Memoria Proyecto de Sistemas Informáticos:

Videojuego Educativo sobre Introducción a la Programación

Autores:
Gómez Gómez, Borja
De Miguel Vicenti, Antonio

Profesor director:
Fernando Rubio

CURSO 2007/2008

Facultad de Informática
Universidad Complutense de Madrid

Índice

Tabla de figuras

A Alfredo, Paula, Gonzalo, Ion, José Antonio y Eva, por habernos cedido su imagen y habernos ayudado tanto.

A nuestras familias por apoyarnos y estar siempre a nuestro lado.

A Carmen y Ainhoa, por aguantar tantas conversaciones sin enterarse muy bien de que iba y animarnos cuando el proyecto y las ganas se venían abajo.

Resumen

El objetivo de este proyecto es la creación y el desarrollo de un Videojuego Educativo sobre Introducción a la programación. Durante el desarrollo del juego, el jugador irá aprendiendo de manera sencilla y amena los principales conceptos de cualquier asignatura de introducción a la programación, sin centrarse en ningún lenguaje en particular. El tipo del videojuego es del llamado *aventura gráfica*.

El proyecto se ha desarrollado utilizando un software de desarrollo de videojuegos llamado Adventure Game Studio. Esta herramienta es de uso libre y no requiere ninguna licencia.

Abstract

The objective of this project is the creation and development of an Educational Videogame about Introduction to Programming. During the game, the player will learn in a simple and enjoyable way the main concepts of any subject about Introduction to the Programming, without focusing on any particular language. The type of game is the one called *adventure game*.

The project has been developed using a software program for programming videogames called Adventure Game Studio. This tool is free to use and requires no license.

Palabras clave

Videojuego educativo, Introducción a la Programación, didáctica de la programación.

1 Introducción

Actualmente hay infinidad de cursos de programación y miles de *tutoriales* sobre los distintos lenguajes y estilos de programación. Pero éstos generalmente requieren una cierta edad para la comprensión de algunos conceptos y sobre todo requieren constancia a la hora de trabajar con el método, lo que los hacen inaccesibles al público más joven. Desde pequeños aprendemos y estudiamos multitud de materias, pero en el ámbito de la informática, lo máximo que aprendemos es cómo usar un procesador de textos o manejar un navegador Web. Sin embargo, pocos conocen qué es la programación, cómo se desarrollan las aplicaciones que uno utiliza, y debido a esto al llegar a la universidad muchos se encuentran con una asignatura (“Introducción a la programación”, “Programación I”, etcétera) de la cual no conocen absolutamente nada.

Este proyecto tiene como objetivo el desarrollo de un software de entretenimiento educativo. La finalidad del mismo es enseñar los principios básicos de la programación imperativa.

El alcance de los contenidos incluye los siguientes conceptos:

- Algoritmo
- Constantes
- Variables
- Asignaciones
- Tipos: Entero, Real, Booleano, Carácter y Vectores
- Expresiones
- Sentencias condicionales IF
- Bucles For y While

El juego está dirigido a un público a partir de los 12 años y el género del juego es del tipo *aventura gráfica*.

El desarrollo del proyecto, a pesar de que no ha sido un trabajo estrictamente lineal, ha tenido tres fases muy diferenciadas, siguiendo el siguiente orden cronológico:

- Desarrollo del guión:

  La primera parte del proyecto. El objetivo era crear una historia jugable a través de la cual poder explicar los distintos conceptos de programación. Para el desarrollo de la misma, utilizamos un sistema *wiki* para facilitar la coordinación de todos los miembros y el director del proyecto.

- Parte gráfica:

  Engloba todos los gráficos del juego. Tanto los personajes, como los escenarios y los objetos son fotografías, en su gran mayoría tomadas por nosotros. Las imágenes de algunos objetos han sido tomadas de Internet, pero todas ellas están libres de copyright y de derechos de autor.

- Implementación:

  Esta parte incluye la incorporación de los gráficos elaborados en la fase anterior, el desarrollo de todos los diálogos del juego, y la programación de los scripts del mismo.

  Para la implementación del juego hemos utilizado el programa Adventure Game Studio (en adelante "AGS"). En la sección 2 se explicará con más detalle qué es AGS y cómo se utiliza.

Requisitos del juego:

Pentium o procesador superior.
64 Mb RAM.
Windows 98, ME, 2000, XP o Vista con DirectX 5 o superior.
Soporta todas las tarjetas de sonido y gráficas soportadas por DirectX 5.
Códecs de audio y video.

2 AGS, ¿Qué es?

Adventure Game Studio (AGS) es una plataforma creada por Chris Jones para la implementación y el desarrollo de juegos para PC. Es el sistema que hemos utilizado para la realización de este proyecto. Permite construir una interfaz de juego totalmente personalizada. Todo el desarrollo es posible por medio de código, sin embargo hay muchas partes de la creación del software en las que AGS facilita al programador esta tarea, por medio de diferentes menús interactivos, ahorrando mucho trabajo y permitiendo al programador desarrollar programas más completos. AGS tiene multitud de funciones ya implementadas y que son comunes a todos los juegos, en especial a los de tipo aventura gráfica. Algunas de las más importantes son:

- Guardar y Cargar el juego en memoria
- Lenguaje propio para la creación de diálogos
- Manejo de eventos
- Gestión del inventario
- Gestión de las imágenes del juego
- Montar las animaciones de los diferentes personajes
- Crear traducciones del juego

Todas las acciones del Script están basadas en objetos, de manera que los métodos están orientados al objeto en cuestión, ya sea un personaje, una habitación, un evento del ratón, etcétera.

La última versión del programa es la 3.0.1. Para el desarrollo del proyecto se han utilizado las versiones 2.7.2 y la 3.0.0.23.

La versión actual del AGS, la 3.x.x utiliza la tecnología .NET Framework para el editor (no es necesario para la ejecución del juego). Gracias a esto, se han subsanado muchas incomodidades de las versiones anteriores, como que por ejemplo sólo se podía tener un *script* abierto.

AGS Editor .NET (Build 3.0.0.23)

v3.0, January 2008

3 Diseñando el juego con AGS

3.1 Creando los personajes con AGS

En el AGS podemos crear distintos personajes, tanto para poder jugar con ellos, como para que formen parte del desarrollo del juego.

Para ello pulsamos el botón derecho del ratón sobre el menú Character, y a continuación seleccionamos *New Character* (en el caso de que queramos crear un personaje nuevo) o Import (en el caso de que queramos importar un carácter de tipo .cha). Se abrirá el menú *character* que tiene una apariencia como esta:

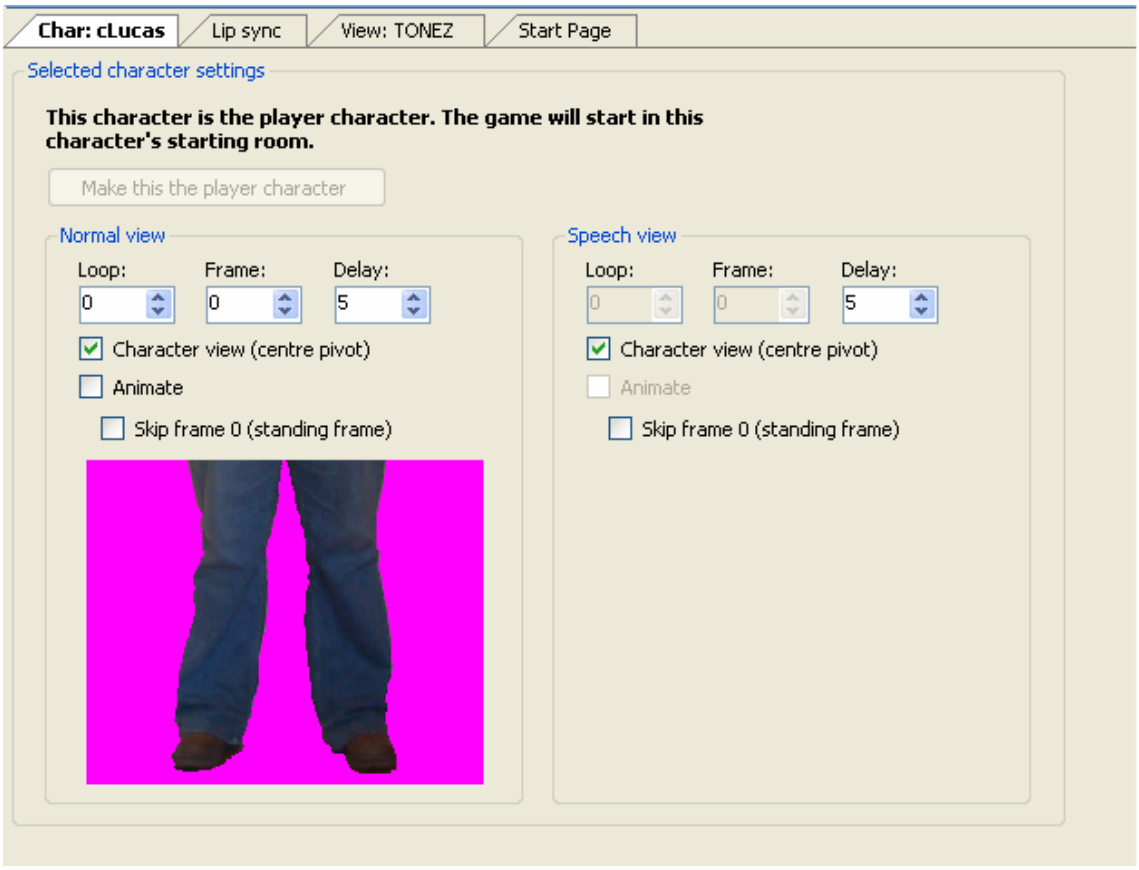

Figura 3.1: Pestaña de caracteres

Además aparece un menú como este:

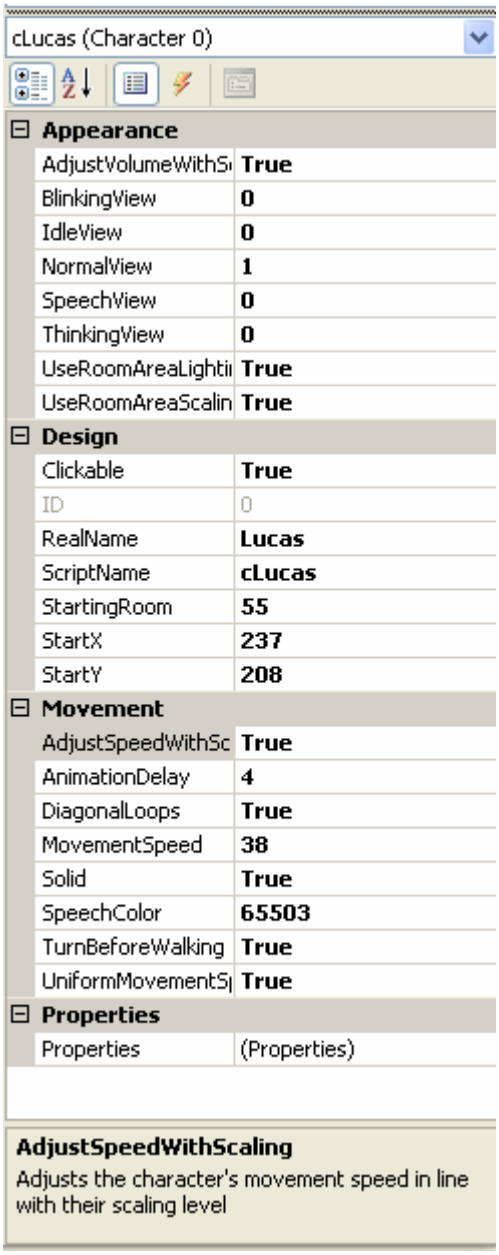

Figura 3.2: Menú de opciones de carácter

A continuación detallamos las opciones:

- ***Adjust volume with scaling:*** esta opción hace que el volumen se ajuste en función de la posición del personaje en la pantalla. Si está lejos, el volumen será más bajo, consiguiendo el efecto de lejanía.
- ***Blinking view:*** establece qué vista se mostrará intermitentemente para la animación del personaje cuando éste esté hablando.

- ***Iddle view:*** establece qué vista se utilizará para la animación del personaje cuando esté en modo espera, sin hacer ninguna actividad.
- ***Normal View:*** establece qué vista es utilizada para la animación normal del personaje caminando. La vista especificada aquí debe tener 4 u 8 bucles, y el primer fotograma de cada bucle es el fotograma para cuando el personaje se queda quieto. Para los personajes con 4 bucles: cuando se camina diagonalmente la dirección en línea recta más cercana es elegida para ser visualizada.

  Con un personaje con 8 bucles, todas las 8 direcciones son visualizadas dependiendo de la dirección del personaje.

  En el modo View se muestra qué bucle representa qué dirección -ej. "Loop 0 (down)" - “Bucle 0 (abajo)”.
- ***Full name:*** Introducimos el nombre que tendrá el personaje en el juego.
- ***Script name:*** Seleccionamos el nombre que utilizaremos en el Script del juego para referirnos al objeto personaje. Como todos los objetos en el AGS, comenzará con la letra “o” seguida del nombre que queramos darle. En nuestro caso, nuestro personaje principal se llama Lucas y el nombre de Script es “oLucas”.
- ***Starting room:*** establece la habitación en la que comienza el personaje. Si el personaje es el personaje jugador (player) en el momento de comienzo del juego, éste empezará en esa habitación.
- ***StartX y StartY:*** establecen las posiciones x e y en las que comenzará el personaje. Es especialmente importante seleccionarlas correctamente, ya que si las coordenadas que introducimos no corresponden a una zona del *“Walkable Area”* nuestro personaje no podrá moverse.
- ***Clickable:*** determina si el personaje es seleccionable o no con el ratón.
- ***Real Name:*** nombre real del personaje.
- ***Script Name:*** el identificador correspondiente al personaje dentro del *script* de programación. Como estándar, se usa una c (de Carácter) seguida del nombre del personaje. De esta manera el script es compatible con el editor de diálogos **Ceditor**.
- ***Adjust speed with scaling:*** determina si el personaje regulará o no la velocidad en función de donde esté en la pantalla. Si esta opción está activada, el personaje se desplazará más despacio al alejarse, creando así una sensación de lejanía.
- ***Animation delay:*** regula el tiempo de espera entre fotograma y fotograma de la animación. Cuanto mayor sea este parámetro, más despacio irá la animación.

- ***Diagonal loops:*** activa o desactiva el movimiento en diagonal del personaje.
- ***Movement speed:*** regula la velocidad de movimiento del personaje.
- ***Solid:*** determina si el personaje es traspasable o no por otros personajes.
- ***Speech colour:*** selecciona el color del texto correspondiente a los pensamientos y los diálogos de este personaje. Esta opción es especialmente importante para distinguir distintos personajes que intervienen en una misma conversación.
- ***Use area Scaling:*** esta opción permite configurar si la vista del personaje será proporcional o no a la zona de la habitación en la que se encuentre. Si está activada podremos establecer dos porcentajes con nombre *max* y *min*. El primero hará referencia al porcentaje del tamaño de la imagen original a la que será reescalada cuando el personaje se encuentre en la zona posterior de la pantalla del walkable area. El *min* será análogo pero para la parte posterior de la pantalla del walkable area.
- ***Use room area lighting:*** esta opción permite configurar si la vista del personaje tendrá en cuenta o no la luz que haya en la zona de la habitación en la que se encuentre.
- ***Turn before walking:*** si esta opción está activada, el personaje se girará hacia el personaje al cual vaya a hablar antes de comenzar.
- ***Interaction:*** el personaje, aún siendo el jugador, es un objeto como otro en AGS, así que podremos interactuar con él. En el menú *interaction* -que sin duda es uno de los más utilizados en la creación de un juego- podemos asociar distintas opciones a cada una de las acciones. Las más comunes son decir algo, cambiar de vista, cambiar de habitación, eliminar un objeto del inventario…

3.2 Creando objetos de inventario

Los objetos de inventario son aquellos que el jugador puede obtener a lo largo del juego, ya sea cogiéndolos de los escenarios, recibidos por parte de algún personaje, o conseguido por otros medios.

En las secciones *Como hacer diálogos* y *Funciones más importantes del juego* se muestran las diferentes formas de añadir un objeto al inventario del jugador en el transcurso del juego.

Para añadir un nuevo ítem en el menú derecho de AGS seleccionaremos *Inventory ítems*, pulsando sobre el botón derecho del ratón y seleccionando la opción *New inventory item* (véase en la figura).

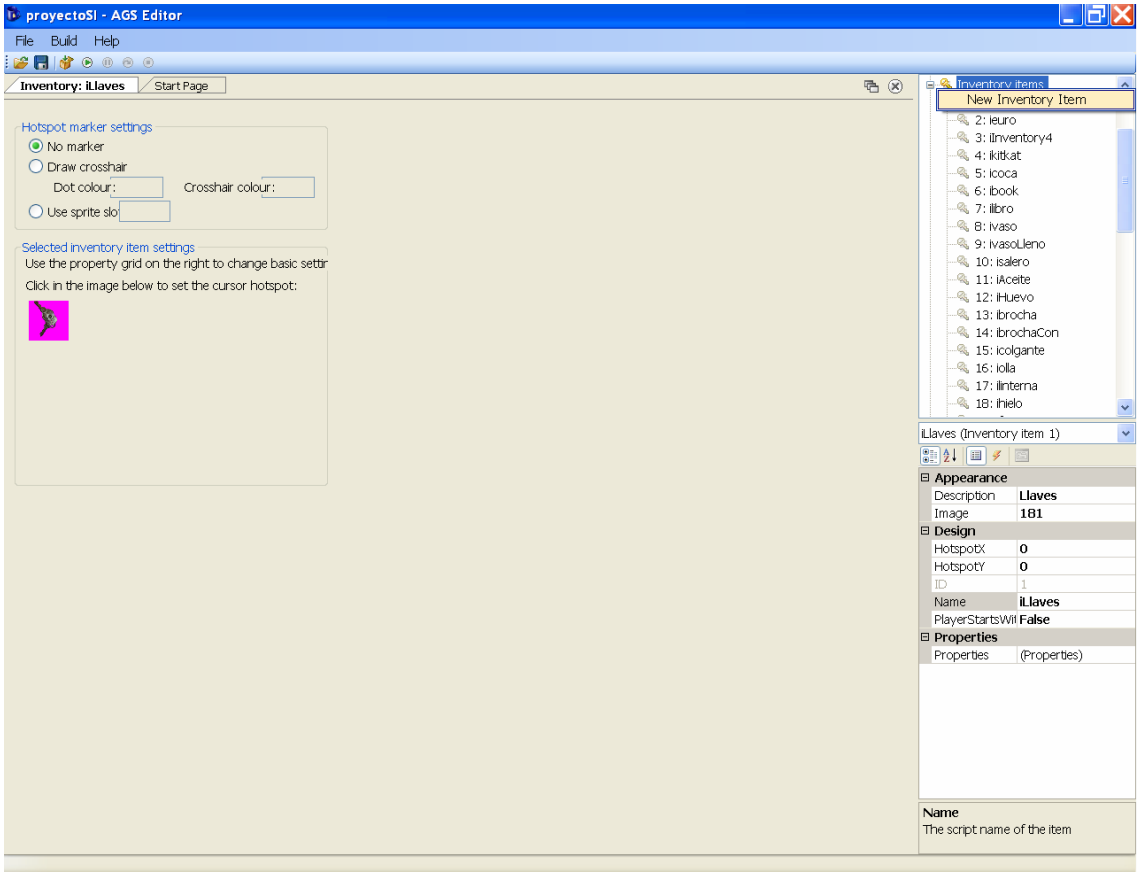

Figura 3.3: Pestaña de ítem de inventario

Como se puede observar en la figura, si desplegamos el menú *Inventory ítems* podremos seleccionar los diferentes objetos que ya hayamos añadido ordenados por ID y nombre en la lista desplegable.

