambio en dicha variable no afectará cualquier variable fuera de la función, a pesar de que tenga el mismo nombre.

En este ejercicio se usaron condicionales tanto en los campos *mientras* de la animación y la función, así como para el desplazamiento de cada partícula. Se puede consultar el apartado sobre condicionales y operadores booleanos para mayor información al respecto. e

13

Funcionalidad intrínseca de Descartes

Descartes cuenta con muchas funciones y variables propias que evitan que el usuario tenga que programar las acciones más básicas. Así pues, el usuario puede bien usar las ya existentes para facilitarse el trabajo.

Es difícil tener en mente todas estas funciones y variables. Por lo mismo, se espera que este apartado funcione como una guía de rápido acceso a las funciones y variables intrínsecas de Descartes.

También se incluyen, cuando se considera importante, algunos ejercicios que agrupan el uso de diversas de estas variables, al igual que se hizo en apartados anteriores.

13.1. Variables intrínsecas de Descartes

Descartes tiene muchas variables asociadas a propiedades de los espacios, a acciones del mouse, a estatus de controles gráficos, etc. Estas variables pueden usarse tanto para conocer el estado de ciertas partes del interactivo (por ejemplo, el tamaño de un espacio), así como para modificar propiedades mismas del interactivo. A continuación se enuncian en distintos apartados dependiendo de a qué tipo de acción o funcionalidad pertenecen.

13.1.1. Variables de Espacio

Las variables de espacio permiten conocer el alto, ancho y escala y de un espacio. Adicionalmente, el usuario puede modificarlas para lograr espacios de características particulares.

Las variables de espacio empiezan siempre con un prefijo que es el identificador del espacio en cuestión. Como notación usamos <Nombre del espacio>para indicar que ahí va el identificador del espacio.

- **<Nombre del espacio>._w**: Esta variable permite conocer el ancho de un espacio en pixeles. El sufijo *w* viene de *width*, que es anchura.
  Por ejemplo, la variable *Esp._w* nos permitiría conocer cuántos pixeles tiene de ancho el espacio *Esp*.

- **<Nombre del espacio>._h**: Esta variable permite conocer el alto de un espacio en pixeles. El sufijo *h* viene de *height*, que es altura.

Por ejemplo, la variable *Esp._h* nos permitiría conocer cuántos pixeles de altura tiene el espacio *Esp*.

- **<Nombre del espacio>.Ox**: Esta variable permite conocer el desplazamiento horizontal del origen en pixeles. El sufijo *Ox* viene de *offset de x*. Por defecto, el origen del plano aparece en el centro del espacio. Pero se le puede asignar un desplazamiento (u *offset*) positivo si se desea moverlo a la derecha, o negativo si se quiere moverlo a la izquierda.
  Esta variable sirve para conocer tanto el desplazamiento horizontal del origen, pero también se le puede asignar un valor si se quiere dar un desplazamiento horizontal. Es análoga a la variable que aparece como el campo de texto *O.x* en el selector *Espacio* cuando se tiene seleccionado el espacio en cuestión.

- **<Nombre del espacio>.Oy**: Esta variable permite conocer el desplazamiento vertical del origen en pixeles. El sufijo *Oy* viene de *offset de y*. Por defecto, el origen del plano aparece en el centro del espacio. Pero se le puede asignar un desplazamiento (u *offset*) negativo si se desea moverlo hacia arriba, o positivo si se quiere moverlo hacia abajo.
  Esta variable sirve para conocer tanto el desplazamiento horizontal del origen, pero también se le puede asignar un valor si se quiere dar un desplazamiento horizontal. Es análoga a la variable que aparece como el campo de texto *O.y* en el selector *Espacio* cuando se tiene seleccionado el espacio en cuestión.

- **<Nombre del espacio>.escala**: Esta variable se usa para conocer la escala (el número de pixeles que hay entre una unidad y otra) del espacio. Por ejemplo, *E7.escala* es la variable asociada a la escala del espacio *E7*. También es posible asignarle un valor a dicha variable para forzar al espacio una escala deseada. Es análoga a la variable que aparece como el campo de texto *escala* en el selector *Espacio* cuando se tiene seleccionado el espacio en cuestión.

- **<Nombre del espacio>.rot.y**: Esta variable sólo es válida para espacios tridimensionales. Guarda la rotación del espacio en grados alrededor del eje *y*. Por ejemplo, *E2.rot.y* sería la variable que guarda la rotación en grados alrededor del eje *y* del espacio tridimensional *E2*. También es posible asignarle un valor a esta variable para forzar que el espacio al inicio tenga la perspectiva deseada.

- **<Nombre del espacio>.rot.z**: Esta variable sólo es válida para espacios tridimensionales. Guarda la rotación del espacio en grados alrededor del eje *z*. Por ejemplo, *E2.rot.z* sería la variable que guarda la rotación en grados alrededor del eje *z* del espacio tridimensional *E2*. También es posible asignarle un valor a esta variable para forzar que el espacio al inicio tenga la perspectiva deseada.

En el apartado sobre el selector Espacios se incluyen ejercicios en donde se echa mano de estas variables.

13.1.2. Variables del Mouse

Las variables del mouse se usan para identificar el estado del mouse. Por ejemplo, si su botón izquierdo ha sido oprimido, o si está siendo oprimido. También se pueden conocer las coordenadas relativas del mouse.

Para usar estas variables es preciso indicar el espacio relacionado (sobre el cual se quiere conocer el estado del mouse) mediante un prefijo, que es el identificador del espacio. Como notación usamos <Nombre del espacio>para indicar que ahí va el identificador del espacio.

- **<Nombre del espacio>.mouse_x**: Esta variable permite conocer la coordenada horizontal relativa al plano cartesiano en que el mouse se encuentra al momento de estar presionado su botón. Su valor se refresca sólo cuando el mouse está oprimido (si el mouse sólo se pasea sin estar oprimido, el valor de esta variable no se refresca). Por ejemplo, la variable *Esp.mouse_x* nos indica la coordenada horizontal del mouse en el espacio *Esp* cuando se hace clic con el mismo.

- **<Nombre del espacio>.mouse_y**: Esta variable permite conocer la coordenada vertical relativa al plano cartesiano en que el mouse se encuentra al momento de estar presionado su botón. Su valor se refresca sólo cuando el mouse está oprimido (si el mouse sólo se pasea sin estar oprimido, el valor de esta variable no se refresca).
  Por ejemplo, la variable *Esp.mouse_y* nos indica la coordenada vertical del mouse en el espacio *Esp* cuando se hace clic con el mismo.

  NOTA: Es posible que las variables del mouse se refresquen aún si el botón del mouse no se encuentra oprimido. Ello se puede lograr con el checkbox sensible a los movimientos del mouse, que se aborda en el apartado *Espacio*. No obstante, como ya se mencionó en el apartado *Espacio*, no se recomienda abusar de esta funcionalidad dado que involucra mayor cantidada de cálculos y podría ralentizar las escenas.

- **<Nombre del espacio>.mouse_clicked**: Esta variable vale la unidad cuando el botón del mouse ha sido soltado tras por lo menos haber hecho clic una vez en el espacio en cuestión. Si el botón del mouse se oprime por primera vez y se mantiene oprimido, su valor es cero. Sólo se vuelve uno hasta soltar el botón mouse. Así pues, nos permite saber si el usuario ha hecho clic en un espacio o no, y siempre y cuando el botón del mouse ya haya sido soltado.

- **<Nombre del espacio>.mouse_pressed**: Esta variable vale la unidad siempre que el botón izquierdo del mouse se encuentra oprimido. Si se suelta dicho botón, el valor de la variable regresa a cero.

Hagamos un ejercicio que permite entender mejor cómo funcionan estas variables y, adicionalmente, permiten usarlas para conocer otros estados del mouse. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Mouse. El

documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio observamos una forma de manipulación de las variables del mouse que nos sirven para conocer otros estados del mismo (como cuando se arrastra el mouse mientras está apretado su botón). Note que para lograr esto se aprovechó el algoritmo *CALCULOS*, que se repite cada vez que el usuario hace algo en el interactivo. Y dentro del algoritmo *CALCULOS* se incluyen una serie de instrucciones que hábilmente, usando una variable auxiliar *MPA*, permiten conocer nuevos estados del mouse. Analicemos que sucede en cada paso.

- La variable auxiliar *MPA* vale 0 al inicio. Cuando oprimimos el mouse se ejecuta el algoritmo *CALCULOS*. *MP* vale 1 (por lo que !*MP* vale 0), *MPA* vale 0 (por lo que *!MPA* vale 1). Así, *MDown* vale 1 & 1 que es 1; *MDragged* vale 1 & 0 que es 0; y *MReleased* vale 1 & !1 (o 1 & 0) que es cero. Sólo la variable *MDown* vale 1, que es lo esperado. Al final del algoritmo, *MPA* adopta el valor de 1 de *MP*.
- La variable auxiliar *MPA* hasta ahora vale 1. Al arrastrar el mouse mientras está oprimido, el algoritmo *CALCULOS* se ejecuta nuevamente. Ahora *MP* vale 1 otra vez (por lo que !*MP* vale 0) y *MPA* retiene su valor anterior de 1 (por lo que !*MPA* vale 0). Así, *MDown* vale 1 & !1 (o 1 & 0) que es 0; *MDragged* vale 1 & 1 que es 1; y *MReleased* vale 1 & !1 (o 1 & 0) que es 0. Sólo la variable *MDragged* vale 1, que es lo esperado. Al final del algoritmo, *MPA* adopta el valor del 1 de *MP*.
- La variable auxiliar *MPA* sigue valiendo 1. Cuando se suelta el botón del mouse, el algoritmo *CALCULOS* se ejecuta nuevamente. Ahora *MP* vale 0 (por lo que !*MP* vale 1) y *MPA* vale 1 (por lo que !*MPA* vale 0). Así, *MDown* vale 0 & !1 (o 0 & 0) que es 0; *MDragged* vale 0 & 1 que es 0; y *MReleased* vale 1 & !0 (o 1 & 1) que es 1. Sólo la variable *MReleased* termina con el valor 1, como es esperado. Al final del algoritmo, se asigna el valor de 0 de *MP* a la variable *MPA*. Es decir, volvemos al valor de *MPA* inicial.

Notamos que la variable auxiliar *MPA* es la que juega un papel crucial en determinar estos nuevos estados del mouse. Con estas instrucciones simples, podemos saber si el mouse sólo está siendo oprimido, o si está siendo oprimido y arrastrado, o si se acaba de soltar. Ello resulta útil cuando se quieren manipular objetos mediante instrucciones del mouse.

Se usaron una gran cantidad de condicionales y operadores booleanos en este ejercicio. Puede consultarlos más a fondo en el apartado sobre condicionales y operadores booleanos.

13.1.3. Variables de los controles campo de texto

Sólo hay una variable de este tipo de controles, y se construye tomando el identificador del control y agregándole el sufijo *.activo*. Por ejemplo para un control con identificador

t1, la variable sería *t1.activo*. Esta variable adopta el valor 1 cuando el control está siendo utilizado (por ejemplo, si el cursor se encuentra parpadeando dentro de él), y 0 de lo contrario. Esta variable es de solo lectura, que implica que sólo cambia de valor mediante manipulaciones del control en juego, y no por asignaciones directas de valores.

13.1.4. Variables de los controles gráficos

Las variables de los controles gráficos se usan para conocer las coordenadas relativas de un control gráfico, así como para asignarle valores a las mismas. También nos permiten saber si un determinado control gráfico está siendo usado o no.

Para usar estas variables es preciso indicar el identificador del control gráfico (sobre el cual se quiere conocer información) mediante un prefijo, que es el identificador del mismo del control. Como notación usamos <Identificador del control>para indicar que ahí va el identificador del control gráfico.

- **<Identificador del control>.x**: Esta variable se puede usar para imprimir el valor de la coordenada horizontal del control gráfico relativa al plano cartesiano. También es posible asignarle un valor a esta variable para colocar al control gráfico en una determinada posición horizontal. Por ejemplo, la variable *g1.x* guarda el valor de la coordenada horizontal de un control gráfico con identificador *g1*.

- **<Identificador del control>.y**: Esta variable se puede usar para imprimir el valor de la coordenada vertical del control gráfico relativa al plano cartesiano. También es posible asignarle un valor a esta variable para colocar al control gráfico en una determinada posición vertical. Por ejemplo, la variable *g1.y* guarda el valor de la coordenada vertical de un control gráfico con identificador *g1*.

- **<Identificador del control>.activo**: Esta variable nos informa si un control está siendo usado o si no. En caso afirmativo, su valor es 1 y de lo contrario es 0. Por ejemplo, la variable *g1.activo* nos indica si el control gráfico *g1* está siendo usado o no. Un control gráfico está activo no sólo si el mouse se encuentra oprimido sobre él controlándolo. También se considera activo si se le hizo un clic y es el último objeto que se seleccionó. Es decir, si se hizo clic en el control gráfico, éste seguirá activo hasta que el mouse haga clic en otro lado distinto al control gráfico.

Hagamos un breve ejercicio con un control gráfico para ver cómo funcionan estas variables. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en ControlGráfico. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *Documentacion-DescartesJS.zip*.

En este ejercicio podemos notar el uso de las variables de los controles gráficos que nos permiten conocer las coordenadas de un control, así como su estado de actividad. Ello resulta útil muchas veces para conocer las coordenadas donde se desea colocar algo

(por ejemplo, un texto). En algunas ocasiones también se desea que se haga algún cálculo cuando un control gráfico ha sido activado, lo cual se puede lograr con un evento cuya condición es el estado de actividad del control gráfico. Finalmente, puede ser necesario regresar los controles gráficos a una posición predeterminada una vez que el usuario los ha modificado, lo cual se logra asignando a sus coordenadas los valores de dicha posición.

Para este ejercicio se usaron controles gráficos. Se puede consultar más información sobre ellos en el apartado sobre controles gráficos.

13.1.5. Variables de controles de audio y video

Existe una variable **<identificador del control>.currentTime** muy parecida a la función *<identificador del control>.currentTime()* (a la que, dándole un tiempo de argumento, posiciona la reproducción en el tiempo indicado del audio o video asociado a un control). La variable (que, a diferencia de la función del mismo nombre, no lleva los paréntesis encontrados en toda función), guarda el tiempo en segundos en que se encuentra el audio o video en ese momento. Si se ha de mostrar como parte de un texto, es necesario tener activada una animación mientras se reproduce el audio o video puesto que la animación hace que constantemente se refresque el texto que muestra la variable.

Puede consultar el ejercicio Controles Audio para revisar cómo se utilizan estas variables.

13.1.6. Variables de controles tipo menú

Sólo hay una variable básica relacionada a los menús, **<identificador del menú>.txt**, que permite recuperar el texto correspondiente a la opción del menú seleccionada.

Así, si hubiera un menú con identificador *color*, y que en su parámetro *opciones* tiene `azul,blanco,rojo`, y se encuentra seleccionada la segunda opción, el valor del identificador (es decir, la variable *color*) valdrá 1, y la variable *color.txt* vale la cadena de texto `'blanco'`.

13.1.7. Variables de vectores y matrices

Hay una variable básica relacionada a los vectores que permite recuperar la longitud de entradas del mismo. Asimismo, hay un par de variables de matrices que permiten conocer el número de columnas y filas de sus dimensiones.

Para los vectores:

- **<identificador del vector>.long**: devuelve el número de entradas de un vector. Por ejemplo, para un vector *v1* con cinco entradas, `v1.long` devuelve 5.

Para las matrices:

- **<identificador de la matriz>.filas**: devuelve las filas de la matriz. Por ejemplo, para una matriz *M2*, la variable *M2.filas* devuelve el número de filas de la matriz.
- **<identificador de la matriz>.columnas**: devuelve las columnas de la matriz. Por ejemplo, para una matriz *M2*, la variable *M2.columnas* devuelve el número de columnas de la matriz.

13.1.8. Variables de ruta

Existe una manera en que el usuario de un interactivo en servidor puede indicar algún parámetro a una escena de Descartes directamente desde la ruta en el navegador que usa hacia el archivo *html* de Descartes que abrirá. El nombre de esta variable es *URL.<etiqueta>*, donde *etiqueta* es un nombre de etiqueta que se da en la ruta, junto con su valor asociado. Conviene ilustrar esto con un ejemplo.

Suponga que la escena interactiva se encuentra, **en servidor**, en el sitio siguiente: *https://sitio.abcd.mx/index.html*. Si en un navegador se abre directamente esa dirección, se abre el interactivo como de costumbre. Sin embargo, si se usara una etiqueta, por ejemplo, *https://sitio.abcd.mx/index.html?vista=horizontal*, entonces la escena internamente tendrá la variable *URL.vista*, cuyo valor es la cadena de texto *horizontal*.

Así, la etiqueta en el ejemplo está indicada por el texto *?vista=horizontal*. La variable en el ejemplo es *URL.vista* ya que *vista* es la palabra clave justo después del signo de interrogación tras la dirección natural del archivo interactivo. Y el valor otorgado a la variable *URL.vista* es *horizontal*, que es lo que sigue del signo =. Si el usario no indica etiqueta alguna, internamente en la escena, la variable *URL.vista* tendrá un valor de 0. Como se puede notar, no es una variable predefinida. El usuario es quien la escoje. Bien podría tratarse de una variable *URL.pos* si la etiqueta de la dirección fuera, por ejemplo, *?pos=loquesea*. Y es importante observar que el prefijo debe ser siempre *URL*.

También es posible dar varias asignaciones simultáneamente separando las etiquetas por el símbolo *ampersand* (&). Siguiendo el mismo ejemplo, suponga que se da la ruta: *https://sitio.abcd.mx/index.html?vista=horizontal&paso=3*. En este caso, internamente en la escena habrá una variable *URL.vista*, cuyo valor es la cadena *horizontal*, pero también habra otra variable *URL.paso* con valor 3.

Este tipo de etiquetas permiten que el usuario indique cosas de antemano antes de que se cargue la escena. Se ha usado, por ejemplo, en editores de libros interactivos para que abran una página particular en lugar de mostrar la primera página por defecto. Para el ejemplo del valor *horizontal*, se ocurre que podría ser usado para especificar si una escena se abre en forma horizontal o vertical.

Es importante notar que estas variable sólo tienen sentido si la escena ha de ser abierta **en un servidor**. Para uso local no tienen sentido.

13.1.9. Variables generales de Descartes

Existe solamente una variable general de Descartes, que es *rnd*. Esta variable, cuyo nombre viene de *random* (o *aleatorio*) genera un valor aleatorio real entre 0 (incluyendo el cero) y 1 (se puede aproximar mucho a uno, pero nunca llega a ser 1) cada vez que es llamada. A veces se quiere tener un valor aleatorio entero definido en un determinado intervalo. Para ello es necesario asociar la variable *rnd* con algunas funciones que nos permitan tener ese comportamiento.