En el menú principal se observa el sprite asociado a este ítem, si es que lo tiene. Es la imagen que el jugador verá en el inventario cuando adquiera el objeto.

En el menú inferior derecho se insertan las diferentes propiedades del objeto. Las más importantes son:

- Descripción: Nombre que recibirá el objeto.
- Image: Sprite asociado al objeto. Para cambiarlo seleccionar este campo y aparecerá el botón  . Pulsando sobre este, se abrirá la ventana de sprites para seleccionar uno. Si se conoce el número de identificación del sprite, también se puede escribir en este campo directamente.
- ID: Número de identificador
- Name: Nombre del objeto en la implementación (por convenio comienzan con la letra i).
- Player Starts with the item: Variable booleana que indica si el jugador comienza con este objeto en el inventario o no.
- Properties: Sección para añadir atributos al objeto si se desea.

Los objetos pueden tener asociados diferentes eventos que podremos ver y tratar pulsando sobre el botón  en el menú inferior derecho. Los diferentes eventos se presentan en la sección 4.1.3.

3.3 Cómo crear habitaciones en AGS

Una habitación en AGS es cualquier zona del juego, ya sea una habitación cerrada o un espacio abierto, y en la cual están los objetos con los que el personaje puede interactuar. Cada habitación tiene un fondo que puede ser una imagen o una animación.

Para hacer una habitación (en adelante "room") para un juego en AGS lo primero que debemos hacer es seleccionar el menú *Room* y a continuación seleccionar *New Room.*

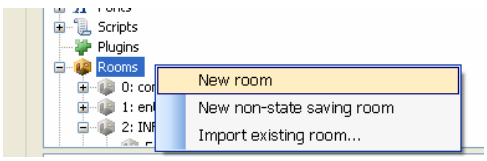

Figura 3.4: Creación de una nueva Room

Después necesitamos tener o crear un fichero de fondo para nuestra habitación. Nos vale cualquier imagen .bmp, .pcx, .png, .gif, tga o incluso .jpg [1]. Seleccionamos la opción *change* y seleccionamos una imagen. Con esto conseguimos establecer el fondo de la room.

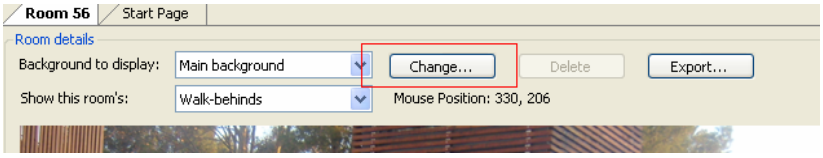

Figura 3.5: Cambio de imagen de fondo de una Room

Como opciones generales de cada habitación tenemos:

- ***PlayMusicOnRoomLoad:*** selecciona un sonido en particular cuando el personaje esté en esta habitación.
- ***SaveLoadEnabled:*** permite configurar la opción de carga y guardado de esta habitación.
- ***PlayerCharacterView:*** si desactivamos esta opción, no se mostrará ninguna vista del personaje **player**. Esta opción es muy usada en los *mini-juegos* ya que en la mayoría de ellos no necesitamos mostrar ningún personaje, pero la única forma de ir a una habitación en AGS es llevando a ella el personaje **player**.
- ***Player character view:*** permite seleccionar la vista a usar del personaje mientras esté en la habitación.
- ***MusicVolumeAdjustment:*** mediante esta opción seleccionamos el nivel de volumen de esta habitación.
- ***Description:*** podemos introducir la descripción de la habitación, que será el nombre que saldrá en el juego.
- ***BackgroundAnimationDelay:*** establece el retardo de las animaciones que forman el fondo de la habitación (opcional).

[1] Al comenzar la implementación del juego, la versión 2.72 de AGS no permitía introducir .jpg. Al migrar a la versión 3.0 descubrimos que admitía este formato, pero era incompatible con todo lo implementado previamente ya que utiliza una codificación distinta de los colores. Debido a esto hemos usado el formato .png para las imágenes del juego.

El *room editor* tiene un sub-menú para cada habitación llamado *Show this Room's*. Mediante este menú seleccionamos lo que queremos mostrar de la habitación.

Figura 3.6: Pestaña desplegable de contenidos de Room

3.3.1 Hotspots

Son las partes de la habitación donde el personaje puede interactuar con el menú. Para "pintarlas" seleccionamos las herramientas de dibujo que hay en la parte de arriba a la derecha. A continuación "pintamos" el área a la que luego añadiremos la acción asociada. No debemos preocuparnos, en el juego no saldrán estos "dibujos".

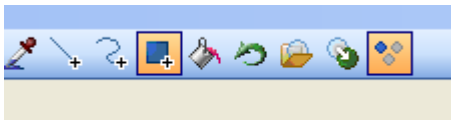

Figura 3.7: Barra de herramientas de las regiones

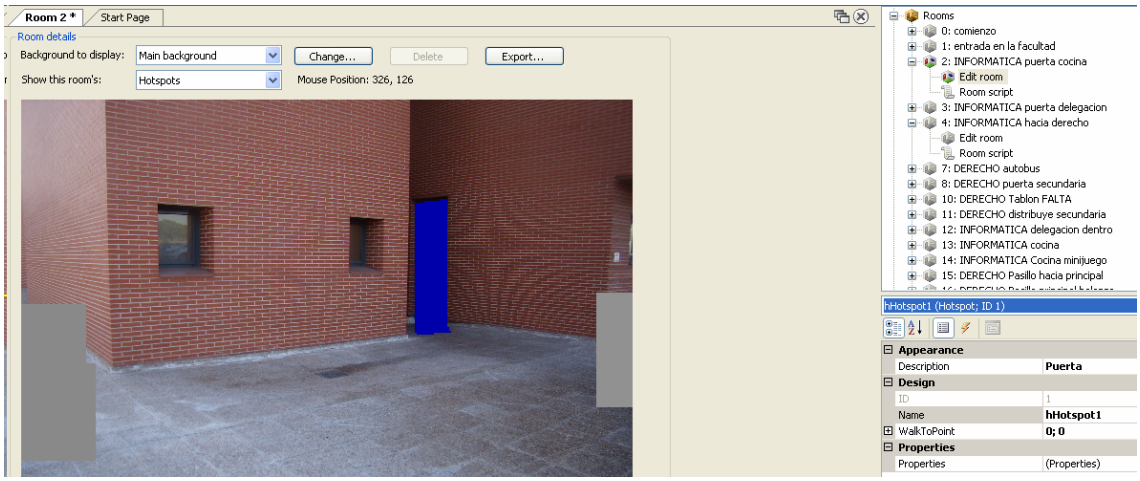

Figura 3.8: Menú de hotspots

Cada hotspot tiene, como de costumbre, su propio menú Eventos. Una característica que conviene resaltar de los hotspots es el WalkToPoint que establece las coordenadas a las cuales se desplazará el personaje.

3.3.2 Walkable Areas

Son las áreas donde puede caminar el personaje. Para cada área se define la escala del personaje cuando está dentro de la zona en cuestión. Para especificar las zonas haremos lo mismo que con los *hotspots*. Ahora, el color que sale es el azul.

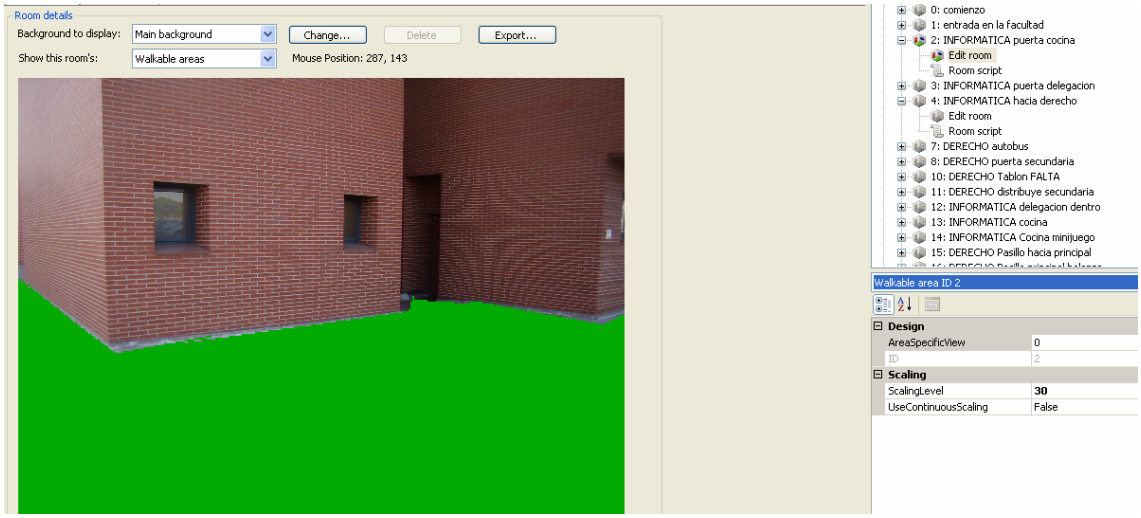

Figura 3.9: Ejemplo de una Walkable area

Una propiedad importante de los **walkable areas** es el escalado. Si seleccionamos *UseContinuousScaling* el personaje cambiará de tamaño en función de dónde esté en el *walkable area*, haciéndose más pequeño si está más arriba, más grande si está más abajo, creando así una sensación de proximidad-lejanía.
Para esto, marcamos los valores *MaxScalingLevel* y *MinScalingLevel* que determinan el porcentaje del tamaño máximo (cuando esté más cerca) y el mínimo (más lejos) de la vista del personaje.

Si no seleccionamos el escalado continuo, el personaje usará el mismo porcentaje de tamaño en todo el *walkable area.*

Es importante resaltar que es recomendable crear unas vistas con un tamaño más grande del que se va a utilizar, y usar un valor del nivel de escalado pequeño, para evitar así que cuando usemos valores mayores la imagen pixele.

3.3.3 Walk-behinds

Son máscaras que ocultan al personaje. Esto es útil para crear una sensación de realismo. Si por ejemplo el personaje pasa por detrás de una mesa, no querremos que se dibuje todo sino la parte que no tapa la mesa. Para cada Walk-behind se debe establecer una línea base (un baseline), que es la altura a la que debe estar el personaje para ocultarlo detrás del walk-behind correspondiente. La altura es el punto más bajo de la vista de un personaje.

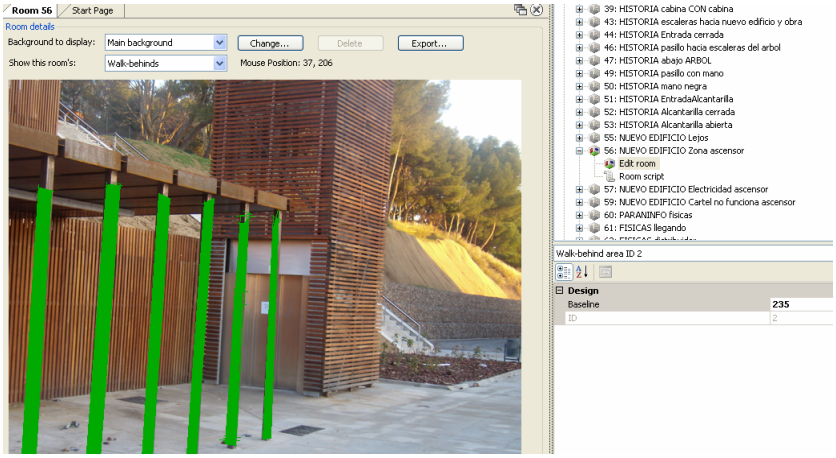

Figura 3.10: Ejemplo de un Walk-Behind area

3.3.4 Regions

Una vez seleccionada, utilizando los Eventos de la región podemos programar qué ocurrirá cuando el personaje entre, salga o permanezca en la región seleccionada.

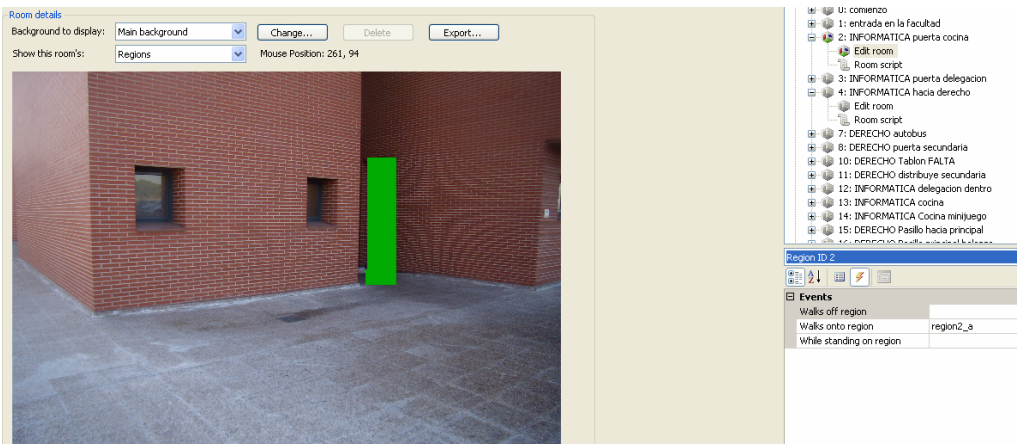

Figura 3.11: Ejemplo de región

3.3.5 Objects

Aquí podemos añadir los objetos que queremos que estén en la room. El personaje puede interactuar con ellos e incluso cogerlos. Para crear un objeto seleccionamos *New object.*. Para moverlo basta con arrastrarlo, y para fijarlo en una posición temporalmente se marca la opción de "*Lock this object in position*". Al igual que un hotspot se definen algunos eventos, imagen, propiedades personalizables, y una línea base que determina cuando está el objeto delante o detrás del personaje.

Sin embargo, los objetos en AGS no pueden añadirse directamente al inventario. Para añadirlos al inventario es necesario crear un *Inventory Item* y asociarle la función *player.AddInventory (inventory_item)* al evento del objeto en la habitación, y a su vez poner en no visible el Object en la room.

3.4 Creando los escenarios con AGS

Los escenarios del juego se componen de habitaciones (rooms) que están enlazadas unas con otras creando así un entorno por el cual los distintos personajes del juego pueden moverse.

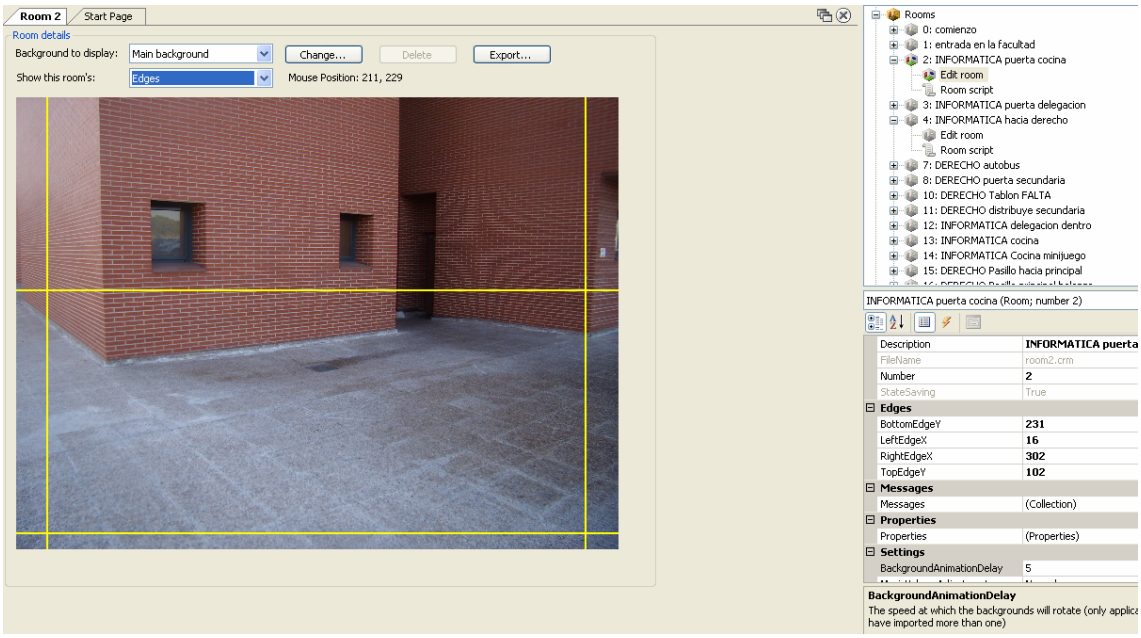

Figura 3.12: Menú de edición de una Room

Como es habitual en AGS, podemos programar distintas opciones de juego con el menú *Events*, los eventos están detallados en la sección 4.1.3.

Las cuatro opciones básicas de movimiento son:

- ***Walk off left edge:*** código que se ejecuta cuando el personaje se mueve al extremo izquierdo de la habitación.
- ***Walk off right edge:*** código que se ejecuta cuando el personaje se mueve al extremo derecho de la habitación.
- ***Walk off bottom edge:*** código de la acción que se ejecuta cuando personaje se mueve al borde inferior de la habitación.
- ***Walk off top edge:*** código la acción que se ejecuta cuando el personaje se mueve al borde superior de la habitación.

A estas opciones generalmente les asociaremos la siguiente acción:
Personaje.ChangeRoom (i, x, y)
que hará que el personaje en cuestión vaya a la habitación i, en las coordenadas (x,y), y así podrá moverse a través de las distintas habitaciones que componen el escenario de juego. Si el personaje que cambia de habitación es el **player.** En el momento de hacer el cambio el juego cambiará a esa misma habitación. Si es un personaje secundario, éste cambiará de habitación permaneciendo el juego en la habitación donde esté el personaje **player**.

Sin embargo, muchas veces tenemos que realizar una u otra acción en función del estado de una variable del juego. Por ejemplo, podemos querer comprobar que la variable de fase sea 3 para poder entrar en una determinada habitación. En este caso, asociaremos a la acción correspondiente un *script* de la forma:

```
If (fase > 3) {
        player.ChangeRoom(i, x, y)
        }
```

Para establecer los márgenes que representan el left, right, bottom y top edge usamos las líneas amarillas que aparecen en la vista general de la habitación. Podemos desplazar estas líneas en su eje vertical u horizontal, seleccionándolas con el ratón y arrastrándolas hasta las coordenadas que deseemos.

3.5 Sprites en AGS

Todas las imágenes de objetos, elementos del inventario e incluso los personajes del AGS están basados en el uso de *sprites*. Éstos se crean utilizando el menú "*Sprite Manager*" en la parte izquierda del árbol de acciones del AGS.

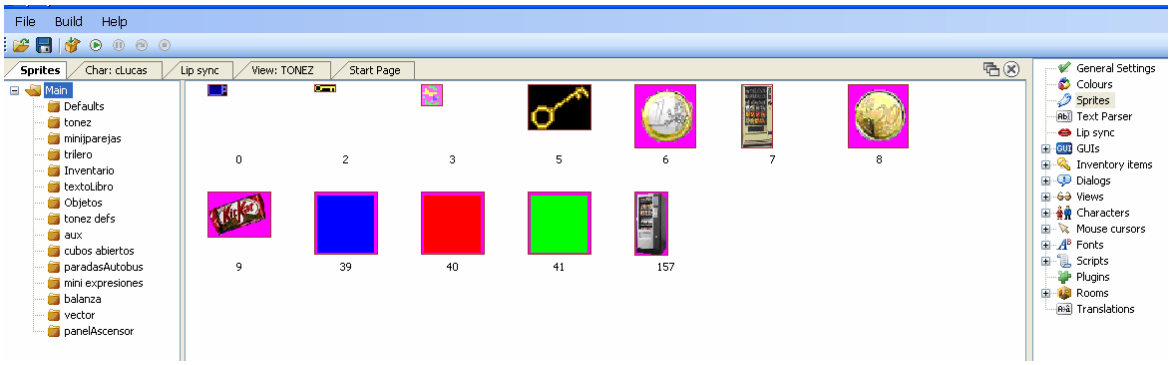

Figura 3.13: Menú de Sprites

Seleccionando con el botón derecho y escogiendo "Import New Sprite from file" se abre un menú de selección del archivo de imagen que se va a importar. Al seleccionar una imagen y pulsar **ok** pasaremos al menú de importación.

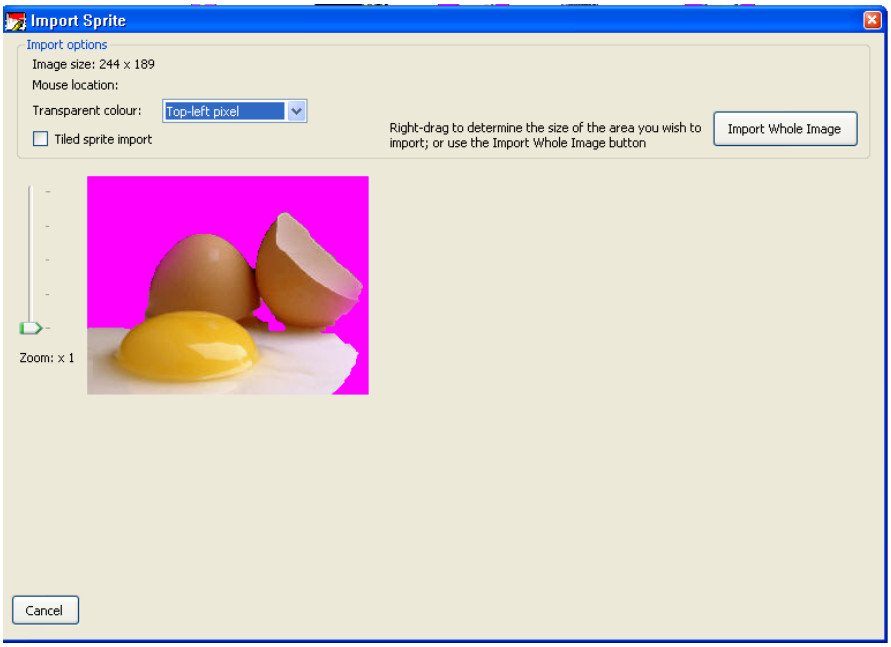

Figura 3.14: Menú de importación de imagen

Por otro lado encontramos en la pantalla el botón “Import whole image”. Si pulsamos sobre este se importará la imagen que hayamos seleccionado entera.

Mediante la opción del zoom podemos seleccionar el tamaño que tendrá la imagen en el juego, en relación al tamaño original. Es recomendable seleccionar 1x ya que si seleccionamos un tamaño mayor, la imagen pixelaría.

Sin embargo, de esta manera importamos imágenes rectangulares, y rara vez nuestros objetos tienen esta forma. Para conseguir que nuestro sprite muestre sólo la imagen del objeto o persona, tenemos que crear un fondo que el AGS ignorará, y así sólo mostrará la parte correcta de la imagen. El color que el AGS toma como fondo es el color Rosa, que se consigue con la siguiente combinación:

ROJO: 255
VERDE: 0
AZUL: 255

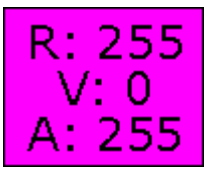

Figura 3.15: Colores de fondo de sprite

Como ejemplo, mostramos la imagen que utilizamos en el juego para unas llaves.

Figura 3.16: Ejemplo de sprite

El desplegable “Transparence color” nos permite seleccionar qué tipo de transparencia queremos utilizar.

- **Palette Index 0:** todos los colores con color índice 0 serán transparentes.
- **Top/Botton – Left/Right index:** toma como color transparente el píxel en la esquina seleccionada.
- **Leave as is**: todos los colores compatibles con las transparencias AGS serán transparentes (todos los que contengan los componentes a 0 o a 255).
- **No transparency:** Importa el sprite completo sin transparencias.

Para la creación de los sprites de los personajes, hemos utilizado una sábana verde que usábamos a modo de ***croma*** que poníamos de fondo a la hora de fotografiar los personajes.

Figura 3.17: Fotografía para hacer sprite de personaje

Después, con un software de retoque fotográfico, seleccionábamos las capas de color verde y las eliminábamos, sustituyendo todos esos píxeles por unos de color rosa.

Figura 3.18: Fotografía convertida en sprite

De esta manera se facilita mucho la creación de las transparencias, y así podemos utilizar fotos en cualquier habitación del juego independientemente del fondo que tengan éstas.

3.6 Creando vistas en AGS

Las vistas, o animaciones, son bucles de imágenes que pueden ser usadas tanto para los movimientos de los personajes, como para hacer más dinámicos algunos objetos o partes de una habitación. Si por ejemplo queremos poner un perro que ladre, pero no queremos que sea ningún personaje de juego, utilizaremos una vista para crear la animación del perro, y se la asignaremos a un objeto *oPerro* que esté en una determinada habitación.

Para la gestión de las animaciones utilizamos el menú *Views* que tiene una apariencia como esta:

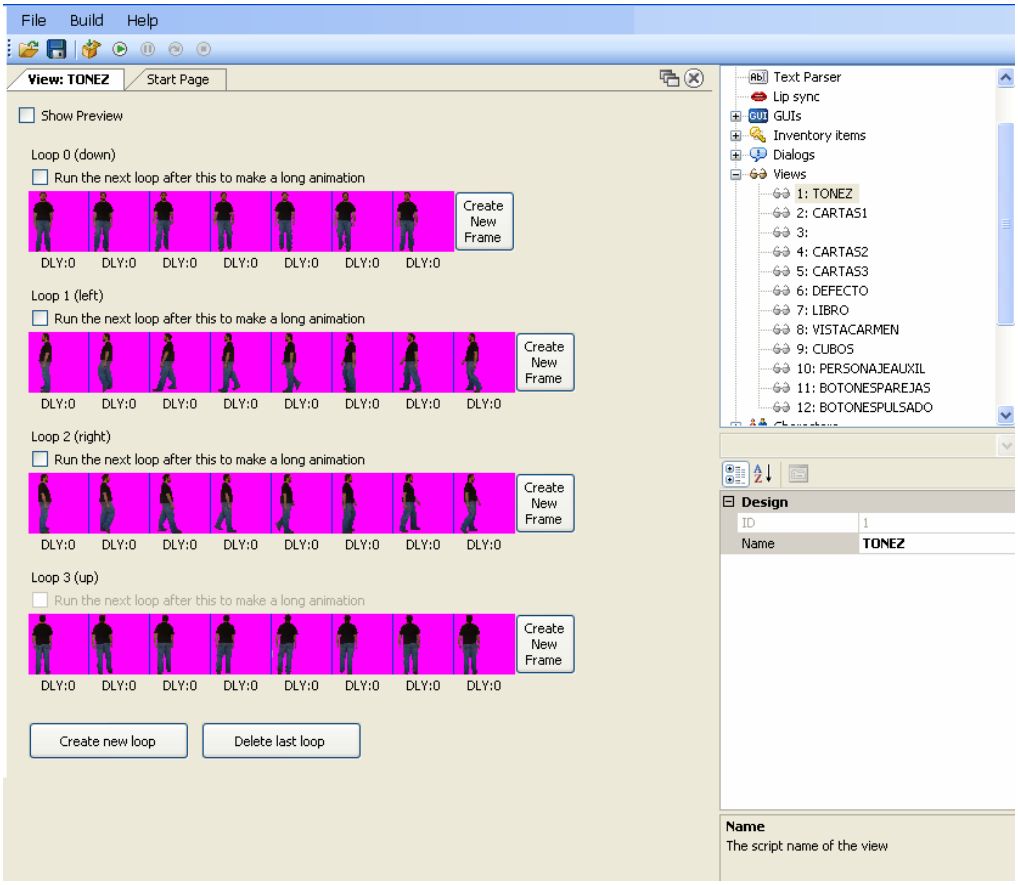

Figura 3.19: Menú de vistas

El menú *Views* nos permite gestionar todas las animaciones que se usarán en el juego.

Pulsando el botón *New View*, crearemos una nueva vista, que aparecerá con un solo bucle el cual aparecerá sin ningún *sprite*. A cada *loop* dentro de la vista debemos asociarle una serie de *sprites* (que previamente hayan sido añadidos en el gestor de sprites) que son los que se repetirán uno detrás de otro mientras se esté ejecutando el movimiento en cuestión. Nosotros hemos utilizado una media de 8 imágenes por movimiento, con las cuales cubrimos toda la acción que corresponde a dar un paso en una determinada posición.

Debajo de cada fotograma de los *loops* hay tres opciones:

- ***SPD: 0*** es la velocidad relativa del fotograma. Este número agrega un retardo al período de tiempo en que el fotograma es mostrado, por lo que un

número más grande hace que se retarde más tiempo. Podemos utilizar números negativos para eliminar retardos.

- ***NO SND*** nos permite agregar un número de sonido que se ejecutará cuando este fotograma se visualiza en la animación. Un buen ejemplo sería, por ejemplo, asociar un sonido de pisada que se ejecutase en cada paso.
- ***NORMAL*** cambia si el fotograma debe ser visto normalmente o invertido de izquierda a derecha. Si en vez de normal se selecciona "*FLIPPED*" los fotogramas se mostrarán invertidos con lo que a partir de un movimiento hacia un lado podremos tener el equivalente al lado contrario.

Cada vista contiene a su vez distintos *loops*, que sirven para animar los distintos movimientos del personaje u objeto. Los más comunes son los cuatro movimientos principales: **up** (cuando el personaje se mueve hacia arriba en la pantalla), **down** (cuando el personaje se mueve hacia abajo en la pantalla), **left** (cuando el personaje se mueve hacia la izquierda en la pantalla) y **right** (cuando el personaje se mueve hacia la derecha en la pantalla). Además de estos hay otros como los movimientos diagonales. Sin embargo, éstos no son imprescindibles ya que AGS combina los cuatro principales para crear el resto de movimientos. Así, si un personaje se mueve hacia la parte superior derecha de la pantalla, si está creado el *loop* up-right se mostrará éste, y si no está creado, se mostrará una combinación del *loop* up con el *loop* right.

Para simular el movimiento de la boca y la cara al hablar, AGS nos facilita una herramienta denominada *Lip Sync*, que está comentada más adelante.

3.7 Creación de diálogos con AGS

AGS tiene su propio lenguaje, distinto al del script para crear diálogos interactivos en el juego. Para crearlos nos hemos ayudado del programa CEditor, que genera un script importable desde AGS.

En primer lugar veremos las instrucciones del lenguaje y posteriormente como se trabaja con este y con CEditor, y un ejemplo de conversación.

3.7.1 Instrucciones

Las instrucciones se dividen en 3 tipos:

- Intervienen en el hilo de la conversación:
  - Option-on X: La opción X del diálogo se hace visible.
  - Option-off X: La opción X del diálogo se hace no visible.
  - Option-off-forever X: La opción X del diálogo se hace no visible para siempre, aunque se vuelva a poner Option-on, o se salga y se entre en el diálogo otra vez.
  - Goto-Dialog X: Se va al diálogo X.
  - Return: Se devuelve el control al jugador para que elija opción.
  - Stop: Finaliza el diálogo.

- Intervienen en el hilo del juego:
  - New-room X: El personaje jugador cambia a la habitación X, interrumpiéndose el diálogo.
  - Goto-Previous: El personaje jugador cambia a la última habitación en la que ha estado, interrumpiéndose el diálogo.

- Generan eventos externos:
  - Add-inv X: Se añade el objeto con identificador número X al inventario del personaje jugador.
  - Lose-inv X: Se elimina el objeto con identificador número X del inventario del personaje jugador.
  - Play-sound X: Se reproduce el sonido número X de la biblioteca de sonidos.
  - Run-script X: Se llama a la función Dialog Request, del script global, con parámetro X.
  - Set-globalint GI VAL: Se asigna un valor VAL a la variable global con número de identificador GI.
  - Set-speech-view Name X: Cambia la vista del carácter con nombre Name a la vista número X.

3.7.2 Creando una conversación con CEditor

Por supuesto utilizar CEditor no es necesario, sin embargo nos facilitará bastante la tarea, no desde el aspecto de implementación si no desde el visual. CEditor nos ayuda a crear el árbol de decisiones de la conversación, siendo más fácil de este modo implementarla posteriormente.

El lenguaje para conversaciones funciona por Opciones que puede seleccionar el jugador a la hora de conversar. Al elegir una conversación se muestra por pantalla lo que el personaje jugador quiere decir, y la contestación del otro personaje con el que se está conversando. El programador del diálogo tiene que escribir todos los cambios de visibilidad en cada opción que desee el diálogo asociado a esta y, posteriormente (o concurrentemente si se desea) ir creando el hilo de la conversación por medio de las funciones OPTION-ON, OPTION-OFF-FOREVER, haciendo que en un determinado momento las distintas opciones estén visibles o no. En principio todas las opciones están no visibles, y es necesario para poder comenzar una conversación seleccionar cuales estarán visibles al ejecutarse esta.

En todas las conversaciones existe una opción llamada Startup, que se ejecutará siempre al comienzo de una conversación, antes de dar a elegir al jugador distintas opciones para conversar. Esta opción se suele utilizar para que hable el personaje narrador (si se quiere utilizar este personaje, utilizar el nombre Narrator), aunque también puede hablar cualquier otro personaje. Por supuesto, si no se desea que haya una introducción a la conversación bastará con dejar vacío el texto de la opción startup. De esta manera al comienzo de la conversación directamente se darán a elegir las opciones iniciales al jugador.

En la imagen se muestra la pantalla CEditor cuando comenzamos a crear una conversación. Hay dos partes claramente diferenciadas. En la superior se muestra el árbol de opciones según se va generando. En la parte inferior vemos diferentes cuadros de texto y opciones: También hay dos Cuadros de texto. El primero (en la imagen pone startup) es el diálogo de opción, que se verá en el cuadro de opciones cuando se tenga que elegir una. Como se puede ver, en la opción “startup” no se da esta opción. Debajo

de este texto se puede dar nombre al diálogo (en este caso New Topic 1) que es común a todas las opciones. Por último, el cuadro inferior, se puede escribir el texto asociado a la opción que hayamos seleccionado en la parte del árbol de opciones.