Hagamos un breve ejercicio cuyo propósito es generar parábolas verticales de la forma $y = ax^2 + bx + c$, donde el coeficiente a puede adoptar valores enteros entre -3 y 3 pero sin incluir el cero (si valiera cero, no se tendría una parábola sino una recta); el b puede adoptar valores enteros entre -4 y 4 incluyendo el 0; y c puede adoptar valores reales entre 0 y 3. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en VariablesGenerales. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *Documentacion-DescartesJS.zip*.

Este ejercicio es muy ilustrativo en muchos aspectos. Por un lado, cada variable tiene una asignación distinta de valores.

- La variable *a* primero extrae un signo de una expresión aleatoria y luego lo multiplica por otra expresión que es el valor entero aleatorio entre 1 y 3. Así, permite los valores -3, -2, -1, 1, 2 y 3 (excluyendo el 0). Note que aunque hay dos distintos *rnd* en su asignación, cada uno lleva un valor aleatorio nuevo (no son el mismo valor).

- La variable *b*, al no tener la restricción de no poder valer cero, se asigna como el valor -4 al que se le suma un valor aleatorio entero entre 0 y 8, de tal forma que puede adoptar valores entre -4 y +4.

- La variable *c*, al no tener la restricción de ser entera, no requiere la función *ent()* que devuelve el valor entero de su argumento.

- Las variables *a* y *b* conllevan un .99999 en su interior. Se deja al usuario que analice cuidadosamente qué pasaría si esa cadena de decimales de 9 no estuviera. Considere la variable *b*: ¿sería igualmente probable obtener un valor de +4 para esta variable que cualquiera de los otros valores? Note también que la variable *c* no lleva esa cadena decimal. Al no estar el argumento sujeto a la función que extrae el entero, si tuviera dicha cadena decimal, *c* podría valer 3.59, y no cumpliría la restricción de tener un valor máximo de 3.

- Se puede notar que el texto enriquecido, a diferencia de aquél sencillo, permite imprimir valores de variables cada uno customizado de forma distinta. Así, es posible en un mismo texto imprimir valores con distinto formato (algunos con decimales, otros no).

Otra conclusión que se puede extraer de este ejercicio es la importancia de usar textos de depuración. Puede ser que el texto que imprime los valores de las variables no deba ser mostrado al usuario. Pero el programador puede echar mano de él para saber si su código está haciendo lo deseado. Una vez que el programa ya quedó listo, dicho texto puede esconderse o eliminarse en su totalidad.

En este ejercicio se usaron funciones como *ent()* y *sgn()* que son propias de Descartes. Se pueden consultar con mayor profundidad en el apartado sobre funciones intrínsecas de Descartes. También se puede consultar más sobre el gráfico utilizado en el apartado sobre el gráfico ecuación.

13.1.10. Variables numéricas

Hay dos variables numéricas predefinidas en Descartes. Una de ellas es *pi* y la otra *e*. La primera es una aproximación al valor de π. La segunda una aproximación a *e*, el número de Euler o constante de Napier.

El usuario puede imprimir o usar en operaciones el valor de cualquiera de las dos. Conviene tenerlas en mente para evitar usar sus nombres (*pi* y *e*) como variables personalizadas o identificadores. Es decir, conviene respetarlas y no asignarles valores.

13.1.11. Variables de información y adecuación de Descartes

- **device**: es una variable cuyo valor es una cadena de texto que indica el tipo de dispositivo en que se está usando la escena de Descartes. Las opciones son *desktop* (que corresponde a un ordenador), *tablet* (que corresponde a una tableta), y *mobile* (que corresponde a un teléfono móvil).

- **dispositivo**: es equivalente a la variable *device*.

- **_NUM_MAX_ITE_ALG_**: es una variable que contiene el número máximo de iteraciones que Descartes permite usar en un algoritmo con repetición. Se decidió que no se pudiera superar un máximo de 100,000 iteraciones (valor de la variable por defecto), con el objeto de que si el usuario por error incurre en un error donde un ciclo se repite indefinidamente, este tope evitaría que la escena se cuelgue. Sin embargo, puede ser necesario superar dicho tope en algunos tipos de simulaciones. Así, el usuario puede reasignar un valor mayor a 100,000 a esta variable, permitiendo así realizar más ciclos.

Las variables *dispositivo* y *device* son útiles al permitir que se use un control con el teclado virtual activado dependiendo de si la escena está abierta en una computadora o en otro dispositivo. Un control, que no usa teclado virtual, se puede mostrar en una escena si ésta está abierta en un ordenador (el caso en que *dispositivo=desktop*), mientras que otro, con el teclado virtual activado, se puede mostrar en su lugar si la escena está abierta en un celular (el caso *dispositivo=mobile*). Recuerde que en los teléfonos móviles y tabletas

se activa por defecto el teclado virtual del dispositivo y que éste desplaza a la escena interactiva rompiendo la estética e incluso la funcionalidad de la misma. Sin embargo, con la alternativa propuesta se evita que se active el teclado nativo del dispositivo y el desarrollador puede ubicar el teclado virtual de Descartes en la posición que desee, manteniendo la estética y funcionalidad de la escena.

13.2. Funciones intrínsecas de Descartes

Descartes tiene una variedad de funciones propias, muchas de las cuales son las típicas utilizadas en casi cualquier lenguaje de programación. No obstante, adicionalmente tiene otras propias relacionadas a audio, video, evaluación de funciones, etc. que se revisarán a continuación. Las funciones aquí presentadas se agrupan según su funcionalidad.

13.2.1. Funciones comunes

A continuación se enlistan alfabéticamente las funciones que Descartes tiene en común con otros lenguajes de programación. La mayoría de éstas son funciones matemáticas. En ocasiones, el nombre de la función puede variar, pero su acción es la misma.

- **abs(x)**: Es una función que recibe un argumento numérico *x* y devuelve el valor absoluto del mismo. Por ejemplo, *abs(-2)* devolverá *2*.
- **acos(x)**: Es una función que recibe un argumento numérico *x* y devuelve su arcocoseno en radianes. Por ejemplo, *acos(-1)* devolverá el valor *3.141...* radianes (que corresponden a *180°*).
- **_AnchoDeCadena_(string,font,style,size)**: Es una función que recibe cuatro argumentos y devuelve una longitud en pixeles. *string* es una cadena de texto, *font* una fuente (cuyas opciones son 'Monospaced', 'Serif'o 'SansSerif'), *style* los atributos de estilo (cuyas opciones son 'PLAIN', 'ITALIC', 'BOLD', 'ITALIC+BOLD'o 'BOLD+ITALIC'), y *size* un tamaño de fuente en pixeles. La función devuelve la longitud en pixeles de la cadena de texto dados estos cuatro argumentos en dicho orden Por ejemplo, *_AnchoDeCadena_('hola, ¿cómo estás?','Monospaced','ITALIC+BOLD',18)* calcula el ancho de la cadena *hola, ¿cómo estás?* en fuente *monospaced*, con estilo itálicas y negritas; y de tamaño de fuente de 18. Esta función resulta útil cuando, por ejemplo, se requiere conocer la longitud de un texto dado para poderlo centrar.
- **aleatorio(a,b)**: Es una función que recibe un par de argumentos *a* y *b* y devuelve un valor aleatorio dentro de una distribución uniforme en `[a,b)` cuando *b>a*, o en `[b,a)` cuando *a>b*. Note que, en el caso *a=0* y *b=1*, la función es equivalente a mandar llamar la variable de *DescartesJS* rnd.
- **asin(x)**: Es una función que recibe un argumento *x* y devuelve su arcoseno en radianes. Por ejemplo, *asin(1)* devolverá el valor *1.570...* radianes (que corresponden a *90°*).

- **atan(x)**: Es una función que recibe un argumento numérico *x*y devuelve su arcotangente en radianes. Por ejemplo, *atan(1)* devolverá el valor *0.785...* radianes (correspondientes a *45°*).
- **ceil(x)**: (de *ceiling*) Es una función que recibe un argumento numérico *x* y devuelve el entero inmediato superior. Por ejemplo, *ceil(-4.2)* devolverá el valor *-4* y *ceil(5.2)* devolverá *6*.
- **charAt(string,n)**: (de *character at*) Es una función que recibe como primer argumento una cadena o texto (en el ejemplo es *string*) como primer argumento, y un número (en el ejemplo es *n*) que funge como índice en el segundo argumento. La función devuelve el caracter de la cadena en la posición dictada por el índice. Se comienza a contar el primer caracter de la cadena como el cero-ésimo. Por ejemplo, *charAt('Saludos',3)* devolverá *u*. Es equivalente a *_letraEn_(string,n)*.
- **cos(x)**: Es una función que recibe un argumento numérico (en el ejemplo es *x*), lo interpreta en radianes, y devuelve el coseno del mismo. Por ejemplo, *cos(pi/2)* (recordamos que $\frac{\pi}{2}$ equivale a 90°) devolverá el valor *0*.
- **cot(x)**: Es una función que recibe un argumento numérico (en el ejemplo es *x*), lo interpreta en radianes, y devuelve la cotangente del mismo. Por ejemplo, *cot(0.785398163)* devolverá un valor cercano a *1* pues *0.785398163* radianes es cercano a *45°*.
- **csc(x)**: Es una función que recibe un argumento numérico (en el ejemplo es *x*), lo interpreta en radianes, y devuelve la cosecante del mismo. Por ejemplo, *csc(pi/2)* devolverá un valor de *1*.
- **ent(x)**: Es una función que recibe un argumento numérico (en el ejemplo es *x*) y devuelve el mismo entero en cuestión pero sin decimales. Por ejemplo, *ent(-3.2)* devolverá el valor *-3*, y *ent(4.71)* devolverá el valor *4*. Se puede ver como una función que trunca (recorta) los decimales.
- **exp(x)**: Es una función que recibe un argumento numérico (en el ejemplo es *x*) y devuelve el valor de la exponencial neperiana del mismo. Por ejemplo, *exp(2)* devolverá el valor de e^2, que es *7.389...*.
- **floor(x)**: Es una función que recibe un argumento numérico (en el ejemplo es *x*) y devuelve el entero inmediato inferior. Por ejemplo, *floor(-4.2)* devolverá el valor *-5* y *floor(5.2)* devolverá *5*. De tal forma que es la función opuesta a *ceil()*.
- **indexOf(string,str)**: Es una función que recibe dos argumentos de tipo cadena de texto. El primero (en el ejemplo es *string*) es la cadena de texto principal. El segundo (en el ejemplo es *str*) es un texto contenido en el primero. La función devuelve un entero que es el índice (contando el primer carácter del texto principal como el cero-ésimo) en el que empieza el texto contenido en la primera instancia encontrada. Si no hay instancias del texto contenido en el texto principal, se devuelve un valor de *-1*. Por ejemplo, *indexOf('hola','ola')* devolverá el valor *1*.
- **_índiceDe_(string,str)**: Función equivalente a *indexOf(string,str)*. Ésta es de las pocas funciones que permite un acento (en la primera letra de *índiceDe*).

- **lastIndexOf(string,str)**: Es una función similar a *indexOf(string,str)* que recibe dos cadenas o textos como argumentos. La primera cadena (en el ejemplo es *string*) es la cadena principal, mientras que la segunda cadena (en el ejemplo es *str*) es la cadena que se busca dentro de la principal. La función devuelve el índice (empezando a contar desde cero) en que la segunda cadena aparece dentro de la primera por última vez. Ejemplo: *lastIndexOf('anitalavalatina','ala')* devolverá el valor *8* dado que *ala* aparece por última vez en la octava posición en la cadena *anitalavalatina*.
- **_letraEn_(string,n)**: Es una función que recibe dos argumentos. El primero (en el ejemplo es *string*) es una cadena de texto. El segundo (en el ejemplo es *n*) es un entero que corresponde al índice de la letra (empezando a contar la primera letra como la cero-ésima) que devolverá la función. Por ejemplo, *_letraEn_('hola',2)* devolverá el texto *l*.
- **_longitud_(string)**: Función equivalente a *strLength(string)*.
- **log(x)**: Es una función que recibe un argumento numérico (en el ejemplo es *x*) y devuelve el logaritmo neperiano del mismo. Por ejemplo, *log(7.389056099)* devolverá un valor cercano a *2*, ya que $e^2 = 7.389056099....$
- **log10(x)**: es una función que recibe un argumento numérico (en el ejemplo es *x*) y devuelve el logaritmo base 10 del mismo. Por ejemplo, *log10(1000)* devolverá un valor de 3, ya que $10^3 = 1000$.
- **max(a,b)**: es una función que recibe un par de argumentos numéricos (en el ejemplo son *a* y *b*) y devuelve aquél que es mayor de ambos. Por ejemplo *max(-6,-3.59)* devolverá -3.59.
- **min(a,b)**: es una función que recibe un par de argumentos numéricos (en el ejemplo son *a* y *b*) y devuelve aquél que es menor de ambos. Por ejemplo *min(-7,-4.2)* devolverá -7.
- **piso(x)**: Función equivalente a *floor(x)*.
- **raíz(x)**: Función equivalente a *sqrt(x)*. Es de las pocas funciones que permiten acentos.
- **random(a,b)**: Función equivalente a *aleatorio(a,b)*.
- **redondeo(x)**: Función equivalente a *round(x)*, explicada en breve.
- **replace(string,str1,str2)**: Es una función que recibe tres cadenas o textos. La primera es la cadena principal (en el ejemplo es *string*). La segunda (en el ejemplo es *str1*) es la cadena a buscar y la tercera (en el ejemplo es *str2*) es la cadena que reemplazará a la segunda. La función sustituye todas las instancias de la segunda cadena en la cadena principal por el texto de la tercera cadena. Por ejemplo, *replace('123123','2','dos)* devolverá la cadena *1dos31dos3*.
- **round(x)**: Es una función que recibe un argumento numérico (en el ejemplo es *x*) y devuelve el entero tras redondear. Por ejemplo, *round(5.8)* devolverá *6*, mientras que *round(5.2)* devolverá *5*. En general, el redondeo de un número *x* se pue-

de entender como el entero inmediatamente inferior de *x+0.5*, de tal suerte que *round(a)=ent(x+0.5)*, o lo que es lo mismo, *round(x)=floor(x+0.5)*. De ahí que el redondeo de un medio entero es el entero inmediatamente superior.

- **sec(x)**: Es una función que recibe un argumento numérico (en el ejemplo es *x*), lo interpreta en radianes, y devuelve la secante del mismo. Por ejemplo, *sec(pi)* devolverá un valor de *-1*.
- **sen(x)**: Función equivalente a *sin(x)*.
- **sgn(x)**: Es una función que recibe un argumento numérico (en el ejemplo es *x*), y de ser éste positivo devuelve el valor +1. De ser el argumento negativo devuelve -1. Es decir, extrae sólo el signo del argumento. Por ejemplo, *sgn(-3)* devolverá el valor *-1*.
- **sin(x)**: Es una función que recibe un argumento numérico (en el ejemplo es *x*), lo interpreta en radianes, y devuelve el seno del mismo. Por ejemplo, *sin(pi/2)* (recordamos que $\frac{\pi}{2}$ equivale a 90°) devolverá el valor *1*.
- **sqr(x)**: Es una función que recibe un argumento numérico (en el ejemplo, *x*) y devuelve su valor al cuadrado. Viene de la palabra *square*. Por ejemplo, *sqr(3)* devolverá el valor *9*.
- **sqrt(x)**: Es una función que recibe un argumento numérico (en el ejemplo, *x*) y devuelve la raíz cuadrada del mismo. Viene de *square root*. Por ejemplo, *sqrt(9)* devolverá el valor *3*.
- **strLength(string)**: (de *string length*) Es una función que recibe como argumento una cadena de texto (en el ejemplo, *string*) y devuelve la longitud en caracteres del mismo. Por ejemplo, *strLength('hola')* devolverá el valor *4*.
- **substring(string,a,b)**: Es una función que recibe 3 argumentos. El primero es una cadena de texto (en el ejemplo, *string*), el segundo (en el ejemplo, *a*) y tercero (en el ejemplo, *b*) son enteros que corresponden a índices de los caracteres de la cadena, contando el primer caracter como el cero-ésimo. La función devuelve una cadena de texto que va del índice del segundo argumento (inclusivo) al índice del tercer argumento (exclusivo). Por ejemplo, *substring('hola',1,3)* devolverá la cadena *ol*.
- **_subcadena_(string,a,b)**: Función equivalente a *substring(string,a,b)*.
- **tan(x)**: Es una función que recibe un argumento numérico (en el ejemplo, *x*), lo interpreta en radianes, y devuelve la tangente del mismo. Por ejemplo, *tan(pi/4)* (recordamos que $\frac{\pi}{4}$ radianes equivalen a 45°) devolverá el valor *1*.
- **techo(x)**: Función equivalente a *ceil(x)*.
- **toFixed(x,n)**: Es una función que recibe primero un real (en el ejemplo, *x*) y luego un entero (en el ejemplo, *n*) como argumentos y devuelve, en forma de cadena de texto, el real ajustado a mostrar sólo el número de decimales equivalente al argumento entero de la función. Por ejemplo, en un gráfico tipo texto con 7 decimales fijos a mostrar, introducir *[toFixed(1/3,3)]* en su parámetro *texto* resultará en mostrar *0.333* en lugar de *0.3333333*.

Y note, también con 7 decimales fijos, la diferencia entre lo que devuelve *[toFixed(1/3,3)]* y *[toFixed(1/3,3)*1]*. En el primer caso, sólo se devuelve *0.333*, dado que *toFixed* tiene precedencia sobre el número de decimales indicado en el gráfico texto. En el segundo caso, al multiplicar por *1* lo que devuelve *toFixed*, toman precedencia los decimales indicados en el gráfico y se devuelve *0.3330000*.
Al hacer productos entre números obtenidos a partir de *toFixed()*, *DescartesJS* los comprende como números y los opera como tales. Sin embargo, si se intenta sumarlos, se devolverá la concatenación de las cadenas (ya que *toFixed()* realmente devuelve una cadena). Por ejemplo, *toFixed(1/3,3)+toFixed(1/7,2)* devuelve la concatenación de las cadenas *'0.333'* y *'0.14*. Es decir, devuelve la cadena *'0.3330.14'*. Para sumarlos como números, se puede echar mano de la función *_Num_()* del lenguaje de *DescartesJS*. Es decir, se aplica dicha función a cada valor devuelto por *toFixed()* y luego se suma. Así, *_Num_(toFixed(1/3,3))+_Num_(toFixed(1/7,2))* devuelve *0.473*.