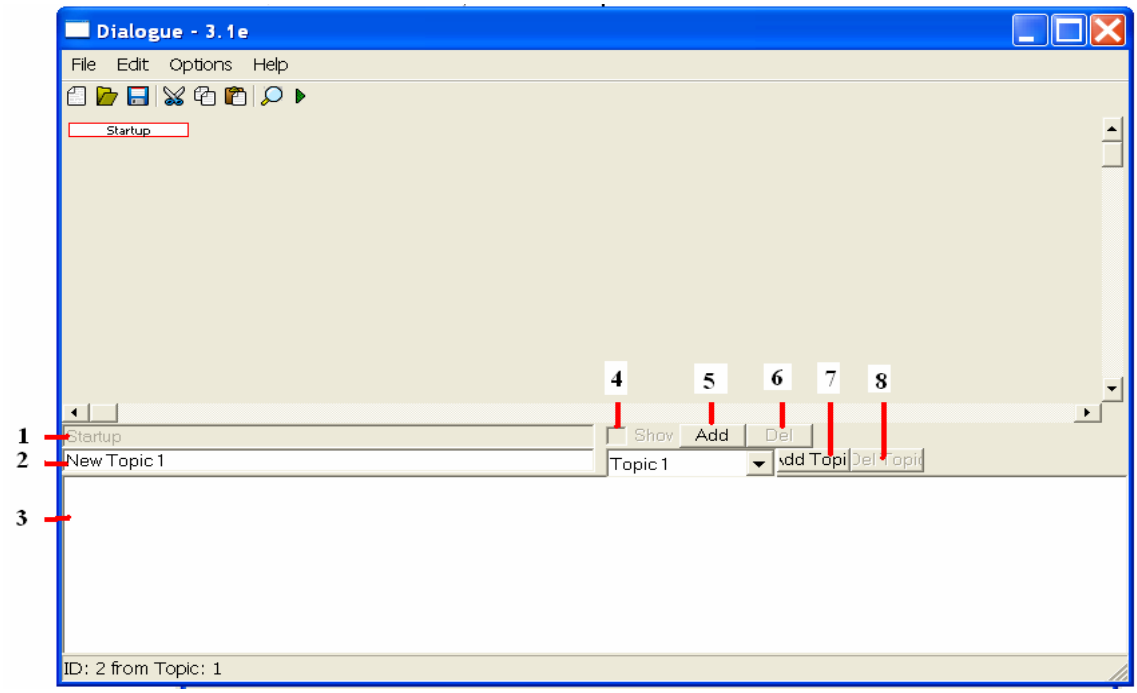

Figura 3.20: CEditor

1. Cuadro de texto en el que se pone el texto que se quiere que aparezca en el cuadro de elección de opciones. (en la opción startup esta opción está deshabilitada).
2. Cuadro de texto para dar nombre al diálogo.
3. Cuadro de texto en el que se escribe el código y el texto de la conversación
4. Selecciona si una opción es inicialmente visible o no. Para que una conversación pueda ejecutarse tiene que haber inicialmente alguna opción con este campo marcado.
5. Botón para añadir un hijo a la opción que haya seleccionada en la parte superior de la pantalla.
6. Botón para eliminar la opción que está seleccionada y todos sus hijos.
7. Crea una nueva conversación.
8. Elimina una conversación.

Para añadir y borrar opciones, hacer clic en los botones Add y Del. De esta manera se crearán y eliminarán hijos sobre la opción seleccionada. Si estando en el estado de la imagen anterior pulsáramos 2 veces sobre Add tendríamos esta pantalla:

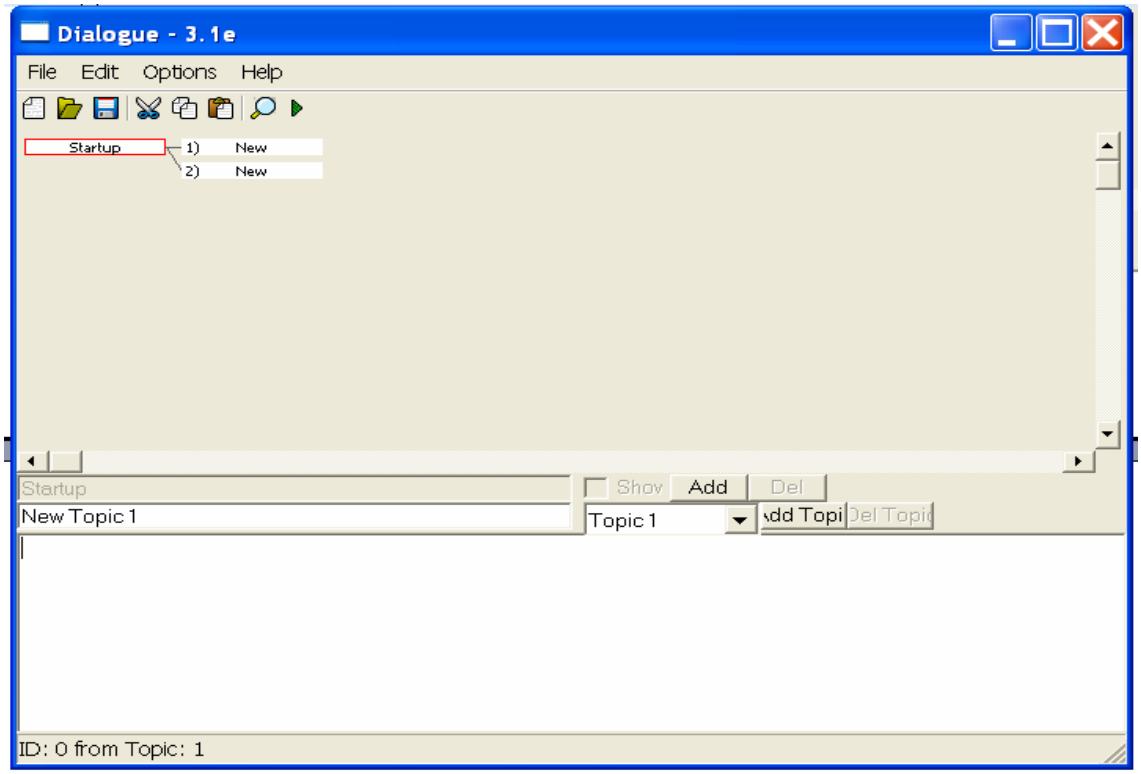

Figura 3.21: Ejemplo de árbol de decisión de un diálogo

Es importante reseñar que el árbol que vamos creando **no es vinculante** con el hilo de la conversación. El hilo se dirige con las instrucciones habilitadas para ello. El árbol es una simple ayuda visual.

En el punto en el que estamos habría que escribir el texto que se desea aparezca por pantalla al seleccionar las distintas opciones, en la imagen se pueden ver las 2 opciones ya con su texto (se puede ver la segunda) y como la segunda tiene asociadas otras dos opciones. La primera tiene una instrucción STOP, lo que implica que si se selecciona la primera opción, tras aparecer el texto pertinente terminará la opción (cEditor nos muestra esto poniendo el recuadro del árbol de esa opción en rojo). La segunda, para la cual hemos creado dos hijos, contiene un breve texto por parte del jugador (identificado con player), y de otro carácter (Carmen). Antes de cada frase es

necesario decir quien la dice por medio del nombre que tenga en AGS seguido del signo de puntuación ":".

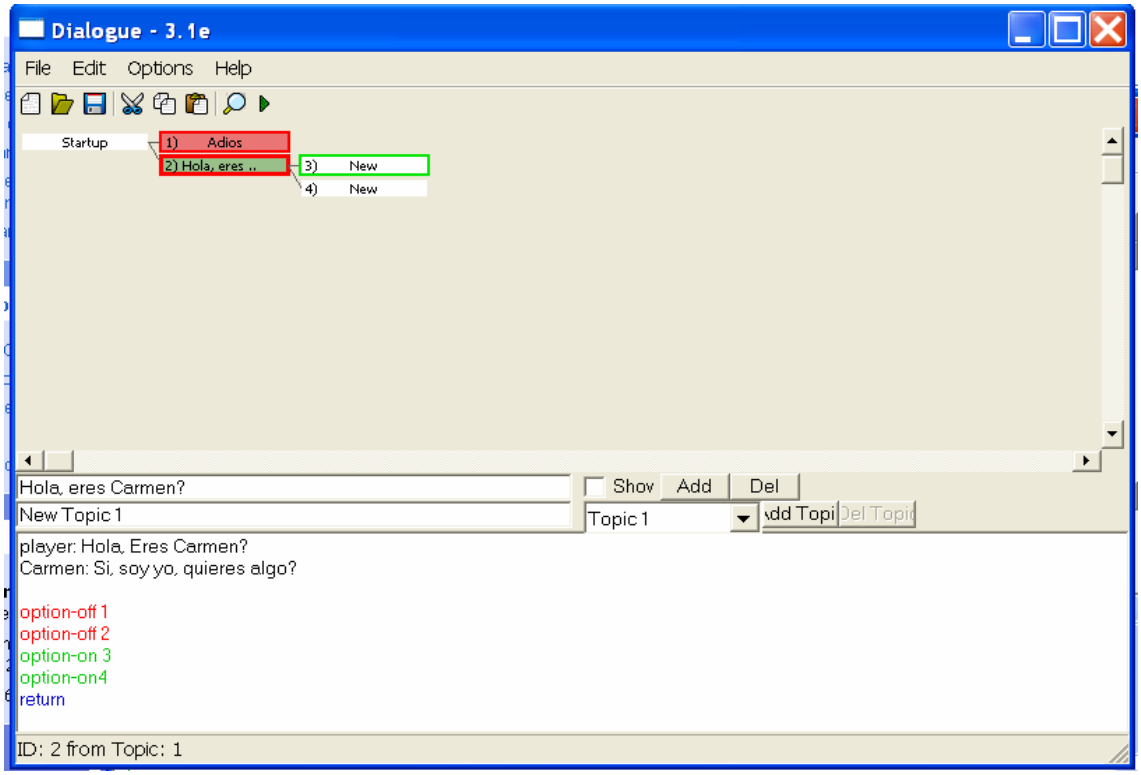

Figura 3.22: Ejemplo 2 de árbol de decisión de un diálogo

Como se puede observar, tras el texto que se quiere que diga cada personaje es necesario controlar el flujo de la conversación, escribiendo las opciones si se quiere cambiar de estado (de visible a no visible y viceversa), si se desea que no cambie el estado de una opción (ya sea visible o no), no es necesario especificarlo.

Por último, tras la gestión de la visibilidad de las opciones, es necesario escribir STOP si se quiere acabar la conversación ahí, RETURN, si se quiere que aparezcan en el cuadro de elección de frases las opciones que haya seleccionadas como visibles en ese momento, o GOTO-DIALOG si se quiere cambiar a otro dialogo.

Para acabar se escribiría el texto de las dos opciones hijas de la opción 2. Que se pondrían visibles cuando se ejecutase esta. En la imagen podemos observar como la conversación ya tiene todas las opciones completadas. Las opciones 3 y 4 también

acabarían con la conversación. En la cuarta (la que aparece en la imagen), el jugador recibiría un bocadillo en su inventario (identificado con el número 10 en gas)

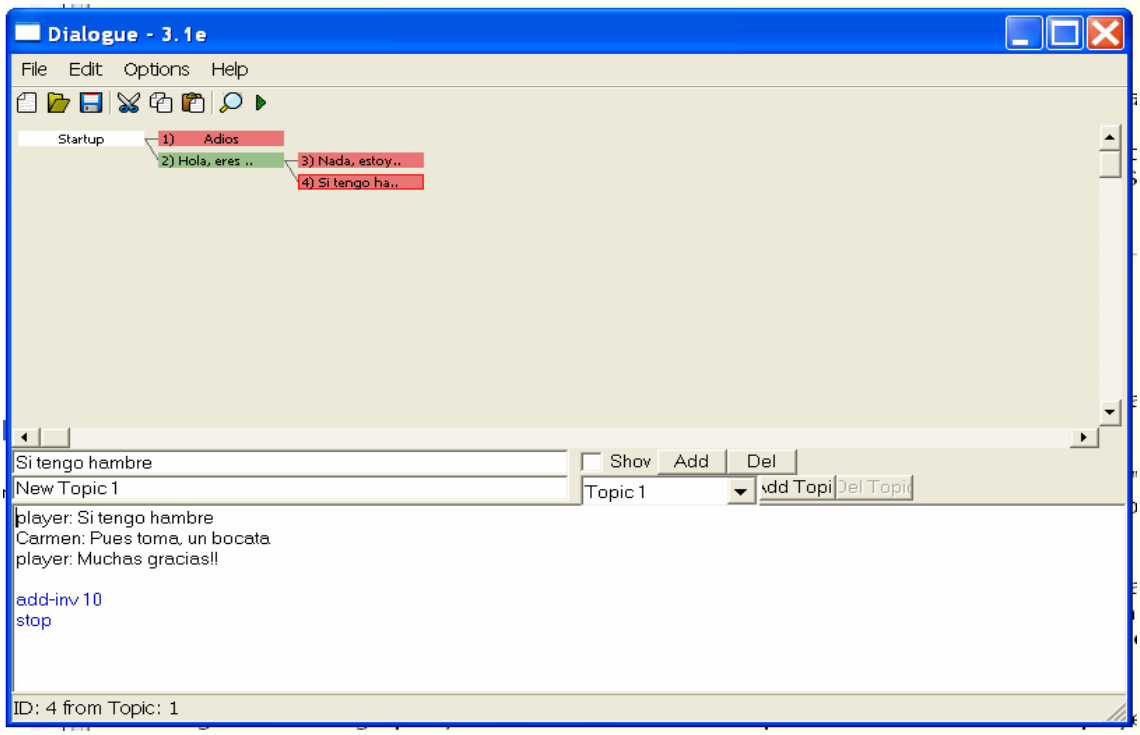

Figura 3.23: Ejemplo 2 de árbol de decisión de un diálogo

3.7.3 Importando una conversación de CEditor a AGS

cEditor genera un script que es fácilmente importable para luego tener la conversación disponible en nuestro juego. Para ello debemos seleccionar en la barra de menú "Options→ view script". Aparecerá una ventana de texto, el cual tendremos que seleccionar todo.

En AGS hacer clic derecho en Dialogs y seleccionar "new dialog". Aparecerá una ventana como la de la imagen.

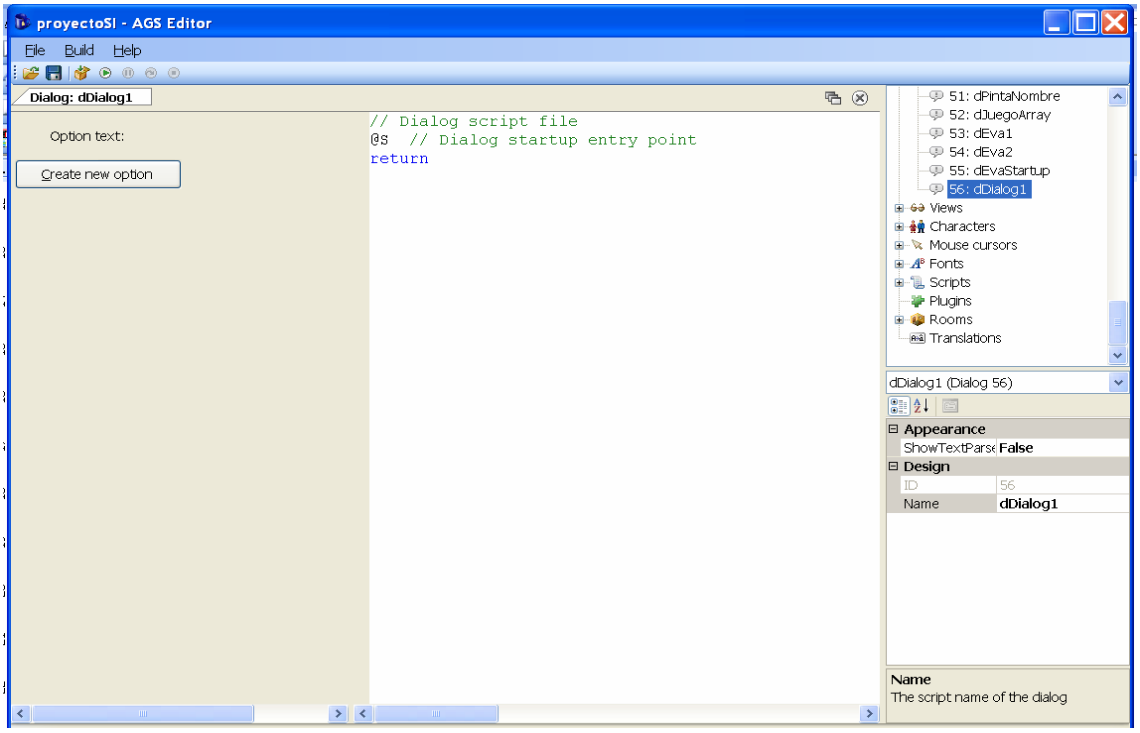

Figura 3.24: Menú de creación de diálogos en AGS

En la parte inferior derecha podremos dar un nombre al dialogo. En la parte central hay que pegar todo el texto que ha generado cEditor. Por último, en la parte izquierda hay que crear tantas opciones como opciones tenga la conversación, pulsando el botón “Create new option”, y escribir el texto que se quiera que aparezca en el cuadro de elección del diálogo. Este texto debe coincidir con el que escribimos en el campo 1 del cEditor para cada opción. Para ayudarnos, este texto aparece al principio del código generado por cEditor. Tendríamos que acabar con algo como en la imagen:

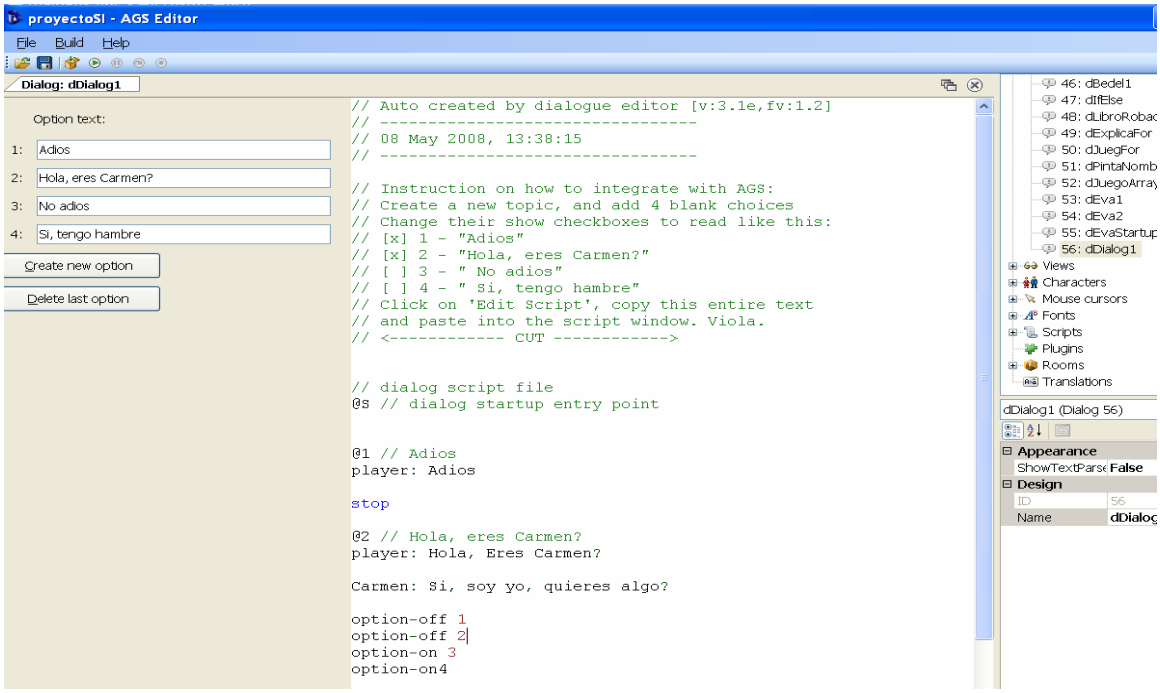

Figura 3.25: Ejemplo de código de diálogo en AGS

Por último seleccionar en la columna show que opciones son iniciales y cuales no. En la columna “Say” se elige si cuando se selecciona la opción, se quiere que el personaje diga (aparezca por pantalla) el texto que aparece en el cuadro de dialogo de selección, o no.

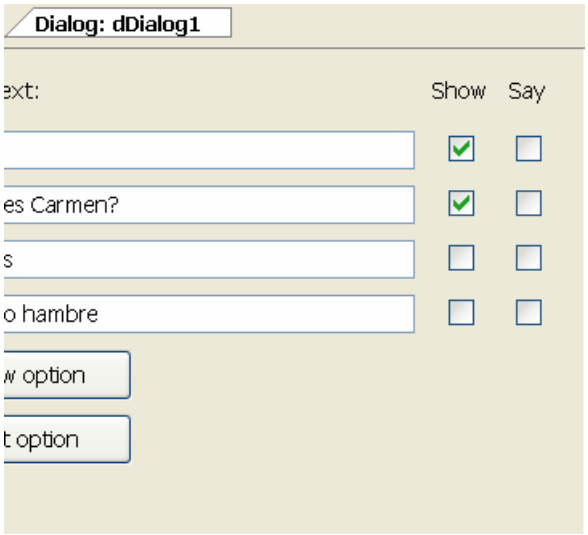

Figura 3.26: Sección del menú de Opciones en un diálogo

3.8 Edición de las Interfaces Gráficas de Usuario

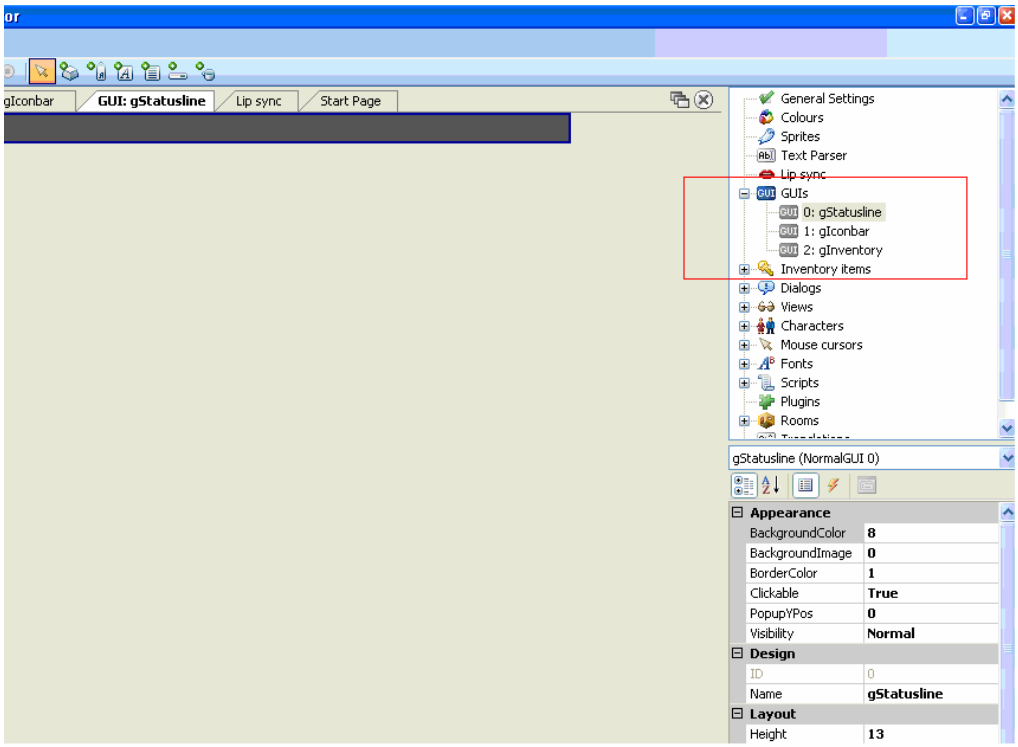

Figura 3.27: Menú de edición de GUIs en AGS

Por defecto, la interfaz del juego se configura para actuar como la interfaz "apuntar-y-seleccionar" de Sierra perteneciente a sus juegos de 1990-93. Para cambiar la interfaz debemos ir al panel "GUIs".

La interfaz del juego se divide en GUIs. Cada GUI es una región rectangular de la pantalla que es dibujada sobre la escena de fondo. Cada una puede ser configurada para:

- O bien estar siempre visualizada (por ejemplo la línea-de-estado de Sierra) denominada STATUSLINE en la plantilla por defecto.
- O bien emerger cuando el ratón se mueve a una determinada posición (ej. La barra-de-iconos de Sierra). Lo podemos ver ejecutando el juego de ejemplo y llevando el ratón hacia la parte superior de la pantalla.
- O bien emerger sólo mediante comandos de script que definamos nosotros mismos.

La interfaz predeterminada está formada por dos GUIs - la línea de estado (STATUSLINE), la barra de iconos (ICONBAR).

Pulsando en la barra de herramientas, en GUI, podemos ver varios botones:

- Import All GUIs. Importa de un fichero una GUI completa creada en otro juego.
- Export All GUIs. Exporta la interfaz entera (ej. todos los GUIs, más los gráficos de los botones) a un archivo.
- Import another GUI. Importa un elemento de una GUI creada en otro juego
- Export this GUI. Exporta a fichero la GUI seleccionada actualmente.
- Edit interface_click. Es un script que es llamado cuando el jugador pulsa un botón en la interfaz. Se le pasan dos parámetros, la GUI sobre la que se ha interactuado y el botón de esta que se ha pulsado.

Sobre la derecha de la pantalla se muestra el elemento de GUI actual, además de una ventana flotante. Esto nos permite editar las diferentes propiedades del GUI, y funciona como la ventana de propiedades de cualquier IDE. En la ventana de Propiedades, podemos cambiar el color de fondo del GUI, establecer una imagen de fondo, y establecer la ubicación y el ancho / alto entre otras cosas.

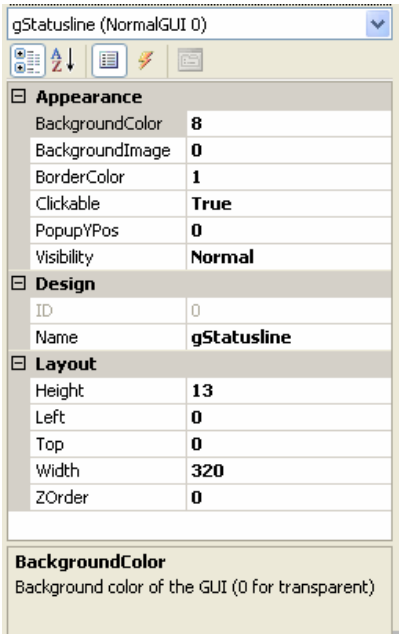

Figura 3.28: Menú de opciones de edición de GUI

La propiedad "Visible" le permite establecer cuándo el GUI se visualizará. Por defecto es "Always" (Siempre), que es como la línea de estado de Sierra: siempre presente en la pantalla. La opción "Script only" (Sólo script) significa que el GUI estará

inicialmente desactivado y deberá ser activado mediante los comandos del script de texto. Con esta opción, el juego se pausará cuando se visualice el GUI.

Si no se desea este comportamiento, se tiene que configurar en "Always" y utilizar GUIOff en la función game_start.

La opción "Mouse YPos" significa que el GUI sólo aparece cuando la posición vertical del ratón se mueva por encima de la coordenada-y establecida con la opción "Popup-YP" (Por ejemplo cuando el ratón baje a la zona inferior de la pantalla).

La casilla "Clickable" en la parte superior de la pantalla nos permite establecer si el GUI y los botones en él responden a selecciones de ratón. Está activada por defecto, pero si la desactivamos y el jugador hace clic sobre el GUI, el juego en realidad procesará el clic como si el jugador hubiera hecho clic detrás del GUI en la pantalla propiamente dicha.

Esta función, que puede parecer inútil, sí que serviría en el caso de GUIs transparentes que son utilizados solamente para mostrar información.

3.8.1 Botones del GUI

Para proporcionar interactividad con la interfaz se utilizan botones. No hay límites para el número de botones por GUI, pero el juego sólo puede contener un total de 80 botones de GUI.

Para agregar un botón, hay que hacer click en el botón "Add button" (Agregar botón), y luego arrastrar un rectángulo con el ratón en el GUI. Se visualizará como un botón de texto, con el texto “New button” escrito encima. La ventana de Propiedades ahora muestra las propiedades de su botón en lugar de las del GUI.

Figura 3.29: Ejemplo de botón para GUI

Utilizando la ventana de Propiedades podemos en su lugar establecer una imagen para el botón, y puede además establecer otros varios atributos auto-explicativos.

Utilizando el atributo “Left click” podemos establecer qué ocurre cuando el jugador hace clic sobre el botón. Éste puede ser configurado en "Do nothing" (predeterminado), y además en “Set mode”, que cambia el modo de puntero especificado en la propiedad "New mode number" (Número del nuevo modo). La otra opción, “Run script” (Ejecutar programa), ejecuta la función interface_click del script de texto, pasando el número de GUI y el número de botón que hemos seleccionado.

Para borrar un botón, lo seleccionamos y luego presionamos la tecla Delete (Suprimir) del teclado.

3.8.2 Texto de interfaz

Podemos mostrar texto estático sobre las interfaces. Por ejemplo, la interfaz al estilo Sierra muestra la puntuación en la barra de estado.

Para agregar texto a un GUI, añadimos una etiqueta (label) haciendo clic en el botón "Add label" (Agregar etiqueta), luego arrastramos un rectángulo como lo hicimos cuando agregamos un botón. Podemos cambiar el texto visualizado en la etiqueta editando la propiedad "Text" (Texto). El texto automáticamente se despliega de manera tal que se ajuste al rectángulo que hemos dibujado.

Figura 3.30: Ejemplo de inserción de una Label

Además de escribir texto normal en la etiqueta, podemos agregar algunos indicadores especiales que le permiten al texto cambiar durante el juego. Los signos siguientes serán reemplazados por los valores pertinentes en el juego:

- @GAMENAME@ El nombre del juego, especificado en el panel Game Settings
- @OVERHOTSPOT@ Nombre del hotspot sobre el cual está el puntero del ratón
- @SCORE@ La puntuación actual del jugador
- @SCORETEXT@ El texto "Score: X of XX" completado con los números pertinentes
- @TOTALSCORE@ La puntuación máxima posible, especificada en el panel Game Settings

Ejemplo: Usted tiene @SCORE@ de @TOTALSCORE@ puntos.

La ventana de Propiedades además nos permite alinear el texto a la izquierda, derecha, o al centro, así como cambiar su fuente y color.

3.8.3 Ventanas de Texto personalizadas

Si queremos agregar un toque personal a las ventanas de texto blancas predeterminadas que muestran todos los mensajes durante el juego, podemos crear un borde usando el Editor de GUIs. Creamos un nuevo GUI seleccionando con el botón derecho el menú GUI y ahí seleccionando New Text Window.

El elemento cambiará de tamaño hasta aproximadamente ¼ de la pantalla, y verá ocho imágenes – una en cada esquina y una en cada lado. Estos son los gráficos del borde.

En el juego, los gráficos de las esquinas serán ubicados en las respectivas esquinas de la ventana de texto, y los gráficos de los lados se repetirán a lo largo del borde de la ventana. Para decirle al juego que use nuestro estilo de ventana de texto personalizado, hay que ir al panel General Settings, y marcar la casilla "Text windows use GUI" (Las ventanas de texto usan el GUI). Luego, escribimos el número de GUI que queremos que usen.

Podemos además establecer una imagen de fondo para la ventana de texto. En el editor de GUIs, simplemente seleccionamos una imagen de fondo para el elemento GUI. En el juego, el gráfico que especificamos no será visualizado como mosaico ni estirado; no obstante, será recortado para que se ajuste a la ventana. La imagen debería ser de al menos 250x80 píxeles para asegurarse de que abarque la ventana entera.

3.8.4 Inventario al estilo Lucasarts

Otra opción dentro del editor de GUIs es el botón Add Inventory (Agregar Inventario). Este nos permite arrastrar un rectángulo que mostrará el inventario actual del jugador, de igual manera en que lo hicieron los juegos de Lucasarts. Para hacer que la ventana de inventario sea desplazable, necesitamos agregar botones de flechas hacia Arriba y Abajo, y adjuntar un código del script de texto para esos botones para utilizar las variables disponibles tales como game.top_inv_item.

3.8.5 Desplazadores (Sliders)

Los desplazadores nos permiten tener una interfaz agradable para el jugador para que cambie opciones como el volumen y la velocidad del juego. Para agregar un desplazador, hacemos click en el botón "Add slider" y arrastramos su rectángulo tal como lo haríamos con un botón. Podemos además cambiarle el tamaño arrastrando la esquina inferior derecha de igual manera como si se tratara de un botón.

Los desplazadores pueden ser verticales u horizontales. La dirección en que se dibujan es automática dependiendo de la dirección que alarguemos el desplazador -si es más ancho que alto obtendremos un desplazador horizontal, de lo contrario obtendremos un desplazador vertical.

Para las propiedades de un desplazador podemos establecer el valor mínimo, máximo y actual que el desplazador pueda tener. En el juego, el jugador podrá arrastrar el indicador desde MIN (mínimo) hasta MAX (máximo), y el desplazador se iniciará configurado en VALUE. Para los desplazadores horizontales, MIN es en la izquierda y

MAX es en la derecha, para los desplazadores verticales MAX es en la parte superior y MIN es en la inferior.

Cuando queramos que el usuario mueva la posición del indicador en el desplazador, se llama a la función interface_click con el número del GUI y el número de objeto para el desplazador. Esto significa que si continuamente se arrastra al indicador arriba y abajo, la función interface_click será llamada repetidamente. Nuestro script puede averiguar los valores del desplazador utilizando el comando GetSliderValue del script de texto.

3.8.6 Cuadros de Texto

Un cuadro de texto es un simple dispositivo que le permite al jugador escribir información dentro del juego. Agregar un cuadro es igual que agregar los otros tipos de controles.

Si el cuadro de texto está sobre un GUI visualizado actualmente, todas las teclas comunes que se presionen (es decir, letras, retorno y borrar) son desviadas al cuadro de texto en lugar de ser pasadas a la función on_key_press. Cuando el jugador presione Retorno en el cuadro de texto, la función interface_click es llamada con el número del GUI del cuadro de texto y el número de objeto. Podemos entonces utilizar la función GetTextBoxText del script para recuperar lo que se escribió en él.

3.8.7 Cuadro de Lista (ListBox)

Los controles de cuadro de lista nos permiten agregar a su GUI una lista de ítems. Esto podría ser útil para hacer una ventana personalizada para Recuperar/Guardar, permitiendo al usuario que elija entre varias opciones, etc.

Podemos utilizar las funciones ListBox del script de texto para manipular el cuadro de lista – por ejemplo, ListBoxAdd para agregar un ítem, o ListBoxGetSelected para obtener la selección actual.

Cuando el jugador hace click en un ítem de la lista, se ejecuta la función interface_click con los números de identificación del GUI y el OBJETO.

4 Implementación del juego

4.1 Programación en AGS

Para implementar los diferentes *scripts* del juego, AGS utiliza un lenguaje de programación orientado a objetos propio bastante básico, muy similar a C++ o Java.

4.1.1 Scripts

En la creación del juego se implementan scripts, en concreto uno global, en el que está el código referente a los personajes, y parámetros generales del juego (como la inicialización de las variables, las fuentes o funciones globales), y uno local por cada Room que se añade al juego.

4.1.2 Importación de variables

Para declarar variables globales a todos los scripts es necesario declararlas en el script global de la siguiente manera:

Tipo nombreVariable;
export nombreVariable;

Además es necesario insertar la siguiente línea en el *script* de cabecera global:

import Tipo nombreVariable;

De esta manera podremos acceder a esta variable desde cualquier script. Si quisiéramos que esta variable no fuera visible y modificable en todos los scripts, en vez de poner el import en la cabecera global, habría que ponerlo en cada script en el que se necesitase.

4.1.3 Eventos

Una de las características más importantes de AGS, y que más facilita al programador el desarrollo del software, es su buena e intuitiva gestión de eventos. La mayoría de los objetos del juego tienen distintos eventos predefinidos (por defecto vacíos) que se pueden activar por medio de la interfaz gráfica de AGS. Para escribir el código interno de un evento hay que seleccionar sobre el entorno que queremos crear el evento. En el menú que aparece en la parte inferior derecha, pulsar sobre el botón  y aparecerá un menú como el de la figura inferior. Después seleccionar el evento y pulsar en  . Se abrirá el script correspondiente donde se implementa la secuencia de instrucciones que se ejecuta cuando se lance dicho evento.

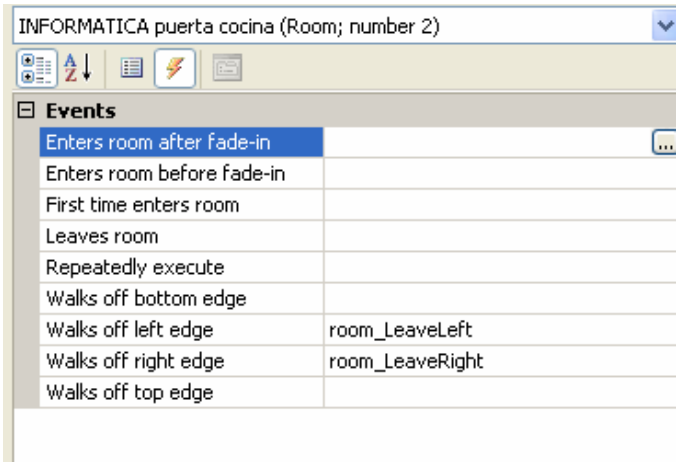

Figura 4.1: Menú de gestión de eventos

4.1.3.1 Eventos en la habitación

- Enters room (alter fade in): Se ejecuta cuando el personaje jugador entra en la habitación, antes del efecto de transición.
- Enters room (before fade in): Se ejecuta cuando el personaje jugador entra en la habitación, después del efecto de transición.
- First time enters room: Se ejecuta la primera vez que el personaje jugador entra en la habitación.
- Leaves room: Se ejecuta siempre que el personaje jugador abandona la habitación.