- **toLowerCase(string)**: Es una función que recibe una cadena (en el ejemplo, *string*) y la devuelve sustituyendo todas las mayúsculas encontradas por sus minúsculas correspondientes. Por ejemplo *toLowerCase('Ahora Todo Está en Minúsculas')* devolverá *ahora todo está en minúsculas*.
- **toUpperCase(string)**: Es una función semejante a *toLowerCase(string)*, pero que, por el contrario, cambia todas las letras de la cadena por sus mayúsculas. Por ejemplo *toUpperCase('Ahora Todo Está en Mayúsculas')* devolverá *AHORA TODO ESTÁ EN MAYÚSCULAS*.
- **trim(string)**: Es una función que recibe una cadena o texto (en el ejemplo, *string*), y devuelve la misma pero recortándole cualquier espacio en blanco que pudiera tener al principio o al final. Por ejemplo, *'.'+'Hola '+'.'* devuelve *. Hola .*, mientras que *'.'+trim('Hola ')+'.'* devuelve *.Hola.* Observe que en el primer caso los puntos están separados de las letras por espacios y en el segundo no.

Existen también funciones trigonométricas hiperbólicas (seno, coseno y tangente). Basta agregar una *h* al final de cada una (*sinh()*, *cosh()* y *tanh()*).

En ocasiones se requiere obtener logaritmos en otras bases, más allá de las funciones logarítmicas existentes en Descartes (*log()* y *log10()*). En general, se puede usar la propiedad de que $log_a(x) = \frac{log(x)}{log(a)}$ (o bien, que $log_a(x) = \frac{log_{10}(x)}{log_{10}(a)}$) para obtener un logaritmo base *a* de un número *x*. Esto se puede asignar a una función nueva creada por el usuario. Por ejemplo, se podría crear una función *logbase(a,x)* que devuelva `log10(x)/log10(a)`. Esta función, que recibe la base y al número a obtener el logaritmo como argumentos, devolvería $log_a(x)$.

13.2.2. Funciones del lenguaje de Descartes

Existen pocas funciones únicas de Descartes. Una de ellas es ***_Eval_(string)***. Esta función recibe un argumento tipo cadena de texto (en el ejemplo, *string*) y permite generar una función a partir de texto del usuario. De esta forma, el usuario puede introducir el

texto de una función y ésta se puede evaluar y graficar. Como primer ejemplo, puede introducir en el paráemtro *texto* de un gráfico tipo texto `[_Eval_('sin(pi/6)']` y, al aplicar los cambios, el texto mostrado debe corresponder a 0.5. Observe que en dicho ejemplo, la cadena fue la función a evaluar (`'sin(pi/6)'`). La función suele usarse para evaluar textos de funciones del usuario. Por ejemplo, si en alguna parte se tiene una asignación del tipo *c1='sin(x)'*, *_Eval_(c1)* estará asociada a la función seno aplicada a la variable *x*. De esta forma, Descartes no se encuentra restringido a, por ejemplo, trazar funciones sólo dadas por el programador, sino que permite al usuario ingresar sus propias funciones.

Hagamos un breve ejercicio para dejar más claros estos aspectos. Este ejercicio consistirá en, por un lado, mostrar que el usuario puede graficar una función que él desee y, por otro lado, evaluar su función para un determinado valor de la variable independiente. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en FuncionesDescartes. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En este ejercicio hay varias cosas dignas de observación.

- Es importante notar que cuando se usa la función *_Eval_()*, su argumento debe ser texto. Ello implica que si su argumento es una variable, ésta debe ser tipo texto. Si es un control numérico de tipo *campo de texto*, éste debe de preferencia ser *solo texto* y su valor asignado debe estar siempre entre comillas sencillas para asegurar que sea sólo texto y no sea interpretado como un valor numérico. Por ejemplo, si el valor de un campo de texto es 1 en lugar de '1', pueden llegar a presentarse errores al usarlo dentro de una función *_Eval_()*.

- También es importante notar que el código es más claro y fácil de manipular cuando las evaluaciones de los campos de texto quedan asociadas a funciones.

- Nótese la diferencia entre evaluar una función completa y evaluarla en un solo punto. El gráfico traza la función *fn()* (que no tiene argumentos), mientras que el texto imprime el valor de una función evaluada *fn2(x)* que tiene como argumento a *x*. Descartes inmediatamente interpreta el valor del interior de *_Eval_()* como la *x* a usar para la función. Muchas veces se busca que el alumno introduzca una fórmula y determinar si es correcta o no. En estos casos se puede usar esta funcionalidad y comparar los valores de la fórmula del alumno, evaluados en varios valores de la variable independiente, contra los valores teóricos. Si coinciden en varios casos, se puede decir que la fórmula del alumno es correcta.

Para este ejercicio se usaron varias funciones a graficar. Para más detalle sobre éstas se puede consultar el apartado sobre las funciones comunes a diversos lenguajes de programación. Se puede consultar también el apartado sobre los controles numéricos tipo *campo de texto* para más información sobre ellos.

La función *_Eval_* puede usarse también para hacer asignaciones y comparaciones booleanas, que permite hacer más versátiles los programas. Para hacer asignaciones de

valores se usa := como parte del texto evaluar, mientras que los operadores booleanos no cambian de notación. Por ejemplo, se podría asociar a un botón, cuya acción sea calcular, la serie de instrucciones:

```
_Eval_('zzz:=zzz+1')
_Eval_('v1:=(zzz%2==0)?1:0')
_Eval_('v2:=(zzz%2!=0)?1:0')
_Eval_('v3:=(zzz<=3)?1:0')
```

Observamos que el primer renglón asgina *zzz+1* a la variable *zzz*. Es decir, incrementa el valor de la variable en una unidad. El segundo renglón se encarga de asignar a una variable *v1* un valor condicional: 1 si *zzz* es par, 0 si es non. El tercer renglón asigna a una variable *v2* un valor condicional: 1 si *zzz* es impar, 0 si es par. Y el cuarto renglón asigna a una variable *v3* un valor condicional, 1 si *zzz* es menor igual a 3 o 0 si es mayor que 3. Si se imprimen las variables *zzz*, *v1*, *v2* y *v3* en un interactivo con un botón así dispuesto, se puede notar el comportamiento de ellas.

La utilidad de poder asignar valores y usar booleanos de esta manera radica en que las variables pueden ser dadas por el usuario del interactivo y no solamente dictaminadas por el programador. Adicionalmente, tiene la virtud de, por ejemplo, poder usar un solo control para cambiar el valor de una gran multitud de variables. Conviene hacer esto en un breve ejercicio para que quede más claro. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en AsignacionesEnEvaluacion. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

ExecBlock(string,label) es una función propia de Descartes, similar a *_Eval_*, pero que permite evaluar no sólo una línea o comando en un texto, sino un texto *string* que contiene varios comandos, y que se encuentra flanqueado por una etiqueta *label* que consiste en un texto correspondiente a una etiqueta en lenguaje HTML. Por ejemplo, suponga que cuenta con un texto en una variable *string* y que es el siguiente:

```
Inicio del texto
<ETIQUETA>
tx=texto vario
x=2
</ETIQUETA>
Y otro texto...
```

Dicho texto pudo haber sido leído de archivo, por ejemplo. Los saltos de línea pueden bien haber sido introducidos mediante la tecla *ENTER* en un editor de texto *sin BOM*, o explícitamente mediante el código `\n` que en textos de Descartes representa también saltos de línea. Así, `_ExecBlock_(string,'ETIQUETA')` buscará en dicho texto el contenido dentro de la etiqueta en cuestión (en este caso entre `<ETIQUETA>y </ETIQUETA>`), y aplicará las asignaciones correspondientes. En este ejemplo, si tras ejecutar la función se imprime en un gráfico tipo texto el valor de la variable *tx*, se observará el texto *texto vario*. Por otra parte, la variable *x* adopta el *2*. No obstante, posteriormente puede operarse el valor.

Conviene hacer un ejercicio para dejar estos conceptos claros. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en EjecucionBloque. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Observe en este ejemplo que el bloque involucra dos asignaciones. Ambas son ejecutadas al ejecutar la función *ExecBlock*. Y observe que sólo los comandos dentro del bloque flanqueado por `<ETIQUETA>...</ETIQUETA>` son ejecutados; lo demás es ignorado. Además, cuando una asignación dentro del bloque es del tipo `x=2` (es decir, una asignación numérica directa), la variable adopta el valor numérico 2, no un texto cuyo contenido es 2. De quererse que adopte el contenido como texto, se tendría que usar la asignación `x='2'` (es decir, flanqueando el texto a incluir por comillas sencillas).

Num(string) es otra función propia de Descartes que recibe una cadena de texto (en el ejemplo, *string*) correspondiente a una expresión numérica decimal y devuelve al número como tal. En caso que la cadena sea algo distinto a una expresión decimal, devolverá *NaN* (*not a number*, o *no es un número*). Para ejemplificar la utilidad de la función, considere que *string* es un campo de texto con su casilla *sólo texto* marcada. En él se supone el usuario introduce la respuesta a la pregunta *1+2=?*. Si el usuario introduce `1+2`, la función *_Num_(c)* devolverá *NaN*. Pero si el usuario introduce `3`, *_Num_(c)* devolverá 3.

Así, esta función evita el uso de la calculadora intrínseca a un control tipo Campo de texto al evaluar una expresión. Si el campo de texto estuviera con su casilla *sólo texto* desmarcada y sólo se pidiera que se devuelva el valor del campo de texto tras pulsar *INTRO*, al introducir `1+2` se devolvería 3, directamente usando la calculadora de Descartes, lo que permite al usuario "*hacer trampa*".

La función *_Num_()* permite también que su argumento, de ser un decimal, use la coma o el punto como símbolo decimal. Para el correcto funcionamiento usando identificadores de campos de texto como argumentos de esta función es necesario que el campo de texto en cuestión tenga marcada su casilla de verificación *sólo texto*.

isNumber(x) es una función de Descartes que recibe un argumento cualquiera *x* y devuelve 1 si es un número real o 0 de lo contrario. Con esta función es posible hacer condicionales que dependen de si el argumento es un *NaN*, *Infinity* (infinito) o cadena de texto (en cuyo caso la función devuelve 0) o si se trata de un número real (donde devuelve 1). Como ejemplo, si se imprime en un gráfico tipo texto `[isNumber(log(-1))]`, o `[isNumber(sqrt(-1))]`, o `[isNumber(1/0)]`, la función devuelve el valor cero, puesto que en los dos primeros ejemplos el argumento equivale a *NaN* y en el último a *Infinity*. Si, por el contrario, se busca imprimir `[isNumber(e^2)]`, o `[isNumber(sqrt(25))]`, la función devuelve 1 dado que los argumento son números reales. Si el argumento de la función es una cadena de texto, también devuelve 0. Por ejemplo, *isNumber(123)* devuelve 1, pero *isNumber('123')* devuelve 0 al ser una cadena de texto compuesta por números. No obstante, *isNumber(_Num_('123'))* nuevamente devuelve 1, pues *_Num_()* convierte la cadena de texto a su valor numérico.

13.2.3. Funciones de espacios HTMLIFrame

Hay otras funciones propias de Descartes que se usan para comunicar información entre un espacio principal y un espacio HTMLIFrame (véase el apartado Espacio HTMLIFrame para más información) contenido en el principal.

- ***<identificador>.set('vr',var)***. Esta función requiere un identificador de un espacio (principal o subordinado) antes del sufijo *.set*, y como argumentos lleva una cadena de texto y una variable. Se usa cuando una escena tiene un espacio principal y uno subordinado (como HTMLIFrame) que requieren comunicarse información. En el ejemplo, el primer argumento (la cadena de texto *'vr'*) representa el nombre de una variable que el espacio receptor de información usará, y el segundo argumento (la variable *var*) es la variable del espacio emisor de información que contiene el valor que se desea comunicar al receptor. El <*identificador*> es aquél del espacio receptor de la información. Es preciso aclarar algunos detalles sobre dicho identificador. Sin embargo, indicar esto no hace que el espacio receptor de información se actualice. Es ahí donde entra la otra función propia <*identificador*>*.update()* que se explica en breve.

- ***<identificador>.exec('fnc',func)***. Esta función manda llamar una función *fnc* que debe encontrarse en el selector *Definiciones* del espacio receptor de información. <*identificador*>*.exec* acepta siempre dos argumentos: el nombre de la función a ejecutar en la escena (la cadena de texto 'fnc'en el ejemplo, donde *fnc* es el nombre de la función) y el segundo es el argumento de la función, en caso de requerir uno (en este ejemplo sería una variable *func*). Si la función *fnc* sólo cuenta con un argumento, puede directamente ir como en el ejemplo. No obstante, en caso que tenga más argumentos, en lugar de *func* va una cadena de texto con los distintos argumentos separados entre ellos por comas. Por ejemplo, <*identificador*>*.exec('fnc','a,b')*, donde los argumentos de la función son *a* y *b*. Entonces podría mandarse llamar usando `<identificador>.exec('func','a,b')`. Si la función no involucra argumento alguno, se puede poner una cadena de texto vacía en su lugar. Un ejemplo de una llamada a una función son argumento es `<identificador>.exec('func','')`

- ***<identificador>.update()***. Como se mencionó, esta función sirve para forzar una actualización de la escena principal posterior a recibir la información de la escena subordinada, por lo que suele incluirse justo después del <*identificador*>*.set* o <*identificador*>*.exec* para que el espacio receptor de información se actualice inmediatamente después de hacer cambios en sus variables. Esta función no involucra argumento alguno dentro de sus paréntesis.

Se usa *parent* como identificador cuando se comunica información del espacio HTMLIFrame subordinado al principal. Por ejemplo, `parent.set('vr',var)`. Siempre se usa *parent* pues el espacio subordinado no sabe cuál es el espacio principal.

Se usa el nombre asignado al espacio HTMLIFrame como identificador cuando se comunica información del espacio principal a su subordinado. Ello pues la escena conte-

nedora sí sabe cuál es el identificador de su espacio subordinado. Si la escena principal tiene un HTMLIFrame subordinado con identificador de espacio *Hijo*, la instrucción será `Hijo.set('var_hijo',var_padre)` para comunicar el valor de la variable *var_padre* del espacio principal a la variable *var_hijo* en la escena subordinada.

Los ejercicios incluidos en esta documentación usan dicha funcionalidad. Puede abrir cualquiera para revisarla. Nótese que, para cualquier ejercicio, el interactivo principal es el *index.html*, que contiene a dos escenas subordinadas: *<Nombre interactivo>_Escena.html* y *<Nombre interactivo>_Texto.html* como espacios HTMLIFrame. En el algoritmo *INICIO* de *<Nombre interactivo>_Escena.html* puede notarse cómo se usa esta funcionalidad para transferir el nombre del interactivo, mediante la variable 'ttl', al interactivo principal o contenedor. Es así como el contenedor *index.html* muestra el nombre de la escena en su esquina superior derecha.

Aunque cualquier otro ejercicio usa esta funcionalidad, se incluye un ejercicio de construcción de una escena principal con un espacio subordinado. No obstante, éste es el único que no se despliega dentro de un contenedor. Ello con el fin de no confundir al lector. Por lo mismo, este ejercicio sólo cuenta con el archivo pdf con las instrucciones y un par de escenas, *Padre.html* y *Esp2.html*, donde la escena principal es la primera y la segunda es la subordinada. No hay una escena *index.html* que las contenga a ambas como para otros ejercicios. En el pdf ubicado en este vínculo se puede consultar las instrucciones para lograr el ejercicio. Los vínculos a los archivos generados por este ejercicio son, para el espacio principal, Padre.html; y para el subordinado, Esp2.html. Se recuerda que estos archivos también están disponibles en el archivo *DocumentacionDescartesJS.zip*, dentro de la carpeta *HTMLIFrame* a su vez dentro de la carpeta *Ejercicios*.

Es importante observar y resumir de este ejercicio que *set* permite pasar el valor de una variable de un espacio a otro bajo un nuevo nombre; *exec* permite mandar ejecutar una función en otro espacio, posiblemente heredándole a la función remota alguna variable del espacio que envía la información; y que *update* refresca el espacio remoto desde el que manda el comando. También es importante notar nuevamente que *parent* como identificador se reserva cuando un espacio subordinado intenta contactar al principal. Un principal pueden tener varios subordinados, y de intentar comunicarse con ellos, el identificador a usar es el del subordinado en su modalidad HTMLIFrame. Se recuerda al usuario que puede revisar el contenedor de cualquier otro ejercicio (el archivo *index.html*) para notar un uso útil y común de toda esta funcionalidad.

Además de las funciones de comunicación entre espacios, hay una función adicional asociada a espacios HTMLIFrame que permite recargar, en uno mismo de estos espacios, distinto contenido html.

- ***<identificador>.changeConf(ruta)***. Es una función que sirve para recargar **sólo el código html** de una escena subordinada. Se usa cuando se requiere que el contenido de un espacio HTMLIFrame pueda adoptar distintos contenidos HTML. Al usar esta función, se evita la recarga del intéprete de DescartesJS, situación deseable cuando se requieren transiciones rápidas entre distintos contenidos HTML asociados al

espacio HTMLIFrame. El contenido es cargado de un archivo *.html*, cuya ruta es el texto contenido en la variable *ruta*, argumento de la función.
Imagine que una escena principal *padre.html* tiene dos subordinadas *A.html* y otra *B.html*, ambas a la misma altura que la escena principal. *padre.html* tiene un espacio HTMLIFrame que, en su parámetro *archivo* contiene `A.html`. Así, por defecto, abre cargando la escena subordinada *A.html*. Si quisiera cambiarse el contenido de la escena subordinada por aquél de *B.html*, sería necesario ejecutar la función `changeConf('B.html')` (recordamos que es la ruta completa al archivo, pero como en nuestro ejemplo mental se encuentra a la misma altura que la escena principal, basta sólo indicar su nombre). Al ejecutarse la función, se recarga el espacio HTMLIFrame con el código del bloque <*ajs*>...</*ajs*> del archivo *B.html*.

Así, esta función *changeConf* permite que una escena principal cambie el contenido HTML de un espacio HTMLIFrame de forma activa, pudiendo cargarlo de archivos existentes. De tal forma que no es necesario agregar un espacio HTMLIFrame por cada archivo html. Un mismo espacio HTMLIFrame puede abrir todos los archivos html usados.