- Repeteadly execute: Este evento se ejecuta constantemente mientras el personaje jugador permanezca en la habitación.
- Walk off bottom edge: Se ejecuta cuando el personaje jugador sobrepasa la frontera de la parte inferior, fijada por el diseñador.
- Walk off left edge: Se ejecuta cuando el personaje jugador sobrepasa la frontera de la parte izquierda de la habitación, fijada por el diseñador.
- Walk off right edge: Se ejecuta cuando el personaje jugador sobrepasa la frontera de la parte derecha de la habitación, fijada por el diseñador.
- Walk off top edge: Se ejecuta cuando el personaje jugador sobrepasa la frontera de la parte superior de la habitación, fijada por el diseñador.

4.1.3.2 Eventos en los hotspots de la habitación

- Any click on hotspot: Se ejecuta cuando se pulsa (teniendo seleccionada cualquier tipo de acción de juego) sobre el hotspot.
- Interact hotspot: Se ejecuta cuando se pulsa en el hotspot teniendo seleccionada la acción de usar.
- Look at hotspot: Se ejecuta cuando se pulsa en el hotspot teniendo seleccionada la acción de mirar.
- Mouse moves over hotspot Se ejecuta cuando se pasa por encima del hotspot el puntero del ratón.
- Stands on hotspot: Se ejecuta mientras el personaje jugador se encuentra dentro de la región del hotspot.
- Talk to hotspot: Se ejecuta cuando se pulsa en el hotspot teniendo seleccionada la acción de usar.
- Use inventory on hotspot: Se ejecuta cuando se pulsa sobre el hotspot teniendo seleccionado cualquier objeto del inventario. Si queremos especificar el objeto habrá que especificarlo en el código del evento por medio de la directiva ActiveInventory, asociada al objeto jugador.

4.1.3.3 Eventos en las regiones de la habitación

- Walks off region: Se ejecuta cuando el personaje jugador abandona la región.

- Walks onto region: Se ejecuta cuando el personaje jugador entra en la región.
- While standing on region: Se ejecuta mientras el personaje jugador permanece en la región.

En los objetos de la habitación, los personajes y los objetos de inventario:

Todos estos eventos se lanzan análogamente a los de los hotspots:

- Any click on [object |character | inventory item]
- Interact object [object |character | inventory item]
- Look at object [object |character | inventory item]
- Talk to object [object |character | inventory item]
- Use inventory on object [object |character | inventory item]

4.1.4 Objetos: atributos y métodos

A continuación se muestran algunos de los atributos y funciones más importantes del juego y de las distintas instancias de objetos que podemos tener. ATENCION: El API de AGS es mucho más extenso, aquí solo se muestran las funciones mas utilizadas a la hora de crear el proyecto que nos ocupa.

4.1.4.1 Objeto Game

Para manejar las opciones globales del juego existe el objeto Game. A continuación mostramos algunos de los métodos más importantes que hemos necesitado utilizar:

- *Debug (int command, int data):*

Esta función provee al desarrollador de los servicios de depuración del sistema. Según el valor introducido en el argumento comando, se pueden realizar las diferentes tareas:

0. El personaje jugador recibe una unidad de todos los objetos que puede haber en el inventario.
1. Se muestra la versión del juego por pantalla
2. Se muestra la parte de la habitación por la que se puede andar (Walkable area).
3. Tele transporte: Aparece una ventana de dialogo en la que se permite introducir a que habitación quiere ir el personaje y la acción cambia automáticamente a esa habitación.
4. FPS: Se muestran los Frames por segundo a los que se está ejecutando el programa. Se puede ocultar y mostrar esta información utilizando el argumento Data de la función debug a 0 y 1 respectivamente.

- *String GetLocationname (int x, int y):*

Devuelve el nombre del objeto que haya (ya sea hotspot, personaje u objeto) en la posición (x,y) de la pantalla.

- *GetLocationType (int x, int y):*

Devuelve el tipo de lo que haya en la posición (x,y) de la pantalla. El valor devuelto pertenecerá al siguiente grupo:

- eLocationNothing: No hay nada en la posición x,y.
- eLocationHotspot: Hay un hotspot en la posición x,y.
- eLocationCharacter: Hay un carácter en la posición x,y.
- eLocation Object: Hay un objeto en la posición x,y.

4.1.4.2 Personajes

Atributos:

- *int X:*

Posición en el eje X del personaje.

- *int Y:*

Posición en el eje Y del personaje.

- *Item* ActiveInventory:*

Es el objeto que en un determinado momento tiene seleccionado el personaje jugador, como mutarlo, con instrucciones del tipo:

Permite tanto acceder al mismo :

cPlayer.ActiveInventory == iObject;

como mutarlo:

cPlayer.ActiveInventory = iObject;

- *String Name:*

Es el nombre del carácter. Se permite acceso y mutación.

- *Int PreviousRoom:*

Devuelve el número identificador de la habitación anterior en la que estuvo el personaje.

Métodos:

- *AddInventory (InventoryItem *item, optional addAtIndex):*

Añade un objeto al inventario del jugador. Opcionalmente en una posición del índice en concreto.

- *LoseInventory (InventoryItem *item):*

Elimina un objeto del inventario del jugador.

- *ChangeRoom (int room_number, optional int x, optional int y):*

Permite que un personaje cambie de habitación, opcionalmente a unas coordenadas en concreto. También existe la función *ChangeRoomAutoPosition*, que elegirá automáticamente unas coordenadas que pertenezcan a un Walkable Area.

- *Say (String Message):*

El personaje dirá la frase *Message*.

- *SetAsPlayer ( ):*

El carácter al que se aplique esta función pasará a ser el personaje jugador.

- *Walk (int x, int y, optional BlockyngStyle, optional WalkWhere):*

Permite que los caracteres anden a una posición (x,y).

El parámetro *BlockyngStyle* permite que la acción del juego se pare hasta que el personaje no haya llegado a la posición deseada.

- eBlock
- eNoBlock

El parámetro *WalkWhere* indica si se desea que el personaje ande hacia la posición indicada caminando únicamente por zonas por las que se puede andar (valor *eWalkableAreas*) o por cualquier zona (valor *eAnyWhere*).

4.1.4.3 Hotspots

Atributos:

- *int WalkToX:*

Valor del eje X en el que el personaje jugador se situará automáticamente antes de ejecutar cualquier acción sobre el hotspot.

- *int WalkToY:*

Valor del eje Y en el que el personaje jugador se situará automáticamente antes de ejecutar cualquier acción sobre el hotspot.

- *String Name:*

Nombre del hotspot.

4.1.4.4 Objetos

Atributos:

- *Int X:*

Posición en el eje X del objeto.

- *Int Y:*

Posición en el eje Y del objeto.

- *String Name:*

Nombre del objeto.

- *Bool Visible*:

Indica si el objeto esta visible o no.

Métodos:

- *Move (int x, int y, int speed, optional BlockingStyle, optional WalkWhere):*

Análogo al método *Walk* de los personajes con un parámetro opcional *Speed* añadido, que indica a que velocidad se moverá el objeto hacia la posición (x,y).

4.1.4.5 Otras funciones importantes

Funciones multimedia:

- *PlayMp3File (string filename):*

Inicia el archivo de sonido MP3 de nombre *filename.*

- *PlayMusic (int music_numer):*

Inicia el archivo de música número *music_number* de la biblioteca de archivos de música del juego.

- *PlayVideo (String filename, int skip, int flags):*

Inicia el video con nombre *filename*.

Parámetro *skip:*

Indica la forma en la que el jugador podrá saltarse el video:

0: El jugador no puede saltarse el video.
1: El video dejará de ejecutarse si se presiona la tecla ESC.
2: El video dejará de ejecutarse presionando cualquier tecla.
3. El video dejará de ejecutarse presionando cualquier tecla o botón del ratón.
Parámetro *flags:*
0: El video se mostrará en su tamaño original (valido para videos de formato AVI).
1: El video se mostrará a pantalla completa.
10: El video se mostrará en su tamaño original y sin sonido.
11: El video se mostrará a pantalla completa y sin sonido.

Funciones de tiempo:

Si se necesita trabajar con unidades de tiempo se puede trabajar con la clase DateTime.

DateTime *dt = DateTime.Now, introduce la hora actual en dt.

Los atributos de DateTime son:

- int DayOfMonth: Día del mes en el momento que se ejecutó la instrucción anterior.
- int Year: Año en el momento que se ejecutó la instrucción anterior.
- int Month: Mes en el momento que se ejecutó la instrucción anterior.
- int Hour: Hora en el momento que se ejecutó la instrucción anterior.
- int Minute: Minuto en el momento que se ejecutó la instrucción anterior.
- int Second: Segundo en el momento que se ejecutó la instrucción anterior.
- int RawTime: Son los segundos transcurridos desde el 1 de Enero de 1970.

Funciones globales:

- *Display (String text)*

Se muestra por pantalla una ventana dentro de un recuadro con el texto text. Suele ser útil para mostrar valores de variables.

El formato de los Strings (también válido para otras funciones que utilicen el tipo String) es:

Código	**Descripción**
%d	Variable entera
%0Xd	Variable Entera con X ceros a la izquierda
%s	Variable String
%c	Variable carácter
%f	Variable real
%.Xf	Variable real con X dígitos decimales
%%	Carácter %
[	Inserta una nueva línea en el mensaje

Ejemplo:

Display ("Variable entera: %d, Variable real: %f", 5, 3.14"):
Mostrará por pantalla el texto:
Variable entera: 5, Variable real: 3.14) ;

- *SetTimer (int timer_id, int timeout):*

AGS facilita al programador 20 temporizadores a los que se puede llamar con esta función, identificándolos por su id (del 1 al 20) e inicializándolos con un tiempo *timeout* en ciclos de juego, aproximadamente 40 ciclos es un segundo.

Con la función *int IsTimerEspired (int id),* el programador sabrá si el temporizador *id* ha terminado ya, o no. Si el temporizador ha expirado devolverá 1, y 0 en caso contrario.

- *Wait (int time):*

Función que pausa el juego durante *time* ciclos (40 por segundo, por defecto, excepto si se cambia la velocidad del juego con la función *SetGameSpeed).*

- *int Random (int max):*

Función que devuelve un número aleatorio entre 0 y max.

- *Dialog_Request (int parameter):*

Esta función global es llamada siempre desde los diálogos, que tienen su propio lenguaje de programación y distinto entorno. En secciones sucesivas veremos como se utiliza esta función al crear los diálogos.

4.2 Cuestiones de implementación

El juego, desde el punto de vista de la implementación es un gran diagrama de estados, en el que cada personaje tiene asociada una variable que marca su estado, y esta cambia cuando se ejecuta cierto evento, generalmente propiciado por alguna acción que realice el jugador.

Cada personaje no jugador (PNJ) tiene asociada una variable *estadoNombre* que cambiará según se avance en el juego y permitirá interactuar de una manera u otra con él. Por ejemplo, si *estadoPersonajeX* tiene valor 1 el carácter X nos hablará de las variables, sin embargo si tiene valor 2 esperará a que se le entregue cierto objeto que el jugador tendrá que encontrar.

El juego también tiene una variable *Fase* más general, que indica en que momento del juego se está. Con esta variable se bloquean y desbloquean los diferentes escenarios del juego. Por ejemplo, una puerta estará cerrada hasta que no se realice determinada acción, o una parte del campus no podrá ser visitada por el jugador hasta que algún PNJ no le pida que vaya.

Por último también se ha utilizado la variable *estadoLibro* que marca hasta que página del “Gran libro de la programación” se ha rellenado con los conocimientos

adquiridos hasta el momento. De manera que cuando el personaje aprende algo nuevo esto se verá reflejado en el libro, cambiando el valor de esta variable.

A continuación se detallan los valores de las variables más relevantes del juego y los eventos que hacen que cambien de valor. En general estos eventos serán propiciados por las acciones del personaje.

4.2.1 Tablas de transición de las principales variables del juego

4.2.1.1 Fase

VALOR	EVENTO	NUEVO VALOR	CONSECUENCIA
0	juego cartas	1	se abre la cocina
1	juego cocina	2	se abre historia
2	juego parejas	3	se abre derecho
3	juego if	4	se abre filología
4	juego while	5	se abre parque de las ciencias
5	recibe sms Eva	6	se abre pasadizo biología
6	Entra una vez en pasadizo	7	

4.2.1.2 estadoLibro

VALOR	HASTA QUÉ PÁGINA SE PUEDE VER
1	Concepto de Algoritmo
2	Constantes y variables
3	Asignación
4	Tipos
5	Vectores
6	If-then-else
7	While
8	For

estadoBorja

VALOR	EVENTO	NUEVO VALOR	CONSECUENCIA
0	Lucas habla con Borja	1	Lucas juega a las cartas
1	Lucas habla con Borja.	2	Se cambia el nombre de Borja de “chico raro” a “Borja”.
2	Lucas da vaso con agua a Borja.	3	Juego del huevo.
3	Lucas hace el huevo.	4	Borja espera a que Lucas vaya a buscar a Carmen y se pone visible la olla en la cocina.
4	Lucas llama a delegación	5	Borja desaparece del despacho y de la cocina.

4.2.1.3 estadoCarmen

VALOR	EVENTO	NUEVO VALOR	CONSECUENCIA
0	Lucas habla con Carmen	1	Lucas tiene que buscar el colgante
1	Lucas da el colgante a Carmen	2	Carmen manda a buscar las variables
2	Lucas tiene todas las variables y le da alguna a Carmen	3	Carmen le explica las variables y le manda a hablar con Gonzalo.
3	Explicación de las variables	4	Carmen se va a derecho.
4	Carmen manda a Lucas a por el snack	5	
5	Lucas le da a Carmen el snack		Lucas hace el juego de las asignaciones
	Lucas gana juego	6	

	asignaciones		
	Lucas abandona el juego asignaciones	-2	Bloquea ahí hasta que gane el juego de las asignaciones
6	Carmen explica IF. Si gana balanza	7	
7	Explica expresiones	8	
8	Explica y empieza el juego del for	-3	
-3	Mientras no termine el juego del for, Carmen le dice "ánimo Lucas"	9	Se termina el juego del for
9	Carmen manda a Lucas a encontrar pintura para la brocha	-4	Carmen espera a que Lucas le de la brocha con pintura
10	Carmen explica el vector a Borja		
10	Lucas pinta bien su nombre	11	Le explica el vector y le manda otro juego
11	Lucas hace el juego	12	Carmen manda a Lucas a ayudar a Eva y se va a buscar a Borja

4.2.1.4 estadoGonzalo

VALOR	**EVENTO**	**NUEVO VALOR**	**CONSECUENCIA**
0	Lucas habla con Gonzalo sin haber hecho las variables	10	Gonzalo no quiere hablar con Lucas hasta que se haga la parte de las variables
0	hago las variables	1	Gonzalo ya puede explicar los tipos
1	Lucas habla con	2	Se hace el juego de los tipos

	Gonzalo		
2	Lucas finaliza el juego	3	Gonzalo habla y explica a Lucas como ir a derecho
3	Gonzalo habla y explica a Lucas como ir a derecho	4	Gonzalo dice que está ocupado y no puede hablar

4.2.1.5 estadoIon

VALOR	**EVENTO**	**NUEVO VALOR**	**CONSECUENCIA**
0	Lucas habla con Ion sin haber mirado la estatua	5	Cuando hable con Ion se ejecuta la conversación 0
1	Habla con Ion		Cuando hable con Ion se ejecuta la conversación 2
2	Habla con Ion		Cuando hable con Ion se ejecuta la conversacion2
3	Habla con Ion	2	Cuando hable con Ion NO se ejecuta conversación
4	Habla con Ion y no habla	4	
5	Habla con Ion		
X	Lucas mete palo en aparato de aire acondicionado	-1	Ion se va y Bedel aparece
0	Mira Estatua	2	
5	Mira Estatua	6	Cuando hable con Ion NO se ejecuta conversación

4.2.1.6 estadoBedel

VALOR	EVENTO	NUEVO VALOR	CONSECUENCIA
0	Lucas habla con Bedel	1	Bedel abre la verja y Lucas puede entrar en el laberinto
1	Lucas consigue el libro de la estatua	2	Bedel explica el while a Lucas
2	Bedel explica a Lucas el while	3	El Bedel está ocupado

4.2.1.7 estadoEva

VALOR	EVENTO	NUEVO VALOR	CONSECUENCIA
0	Carmen manda a Lucas a ayudar a Eva	1	Eva aparece en su situación inicial
1	Lucas habla con Eva y nos explica lo del ascensor	2	Eva espera a que se arregle el ascensor
2	Lucas arregla el ascensor	3	Eva baja y recibe SMS de Carmen
3	Lucas habla con Eva	3	Eva le dice que ayude a Carmen

4.3 Como implementar mini-juegos: un ejemplo práctico

AGS, como se ha comentado, es un entorno orientado principalmente al desarrollo de juegos de entretenimiento tipo aventura gráfica. Sin embargo, sabiendo aprovechar sus características, y con un poco de imaginación, se puede conseguir crear también juegos de acción, arcade, puzzles etc.

Por supuesto, cada juego que se quiera crear será distinto. Nosotros en esta sección nos centraremos en algunas características generales de los *mini-juegos* que hemos implementado.

Como ejemplo sobre el que trabajar hablaremos del juego del trilero, en el que aparecen 3 cartas, se nos dice que nos fijemos en una, y tras unos movimientos de las cartas, se pedirá al jugador que acierte en qué posición se encuentra.

En primer lugar será necesario crear una o varias habitaciones en las que discurrirá el mini juego. En este caso sólo será necesaria una, en la que deshabilitaremos la opción de que aparezca el personaje seleccionando la opción *"Hide Character"*, puesto que sólo queremos que aparezcan las cartas.

Pondremos un fondo de pantalla. Para el ejemplo podríamos elegir un fondo marrón que simulará que las cartas están sobre una mesa. Posteriormente añadiríamos a la habitación los objetos que necesitamos, en este caso serán necesarias tres, que serán las cartas.