13.2.4. Funciones de controles de audio y video

Existen algunas funciones intrínsecas asociadas a controles de audio y video que comienzan con el nombre del identificador del control de audio o video. Por ejemplo, <*identificador del control*>.*play*() con la que se puede controlar la reproducción del archivo. Estas funciones son:

- **<identificador del control>.play()**: Cuando se llama a esta función, el archivo de audio o video asociado al identificador del control se reproduce desde el principio. Por ejemplo, si el identificador del control es *a2*, la función sería *a2.play()*.
- **<identificador del control>.stop()**: Cuando se llama a esta función, se detiene la reproducción del archivo de audio o video asociado al control. Por ejemplo, si el identificador del control es *a2*, la función sería *a2.stop()*.
- **<identificador del control>.pause()**: Cuando se llama a esta función, se pausa la reproducción del archivo de audio o video asociado al control. Si eventualmente se vuelve a reproducir, continuará donde se quedó. Por ejemplo, si el identificador del control es *a2*, la función sería *a2.pause()*.
- **<identificador del control>.currentTime(t)**: Ésta es una función que sí maneja argumentos. Su argumento es numérico (en el ejemplo, *t*), y corresponde al valor en segundos del tiempo en el que se colocará un archivo de audio o video para su subsecuente reproducción. Por ejemplo, *a2.currentTime(5)* hará que el archivo asociado al control *a2* se coloque en el segundo 5, y no en el principio. Si posteriormente se implementa una instrucción *a2.play()*, el archivo asociado al control se reproducirá desde el tiempo elegido.

Se puede revisar un ejercicio relacionado con esta funcionalidad en este vínculo.

13.2.5. Funciones de controles numéricos tipo menú

Existe una función asociada a este tipo de controles.

- **<identificador del menú>.setOptions(str)**: Esta función permite asignar las opciones que tendrá el menú mediante una cadena de texto, que en este ejemplo es *str*. Recordamos que este tipo de controles cuenta con un parámetro *opciones* en el que se introducen las opciones del menú separadas por comas. En ocasiones, es deseable poder asignar estas opciones a partir de una cadena de texto.
  Como ejemplo, considere el caso en que el usuario llena unos campos de texto (de tipo *sólo texto*), *t1* y *t2*, y se desea que las opciones de un menú *m1* sean lo que el usuario introdujo en los respectivos campos. Entonces puede asignarse a ambos campos de texto la acción *calcular*, y como parámetro de cálculo:
  ```
  str=t1+','+t2
  m1.setOptions(str)
  ```
  La primera asignación hace que la cadena *str* esté compuesta por una concatenación del texto incluido en el primer campo de texto (*t1*), seguido de una coma (,), seguida del texto del segundo campo de texto (*t2*). Note que la cadena de texto corresponde a como se introduciría en el parámetro *opciones* del menú. La segunda línea comienza con el identificador del menú (*m1*) seguido de `.setOptions()`, y el argumento de la función es la cadena *str*. Al ejecutar esta función, las opciones del menú *m1* cambian automáticamente a lo que contengan los campos *t1* y *t2*. Esta función es particularmente útil pues permite hacer de los menús un tipo de control más dinámico.

13.2.6. Guardado de imágenes de espacios

Sólo hay una función asociada al guardado de la disposición gráfica de un espacio como imagen.

- **<identificador del espacio>.download_image(str)**. Cuando se ejecuta esta función, se lanza un diálogo del sistema operativo para guardar una imagen. El nombre sugerido para la imagen es el incluido en el argumento como una cadena de texto (*str* en el ejemplo).
  En ocasiones, puede ser útil guardar la disposición gráfica de un espacio como imagen, sobre todo al generar material gráfico usando *DescartesJS*. Esta función puede asociarse, por ejemplo, como la instrucción llamada por un determinado botón. La instrucción podría ser, por ejemplo `E1.download_image('Imagen_E1')`. Una vez que se logra una disposición a guardar, al oprimir este botón se lanza el diálogo, y el nombre sugerido para el archivo sería *Imagen_E1.png*.
  El *png* generado incluye sólo los elementos gráficos no considerados parte del fondo del espacio. En este sentido, los ejes y los elementos de la red del espacio no se

incluye en la imagen. También se excluyen los elementos gráficos declarados como parte del fondo. Igualmente, se excluye cualquier tipo de control de la imagen.

13.2.7. Transferencia de información entre matrices y vectores con variables de texto

Primero veamos las funciones involucradas para transferir información entre matrices y variables de texto. Posteriormente, y dado que la información aquí incluida es complicada, se incluye un ejemplo con el fin de aclarar dudas.

- **_MatrixToStr_('<identificador de la matriz>')**. Su nombre viene de *matrix to string*, o *matriz a cadena*. Su argumento es el identificador de la matriz flanqueado por comillas sencillas.

  Esta función devuelve, en forma de cadena de texto, el contenido de la matriz, por lo que suele asignarse la función a una variable.

  El texto devuelto usa formato de etiquetas. El primer renglón consiste en el identificador de la matriz dentro de <>. Por ejemplo, para una matriz *M* se tendría `<M>`. Después viene un salto de línea. Luego vienen los valores de las entradas (o columnas) de la primera fila, cada una separada por el símbolo especial ¦. Y así van varios renglones, cada uno representando una fila de la matriz, y separado de otro por saltos de línea. Después de la última fila, y también en un nuevo renglón, se cierra la etiqueta con el identificador del a matriz dentro de </ >. Por ejemplo, para la matriz *M*, se tendría `</M>`.

  Por ejemplo, `cad=_MatrixToStr_('M')` volcará el contenido de la matriz *M* en la variable *cad*. Si *M* es una matriz de 2×2, y sus valores son *M[0,0]=1*, *M[0,1]=2*, *M[1,0]=3*, y *M[1,1]=4*, la variable *cad* guarda:

```
<M>
1 ¦ 2
3 ¦ 4
</M>
```

- **_StrToMatrix_(cadena,'<identificador de la matriz>')**. Su nombre viene de *string to matrix*, o *cadena a matriz*. Su primer argumento es una variable que contiene una cadena de texto dentro de la cual vienen los datos de la matriz, y su segundo argumento es el identificador de la matriz en la que se volcarán esos datos, flanqueado por comillas sencillas.

  Esta función hace lo opuesto que *_MatrixToStr_()*. Esto es, vuelca el contenido de una cadena en una matriz. No devuelve un valor como tal, sino que usa sus dos argumentos para transferir la información.

  Por ejemplo, supongamos que queremos volcar la información de una variable *cdn* a una matriz *M3*. Supongamos que la variable *cdn* tiene en su interior:

```
<M>
1 ¦ 2
3 ¦ 4
</M>
<M3>
4 ¦ 3
2 ¦ 1
</M3>
```

Note que *cdn* trae la información de la matriz *M*, y de otra *M3*. Cuando se usa la función `_StrToMatrix_(cdn,'M3')`, se busca dentro del texto en *cdn* aquel correspondiente a una matriz *M3* (pues el segundo argumento de la función nos indica qué matriz vamos a usar), y ese se volcará en la matriz *M3*. Así, la entradas de dicha matriz quedarán *M3[0,0]=4*, *M3[0,1]=3*, *M3[1,0]=2*, y *M3[1,1]=1*.

La matriz *M3* **puede o no estar declarada** previamente en el selector *Definiciones* de la escena. De estar declarada, los datos de la cadena de texto se vuelcan sobre ella, sobreescribiendo cualquier contenido que pudiera tener. De lo contrario, se genera internamente la matriz con dicha información. Igualmente, las variables asociadas a la matriz que indican su número de columnas y filas (*<id de la matriz>.columnas* e *<id de la matriz>.filas*, respectivamente) se ajustan a la información de la cadena, sobreescribiendo cualquier valor dado sobre la matriz en el selector *Definiciones*.

Hagamos un ejercicio para entender mejor cómo se manejan estas dos funciones.

Este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Matriz Cadena. El documento del interactivo como tal se encuentra en este vínculo.

En ese ejercicio primero se muestra cómo el contenido de una matriz *M*, con entradas 1, 2, 3, y 4, se puede transferir a una variable *cad*. Después, se re-escribe el contenido de la matriz a que todas sus entradas sean cero. Luego, se le vuelven otorgar sus valores iniciales, pero ahora leyéndolos de la variable *cad* donde se guardaron inicialmente. Pero es importante que, cuando se retoman los datos de *cad*, en la función *_StrToMatrix_()* coincida el nombre de la matriz en la que se volcará con aquél dentro de la variable *cad*. Como se menciona en las instrucciones, la variable *cad* puede tener información más allá de la matriz, pero la forma en que encuentra los valores deseados es por la correspondencia de nombres de las matrices en las dos instancias. Finalmente, en el último paso se muestra que es posible volcar datos de una cadena en una matriz, aún si ésta no ha sido defiinida, como fue el caso del volcado de datos de la variable *cad2* en la matriz *M2*. Esta última matriz, al no haber sido definida, se crea al vuelo internamente en *DescartesJS*.

La información de vectores también se puede guardar a cadena de texto y viceversa de forma similar a como se hace con las matrices.

- **_VectorToStr_('<identificador del vector>')**. Su nombre viene de *vector to string*, o *vector a cadena*. Su argumento es el identificador del vector flanqueado por comillas sencillas.

  La función devuelve, en forma de cadena de texto, el contenido de vector, y suele asignarse por lo mismo a una variable. Por ejemplo `vc=_VectorToStr_('Vc')` asignará a la variable *vc* el contenido del vector *Vc*.

  El formato es, también, similar al de las matrices. En un primer renglón viene el identificador del vector flanqueado por <, y >. Por ejemplo, `<Vc>`. Luego un salto de línea y la primera entrada del vector. Y así cada entrada del vector en una línea distinta. Al final viene una última línea de cierre con el identificador del vector. Por ejemplo, `</Vc>`.

- **_StrToVector_(cadena,'<identificador del vector>')**. Su nombre viene de *string to vector*, o *cadena a vector*.

  Muy similar a con matrices, esta función vuelca el contenido de una cadena en un vector. Por ejemplo `_StrToVector_(vc,'Vc')` volcará en un vector *Vc* el contenido de la variable *vc*. El formato de la cadena de texto es el mismo ya visto que se adopta cuando se vuelca información de un vector en una cadena de texto.

  Cabe decir que, al igual que para las matrices, la cadena de texto puede contener información de diversos elementos (matrices y vectores distintos). La función volcará sólo la información referente al vector indicado en su argumento. Y si el vector no ha sido definido previamente en el *Definiciones*, se definirá internamente al vuelo, al igual que ocurre con las matrices. Si existía previamente, se sobreescribirá la información de sus entradas, así como el número total de entradas del mismo.

Observe que el formato de la cadena de texto que guarda la información de una matriz es muy similar al de una que guarda información de un vector. La diferencia es que para el vector cada renglón es una entrada del mismo, mientras que para la matriz hay que definir, para cada línea, el número de columnas usando el símbolo ¦ entre cada elemento de la línea.

El manejo entre matrices y cadenas, así como de vectores y cadenas, visto hasta ahora es interno a la escena. En ocasiones puede ser necesario guardar la información en archivos aparte. Ello se aborda en el apartado sobre la funcionalidad de guardado y recuperado de datos generados.

13.3. Condicionales y operadores booleanos

Los operadores booleanos permiten hacer comparaciones entre valores de elementos de Descartes y devolver el valor 1 o 0 dependiendo del resultado de la comparación. Los elementos que se comparan pueden ser constantes, variables, e inclusive valores de retorno de funciones.

Cuando el resultado de comparación es verdadero, se arroja un valor de 1, y cuando es falso, se arroja un valor de 0. A continuación se muestran los operadores booleanos en Descartes.

13.3.1. Los operadores booleanos y su uso en condicionales

A continuación se presenta una lista de cuáles son los operadores, así como la forma en que se usan para comparar valores.

- **==**: El operador == (dos signos de igualdad uno tras otro) compara el elemento antes del operador con aquél después. Si los valores de estos elementos son iguales se considera que el resultado de la comparación es verdadero (que arroja un valor de 1), y de lo contrario será falso (que arroja un valor de 0). Por ejemplo, *(2==2)* arrojaría un valor de 1 al ser una condición verdadera, pero *(2==1)* arrojaría un 0 al ser falsa. Aunque Descartes tolera hacer este tipo de comparación con un solo signo de igualdad, se recomienda siempre usar el doble signo para evitar confusiones. Recuerde que un solo signo de igualdad corresponde a una asignación.

- **!=**: El operador != (una admiración de cierre seguida por un signo igual) compara el elemento antes del operador con aquél después. Si los valores de estos elementos son distintos se considera que el resultado de la comparación es verdadero (que arroja un valor de 1), y de lo contrario será falso (que arroja un valor de 0). Por ejemplo, *(2!=2)* arrojaría un valor de 0 al ser una condición falsa, pero *(2!=1)* arrojaría un 1 al ser verdadera.

- **<**: El operador < compara el elemento antes del operador con aquél después. Si el valor de aquél antes del operador es menor que aquél después, se considera que el resultado de la comparación es verdadero (que arroja un valor de 1), y de lo contrario será falso (que arroja un valor de 0). Por ejemplo, *(2<1)* arrojaría un valor de 0 al ser una condición falsa, pero *(1<2)* arrojaría un 1 al ser verdadera.

- **>**: El operador > es parecido al <. Si el valor del elemento antes del operador es mayor que aquél después, se considera que el resultado de la comparación es verdadero (que arroja un valor de 1), y de lo contrario será falso (que arroja un valor de 0). Por ejemplo, *(2>1)* arrojaría un valor de 1 al ser una condición verdadera, pero *(1>2)* arrojaría un 0 al ser falsa.

- **<=**: El operador <= (un signo de *menor que* seguido de un igual) se comporta como el <, pero permite también la igualdad. Si el valor de aquél antes del operador es menor **o igual** que aquél después, se considera que el resultado de la comparación es verdadero (que arroja un valor de 1), y de lo contrario será falso (que arroja un valor de 0). Por ejemplo, *(2<=2)* arrojaría un valor de 1 al ser una condición verdadera, pero *(3<=2)* arrojaría un 0 al ser falsa.

- **>=**: El operador >= (un signo de *mayor que* seguido de un igual) se comporta como el >, pero permite también la igualdad. Si el valor de aquél antes del operador es mayor **o igual** que aquél después, se considera que el resultado de la comparación es verdadero (que arroja un valor de 1), y de lo contrario será falso (que arroja un valor de 0). Por ejemplo, *(3>=3)* arrojaría un valor de 1 al ser una condición verdadera, pero *(3>=4)* arrojaría un 0 al ser falsa.

- **!**: El operador ! (un signo de admiración de cierre) se aplica antes de alguna condición o valor. Lo que hace es negar el resultado de dicha comparación. Es decir, si la comparación era verdadera (dando un valor de 1), la vuelve 0 (o falsa), pero si era falsa (dando un valor de 0), la vuelve 1 (o verdadera). Por ejemplo, *!(3>4)* devolvería un 1 (verdadero), mientras que *!(3<4)* devolvería 0 (falso). Otro ejemplo más directo es *!1*, que devolvería 0, o *!0*, que devolvería 1.

- **|**: El operador | (o *pipe*) consiste en una barra vertical que se puede introducir normalmente oprimiendo la tecla a la izquierda del número 1 en el teclado. Separa a dos condiciones (o valores de 0 ó 1) y, si al menos una de las dos es verdadera, de estas dos condiciones se arroja un resultado de 1 (o verdadero). Por ejemplo, *(2>3)|(3>1)* arrojaría un 1 (o verdadero) pues la condición a la derecha es verdadera. Igualmente, *(2>3)|1* arrojaría 1, pero *(2>3)|(3>4)* arrojaría 0. A este operador se le conoce como *or*.

- **&**: El operador & (o *ampersand*) separa a dos condiciones (o valores de 0 ó 1) y, si al menos una de las dos es falsa, de estas dos condiciones se arroja un resultado de 0 (o falso). Por ejemplo, *(2>3)&(3>1)* arrojaría un 0 (o falso) pues la condición a la izquierda es falsa. Igualmente, *(2>3)&1* arrojaría 0, pero *(2<3)&(3<4)* arrojaría 1 al ser ambas condiciones verdaderas. A este operador se le conoce como *and*.

Las condicionales usan condiciones para asignar valores a una variable. Digamos que se quiere asignar a la variable *a* el valor de 10 si el valor de la variable *u* es mayor o igual que 50, pero de lo contrario se le desea asignar 20.5. Entonces se debe introducir el siguiente texto:

```
a=(u>=50)?10:20.5
```

Note que la instrucción tiene primero a la variable *a* a la que se le asignará el valor. Viene seguida, como de costumbre, del signo = de asignación, pero inmediatamente después de éste hay un paréntesis dentro del cual está la condición deseada. Después del paréntesis viene un signo de interrogación de cierre. Justo después de éste viene el valor a asignar si la condición es verdadera. Después viene un signo de dos puntos (:). El valor de asignación si la condición no es verdadera viene después de este signo. Considere como otro ejemplo la siguiente asignación:

```
a=((c<2)|(d>1))?j:sqrt(k)
```

Para interpretar esta asignación, considere que las variables *c* y *d* tienen valores 1 y 0, respectivamente. *c<2* devuelve un valor de 1 (pues la condición es verdadera), pero *d>1* devuelve un 0 (pues la condición es falsa. Después se comparan los resultados de ambas

condiciones. Es decir, *1|0*. Como al menos uno de los resultados de las condiciones fue 1 (o verdadero), el resultado del operador | es 1. Así, se asigna el valor después de ? a la variable *a*, que es el valor que tenga la variable *j*. Si las variables *c* y *d* tuvieran valores 3 y 0, respectivamente, entonces a *a* se le asignaría el valor de la raíz cuadrada del valor de la variable *k*.

Hagamos un ejercicio para practicar estos conceptos. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en Condicionales. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

En el texto mostrado en este ejercicio se puede notar el comportamiento de las condiciones unitarias (donde se compara el valor de una variable con el de otra). También se nota el comportamiento de un par de condiciones en la asignación a la variable *mostrarrojo*. Asimismo, también se vio el comportamiento de las condicionales, en las que se usa una o varias condiciones para determinar qué valor asignar a una variable dada. Por último, se puso en práctica el uso del campo *dibujar si* cuando se quiere que un gráfico, control, etc. se muestre sólo si se cumple una condición dada.

Un detalle importante que se puede notar en el ejercicio previo es que, cuando se decide si algún elemento del interactivo ha de mostrarse o no mediante una variable, si esta variable sólo puede adoptar valores 0 ó 1, entonces la variable como tal puede introducirse como elemento único en el campo *dibujar si* del elemento. Por ejemplo, suponga que un gráfico ha de pintarse sólo si la variable *a* vale 1. Pero dicha variable sólo adopta valores entre 0 y 1. Entonces, no es necesario introducir explícitamente `a==1` en el campo *dibujar si* del gráfico; basta solamente introducir `a`. Por el contrario, suponga que la variable *a* controla si se muestra un texto de retro alimentación. Se asigna el valor 0 para cuando no se quiere mostrar retroalimentación, 1 para cuando la retroalimentación es *Correcto* y 2 para cuando es *Incorrecto*. En este caso, *a* tiene más valores que 0 ó 1. En este caso, sí conviene hacer explícita la comparación en el campo *dibujar si* del gráfico.

Se pueden comparar varias condiciones a la vez usando los operadores & y |. En este caso es importante hacer un uso adecuado de los paréntesis que flanquean las condiciones para que los operadores tengan el funcionamiento deseado.

Para este ejercicio se usaron colores en varios gráficos. Puede consultar el apartado sobre la herramienta de control de colores para una descripción más profunda de este funcionamiento.

13.3.2. Uso de variables mudas para condicionar el llamado de funciones

Algunas veces se necesita usar condicionales para determinar si se llama a una función o no. No hay una forma explícita de decirle a Descartes que ejecute una función si se cumple una condición y que de lo contrario no la ejecute. Esto ocurre frecuentemente cuando se desea ejecutar de forma condicionada funciones.

Para resolver este problema, se pueden usar *variables mudas*. Estas variables no sirven realmente para nada más que para permitir asociar la condicional a una función. Por ejemplo, considere un ejemplo en que se desea asignar una serie de valores iniciales a varias variables mediante una función, pero que dicha función no devuelve un valor (el campo después del signo = de la función está vacío). Una posible asignación sería la siguiente:

```
bla=(z>3)?GeneraValores():bla
```

Aquí se desea que si la variable *z* es mayor que 3, se ejecute una función *GeneraValores()*. Esta función no tiene argumentos (por eso su paréntesis se encuentran vacío). Pero además es una función tipo algoritmo que le asigna valores a otras variables. Esta instrucción le devuelve un valor a la variable *bla* que no está definido. Pero eso no importa, pues lo que nos interesa solamente es que se ejecute la función, con lo que las instrucciones dentro del algoritmo de la misma se ejecutarán. Si *z* es menor o igual a 3, el valor que se le asigna a la variable *bla* es el mismo valor que tenía, y en este caso la función no se ejecuta. Así, la variable *bla* es muda pues no se usa explícitamente en el programa. Sólo se usa como un medio para llamar a una función de forma condicional.

13.4. Operadores generales

Los operadores generales consisten en los signos matemáticos que se usan comúnmente para asignar valores, sumar, restar, multiplicar, dividir y agrupar, entre otros que tienen funciones más específicas. Estos operadores no sólo sirven para cuestiones matemáticas. Por ejemplo, pueden tener también la función de concatenar textos.

Los operadores pueden trabajar sobre constantes, variables, retornos de funciones y, como se mencionó, hasta textos.

- =: El operador *igual* sirve para hacer asignaciones. Las asignaciones toman el valor de una constante, o variable o expresión colocadas a la derecha del signo y se lo asignan a la variable a la izquierda del signo. Estas asignaciones pueden pasar tanto valores numéricos como cadenas de caracteres. Por ejemplo, una asignación *a=2* toma el valor de 2 a la derecha de la igualdad y se lo asigna a la variable *a*. La asignación *b=2*3* toma el valor de la expresión a la derecha del igual, que es 6, y se lo asigna a la variable *b*. Una asignación *c='hola'* toma la cadena de caracteres a la derecha del signo y se lo asigna a la variable *c*, de tal forma que si se imprime su valor, se imprimirá el texto *hola*.
  Es importante diferenciar entre el operador = y el ==. El operador == se usa para comparar valores de expresiones, y su uso se detalla en el apartado sobre condicionales y operadores booleanos.

- +: Éste es el operador *suma*, que añade los valores de variables. También sirve para concatenar cadenas de texto. Por ejemplo, *a=2+3* hace que la variable *a* adopte el valor de 5. Pero *b='te'+'saludo'* hará que la variable *b* contenga ambos textos (cada uno

entre comillas sencillas) terminen concatenados, de tal suerte que si se imprime, se observará el texto *te saludo*.

- *****: El operador *producto* multiplica los valores de las expresiones a sus extremos. Por ejemplo, *a=2*pi* hará que la variable *a* tenga el valor de 6.283...

- **/**: El operador *división* devuelve el cociente entre dos números. Por ejemplo, si se tiene una variable *a* que vale 3 y una *b* que vale 1.5, la asignación *c=a/b* hará que la variable *c* termine con el valor 2.

- **^**: El operador *potencia* toma el elemento a su izquierda y lo eleva una potencia que es su elemento a la derecha. Por ejemplo, *q=2^3* hará que *q* termine con un valor de 8. Cabe decir que este operador se puede usar para extraer raíces más allá de la cuadrada. Por ejemplo, *r=3^(1/3)* corresponde a extraer la raíz cúbica del número 3 y asignar dicho valor a la variable *r*.

  Cabe notar un detalle al usar el operador ^, sobre todo en el ámbito de graficación de funciones de tipo potencia. Sea `y1=x^0.6` y `y2=x^(3/5)`. Matemáticamente, ambas funciones corresponden a $y = x^{3/5}$. Sin embargo, si se graficaran dichas ecuaciones, la gráfica sólo mostraría la parte correspondiente a *x>0*. Ello se debe a que una función del tipo $y = x^p$, con p no entero, en DescartesJS sólo está definida para *x>0*. Los exponentes 0.6 y (*3/5*) se toman como reales no enteros y no se traza la parte de la función correspondiente a cuando el argumento es negativo.

  Así, al graficar una función del tipo $y = x^{\frac{p}{q}}$, con q impar positivo, para hacer explícito que el denominador q ha de tomarse como la raíz q-ésima, es necesario introducir, según nuestro ejemplo, `y3=(x^3)^(1/5)`. Esta función estará definida para x en los reales.

  En resumen, para indicar que una función es la raíz q-ésima es preciso introducir `^(1/q)` (debe llevar estar elevada a la **uno** entre el número correspondiente a la raíz que se extrae). Con esto, la función estará definida para todo real, y de graficarse, su gráfica se mostrará completa.

- **%**: Éste es el operador *módulo*. Devuelve el residuo de una división. Por ejemplo, *a=23%7* asignará el valor de 2 a la variable *a*, ya que 7 cabe 3 veces en 23 y sobra un residuo de 2.

- **()**: El par de paréntesis, como operadores, se usan para agrupar expresiones en asignaciones o condiciones en una condicional. Mediante paréntesis es posible indicarle a Descartes que ignore el orden de operaciones predeterminado y que haga algunas operaciones antes que otras. Para más información sobre el orden de operaciones, consulte el apartado sobre el orden y jerarquía de operaciones matemáticas.

Es importante tener en mente los errores que se pueden cometer al hacer asignaciones y operaciones en general.

- *Infinity*: En ocasiones, se puede hacer una división donde el divisor eventualmente vale cero. Usualmente, este tipo de errores se reporta si la variable involucrada ha de ser impresa en el editor. En lugar de mostrarse un valor, se imprime el texto *Infinity*.
- *NaN*: Este error, que también se muestra en el editor sustituyendo el valor de una variable cuando ésta se intenta imprimir, responde a alguna operación no válida. Por ejemplo, una raíz cuadrada con argumento negativo. O cuando se trata de operar variables con cadenas de texto mediante los operadores distintos a + (que sirve para concatenar).
- Errores en la consola: Estos errores aparecen explícitamente en la consola del programa, por lo que es buena práctica mantenerla abierta cuando se está trabajando particularmente con muchas asignaciones matemáticas. Los errores explícitamente indican la naturaleza del problema. Por ejemplo, "*faltan paréntesis por cerrar*" (para la falta de consistencia de apertura / cierre de paréntesis en fórmulas matemáticas), o bien "*condicional incompleta*" en asignaciones condicionales. La ubicación del error no siempre se indica explícitamente, por lo que se recomienda al usuario aplicar los cambios de forma constante (no esperar a tener muchos cambios para aplicar). De esta forma, los errores pueden ser ubicados rápidamente. Igualmente, se recuerda que los errores desplegados no necesariamente corresponden a la última vez que se aplicaron los cambios, por lo que es recomendable cerrar y abrir constantemente la consola (para vaciar los errores y que aparezca limpia al reabrir) cuando se intenta depurar un error.

13.5. Orden y jerarquía de operaciones matemáticas

Al igual que cualquier otro lenguaje de programación, o inclusive cualquier lenguaje de calculadora, Descartes sigue un orden predeterminado de operaciones. Cuando hay varias operaciones en una instrucción, Descartes primero atiende las potencias, luego las multiplicaciones y divisiones, y finalmente las sumas y restas, todo en orden de izquierda a derecha. Considere la siguiente expresión.

```
a=3+2*3+4/5^2
```

Primero se aplican las potencias, por lo que la expresión se reduce a

```
a=3+2*3+4/25
```

Posteriormente se aplican los productos y divisiones, de izquierda a derecha, con lo que queda primero

`a=3+6+4/25`, y luego `a=3+6+0.16`

Después se aplican las suma y restas, de derecha a izquierda, con lo que queda primero

`a=9+0.16`, y luego `a=9.16`

Así pues, muchas veces es necesario indicarle a Descartes que haga las operaciones de forma distinta. Esto se logra usando paréntesis para agrupar las operaciones. Por ejemplo,

si se desea extraer la raíz quinta de 32 ($\sqrt[5]{32}$), ella se obtiene usando $32^{\frac{1}{5}}$. Un error común en este caso suele ser la expresión

`32^1/5`, en lugar de `32^(1/5)`

La primera expresión primero hace la potencia, con lo que se obtiene

`32/5`, que finalmente da `6.4`

Así, es preciso usar el paréntesis para indicar el orden de operaciones. Si se usa

```
32^(1/5)
```

Descartes primero hará lo que está en el paréntesis que da 0.2 y luego elevará 32 al exponente 0.2, con lo que se obtiene el valor de 2 deseado.

Así, las reglas de orden de operaciones cambian al incluir paréntesis. Lo primero que se resuelve en la expresión son los paréntesis, y si dentro de ellos hay operaciones, éstas se resuelven ahí dentro de la manera convencional. Una vez que se resolvió un paréntesis y se tiene un valor que representa a toda esa expresión, se aplican las siguientes operaciones en el orden tradicional.

Es importante notar que se pueden anidar paréntesis dentro de otros. Por ejemplo, considere la siguiente expresión

```
2*(1+3/(2+1))
```

Descartes primero se encuentra el paréntesis externo. Dentro de él busca si hay más paréntesis y se encuentra el interno, que es el primero que resuelvo, de donde se obtiene

```
2*(1+3/3)
```

De ahí, sigue operando el interior del paréntesis externo. Busca primero si hay productos o divisiones y las resuelve. obteniéndose

```
2*(1+1)
```

De ahí pasa a hacer las sumas y restas en ese paréntesis, de donde se obtiene

```
2*2
```

Y de ahí finalmente se tiene el valor de 4.

Así, los paréntesis permiten una mayor flexibilidad al indicarle a Descartes cómo se desea hacer las operaciones. Muchas veces se cometen errores pues a un paréntesis de apertura no corresponde uno de cerradura o viceversa. Una práctica útil para evitar cometer este tipo de errores es que cada vez que se abre un paréntesis, primero se cierre y después se introduzca su contenido. El tipo de errores mencionados suele reportarse en la consola Descartes ya descrita, como se muestra en la Figura 13.1.

Note en dicha figura que el error resulta de un gráfico tipo *texto* que desea imprimir el valor de `(3+2*(1/3)`, en el cual falta un paréntesis de cierre. Si se incluyera dicho paréntesis: `(3+2*(1/3))`, el valor de dicha expresión sería 3.6666..., que finalmente sí imprime

Descartes en la parte superior izquierda de la escena. Descartes en ocasiones puede *intuir* lo que quiso decir el usuario, pero sólo hasta cierto grado. Este tipo de mensajes sólo se observan en la consola, y es bueno tenerla abierta sobre todo cuando se está haciendo programación complicada.

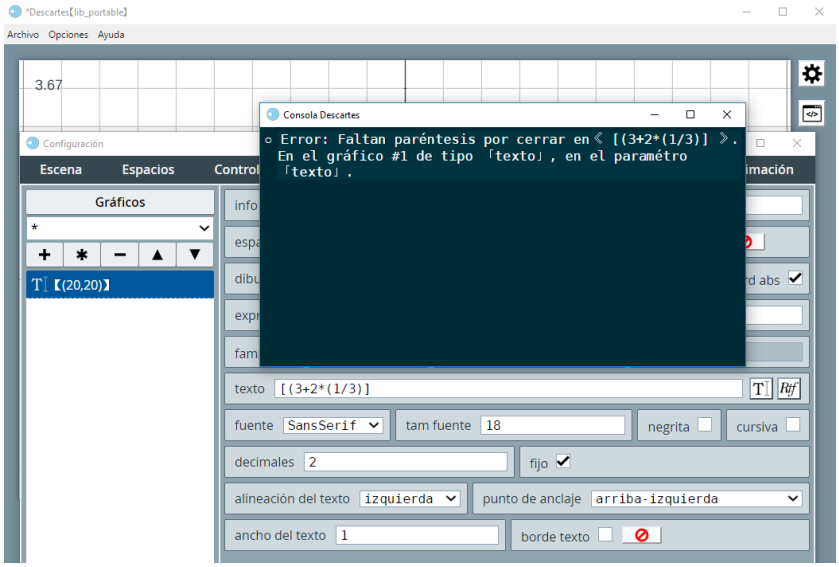


Figura 13.1: Mensaje de error al querer aplicar una expresión sin concordancia de cierre de paréntesis.

13.6. Orden de actualizaciones al iniciar una escena y al interactuar con ella

En este apartado se aborda el orden de ejecuciones y actualizaciones que hace *DescartesJS* al lanzar una escena de forma inicial, o bien cuando ocurren cambios o interacciones subsecuentes dentro de la misma. En general, el usuario no necesitará tener esto en mente todo el tiempo, pero algunas veces permite explicar algunas situaciones. Como ejemplo, considere una escena de *DescartesJS* que contiene un pulsador con identificador *n1*. ¿Qué pasa cuando se lanza esta escena si en el valor inicial del control pulsador se le asigna, digamos, 3, pero en el algoritmo *INICIO* se hace la asignación `n1=7`? ¿Y si ademas incluimos un evento que le asigna otro número? ¿Cuál será el valor asociado a su variable al inicio de la escena? ¿Y cómo cambiará después del inicio, por ejemplo cuando el usuario interactúa con la escena, o a lo largo digamos de una animación?

Estas preguntas justo se pueden responder entendiendo el orden de ejecución predeterminado en *DescartesJS*.

13.6.1. Actualizaciones al iniciar la escena

Al cargar por primera vez la escena se ejecuta lo siguiente en orden:

1. **Construcción interna de objetos**. En un primer momento se construyen todos los objetos que conforman la escena con sus valores por omisión.
2. **Fase de inicio**. Ya construidos los objetos se pasa a este segundo momento. En esta fase se actualizan en orden:
   a) Las definiciones (vectores, matrices, etc.).
   b) Los algoritmos del programa.
   c) Los controles.
   d) Los espacios. Esta actualización incluye el cálculo de variables asociadas a los espacios y la actualización de los elementos gráficos contenidos en ellos.
3. **Fase de actualización de todos los valores**. Esta fase se incluye con el fin de asegurar que los objetos echen mano de valores actualizados en lugar de los predefinidos (valores por defecto), y es importante para la ejecución de los eventos.

Cada elemento de este listado puede tener varias ejecuciones. Por ejemplo, en las definiciones pueden haber varias asignaciones de, digamos, vectores y matrices. Éstas se realizan en el orden en que aparecen en el listado. Y lo mismo ocurre para los demás elementos.

Esto define cómo se muestra la escena de inicio. Y corresponde al resultado de pulsar el botón *Aplicar* o *Aceptar* en el editor de configuraciones. Sin embargo, también es útil saber qué pasa una vez iniciada la escena cuando el usuario interactúa con ella, o ésta sufre cambios resultantes de, por ejemplo, una animación.

13.6.2. Actualizaciones subsecuentes de la escena

Cada interacción subsecuente del usuario, o bien cada paso de una animación (de haberla) resulta en un ciclo de actualización con el siguiente orden:

1. **Actualización de auxiliares**. Nuevamente, primero se actualizan los elementos auxiliares de las *Definiciones*.
2. **Actualización de controles**. Una actualización inicial de los *Controles*.
3. **Actualización de eventos**. La actualización de los eventos de *Programa*.
4. **Actualización de controles**. Una segunda actualización de los *Controles*, cuyo objeto es asegurar que los controles queden todos actualizados correctamente.
5. **Actualización de espacios**. Actualización de los elementos de los distintos *Espacios*.

La segunda actualización de controles responde a situaciones en que un control al final de la lista de actualización bien puede actuar modificando a uno que está antes de él. En dicho caso, la modificación en cuestión sólo se implementa realizando una segunda actualización.

A manera de ejemplo, considere un par de controles numéricos con identificadores *a* y *b*. *a* tiene un valor inicial de 0 y un valor mínimo correspondiente al valor actual de *b* (es decir, el parámetro *min* de *a* tiene `b` dentro de él). Si se lanza la escena, se tiene a *a* con un valor de 0, y posteriormente se manipula *b* tal que su valor sea mayor que cero, se observa que el valor de *a* se ajusta al valor de *b*, ya que se forza que respete su condición de que mínimo vale lo que *b*. Cada que se manipula *b*, internamente *a* se actualiza gracias a la segunda actualización de controles. De no existir ésta, simplemente se cambiaría el valor de *b* y *a* ya no se volvería a visitar, por lo que no adoptaría el valor de *b*.

Como se mencionó, lo visto en este apartado no es indispensable para crear una escena en *DescartesJS*, pero puede ayudar al usuario a comprender por qué una variable adopta tal o cual valor si en otro lado se le da un valor distinto. Este tipo de comportamientos "*raros*" muchas veces responden al orden de actualización y ejecución de elementos de *DescartesJS*.

14

Guardado y recuperado de datos generados

Este capítulo involucra el guardado y recuperado de archivos con información generada por en escenas de *DescartesJS*. De forma general, este manejo de datos se hace a través de matrices, por lo que en este capítulo también se usará mucho este tipo de definición.

En ocasiones es necesario guardar una gran cantidad de datos de matrices a una variable tipo texto. Cuando esto se hace, *DescartesJS* separa los datos de las entradas con símbolos especiales para que se puedan leer fácilmente después. Eventualmente, es posible guardar los datos de la variable a un archivo de texto. Y, luego, se pueden volcar los datos de este archivo de nuevo a una matriz.