A continuación procedemos a importar los *sprites* necesarios. Tres cartas diferentes por un lado, y una carta del reverso. Cada objeto tendrá diferentes vistas dependiendo del juego (cada carta podrá estar del reverso o del anverso), por lo que asociaremos a las cartas vistas, de manera que con la función *SetView* podremos, según el estado del juego, asignar una u otra vista.

Una vez asociada cada imagen a cada objeto de la room podemos empezar a implementar nuestro juego.

Lo primero que debemos hacer será separar el juego en estados en este caso hemos optado por esta distribución:

- Estado 0: Asociamos un evento cuando el jugador entra en la habitación en el menú inferior derecho de la habitación (*Enters room after fade-in)*

- Se explica el juego

```
cBorja.Say ("Mira las cartas y adivina dónde esta el rey de oros");
```

- Se muestran las cartas, se espera 1 segundo y se las da la vuelta:

```
oCartauno.SetView(4);
oCartados.SetView(2, 0, 1);
oCartatres.SetView(2,0, 2);
Wait(30);
oCartauno.SetView(2);
oCartatres.SetView(2,0, 0);
oCartados.SetView(2, 0, 0);
```

- Se espera una tecla para que se comience:

```
Display ("pulsa una tecla");
WaitKey(1);
```

- Estado 1: Se mueven las cartas y se espera a que el jugador seleccione dónde cree que está la carta que debe acertar. En este caso haremos 2 tipos de movimiento aleatoriamente 15 veces.

```
while (cont != 15) {
        int aleat = Random(2);
        if (aleat == 1)movtipo2();
        if (aleat == 2)movtipo1();
        cont = cont+1;
}
```

- Estado 2: Se hace la comprobación de acierto o no y se comunica al jugador. Para esto, creamos un evento *repeteadly execute* que comprobará constantemente si se cumple alguna condición de fin de juego.

```
if (faseCartas== 1) {
cBorja.Say("Muy bien, has ganado");
}

if (faseCartas==0){
cBorja.Say("ohhh, has perdido");
}
```

La variable acierto habrá sido declarada e inicializada a -1 al comienzo del script privado de la habitación. Para que el valor de acierto cambie y se ejecute uno de estos if´s, de manera que el juego termine, será necesario seleccionar alguna de las cartas y que el valor de esta variable pase a valer 0 o 1 según sea el rey de oros o no. Para esto asociaremos a cada una de las tres cartas un evento del tipo *Any click on object* que dará la vuelta a la carta y cambiará el valor de la variable acierto, por ejemplo, a una de las cartas que no sean el rey de oros le asociaríamos este código:

```
oCartauno.SetView(5,0,1);
acierto=0;
```

5 El juego

En esta sección se explica de manera detallada todo lo relacionado con el guión y la historia del juego: desde cómo utilizar el juego hasta cómo fue el desarrollo del guión. Además se incluye una guía en la cual se explica la trama del juego incluyendo aquellas tareas que el jugador deberá resolver para terminar el juego.

5.1 Guía de usuario: cómo jugar

En este capítulo se enseña a un usuario sin experiencia en juegos de tipo aventura gráfica cómo dar sus primeros pasos. En general consideramos que es bastante intuitivo, sin embargo si el jugador nunca se ha enfrentado a este tipo de interfaz, será de agradecer una explicación detallada como esta.

5.1.1 Seleccionando la acción

En el juego disponemos de 4 acciones básicas:

- Ir a
- Mirar
- Hablar a
- Coger/usar

Tenemos 2 formas de seleccionar una opción:

1. Pulsando el botón derecho del ratón repetidamente hasta que el cursor coincida con la acción que queremos realizar.

Figura 5.1: Cursores de acción

2. Seleccionando en la barra emergente. Ésta se oculta automáticamente. Para visualizarla, basta con pasar el puntero del ratón por la zona superior de la pantalla.

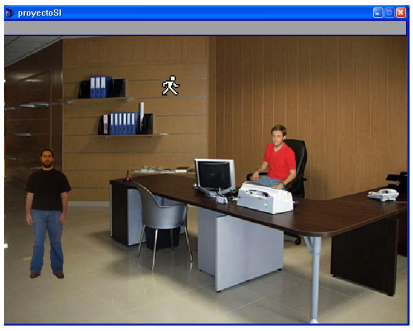

Figura 5.2: Barra oculta

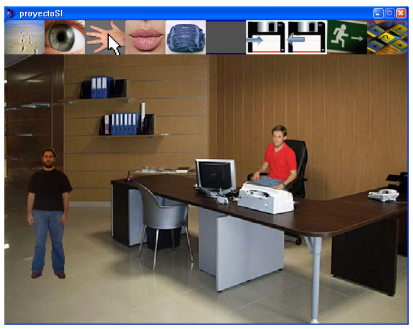

Figura 5.3: Barra visible

Los botones de la barra, ordenados de izquierda a derecha son los siguientes:

1. Andar
2. Mirar
3. Coger/Usar
4. Hablar
5. Abrir inventario
6. Objeto seleccionado del inventario (en gris si no hay ninguno seleccionado)
7. Salvar la partida
8. Cargar la partida
9. Salir del juego
10. Información sobre el juego

Figura 5.4: Barra de acciones

5.1.2 Inventario

Para acceder al inventario tendremos que seleccionar la mochila que hay en la barra emergente. Entonces se abrirá una ventana en la que podemos ver todos los objetos que tenemos disponibles. Cada objeto del inventario puede ser inspeccionado o seleccionado. Para mirarlo tendremos que pulsar sobre el icono de ojo que hay dentro del menú del inventario. Si por el contrario lo que queremos es seleccionar un objeto para usarlo fuera del inventario, tendremos que hacer clic sobre el objeto teniendo el cursor normal activo, y una vez que tengamos el objeto en cuestión como cursor, pulsar al OK. De esta manera se cerrará el menú de inventario y el objeto para ser utilizado en el escenario.

5.1.3 Las acciones

1. Ir a: lo utilizaremos para que el personaje se mueva por las habitaciones y para cambiar de habitaciones. Cuando entremos en ciertas zonas de las habitaciones, haremos que el personaje cambie de habitación (generalmente son los extremos de las habitaciones, y las puertas).

2. Mirar: Esta acción nos dará generalmente más información sobre un objeto, persona o parte del escenario que no vemos a simple vista y que puede resultarnos útil en el desarrollo del juego. Por ejemplo, podemos ver que hay una boca de incendios en el escenario, pero sólo si seleccionamos la acción de mirar, y pulsamos sobre la boca de incendios, nos daremos cuenta de que esta gotea, de manera que si utilizamos un vaso sobre la boca de incendios, podríamos llenarlo.

3. Hablar a: Generalmente utilizaremos esta acción con personas para comenzar a desarrollar una conversación con los distintos personajes del juego. En contadas

ocasiones, podemos hablar con objetos o animales (por ejemplo, si hubiera un loro, tendría sentido intentar hablar con él).

4. Coger/usar: Esta acción puede realizar indiferentemente dos cosas:

    a. Coger: Si teniendo seleccionada esta acción pulsamos sobre un objeto, el protagonista lo cojerá y lo añadirá a su inventario en el caso de que este sea interesante. Por ejemplo, si hay un salero y lo seleccionamos, se añadirá a nuestro inventario y desaparecerá del escenario.

    b. Usar tiene 2 vertientes:

        i. Un objeto del escenario puede tener una función por si mismo, por ejemplo un interruptor. Si lo seleccionamos, probablemente se encienda la luz. Para esto bastará con tener seleccionada la acción de usar, y pulsar sobre el objeto interruptor.

        ii. Usar con: A diferencia de las demás acciones esta no tiene su propio icono en la barra de herramientas ni en la serie de iconos que aparecen pulsando el botón derecho del ratón. Para realizar este tipo de acciones tendremos que seleccionar el objeto del inventario, pulsar sobre el botón de OK, y posteriormente utilizarlo sobre lo que queramos. Un objeto puede usarse sobre:

            1. Otro objeto. Por ejemplo, si utilizamos una moneda de euro con una máquina conseguiremos un snack que será añadido a nuestro inventario. Para realizar esta acción tendremos que seleccionar del inventario la moneda de euro; una vez que esta aparezca en el cursor, podremos moverla hacia el objeto con el que queramos interaccionar y pulsaremos sobre él.
            2. Un personaje: De este modo le intentaremos dar el objeto al personaje seleccionado.

3. El propio jugador: Por ejemplo, si queremos que el protagonista se ponga un casco, tendremos que seleccionar el casco en el inventario y pulsaremos sobre el propio personaje.

5.1.4 Cómo cambiar de habitación

En el desarrollo del juego, el personaje deberá cambiar de una habitación a otra. Para ello el jugador deberá seleccionar la acción andar y seleccionar en una zona de la habitación. Si la zona seleccionada no es accesible, el personaje se acercará lo máximo posible a ese punto, siempre manteniéndose dentro de la zona *walkable*. Normalmente, las zonas de transición están marcadas de manera que al pasar el ratón por encima saldrá algún nombre en la barra superior. Normalmente estas zonas son los extremos de las habitaciones y las puertas.

En el desarrollo del juego, hay algunas zonas que pueden estar bloqueadas hasta que no se realice una acción determinada. También hay algunas zonas a las que se accede realizando alguna acción. Por ejemplo, si hay una puerta y al intentar pasar se dice que está cerrada, el personaje deberá usar algún objeto para abrir la puerta y pasar así a la nueva zona o habitación.

5.2 Creación del guión

Para la creación del guión, la primera tarea realizada fue buscar información sobre el estado del arte de los Videojuegos Didácticos de Introducción a la Programación. Pudimos comprobar que no hay prácticamente nada sobre este tema y que lo poco que encontramos no aborda los conceptos generales de programación, sino ciertas funciones y conceptos de un lenguaje y entorno determinados.

En un primer momento pensamos en un guión que se desarrollase en el interior de la facultad, pero debido a posibles problemas legales al necesitar fotos del interior de la misma, decidimos realizarla en los exteriores del campus de la UCM.

Debido a que el guión comenzaba de cero, empezamos realizando *tormentas de ideas* tanto para la historia del guión como para posibles formas de explicar los conceptos de programación. Pronto vimos que la mejor manera de realizar estas tormentas de ideas no era en reuniones convencionales, sino en algún espacio que nos permitiese a cada uno lanzar una idea en el momento que se nos ocurriese, y poder discutir sobre ella de manera fluida sin tener que estar todos reunidos. Para esto creamos un espacio *wiki* en el que fuimos lanzando distintas ideas que se nos ocurrían. La dirección del espacio *wiki* es http://proyectosi.wikispaces.com.

Una vez que ya teníamos una idea sobre la historia del juego, y teníamos también algunas ideas sobre qué tipo de escenarios podíamos necesitar, realizamos la toma de fotografías. En distintos momentos fuimos fotografiando las distintas zonas del campus que queríamos incluir en el juego. Al realizar éstas, fuimos confirmando aquellas ideas que eran posibles de implementar y rechazando aquellas que no iban a ser posibles. Por ejemplo, confirmamos la idea de llenar un vaso de agua, ya que comprobamos que en el escenario del juego hay una boca de incendios. Sin embargo, descartamos la idea de que Lucas tuviese que sacar un libro de la biblioteca envolviéndolo en papel de plata para evitar activar la alarma.

Si bien es cierto que la mayoría de las ideas surgieron en los primeros meses, algunas de ellas surgieron al final y otras de ellas fueron modificándose una vez que íbamos comprobando las posibilidades y limitaciones del AGS.

Para el debate de algunas ideas utilizamos las discusiones que el espacio *wiki* nos aportaba, de manera que íbamos completando las ideas más prometedoras y descartando aquellas que no encajaban con el juego, o no eran factibles de realizar. Sin embargo, pese a que estas discusiones nos fueron muy útiles, no eran definitivas. Todas las ideas, tanto del guión como de la forma de abordar los distintos conceptos, han sido validadas y corregidas por el director del proyecto en las distintas reuniones que hemos tenido a lo largo de todo el curso.

5.3 Guía del juego

Lucas A. es un futuro alumno de Ingeniería Informática en la UCM. A finales de agosto, Lucas se acerca a su futura facultad para informarse de la fecha del acto de bienvenida para los alumnos de primer curso. Al llegar encuentra que todo está bastante vacío en la zona de la entrada de la facultad, por los alrededores sólo ve un chico bebiendo un mini, sentado a la sombra.

Lucas sigue paseando un poco y sólo encuentra abierta la delegación de alumnos, donde hay un chico en un ordenador que parece ignorarle. Cuando Lucas habla con él, este tendrá una amable conversación. Bromeará con Lucas y le dirá que si quiere ser informático tendrá que engordar unos cuantos kilos, tener siempre una coca cola a mano, y llevar camisetas frickies (tras esto emite una estruendosa risa). Seguidamente, y en un tono más serio, le dice a Lucas que lo que sí que necesitará realmente será aprender a programar y le preguntará si quiere aprender. Lucas responde que le da pereza, que está de vacaciones. Sin embargo parece que el chico de delegación (a partir de ahora, Borja) tiene mucho interés en que Lucas aprenda a programar, puesto que necesita que alguien pruebe el juego tipo gymkhana que ha creado para enseñar los conceptos básicos de la programación.

Borja enseña a Lucas un abono transporte, que parece ser que Lucas ha perdido por ahí. Borja le dice a Lucas que si quieren se lo apuestan. Borja enseñará tres cartas a Lucas y las moverá. Si Lucas acierta dónde esta el rey, Borja le devolverá el abono, si no, Lucas tendrá que probar el juego y aprender a programar. Lucas pierde, y aquí es donde empieza el juego de verdad.

En primer lugar Borja parece que tiene sed, así que manda a Lucas a llenar un vaso de agua. Para llenarlo Lucas lo podrá llenar en una boca de incendios que hay cerca de la puerta de delegación.

Una vez entregado el vaso a Borja (que, para fastidiar, dirá que no lo quiere), éste le empezará a explicar el concepto de algoritmo.

Borja le dirá a Lucas 5 tareas para conseguir un huevo frito, y este tendrá que decirlas en el siguiente orden:

- Echar aceite
- Encender el fuego
- Echar el huevo
- Echar sal
- Sacar el huevo.

Cuando Lucas lo completa correctamente Borja le dirá que muy bien hecho, y le lleva a la cocina, (cosas que tiene ser de delegación...). A partir de este momento Lucas podrá entrar a la cocina, que hasta ahora estaba cerrada. Podrá acceder desde la puerta exterior, a la izquierda de la entrada principal.

En la cocina Lucas tiene que realizar la tarea de freír un huevo tal y como lo hizo en la conversación con Borja. Hasta que no lo acabe no podrá salir de la cocina. Se encontrará un mostrador con los ingredientes necesarios, y la sartén encima de la cocina. Pulsando el botón de la derecha se encenderá el fuego.

Cuando lo complete, Borja le dará "El gran libro de la programación", en el cual, a partir de ahora podrá apuntar los conocimientos que vaya adquiriendo sobre programación.

Tras echar un vistazo al libro y a la explicación de Algoritmo, Borja le pedirá un favor a Lucas. Necesita que este vaya a Historia a buscar a su novia Carmen para decirle que no podrá acudir a una cita. Lucas podrá ir a historia en autobús, cogiéndolo en la puerta de informática.

En la entrada lateral de derecho se encuentra con una chica que resulta ser Carmen. Tras dialogar un poco con ella, parece que Carmen le va a enseñar algunas cosas sobre programación. En primer lugar le pide que busque algo constante.

Cuando empieza la búsqueda, Lucas puede hablar con una chica que parece nerviosa. Si lo hace ella le dirá que busca un colgante que tiene mucho tiempo. Esto es

una pista para que Lucas sepa que por la zona debe haber algo que lleva mucho tiempo sin cambiar, porque es muy antiguo.

El colgante resulta estar en una alcantarilla (que hay que abrir) en la facultad de historia. Para poder verlo hay que iluminar la alcantarilla utilizando una linterna que se encuentra en unas obras cerca de historia. Para encontrar la linterna hay que mirar dentro de un montón de arena.

Una vez conseguido el colgante, Lucas se lo debe dar a Carmen y ella le hará una breve explicación de lo que es una variable, y le pedirá que consiga algo que tenga varios estados, concretamente agua. Lucas ya tiene un vaso de agua, por lo que sólo le quedarán los estados sólido y gaseoso.

- gaseoso: Borja se está haciendo unos espaguetis en la cocina. Hay que llamarle (desde la cabina) al despacho para que se vaya de la cocina y poder coger la olla. (para tener el número, hay que haber mirado el teléfono del despacho). Aquí Lucas se hace pasar por el rector y le hará un encargo, sacando así a Borja de la cocina y del despacho de delegación.
- hielo: El chico que está fuera de la facultad de informática con un mini, si le pides muchas veces por favor que te de un hielo al final te lo dará. Sin embargo, si Lucas intenta llevárselo así a Carmen, el hielo llegará derretido y tendrá que volver a pedírselo al chico del mini. La manera de conseguir llevarlo será metiéndolo en una bolsa isotérmica que hay en la cocina.

Una vez que Lucas le entrega las tres variables a Carmen, ésta le explica el concepto de variable, a continuación le comenta que necesita un libro que le ha dejado a su amigo Gonzalo. Carmen se tiene que ir a hacer unas cosas, por lo que quedan en Derecho para cuando Lucas consiga el libro. Al preguntarle Lucas a Carmen dónde está Gonzalo, ésta le contesta que está segura de que llegará a lo más alto. Gonzalo se encuentra en un despacho de la facultad de historia. Como es agosto, la facultad está cerrada, así que para entrar Lucas se tendrá que colar por una ventana trepando por un árbol. Pero a Lucas le da un poco de miedo, por lo que tendrá que escalar usando un casco que encontrará en la obra de cerca de historia. Lucas se dará cuenta de que siente

algo especial por Carmen, así que termina poniéndose el casco y subiendo para ayudarla.

Una vez dentro de historia Lucas se encuentra en un pasillo con una zona fregada, por lo que no puede subir donde podrá coger una fregona. También encuentra a Gonzalo, que le explica el concepto de Tipos por medio de un *mini-juego* de hacer parejas. Lucas tiene que unir dos ejemplos de los cuatro tipos diferentes que le explica Gonzalo, Enteros, Reales, Booleanos y Caracteres. Una vez explicado el juego y tomados los pertinentes apuntes, Gonzalo le dice que no tiene el libro de Carmen, que no llegó a dárselo. Lucas podrá ir a Derecho al encuentro de Carmen aunque sin el libro. Tras la explicación de variables se le hará una breve introducción al concepto de tipos y hará el mini-juego.