Esto resulta útil, por ejemplo, cuando la generación de los datos de una matriz requiere de mucho procesamiento y puede tomarle a *DescartesJS* mucho tiempo en hacer. En estos casos, como no se quiere que quien usa la escena tenga que esperar mucho tiempo, puede ser preferible contar con ejemplos pre-generados de las matrices que sólo muestran el resultado final.

14.1. Guardado y recuperado de información en archivos

DescartesJS puede guardar los datos de una cadena a un archivo de texto. También es posible cargar los datos de texto de un archivo a una variable como una cadena de texto.

- **_Save_**(). Sirve para guardar algún contenido en archivo.

  Su primer argumento es el nombre del archivo de texto en que se guardará la información flanqueado por comillas sencillas. Es preciso incluir la extensión del archivo a usar. Su segundo argumento es el contenido a guardar. Por ejemplo, `_Save_('Arch.txt',cdn)` hará que, cuando se ejecuta, se lance un diálogo de guardado de *DescartesJS*. El nombre propuesto por defecto en este diálogo será *Arch.txt*. Y lo que guarda es el texto contenido en la variable *cdn*. Si se usa una variable, esta debe contener una cadena de texto. O bien, en lugar de la variable, se puede usar directamente un texto flanqueado por comillas sencillas (a manera de ejemplo de esto, `_Save_('Arch.txt','hola')`).

  Para que esta función sirva a la perfección, conviene tener la escena guardada en alguna ubicación. Y luego incluir en el primer argumento la ruta al archivo. Así, la forma más correcta de usarla sería, si el archivo se guarda a nivel de la escena `_Save_('./Arch.txt',cdn)`. La ruta se puede ignorar si la escena ha accesado a dicha ubicación previamente, pero de lo contrario, conviene incluirla.

- **_Open_()**. Sirve para abrir un archivo elegido por el el usuario de la escena. No puede abrir archivo pre-elegidos dentro de la escena, sino que solamente permite lanzar un diálogo mediante el cual el usuario puede escoger un archivo local y volcar el nombre del archivo, así como su contenido a un par de variables.

  Su argumento es una cadena de texto (por lo que va flanqueada entre comillas sencillas) que corresponde al nombre de una función (sin los paréntesis de argumentos) en la que se usan un par de variables en que se volcarán los datos de nombre y contenido del archivo.

  Un ejemplo de uso sería una escena que contiene un control cuya acción es calcular, y su parámetro de cálculos es *_Open_('obtenerDatos')*. En la escena debe existir una función en *Definiciones* llamada *obtenerDatos()*, en la que en sus cálculos se pueden llamar dos variables *DJS.fileContent* y *DJS.fileName*, que guardarán, respectivamente, el contenido del archivo y el nombre del mismo. Al pulsar el botón, se lanza un diálogo de sistema mediante el cual el usuario puede elegir el archivo a leer.

- **_Load_()**. Esta función sirve para abrir un archivo de disco. La función devuelve el contenido leído del archivo, por lo que suele asignarse a una variable como cadena de texto.

  Su argumento es la ruta al archivo a leer entre comillas sencillas.

  Por ejemplo, si se usa `cdn=_Load_('Arch.txt')`, se volcará el contenido del archivo como cadena de texto a la variable *cdn*. O bien, si el archivo *Arch.txt* estuviera en una carpeta *datos* a nivel de la escena, `_Load_('./datos/Arch.txt')` sería la función a usar.

  Es **muy importante** tener en mente que, por cuestiones de seguridad en los navegadores, esta función **sólo funciona en el editor de *DescartesJS***.

  Así pues, las diferencias esenciales entre *_Open_()* y *_Load_()* son que, por un lado *_Load_()* sí permite accesar archivos indicados dentro de la escena (se le puede decir dentro de la escena qué archivo leer y no es necesario escogerlo mediante un diálogo), y por otro, que *_Load_()* sólo funciona en editor y no en navegador. Por esta última razón, *_Load_()* suele usarse sólo al preparar datos de una escena que, en su fase final de presentación, los llevará dentro.

Hagamos un ejercicio para entender mejor estas funciones. El ejercicio, junto con las instrucciones para lograrlo, se encuentran en Lee Guarda. El documento del interactivo como tal se encuentra en este vínculo.

En este ejercicio se abre un archivo de texto previamente generado en un editor de texto como *Notepad++*. Su codificación se elige como *UTF-8 sin BOM*, con el fin de que no guarde caracteres que puedan estorbar la interpretación del mismo por *DescartesJS*. Los datos se leen con la función *_Open_()*. Los datos leídos son posteriormente guardados en

otro archivo *Arch2.txt* mediante *_Save_()*. Finalmente, son leídos también con la función *_Load_()*.

Se enfatiza que un detalle importante es que es preciso notar que, por cuestiones de seguridad de los navegadores, no es posible que *DescartesJS* lea un archivo determinado usando *_Open_()* sin que antes sea indicado por el usuario cuál archivo se leerá. Es decir, no se puede pedir a la escena que directamente abra el archivo. Así, si la escena se usa en navegador, sólo se puede pedir al usuario que lo eliga en el diálogo antes y de ahí lo abra. Alternativamente, la función *_Load_()* se puede usar para dicho propósito, pero tiene la desventaja que, por cuestiones de seguridad, sólo funcionará bien en el editor, y no en navegador. El usuario puede corroborar esto usando el botón *Cargar* en la escena abierta en un navegador, y notará que no carga los datos a la escena. Por último, en el ejercicio se incluye en el contenido de *Arch.txt* los datos de una matriz. El usuario puede practicar volcando, mediante la funciones *_StrToMatrix_()* y/o *_StrToVector_()*, esos datos de matriz a una matriz, o a un vector, según sea el caso.

14.2. Guardado y recuperado de datos dentro del archivo de escena

En muchas ocasiones es preferible guardar un archivo con datos dentro de la escena misma. Esto garantizará que no habrá problemas de permisos al leerlo.

Por ejemplo, suponga que hay datos de una matriz que tardan mucho en generarse en una simulación debido a la complejidad de los cálculos. Una escena así puede trabarse, o bien el navegador comenzará a lanzar diálogos indicando al usuario que los cálculos están tardando mucho, lo que puede confundir al usuario. En estos casos, se puede optar por guardar los datos de la simulación en una matriz (luego de haberse sido generados mediante la funcionalidad descrita en el apartado sobre transferencia de información entre matrices y variables de texto) en archivo para luego sólo mostrarlos al usuario. Pero además, se quiere asegurar que el archivo se leerá sin problemas de permiso. Así que el código del archivo, que lleva los datos de la matriz, se puede embeber en el código *html* del archivo de escena dentro de un bloque *script*.

El bloque *script* en cuestión abre con una etiqueta `<script id="ruta al archivo" type="descartes/archivo" >` y cierra con una etiqueta `</script>`. Por ejemplo, `<script id="./dat.txt" type="descartes/archivo" >` indicaría que se lee un archivo *dat.txt*. Entre la etiqueta de apertura y cierre va el contenido del archivo. El archivo de datos se puede generar primero, luego copiarse los datos de él y embeberlos de esta forma en el bloque *script* dentro de la escena de *DescartesJS* que los leerá, con lo que se asegura que la lectura de los mismos no estará impedida por permisos. La ubicación del bloque *script* se sugiere sea tras el cierre del bloque *div* de la escena.

Hay unos cuantos detalles a notar sobre embeber contenidos de archivos como un bloque script. Aunque no es indispensable, se sugiere usar el ./ y no sólo el nombre del archivo dentro de la etiqueta de apertura. Ello con el fin de asegurar una correcta leída de archivo. Otro detalle importante es que los datos de archivo dentro de un bloque *script* **no** son visibles para el editor. Es decir, el editor puede leer datos externos, pero no aquellos embebidos en la escena *html*. Sin embargo, son visibles para la escena abierta desde navegador. Así, siempre hay que verificar la correcta lectura de los mismos abriendo la escena en navegador.

Hagamos una escena para aclarar el funcionamiento de los archivos embebidos. El ejercicio resultante, junto con las instrucciones para lograrlo, se encuentran en Guard script. El documento del interactivo como tal se encuentra en este vínculo.

Algo a notar de este ejercicio es que se usa la función *_Load_()*, que no permite cargar archivos ajenos a la escena. Sin embargo, note que el archivo *Datos.txt* está incluido dentro de la misma. Está dentro de un bloque *script* embebido en el código *html* de la escena. Así que, en este caso, se puede usar *_Load_()* tal cual.

Observe que la escena fue creada primero en *DescartesJS* y después se le hicieron cambios al código en un editor de texto. Pero antes de hacer esto, se cerró *DescartesJS*. Ello se hace con el objeto de que cambios hechos en uno no alteren los cambios hechos en el otro. Esto no suele suceder, pero es buena medida sólo editar con una de las dos herramientas a la vez.

Note también que en el último paso de las instrucciones se indica abrir la escena en un navegador y pulsar el botón *Cargar*. Ello responde a que, como se mencionó, el editor no accesa el bloque *script* incluido en la escena *html*, pero el navegador sí.

Finalmente, note que, aunque el ejemplo propuesto aquí es muy sencillo (una matriz de 2×2), se puede extender a casos de matrices enormes y donde cada entrada puede tomar un tiempo considerable en ser generada. Así que esta estrategia de guardado permite, por un lado, sólo cargar el ejemplo sin hacer los cálculos, y por otro, al estar embebidos los datos dentro del archivo *html*, garantiza su lectura sin problemas de permisos del navegador.

15

Herramientas generales

Algunas herramientas no pertenecen a un selector en particular, sino que se encuentran en una gran variedad de los mismos. Estas herramientas se discutirán en este capítulo.

15.1. Herramienta del control de colores e imágenes

Esta herramienta permite aplicar colores o imágenes a los diversos elementos que lo admitan. Por ejemplo, botones, fondos de espacios, figuras geométricas como rectángulos, etc. La herramienta cuenta con tres pestañas superiores. Cada pestaña permite la introducción de algún color o gradiente, o bien una imagen. A continuación se describe el funcionamiento de cada pestaña:

15.1.1. RGB

Esta pestaña permite la asignación de un color único a alguna figura. El formato de introducción es por colores primarios (*RGB* son las iniciales en inglés para *red*, *green* y *blue*). También incluye un control de transparencia. En la Figura 15.1 se muestra la pestaña para el control de colores. El color usado en esta ventana es 8E44AD y la transparencia es de 0.

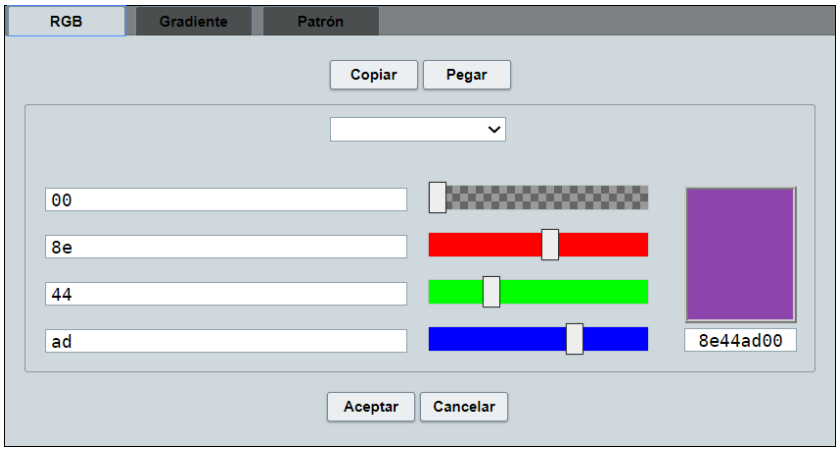


Figura 15.1: Pestaña *RGB* de la herramienta de control de colores

Esta pestaña cuenta con los siguientes elementos:

- *copiar*: un botón que permite copiar el color actual en la ventana a un portapapeles interno a Descartes para pegarlo a otros elementos en el interactivo. Es útil cuando muchos objetos han de tener el mismo color y transparencia, ya que reduce el tiempo de ingresado manual de la información.
- *pegar*: un botón que permite pegar un color previamente copiado de tal forma que el elemento en edición termine con el mismo color que se había copiado.
- menú de colores: un menú desplegable que permite elegir entre algunos colores prediseñados.
- *campo de texto y barra de transparencia*: consiste en un campo de texto y una barra horizontal. Ambos controlan la transparencia del objeto (que es qué tanto se permite ver de los objetos que se encuentran detrás del objeto siendo editado). Este control es útil cuando se prefiere dejar ver información u objetos detrás de aquél en cuestión. Es decir, cuando no se desea que el objeto cubra su fondo de forma total. Se puede introducir en forma de texto o controlando la barra. Adelante se detalla cómo se introducen los datos en forma de texto.
- *campo de texto y barra del color rojo*: también consiste en un control que se puede editar introduciendo texto en el campo, o bien a partir de la barra horizontal correspondiente. Este control define el componente del color rojo que tiene el objeto.
- *campo de texto y barra del color verde*: también consiste en un control que se puede editar introduciendo texto en el campo, o bien a partir de la barra horizontal correspondiente. Este control define el componente del color verde que tiene el objeto.
- *campo de texto y barra del color azul*: también consiste en un control que se puede editar introduciendo texto en el campo, o bien a partir de la barra horizontal correspondiente. Este control define el componente del color azul que tiene el objeto.
- panel de muestra del color: un cuadro que permite una vista previa del color del elemento editado, así como su código en hexadecimal. En la Figura 15.1 se muestra el panel a la derecha. Abajo de panel está un código de 8 dígitos. Los primeros 6 corresponden al código hexadecimal del color y los últimos dos a la transparencia.

Adicional a esta funcionalidad, también es posible seleccionar un color con funcionalidad tipo *gotero*. Para ello, basta pulsar en el panel de previsualización del color (el rectángulo sobre el código hexadecimal del color). Con esto, aparece un selector de color en forma de un círculo con una cuadrícula dentro. Éste sigue al ratón y se puede colocar sobre el color que se desea muestrar (dentro o fuera de *DescartesJS*). Cuando se encuentra el color a muestrear (dentro del cuadrado central en el círculo) se pulsa, obteniendo así el código hexadecimal del color en el parámetro debajo del panel de previsualización del color. Esta herramienta permite evitar el usar herramientas externas para obtener colores, agilizando así el trasladado de códigos de color.

Si el color ha de introducirse en forma de texto, se puede introducir en forma hexadecimal y forma decimal, como se detalla a continuación:

Forma hexadecimal de introducción de color

Cuando se introduce un color en hexadecimal (también conocido como base 16), cada componente de color y la transparencia tiene un total de 256 distintos valores posibles para cada color y la transparencia. Los valores van como sigue: 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0a, 0b, 0c, 0d, 0e, 0f, 10, 12, 13 ... 9d, 9e, 9f, a0, a1, a2 y así sucesivamente hasta el valor ff. El valor 00 corresponde al valor más bajo y el valor ff corresponde al más alto. Si el código hexadecimal involucra letras, éstas pueden ser mayúsculas o minúsculas.

Cuando se introduce un valor hexadecimal en alguno de los campos de texto, se puede presionar *INTRO* y la barra horizontal correspondiente se actualizará. También el cuadrado de vista previa del color lo hará. De forma semejante, el arrastrar una de las barras horizontales automáticamente actualizará el valor del campo de texto y el color del cuadro de vista previa.

En caso de no estar familiarizado con la numeración hexadecimal, resulta un buen ejercicio mover una de las barras horizontales aumentando un valor a la vez para entender mejor este tipo de numeración.

Forma decimal de introducción de color

Aunque la notación hexadecimal es la más usada para determinar los colores, Descartes también permite una notación decimal. En este caso, el valor mínimo es el 0 y el valor máximo es el 1. Esto resulta útil cuando el color es determinado por el valor de alguna variable, en lugar de introducir sólo un valor constante determinado en forma hexadecimal. Por ejemplo, en el campo de texto de color rojo se puede poner la variable *ColorRojo* (que acepta valores entre 0 y 1), que en algún otro lado cambia de valor según el comportamiento del interactivo. De tal forma que el cambio de color puede ser dinámico. Sin embargo, note que si se introduce una variable para controlar un cierto color, la barra del color correspondiente automáticamente se recorrerá al extremo izquierdo y el color del panel no necesariamente corresponderá al color en cuestión. Ello sucede pues, al usar una variable para controlar el color, la herramienta de control de colores no sabe qué color usar y no mostrará el correcto.

Ahora bien, nos compete también hacernos la pregunta: ¿cómo entonces convertimos entre hexadecimal y decimal?

Transformación de hexadecimal a decimal en la introducción de color

Muchas calculadoras científicas cuentan con un convertidor entre la numeración decimal y hexadecimal. Algunas páginas tales como la incluida en este vínculo y éste permiten estas conversiones también. Así, podemos, por ejemplo, introducir el valor hexadecimal 9a y convertirlo a decimal: 154. Dado que Descartes maneja valores en decimal entre 0 y 1, el valor obtenido ha de dividirse entre 256 (el valor máximo admitido), lo cual nos da un valor de 0.6015625. Éste es el valor en forma decimal, y resulta igual introducirlo en el campo de texto que introducir 9a.

La desventaja de introducir valores decimales es que, aún cuando se presione *INTRO*, la barra no se actualizará. Tampoco lo hará el cuadrado de vista previa de color. Pero el color será mostrado al presionar el botón *Aplicar*. En caso que se introduzca un valor menor a 0 o mayor a 1 en alguno de los colores o la transparencia, el color mostrado será negro dado que no se podrá interpretar dicho valor. Así pues, el panel de vista previa del color sólo servirá si el color es introducido en forma hexadecimal.

15.1.2. Gradiente

Esta pestaña de la herramienta de colores permite usar un degradado de colores. En la figura 15.2 se muestra la pestaña para introducción de gradientes o degradados de colores.

Figura 15.2: Pestaña *Gradiente* de la herramienta de control de colores.

Esta pestaña cuenta con los siguientes elementos:

- *copiar*: un botón para copiar la definición actual en esta pestaña de *Gradiente*.
- *pegar*: un botón con el que se implementa la definición copiada de la pestaña *Gradiente* con el botón *Copiar* mencionado anteriormente.
- Barra de muestra: una barra con una muestra de cómo se verá la implementación de la definición actual de un gradiente.
- x_1, y_1, x_2, y_2: campos de texto en los que se introduce, respectivamente y usando **coordenadas absolutas**, la coordenada horizontal del primer extremo del segmento sobre el que se define el gradiente, la vertical del primer extremo, la horizontal del otro extremo, y la vertical del otro extremo. El gradiente se traza en la dirección que sigue dicho segmento. Las coordenadas absolutas se toman respecto al contenedor donde se aplicará el gradiente (el fondo de un espacio, el área de un botón, etc., dependiendo del tipo de objeto al que se aplica).
- +: un botón con el que se agregan paradas. En cada parada se puede definir un gradiente distinto. Y cada parada tiene los siguientes controles:

- *Parada*: un campo de texto en el que se introduce un número entre 0 y 1 que indica donde empieza el gradiente definido en esa parada. 0 corresponde al inicio del canvas o lienzo (el inicio del segmento de dirección), y 1 al final.
- selector de color: un botón, con un recuadro de color, que al oprimirlo despliega una paleta de colores. El pulsar en alguna parte de la paleta se traduce en seleccionar el color correspondiente, y éste queda introducido en forma de código en la parte inferior de la paleta de colores. Es también posible introducir numéricamente el color en el código inferior de la paleta. Y se puede alternar entre formato de color RGB, HSL, o bien hexadecimal.
- -: un botón con el que se elimina la parada en cuestión.

Conviene atender la barra superior de ejemplo para notar si el gradiente es lo que el usuario quiere. Los botones asociados a cada parada muestran el color inicial de la misma en su interior. Una vez implementado el gradiente, el botón de la herramienta de color que se usó (ya sea en un botón, o en un fondo de espacio, etc.) aparece con el gradiente de ejemplo en su interior. En la figura 15.3 se muestra la paleta de colores lanzada al oprimir un botón de parada.

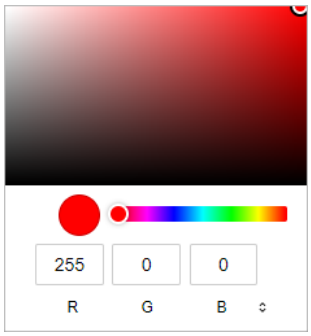


Figura 15.3: Paleta de colores de una parada en el gradiente.

A manera de ejemplo, considérese un gráfico segmento con las siguientes coordenadas absolutas: `(0,0)(200,200)`. Si en la pestaña *Gradiente* de la herramienta de colores se introduce $x_1 = 0$, $y_1 = 0$, $x_2 = 200$, $y_2 = 200$, el gradiente va en la misma dirección del vector (de izquierda a derecha y de arriba a abajo). Si en una parada con número 0 se coloca un color amarillo, y en aquella con número 1 se coloca un rojo, se trazará un gradiente de amarillo a rojo en la dirección en que se recorre el segmento. Sin embargo, si se dan los valores $x_1 = 200$, $y_1 = 200$, $x_2 = 0$, $y_2 = 0$, los colores terminan al revés, con el amarillo abajo y el rojo arriba. E, inclusive, si se asigna $x_1 = 0$, $y_1 = 200$, $x_2 = 200$, y $y_2 = 0$, todo el segmento queda de un mismo color anaranjado, pues la dirección del gradiente es perpendicular a la dirección del vector, y éste queda a la mitad (naranja) entre el amarillo y el rojo.

Como otro ejemplo, considere un gráfico rectángulo con las siguiente coordenadas absolutas: `(0,0,200,100)`. Como relleno, se elige un gradiente con $x_1 = 0$, $y_1 = 0$, $x_2 = 200$ y $y_2 = 100$. En la primera parada (con valor 0) se toma un color azul marino y en la segunda

parada (con valor 1) se toma un color negro. El degradado es entonces diagonal de arriba a abajo y de izquierda a derecha. Pero si $x_1 = 0$, $y_1 = 100$, $x_2 = 200$ y $y_2 = 0$, el degradado va de la esquina inferior izquierda a la superior derecha del rectángulo. Si $x_1 = 0$, $y_1 = 0$, $x_2 = 200$ y $y_2 = 0$, el degradado es horizontal de izquierda a derecha (parejo verticalmente). Y si $x_1 = 0$, $y_1 = 0$, $x_2 = 0$ y $y_2 = 100$, el degradado es vertical, de arriba a abajo (parejo horizontalmente).

Como un último ejemplo, considere el anterior rectángulo, sólo que ahora coloque la primera parada en 0 y la segunda parada en 0.1. Y use un degradado horizontal de izquierda a derecha ($x_1 = 0$, $y_1 = 0$, $x_2 = 200$ y $y_2 = 0$). El degradado de azul a negro se da sólo en el primer 10 % horizontal del rectángulo. Todo el 90 % restante de la derecha se traza del último color elegido (negro). Ello se debe a que la parada final está en 0.1 (el 10 % de la unidad). Este efecto también se puede lograr si la primera parada se pone en 0, la segunda en 1, pero como dirección del gradiente se elige $x_1 = 0$, $y_1 = 0$, $x_2 = 20$ y $y_2 = 0$. El efecto es el mismo puesto que se indica, en el vector de gradiente, que el punto final es en 20, y el ancho del rectángulo es 200. Así, 20 es el 10 % de 200.

15.1.3. Patrón

En la pestaña *Patrón* de la herramienta de control de colores se puede incluir una imagen (*jpg*, *png*, o *gif* estática) en el objeto en que se define (un espacio, el fondo de un botón, etc.). En la figura 15.4 se muestra un ejemplo de esta pestaña.

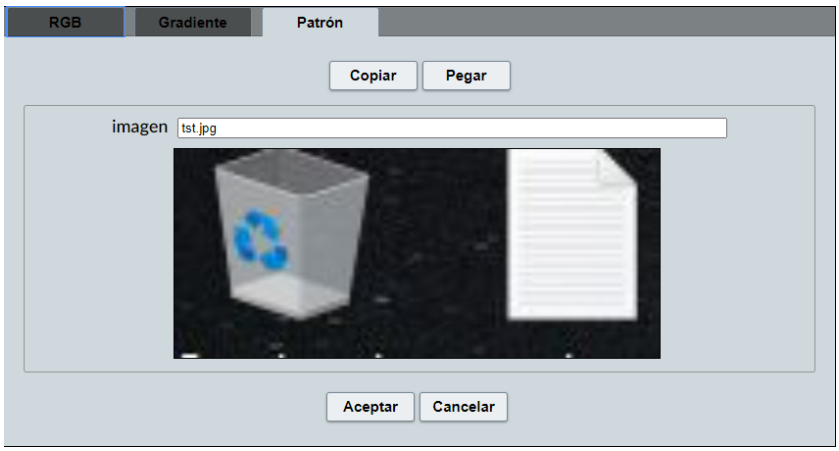


Figura 15.4: Pestaña *Patrón* en la herramienta de control de colores.

Los elementos son:

- *copiar* y *pegar*: los botones para copiar y/o pegar una definición de patrón de imagen a incluir, y que funcionan como para las otras dos pestañas.
- *imagen*: un campo de texto en que se introduce la ruta a una imagen a usar. De usarse una dirección relativa a la ubicación del archivo *html* de la escena, conviene que ésta esté previamente guardada.

La imagen comienza a dibujarse en el origen del lienzo o canvas. De ahí, si sobra espacio, se repite (a manera de mosaico) hasta cubrirlo todo.

Cuando se usa un patrón, en el botón del elemento en cuestión se despliega un paisaje genérico para indicar que es un patrón el que se está usando.

Cuando se usa un gradiente en un control tipo barra, el color de la misma cambia al desplazar la barra. Y si es un patrón, el manipulador de la barra muestra el mosaico de la imagen si el patrón se asocia al *color interior* de la barra, o se muestra el mosaico detrás del manipulador si el patrón se asocia al *color* de la barra.

15.2. Herramienta de introducción de textos

La herramienta de introducción de textos se encuentra disponible para una variedad de gráficos tales como *punto*, *texto*, etc. así como para textos relacionados a otros elementos del interactivo más allá de los gráficos. No obstante, su funcionamiento es el mismo en todos los casos.

Normalmente se presenta un campo de texto donde se puede introducir el texto al vuelo. Después del campo de texto hay un botón *T*. El texto se puede introducir directamente en el campo de texto, o bien se puede oprimir el botón *T* para introducir texto sencillo, o bien hacer clic en el botón *Rtf* para texto enriquecido (*Rich text format*).

15.2.1. Introducción de texto simple

Si se presiona el botón *T* adyacente a un campo tipo *texto* en un gráfico se abre una ventana donde se pueden introducir varias líneas más cómodamente. En la figura 15.5 se muestra dicha ventana.

Es posible elegir de tres fuentes distintas para el texto: *SansSerif*, *Serif* y *Monospaced* (monoespacio). Esta fuente se aplica a todo el texto introducido en la ventana. Esto es, no es posible tener distintas fuentes en un texto simple. Cuenta con un menú del cual se puede elegir el tamaño de la fuente. Tiene también un checkbox **negrita** para que el texto se muestre en negritas y otro ***cursiva*** para que se muestre en cursiva. Nuevamente, estos controles se aplican por igual a todo el texto introducido en esta ventana. Por último, tiene un botón con diversos símbolos para la introducción de caracteres especiales. Este botón lanza una ventana adicional como se muestra en la Figura 15.6 con las diversas opciones de símbolos. A la izquierda de esta ventana hay un menú para navegar más ágilmente entre los caracteres dependiendo del conjunto de caracteres que se desee introducir. Este panel es muy útil particularmente cuando se desea introducir operadores matemáticos y letras griegas.

Es posible introducir saltos de línea en esta ventana simplemente oprimiendo *INTRO*. Una vez introducido el texto en esta ventana, se puede elegir *Aceptar* el cambio o descartarlo con el botón *Cancelar*. En caso de aceptar el cambio, el texto es trasladado al campo de texto *texto* localizado al lado del botón que se oprimió para lanzarlo en primer lugar. El

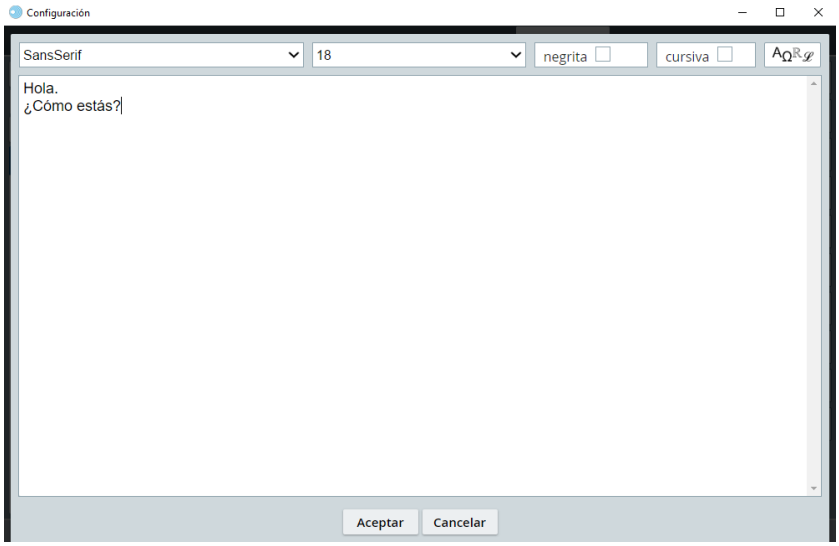


Figura 15.5: Ventana de introducción de texto simple

texto aparece en una sola línea en ese campo de texto. Los saltos de línea son representados por \n (la diagonal invertida seguida de una *n*). Alternativamente es también posible introducir \n directamente dentro de este campo de texto para que se imprima un salto de línea.

Aunque para añadir texto realmente enriquecido conviene ver el apartado sobre introducción de texto enriquecido, se puede incluir cierto grado de texto enriquecido dentro del texto simple, mediante instrucciones parecidas a las del editor LaTeX. Esta edición se limita a cambios en el color, negritas e itálicas para sólo parte del texto, y también permite introducir subíndices y superíndices. También es posible cambiar la alineación de parte del texto a centrada o a la derecha.

- **Cambio de color**: Cuando se quiere cambiar el color de una parte del texto se usa `\color{}{}`. Dentro del primer par de llaves se incluye el código hexadecimal del color a usar, mientras que dentro del segundo par de llaves se incluye el texto a usar dicho color.

  Por ejemplo, al introducir `Hola, ¿\color{ff0000}{cómo} estás?` y aplicar los cambios, se debería ver así: *Hola, ¿cómo estás?*

- **Negritas**: Las letras en negritas deben ir dentro de las llaves de `\b{}`.

  Por ejemplo, si se introduce `Descartes\b{JS}`, esto se muestra como *Descartes**JS***.

- **Itálicas**: El funcionamiento para introducir itálicas o cursivas es igual al de negritas. Sólo basta cambiar la *b* por *i*. Es decir, el texto a imprimir en itálicas debe ir dentro de las llaves de `\i{}`.

- **Lenguaje matemático**: Cuando se quiere incluir lenguaje matemático, es necesario flanquearlo entre las llaves de `\${}`.

Figura 15.6: Ventana de selección de caracteres especiales dentro de la herramienta de introducción de texto simple.

- **Superíndices**: El elemento dentro de la fórmula matemática que ha de quedar como superíndice debe ir después del símbolo de acento circunflejo (`^`). Si el superíndice ha de aplicarse a más de un caracter, es preciso flanquear los caracteres en cuestión entre llaves (`{}`). Si solo consta de un caracter, las llaves no son necesarias y el caracter en superíndice será aquél inmediatamente tras el acento circunflejo.

  Por ejemplo, si se desea mostrar A^{BC}, es necesario introducir `\${A^{BC}}`.
- **Subíndices**: Su funcionamiento es prácticamente igual al de los superíndices. Sólo basta reemplazar el acento circunflejo por guión bajo (es decir, en lugar de usar `^` hay que usar `_`).

  Por ejemplo, si se desea mostrar A_{BC}, es necesario introducir `\${A_{BC}}`.
- **Fracciones**: El elemento dentro de la fórmula matemática a introducir está dado por `\frac{}{}`. Dentro del primer par de llaves va el numerador, dentro del segundo el denominador.

  Por ejemplo, para mostrar la fracción $\frac{e^x}{b_2}$ sería necesario introducir lo siguiente: `\${\frac{e^x}{b_2}}`.
- **Raíces**: Para introducir una raíz se usa el código `\sqrt{}{}` dentro de la fórmula matemática. En las primeras llaves va el índice de la raíz, y en las segundas el radicando. Es posible omitir el índice dejando vacío el contenido de las llaves.

  Por ejemplo, `\${\sqrt{3}{sin(a)}}` mostraría $\sqrt[3]{\sin(a)}$.

  Pero `\${\sqrt{}{sin(a)}}` mostraría $\sqrt{\sin(a)}$.
- **Integrales**: Para introducir una integral se usa el código `\int{}{}{}` dentro de la fórmula matemática. En las primeras llaves va el límite inferior de la integral,

en las segundas, el límite superior, y en las terceras, el integrando con todo y su diferencial.

Por ejemplo, `\${\int{a}{b}{x^2 dx}}` muestra $\int_a^b x^2 dx$.

En caso de no incluirse límites de integración, basta dejar vacío el interior del primer par de llaves, al igual que el segundo.

- **Sumas**: Para introducir una suma es preciso incluir el código `\sum{}{}{}` dentro de la fórmula matemática. Dentro de las primeras llaves va el valor inicial del índice de la suma, dentro de las segundas el valor máximo, y dentro de la terceras el argumento de la suma.

  Por ejemplo, si se introduce `\${\sum{i=1}{n}{i^2}}`, se logra $\sum_{i=1}^{n} i^2$.

- **Productos**: Su comportamiento es muy similar al de sumas e integrales. Debe introducirse el código `\prod{}{}{}` dentro de la fórmula matemática. Dentro de las primeras llaves va el valor inicial del índice del producto, dentro de las segundas el valor máximo, y dentro de la terceras el argumento del producto.

  Por ejemplo, si se introduce `\${\prod{i=1}{n}{A_i}}`, se logra $\prod_{i=1}^{n} A_i$.

  Si el parámetro *fuente* del gráfico tipo texto se encuentra usando la fuente *Serif*, el lenguaje matemático se mostrará en itálicas, ya que dicha fuente es la que típicamente se utiliza para mostrar ecuaciones y demás lenguaje matemático.

- **Alineación centrada**: Basta incluir el texto a centrarse entre las llaves de `\c{}`. Si el parámetro *ancho del texto* del gráfico tipo texto se encuentra con valor 1 (es decir, que no se le pone límite horizontal al texto), entonces el texto a centrar se centrará respecto al renglón más largo presente. Si dicho parámetro tiene otro valor (por ejemplo, 500), el texto quedará centrado respecto a un renglón imaginario de 500 px de ancho. Conviene estudiar el apartado sobre ancho del texto para recordar cómo funciona.
- **Alineación a la derecha**: Su funcionamiento es análogo al del centrado de texto, sólo que alinea el texto incluido entre las llaves al margen derecho. La expresión a usar es `\r{}`. Es decir, hay que cambiar la `c` (de *centrado*) por `r` (de *right* o *derecha*).
- **Alineación a la izquierda**: el texto entre llaves de `\l{}` quedará alineado con el margen izquierdo del bloque de texto. La letra usada es *l* (por *left* o *izquierda*).
- **Alineación justificada**: el texto entre llaves de `\j{}` quedará justificado respecto al parámetro ancho de texto del gráfico texto.

Se pueden anidar las distintas ediciones al texto. Por ejemplo, para mostrar el texto ***Hola***, donde el texto es de color azul y la ***o*** está en negritas, habría que introducir `\color{0000FF}{H\b{o}la}`.

Es preciso notar que el texto enriquecido abordado es útil sólo para cambios pequeños de edición. Para una edición realmente de texto enriquecido conviene estudiar el apartado a continuación.

15.2.2. Introducción de texto enriquecido

Si se presiona el botón *Rtf* adyacente a un campo de introducción de texto, la ventana que se muestra para la introducción de texto es diferente. Cuenta con muchos botones en la parte superior, los que permiten tener diferentes tipos de texto así como diversos símbolos matemáticos. En la Figura 15.7 se muestra dicha ventana.

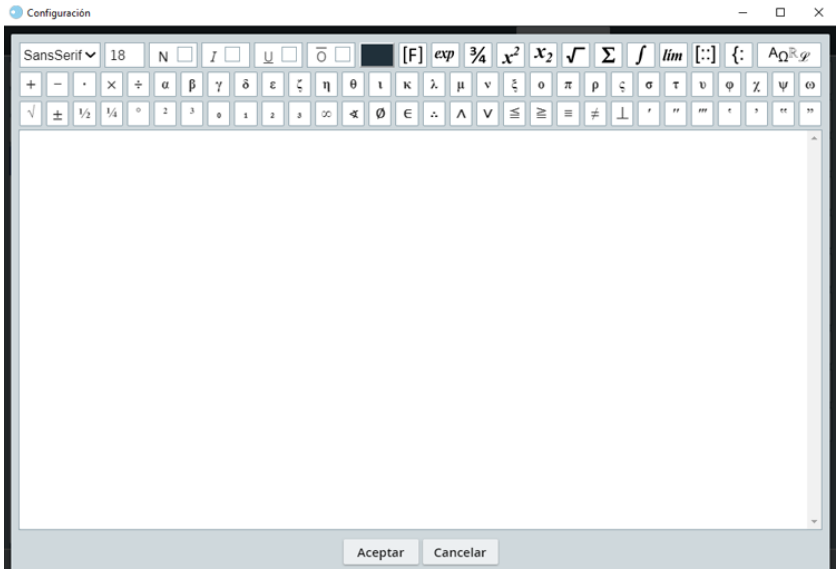


Figura 15.7: Ventana de introducción de texto enriquecido.

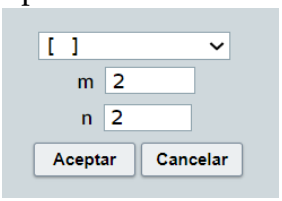
En esta ventana se puede introducir texto igual que en la de texto simple. No obstante, es posible seleccionar parte del texto y aplicarle cambios de formato sólo a esa parte. Se puede usar el menú para el tipo de fuente, el menú para el tamaño, y los checkboxes de cursivas y negritas para aplicar los cambios a dicha selección.

Cuando se coloca el cursor al final del texto en esta ventana y se continúa escribiendo, algunas veces el formato de este nuevo texto no coincide con aquél a partir del cual se continuó escribiendo como ocurre en otros editores de texto. Cuando la intención es usar un texto con algunas características dadas para la ventana de texto enriquecido, es conveniente primero usar la ventana de texto simple para dar dichas características y, una vez hecho esto, editar el texto con la ventana de texto enriquecido. De esta forma suelen evitarse este tipo de problemas.

Aparte de los primeros controles en esta ventana existen algunos más sofisticados:

- **botón de subrayar**: tiene una U subrayada y se encarga de colocar una subraya bajo el texto seleccionado.
- **botón de suprarayar**: tiene una O suprarayada y se encarga de colocar una raya encima del texto seleccionado.

- **botón de color**: botón que muestra el color actual del texto. Al oprimirlo, muestra la herramienta de edición de color para asignarle un nuevo color al texto seleccionado.
- **botón [F]**: introduce a la derecha del cursor una fórmula. Esto se visualiza como una caja de texto de borde punteado. Una vez introducida, se puede colocar el cursor dentro de ella e introducir texto en lenguaje matemático. Los botones siguientes lanzan una caja de fórmula si son oprimidos cuando el cursor está fuera de una de estas cajas; y si están dentro de una, simplemente introducen el símbolo dentro del cuadro de fórmula existente. El atajo para introducir una fórmula es *CTRL* + *f*.
- **botón *exp***: es un botón para introducir expresiones. En las expresiones se pueden introducir variables, constantes, o expresiones a calcular. Una vez evaluadas, lo que se imprime en el interactivo es el valor de las mismas. Por ejemplo, en alguna parte del programa se puede calcular $a = 2 + 1$. Si dentro de la expresión se introduce *a*, el valor impreso sería 3.00. Para introducir un cálculo en la expresión, se hace doble clic sobre el *expr* que aparece en el editor de texto enriquecido. Ello desplegará una ventana en la cual se introduce el cálculo que habrá de hacerse. Las expresiones se pueden introducir en cualquier lugar del editor de texto enriquecido donde se pueda introducir texto simple en un cuadro de fórmula. Por ejemplo el límite superior de una integral puede ser una expresión que cambia según la manipulación del interactivo. Se puede introducir con un atajo siempre y cuando el cursor ya esté dentro de un cuadro de función, y el atajo es *CTRL* + *e*. Dentro del diálogo de introducción de la expresión se puede decidir si para esa expresión en particular se habrán de mostrar decimales de forma fija y cuántas decimales se mostrarán.
- **botón fracción**: se muestra como un botón dentro del cual hay un 3/4. Al oprimirlo introduce una fracción. Se puede colocar el cursor en el numerador o en el denominador para editarlo. El atajo para introducir una fracción es *CTRL* + / (la diagonal sencilla). También se puede insertar una fracción con *CTRL* + *f* siempre y cuando el cursor ya esté en modo de edición de función.
- **botón elevar a una potencia**: se muestra como un botón que dentro tiene un x^2. Agrega una base y exponente. Se puede colocar el cursor en cualquiera de los dos para introducir texto en ellos. Se puede introducir este símbolo con *CTRL* + ↑ (la flecha hacia arriba del teclado).
- **botón agregar subíndice**: se muestra como un botón que dentro tiene un x_2. Permite agregar un subíndice. Automáticamente agrega un campo de fórmula con la posición para colocar la letra y el subíndice si el cursor no está dentro de uno ya. Si está dentro de un campo de fórmula, agrega sólo la posición para el subíndice ya que siempre se puede introducir la letra en el campo de fórmula. Cuenta con el atajo *CTRL* + ↓ (la tecla en el teclado para mover el cursor hacia abajo).
- **botón extraer raíz**: tiene el símbolo de un radical en su interior. Si se oprime agrega un símbolo de radical. Se puede colocar el cursor en el radicando o en el índice del radical para su edición. Se puede introducir este símbolo con el atajo *CTRL* + *r*.

- **botón suma**: tiene la letra *sigma* griega mayúscula que corresponde a sumas sobre índices. Si se oprime agrega un símbolo de suma. Se puede editar colocando el cursor en su parte inferior (para indicar el índice de la suma y su valor inicial), en su parte superior (para indicar hasta donde llega el índice) y a la derecha (para indicar los términos que se habrán de sumar. El atajo para introducir este símbolo es *CTRL + s*.
- **botón integral**: al oprimirlo agrega el símbolo de una integral. El cursor puede colocarse debajo del símbolo para editar el límite inferior de la integral, arriba del símbolo para el límite superior de la misma, y a la derecha para el argumento de la integral.
- **botón *lím***: al oprimirlo agrega el símbolo de límite con sus posiciones para el argumento del límite, la variable en cuestión y el valor o variable al que tiende. Puede agregarse dentro de un campo de fórmula o directamente en el texto (en cuyo caso se agregará un campo de fórmula). Cuenta con el atajo *CTRL + l*.
- **botón de matriz**: tiene en su interior un paréntesis con cuatro puntos. Abre una ventana en la que se especifican las características de la matriz: 

El menú permite asociar corchetes cuadrados, paréntesis, llaves, barras verticales simples, y barras verticales dobles como delimitantes de la matriz. Y en los parámetros *m* y *n* se indica el número de elementos de las filas y las columnas de la matriz, respectivamente. Una vez agregada la matriz, el cursor se puede colocar en cualquiera de las entradas de la matriz para su edición.
- **botón definición por partes**: tiene en su interior una llave para definir funciones por partes (parecida a la llave de un cuadro sinóptico). Cuando es oprimido, abre una ventana en la que se puede introducir el número de partes en que se definirá la función. Al aceptar, aparece la llave con un número de entradas correspondiente. Se puede colocar el cursor en cualquiera de las entradas para su edición.
- **botón tabla**: abre una ventana con una variedad de caracteres especiales que se pueden introducir. Su funcionamiento es el mismo que para la ventana de introducción de texto simple.

Cuando se inserta una forma de texto enriquecido, ésta aparece con unos marcadores de posición. Por ejemplo, la integral tiene 3: el límite inferior, el superior, y aquél del integrando. El cursor se puede colocar en cualquiera de estos marcadores de posición pulsando el mouse. Sin embargo, también se pueden navegar con las teclas de flechas del teclado.

Note que el control de tamaño en la ventana de introducción de texto enriquecido es un campo de texto en lugar de un menú, y permite elegir tamaños entre 8 y 200.

También se incluyen dos renglones para introducir una gran variedad de caracteres matemáticos y letras griegas que se usan frecuentemente en las fórmulas. Para introducir letras del alfabeto griego en mayúsculas basta pulsar el botón de mayúsculas (*SHIFT*) en el teclado. Mientras esté pulsado, las letras a elegir aparecerán en mayúsculas, y pulsar en una de ellas resultará en introducir la letra correspondiente en mayúscula.

Cuando se agrega algún símbolo y aparece en un cuadro de fórmula, siempre es posible insertar texto matemático a la izquierda del símbolo colocando el cursor en esa posición. Adicionalmente, los símbolos (por ejemplo, la fracción, el radical, etc.) pueden eliminarse colocando el cursor a su derecha y oprimiendo la tecla de retroceso. Es decir, se comportan como caracteres tal cual. Es preciso tener cuidado con qué es lo que se borra. Por ejemplo, si se tiene una expresión matemática con una integral y se coloca el cursor a la derecha de la misma pero fuera del cuadro de fórmula, el oprimir la tecla de retroceso borrará toda la integral, y no el último carácter dentro de la caja. Si lo que se quisiera borrar es el último carácter, hay que asegurarse que el cursor esté dentro de la caja de fórmula y a la derecha del carácter que se desea borrar. Igualmente, si se desea añadir una línea nueva después de alguna fórmula matemática, es preciso recorrerse a la derecha hasta salirse de la fórmula con las flechas del teclado (o bien hacer clic hasta la derecha de la línea) y luego oprimir *INTRO*.

Es posible también combinar las expresiones para generar fórmulas matemáticas complicadas. Por ejemplo, se puede generar una fracción contínua del tipo $\frac{1}{1+\frac{1}{1+\ldots}}$ oprimiendo el botón de fracción cuando el cursor está en el denominador de otra fracción. Los tamaños de los textos se ajustan automáticamente según el nivel en que se encuentra el texto.

Una mejora reciente permite ahora incluir distintos estilos dentro de una misma caja de fórmula. Por lo mismo, ahora es posible tener distintos colores, fuentes, etc. dentro de una misma fórmula.

Cuando se oprime el botón *Aceptar*, la ventana se cierra y se muestra de nuevo el editor con el texto editado. El campo de texto con el texto a imprimir se encuentra lleno de caracteres raros. Esto es un código para que el texto enriquecido se muestre correctamente. Ello se puede ver al oprimir el botón *Aplicar*.

Hagamos un breve ejercicio practicando el editor de textos. El interactivo de este ejercicio, junto con las instrucciones para lograrlo, se encuentran en EditorTextos. El documento del interactivo como tal se encuentra en este vínculo. Todos estos archivos están también disponibles en el archivo *DocumentacionDescartesJS.zip*.

Se pudo observar con este ejercicio que el editor de texto enriquecido, a pesar de ser un poco más difícil de usar que el de texto simple, permite una visualización más estética de texto matemático. Adicionalmente, permite una funcionalidad más rica, valga la redundancia, como el poder determinar el número de decimales mostrados para cada expresión, en lugar de tener una sola opción para todas las expresiones del texto.

15.3. El teclado virtual

Con el fin de que el teclado nativo presente en los dispositivos móviles no estorbe cuando se pulsa el campo de texto en un control, se puede echar mano de el teclado virtual de *DescartesJS*. Los teclados nativos en ocasiones ocultan parte del interactivo, en otros casos hacen que el interactivo ocupe la mitad de espacio, reduciéndolo en escala, con el fin de permitir que el teclado nativo se visualice completo.

El teclado virtual tiene la virtud de aparecer **dentro** del interactivo, y en la posición que el usuario indica. Sólo aparece si se encuentra marcada la casilla teclado del campo de texto de un control.

El teclado a seleccionar depende de para qué se piensa usar el campo de texto. Si se desea que el usuario del interactivo introduzca una respuesta puramente numérica, convendría por ejemplo, el 7×2 o el 14×1. El 4×4 permite operaciones básicas de suma y resta. Si se desea permitir al usuario tener acceso a más operaciones, entonces podría dársele esta libertad con el teclado de 5×4. Si se le pregunta su nombre, podría usarse el $10 \times 4\_alfa$. Así, el usuario puede echar mano de un teclado que se ajuste al tamaño y funcionalidad de su escena.

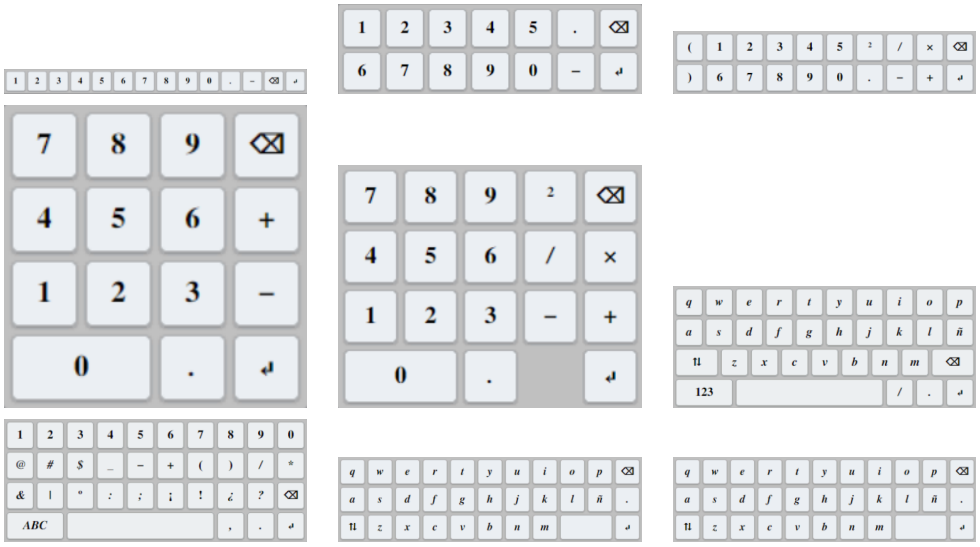


Figura 15.8: Los distintos tipos de teclados disponibles para controles con campo de texto.

Al pulsar una tecla en el teclado, se introduce el caracter en la posición del cursor dentro del campo de texto. Si se desea salir del teclado, es preciso pulsar la tecla de retorno, con lo que se implementa el cambio en el campo de texto.

Cabe mencionar que el teclado sólo está visible cuando el usuario pulsa en un campo de texto de un control con la funcionalidad de teclado virtual implementada. Al salir del campo de texto, se deja de visualizar el teclado. Asimismo, mientras está mostrado un

teclado virtual, la escena se vuelve ligeramente más opaca, indicando que el usuario no puede controlar ningún otro aspecto de la escena mientras está el teclado activo.

Algunos caracteres del teclado incluyen otros no directamente visibles en el teclado. Por ejemplo, si se pulsa y mantiene la pulsación de una letra que se pueden acentuar de distintas formas, encima del teclado aparece una serie de teclas adicionales mostrando las opciones a elegir. Lo mismo ocurre si se pulsa el punto decimal, con lo que se muestra encima del teclado una serie de teclas correspondientes a diversos símbolos numéricos adicionales, que bien pueden no estar visibles en el teclado mostrado. En este sentido, es preciso que si el usuario desea echar mano de esta funcionalidad, conviene que el teclado no se encuentre en la parte superior del espacio, sino dejar un colchón de 50 px sobre él para poder desplegar los símbolos. Por ejemplo, introduciendo `(100,50)` en el parámetro posición del teclado se deja espacio para dichas teclas.

15.4. Atajos de teclado

15.4.1. Atajos al editor de configuraciones y sus selectores

DescartesJS cuenta con diversos atajos de teclado para hacer la vida más fácil al usuario. A continuación se enlistan dichos atajos y su descripción. La tecla *CONTROL* se abrevia *CTRL* en la descripción, y la tecla de retorno se abrevia *INTRO*. A la tecla de retroceso se le denota *BACKSPACE*. Para indicar que se pulsan dos teclas simultáneamente, se usa el signo + entre ellas.

Atajos usados mientras se está dentro del editor general de *DescartesJS*.

- **CTRL+E**: Si se pulsa la tecla *CONTROL* y simultáneamente la tecla *E*, estando en la ventana principal de *DescartesJS*, lanza el editor de configuraciones.

Atajos usados mientras se está dentro del editor de configuraciones de *DescartesJS*.

- **CTRL+1**: Al usar esta combinación, se despliega el editor de configuraciones en el selector Escena.
- **CTRL+2**: Al usar esta combinación, se despliega el editor de configuraciones en el selector Espacios.
- **CTRL+3**: Al usar esta combinación, se despliega el editor de configuraciones en el selector Controles.
- **CTRL+4**: Al usar esta combinación, se despliega el editor de configuraciones en el selector Definiciones.
- **CTRL+5**: Al usar esta combinación, se despliega el editor de configuraciones en el selector Programa.

- **CTRL+6**: Al usar esta combinación, se despliega el editor de configuraciones en el selector Gráficos 2D.
- **CTRL+7**: Al usar esta combinación, se despliega el editor de configuraciones en el selector Gráficos 3D.
- **CTRL+8**: Al usar esta combinación, se despliega el editor de configuraciones en el selector Animación.
- **CTRL+INTRO**: Al usar esta combinación, se ejecuta la acción de aplicar los cambios, correspondiente al botón Aplicar del editor de configuraciones.
- **CTRL+ALT+INTRO**: Al usar esta combinación, se ejecuta la acción de aceptar los cambios, correspondiente al botón Aceptar del editor de configuraciones.
- **CTRL+BACKSPACE**: Al usar esta combinación, se ejecuta la acción de cerrar el editor de configuraciones, correspondiente al botón Cerrar del editor de configuraciones.
- **CTRL+SHIFT+C** (o *Command+SHIFT+C* en MacOS): Este atajo sólo funciona en una escena presentada en navegador. Su función es copiar al portapapeles el código de configuración de la escena activa. Este código es el que se muestra a manera de ejemplo en la figura 5.2.

15.4.2. Atajos de navegación de elementos en el panel de lista

Como se ha visto, los selectores *Espacios*, *Controles*, *Definiciones*, *Programa*, *Gráficos* y *Gráficos 3D* contienen un panel en su margen izquierdo en el que se enlistan los distintos elementos presentes para dicho selector. Con el objeto de agilizar el uso del editor de configuraciones, ahora es posible usar algunos atajos de teclado, **siempre y cuando el pánel esté seleccionado**. Cuando el panel está seleccionado, aparece enmarcado con un rectángulo de color azul oscuro.

Cuando el pánel está seleccionado y contiene más de un elemento, las teclas de flecha superior (*ARRIBA*) y flecha inferior (*ABAJO*) del teclado pueden usarse para cambiar la selección del elemento a aquél que está arriba o abajo del actualmente seleccionado.

Si, en cambio, en una lista de varios elementos, se encuentra uno seleccionado y se usa *CTRL+ARRIBA*, el elemento seleccionado **se desplazará** un lugar hacia arriba de la lista. Con *CTRL+ABAJO* el desplazamiento es hacia abajo.

Estos atajos permiten una organización y visualización más ágil de los elementos de una lista, especialmente para usuarios que manejan más el teclado.