Lucas llega a derecho y se encuentra allí con Carmen. Al hablar con ella, Carmen le dice que le va a explicar el concepto de asignación, pero antes Lucas tendrá que conseguirle una chocolatina. Carmen le da una moneda de un euro y se queda esperando en la puerta principal de derecho. Lucas tendrá que buscar una máquina de snacks que se encuentra en la puerta secundaria de derecho. Allí podrá comprar una chocolatina para Carmen.

Cuando le dé la chocolatina, Carmen le explicará la asignación y le explicará que antes tenía 100 céntimos (un euro) y ahora tiene 20 céntimos y una chocolatina. A continuación Carmen le hará un test sobre asignaciones a Lucas. Las respuestas de este test son:

Pregunta 1

x:= 10; y:= 5; z:= 3;

y:= y-z;

x:= x-y;

¿Cuánto vale x?

1. 15
2. 5
3. 8 CORRECTA

Pregunta 2

x= 4, y= 0.5

x= x * y

¿Cuánto vale x?

1. 4.5
2. 2 CORRECTA
3. 5

Pregunta 3

var1= 10, var2= 20

var1= var2 * 2

¿Cuánto vale var1?

1. 40 CORRECTA
2. 20
3. 30

Pregunta 4

x= 10, y= 15

x= x + x;

¿Cuánto vale x?

1. 10
2. 20 CORRECTA
3. 30

Pregunta 5

x = 10, y = 2.5, z= 3...

x= (z * x) - y;

¿Cuánto vale x?

1. 27.5 CORRECTA
2. 24
3. 35.5

Pregunta 6

x= 5, y= 6, z= 2....

x= (x - 1) * z

¿Cuánto vale x?

1. 9
2. 8
3. 10 CORRECTA

Pregunta 7

var1 = 10, var2= 3

var1= var1+ (1 -3)

¿Cuánto vale var1?

1. 10
2. 8 CORRECTA
3. 12

Pregunta 8

var1 = 2, var2= 6

var2= var1+ var2

¿Cuánto vale var1?

1. 10 CORRECTA
2. 8
3. 16

Pregunta 9

x= 3, y= 2 z = 4

x= (3 - z) * y

¿Cuánto vale x?

1. -1
2. 2
3. -2 CORRECTA

A continuación Carmen le explicará la estructura IF y Lucas tendrá que realizar el *mini-juego* de la balanza. Para equilibrar la balanza, Lucas deberá elegir una combinación de cubos cuyo peso sume 12, igual que en el brazo izquierdo. Una vez conseguido, Lucas le dice a Carmen que Gonzalo no tenía el libro, y Carmen recuerda que el libro no se lo llegó a dar, y que lo escondió en una estatua cerca de allí, por la zona de filosofía. Carmen manda a Lucas a buscarlo y quedan en verse en la puerta de la facultad de matemáticas, donde tiene que reunirse con su amiga Eva.

Para ir a la facultad de filosofía, donde parece que ahora se encuentra el libro, Lucas podrá coger el autobús, o ir andando desde derecho.

En filosofía hay una estatua con un libro en la mano. Es probable que ese libro sea el de Carmen, pero hay una verja que impide a Lucas acercarse a cogerlo.

Hay un chico en la puerta, que le dice a Lucas que tal vez el bedel le pueda abrir la puerta, pero que está encerrado en la facultad y suele salir poco. Para conseguir que salga habrá que romper el aparato de aire acondicionado que hay cerca del despacho, de manera que por el calor el bedel salga a la calle. Para romperlo Lucas tendrá que utilizar la fregona que encontró en historia (cerca de Gonzalo) con el aparato de aire acondicionado.

Una vez que el bedel salga Lucas podrá hablar con él y pedirle que le abra la verja. El bedel accederá pero no le pondrá las cosas fáciles a Lucas. Lucas tiene que pulsar un botón y cruzar un laberinto corriendo a lo “Indiana Jones” antes de que la puerta del otro lado del laberinto se cierre. Si Lucas no llega a tiempo tendrá que volver al principio del laberinto a pulsar el botón para que la puerta se vuelva a abrir.

Una vez que lo consiga y tenga a mano el libro, Lucas no se atreverá a dejar la estatua sin libro, por lo que continuando con el estilo “Indiana Jones”, tendrá que cambiar el libro por otro libro. Podría intentar cambiarlo por el “Gran libro de la programación”, sin embargo es lo suficientemente importante como para no dejarlo. La única opción será buscar otro libro. Puede valer el libro que hay en el despacho de delegación de alumnos de informática, titulado “Alicia en el país de los bits”. Si Lucas

hace esto podrá coger el libro de Carmen e ir a matemáticas, donde ha quedado con ella para dárselo.

Al llegar a la facultad de matemáticas Lucas se encontrará de nuevo con Carmen. Entonces se irán a una zona con siete contenedores de basura. Allí le explicará la estructura for y para ello le dará a Lucas unas cartulinas para que cumpla la sentencia for que hay escrita en un cartel sobre los cubos. Para esto Lucas deberá empezar por el primer cubo y poner la cartulina con el cero. A continuación, poner una cartulina negra Después poner el dos. Así hasta el final, poniendo una negra en los cubos impares y el número correspondiente en los pares.

Para explicarle el vector, Carmen le da una brocha a Lucas y le dice que vaya a por pintura. Si Lucas se mueve un poco por la zona de matemáticas verá que hay una verja que está cerrada con un candado, detrás de la cual hay un cubo grande con pintura. Para abrir este candado necesitará algo, puesto que no le vale nada de lo que tiene hasta ahora. Lucas irá hacia la parte del edificio de Biología, y ahí encontrará una puerta que está abierta y que parece que conduce a un cuartito. Al entrar, Lucas se tropezará con algo y caerá al suelo golpeándose con varias cosas y rompiendo la linterna al caer. La luz está apagada, así que Lucas saldrá de la habitación, no sin antes recoger algo que se clavó al caerse al suelo. ¡Son unas llaves! Lucas intenta abrir el candado de la verja con las llaves y, sorprendentemente, funciona. Una vez dentro, Lucas utilizará la brocha con el cubo de basura e irá a la zona de los cubos.

Carmen le pedirá que pinte su nombre en los cubos de basura. Tras mostrarse un poco reticente, Lucas accede y pinta su nombre. Entonces Carmen le explica los vectores, y le pide que para continuar limpie los cubos antes de que antes los encuentre así. Para ello Lucas debe encontrar un trapo que hay en la zona del parque de las ciencias y usarlo con los cubos.

Entonces, Carmen le muestra otro cartel, y le pide a Lucas que introduzca las cartulinas de manera que se cumpla la condición del cartel. Para ello, Lucas deberá abrir el cubo número uno e introducir la cartulina con valor seis, abrir el tres e introducir el

tres y abrir el cuatro e introducir el cero. Lucas podrá realizar esta tarea en cualquier orden, porque no afecta para nada en el vector.

Una vez conseguido esto, Lucas le dará a Carmen el libro que tanto ha buscado. Al abrirlo, Carmen se muestra muy sorprendida y preocupada. Borja le ha robado los derechos del método de aprendizaje e intentará venderlos para hacerse rico, cuando ella lo que quería era ayudar a la gente que empieza a programar. Lucas le dice que no pueden consentirlo. Entonces Carmen le pide un último favor, su amiga Eva tiene algún problema y está en el ascensor que conduce a la facultad de historia. Carmen y Lucas se separan, Lucas va a ayudar a Eva y Carmen va en busca de Borja para impedirle salirse con la suya.

Lucas tendrá que ir a la zona del edificio nuevo, donde encontrará a Eva, la amiga de Carmen, que está con muletas.

Eva necesita ayuda, puesto que el ascensor está roto y necesita ir a la parte de abajo, pero sólo hay unas escaleras largas y empinadas por las que difícilmente podría bajar alguien con muletas. Parece que la única solución será arreglar el ascensor de alguna manera.

Si Lucas mira el tablón de derecho, encontrará un cartel de los estudiantes de físicas titulado *"¿Cómo funciona un ascensor?"* En el que aparecen la mitad de unas instrucciones para activar un ascensor y se anima a los estudiantes a pasar por físicas para hacer unos cursos.

Si Lucas va a físicas, mirando los carteles que hay cerca de la puerta encontrará otro cartel con la segunda parte de las instrucciones.

De vuelta al ascensor, cerca de este encontrara un panel eléctrico que podrá abrir utilizando una palanca que hay en químicas, apoyada en un coche.

Una vez que Lucas haya abierto el panel, podrá proceder a aplicar las instrucciones. Teniendo que activar los interruptores 1, 3, 4 y 5 y pulsando después el botón 1.

Cuando funcione el ascensor Eva lo utilizará y bajará a donde está Lucas. Una vez que le haya dado las gracias Eva recibirá un extraño mensaje de Carmen al móvil. Parece ser que Carmen está en problemas y pide ayuda, está en el pasadizo de biológicas. Si Lucas ya ha intentado entrar alguna vez no habrá podido entrar ya que da bastante miedo, sin embargo, una vez más tendrá que afrontar sus miedos para ayudar a su amada.

Una vez que Lucas entre en el pasadizo de biológicas se encontrara con Borja, que parece estar loco. Ha encerrado a Carmen y no la dejará salir a menos que Lucas supere una última prueba. Una especie de examen en el que tendrá que responder 5 preguntas seguidas correctamente. Lucas podrá intentarlo tantas veces como quiera, pero no liberará a Carmen hasta que no lo consiga.

Preguntas:

- **Pregunta 1**

for (x=1 to 5)
v[x] = x*2
¿Cuánto vale v[3]+v[5]?

1. 16 (CORRECTA)
2. 14
3. 12

- **Pregunta 2**

x:=7;
while (1!=1)
x:= x +1 ;
¿Cuál es el valor de x después de ejecutar este código?

1. 1
2. 7 (CORRECTA)
3. 8

- **Pregunta 3**

x:=0;

while (1= =1)

x:= 4+x ;

¿Cuál es el valor de x después de ejecutar este código?

1. 0
2. 4
3. Ese bucle es infinito, no se saldría nunca del bucle (CORRECTA)

- **Pregunta 4**

x:=3; y:=6; z:=1;

if (x==4)

y:= x+y+z ;

else

y:= z*2;

¿Cuánto vale y después de ejecutar este programa?

1. 6
2. 9
3. 2 (CORRECTA)

- **Pregunta 5**

x:= true; y:= false; z:=0;

if ((x and y) == true)

z:= 10;

¿Cuál es el valor de z después de ejecutar este código?

1. 10
2. 0 (CORRECTA)
3. Hay un error de tipos

- **Pregunta 6**

x:=true; y:=false; z:=5;

if (x==false)

z = x+y;

else

y:= z*2;

¿Cuánto vale y después de ejecutar este programa?

1. True
2. False
3. Hay un error de tipos (CORRECTA)

- **Pregunta 7**

x:=true; y:=false; z:=true;

if (x==false)

z:= x or y or z

else

z:= x and y and z

¿Cuánto vale y después de ejecutar este programa?

1. False (CORRECTA)
2. Hay un error de tipos
3. True

- **Pregunta 8**

y= 2;

for (x:=2 to 4)

y = y*y;

¿Cuánto vale y después de ejecutar este programa?

1. 68
2. 512
3. 256 (CORRECTA)

- **Pregunta 9**

x:=true; y:=true;

while (x==true)

{ x := x and y;

y := false; }

¿Cuánto vale x después de ejecutar este programa?

1. True
2. False (CORRECTA)
3. El bucle es infinito

- **Pregunta 10**

for (x:=1 to 5)

v[x] = x;

¿Cuánto vale v[1] + v[2] + v[3] + v[4] + v[5] después de ejecutar este programa?

1. 5
2. 10
3. 15 (CORRECTA)

- **Pregunta 11**

x:=1; y:=2;

while (x<= 101)

{ y:= x;

x:=x+1; }

¿Cuánto vale x después de ejecutar este programa?

1. 101
2. 102 (CORRECTA)
3. El bucle es infinito

- **Pregunta 12**

x:= true; y:= false; z:= false;

if (x or y)

z := x and y;

else

z:= x or y;

¿Cuánto vale z después de ejecutar este programa?

1. True (CORRECTA)
2. False
3. Hay error de tipos

- **Pregunta 13**

x:= 1; y:= 2;

if (x+1 == y)

x := x + y;

else

x:= y-x;

¿Cuánto vale x después de ejecutar este programa?

1. 1
2. 2
3. 3 CORRECTA

- **Pregunta 14**

for (i:=1 to 4{

v[i] = true}

v[3] = false;

¿Cuánto vale v[1] and v[2] después de ejecutar este programa?

1. False
2. True (CORRECTA)
3. Hay error de tipos

- **Pregunta 15**

```
x:= 4; y:=1;
while (x>0){
y:= y+ 2;
x:= x-1;
}
```

¿Cuánto vale y después de ejecutar este programa?

1. 1
2. 8
3. 9 (CORRECTA)

- **Pregunta 16**

```
v[2]= 'a'; v[3]='s';
v[1]= 'c'; v[4]='a';
```

¿Qué palabra contiene v?

1. Ninguna
2. casa (CORRECTA)
3. asac

Además de estas preguntas también podrán aparecer las del examen que le hizo Carmen sobre asignaciones e if´s.

Una vez superada la prueba Borja tendrá que liberar a Carmen y escaparse, pero durante la huida se le caerán los papeles de la propiedad intelectual del método de aprendizaje, y por fin Lucas y Carmen podrán estar juntos.

6 Abordando los objetivos

En esta sección se exponen los conceptos de programación que se han explicado en el juego, así como una explicación del método que se ha utilizado para ello.

6.1 Objetivos y metodología.

Para explicar cada concepto de los objetivos, hemos utilizado un método distinto de enseñanza. Muchos de éstos conceptos se explican mediante *mini-juegos* [2] y otros se explican mediante diálogos interactivos o usando objetos del inventario del personaje. Es importante resaltar que casi todos los conceptos tienen un diálogo asociado que permiten al personaje "pedir" que se repita la explicación teórica tantas veces como sea necesario para entender el concepto; sin embargo, un concepto no se considera explicado en su totalidad hasta que el personaje ha realizado la tarea o el mini-juego asociado (y ha leído la teoría sobre ese concepto en el libro).

A continuación se detalla cómo hemos explicado los distintos conceptos.

6.1.1 Algoritmo

Para explicar el concepto de algoritmo, hemos hecho un símil con una receta de cocina, que se compone de pasos que deben realizarse en un determinado orden. Mediante un diálogo interactivo, Lucas debe ordenar las tareas necesarias para freír un huevo frito. Las tareas son éstas:

- Echar el aceite
- Encender el fuego
- Echar el huevo
- Sacar el huevo
- Echar la sal.

[2] Entendemos como *mini-juego* una parte del juego que se desarrolla, normalmente, en una sola habitación, y en la cual el personaje no aparece. Éste concepto nos permite desarrollar partes del juego conceptualmente distintas al resto, ya que el personaje no debe moverse a ninguna parte y debe realizar una serie de tareas en orden para conseguir ganar el *mini-juego*.

Después de ordenar estas tareas, Lucas deberá realizar el ***mini-juego del Huevo Frito.*** El juego cambiará a una habitación nueva que es una cocina vista de cerca en la que tendrá que freír un huevo. En este *mini-juego* las tareas deberán realizarse en el orden correcto. Si no, Borja le indicará a Lucas que se ha equivocado. De esta manera el jugador se dará cuenta de la importancia del orden de ejecución de las distintas tareas de un algoritmo.

6.1.2 Constante

Para explicar el concepto de una constante, Carmen enviará a Lucas a buscar algo que lleve mucho tiempo sin cambiar. Cuando le entregue el objeto correcto, se desarrollará un diálogo en el que Carmen le explicará el concepto de constante.

6.1.3 Variable

Para explicar el concepto de variable, hemos utilizado el agua como ejemplo de algo que puede cambiar de estado. Carmen le explicará mediante un diálogo las diferencias entre variable y constante y Lucas deberá elegir, entre las distintas opciones, algo que pueda tener distintos estados. Finalmente deberá elegir el agua, que tiene tres estados. Entonces Lucas deberá conseguir una olla de la que sale vapor de agua, un cubito de hielo y el vaso de agua que llenó para Borja anteriormente.

6.1.4 Tipos

Para explicar los tipos predefinidos de programación Gonzalo le explicará a Lucas los distintos tipos que existen. A continuación, Lucas tendrá que resolver el ***mini-juego de los tipos***. Éste consiste en una pantalla en la que aparecen distintos cuadros de texto; en unos de ellos aparece el nombre de un tipo (char, entero, boolean, real) y en otros aparecen valores de esos tipos. Lucas deberá seleccionar el nombre de un tipo y a continuación seleccionar uno de los valores de ese tipo. Todos estos cuadros se moverán aleatoriamente por la pantalla cada dos segundos.

6.1.5 Asignación

Para explicar la asignación, hemos utilizado un símil con dos variables que puede tener Lucas: dinero y tieneSnack. Carmen le dará un euro a Lucas y le pedirá que consiga un snack. Cuando Lucas consigue el snack le explica cómo antes dinero=100 (céntimos) y ahora dinero=20, y tieneSnack antes era falso y ahora es cierto.

A continuación se realizará el ***mini-juego de las asignaciones*** que consiste en un diálogo interactivo a modo de test en el cual Carmen irá poniendo ejemplos de asignaciones. Estos ejemplos se elegirán de manera aleatoria de una lista de ejemplos para evitar hacer siempre los mismos. El jugador tendrá que elegir una de las tres opciones que tiene cada ejemplo. Para conseguir ganar este *mini-juego* será necesario que el jugador acierte tres preguntas seguidas. Si falla, volverá a empezar de cero.

A lo largo de la ejecución del test, el jugador podrá seleccionar acabar el test. De esta manera, podrá estudiar la teoría del libro de programación y volver a realizar el test cuando esté preparado.

6.1.6 Expresiones

Para explicar las expresiones en programación, tras explicarle cómo durante la ejecución de un programa se evalúan distintas expresiones, le pone algunos ejemplos de comparación entre enteros y comparaciones de igualdad entre booleanos. También introduce el concepto de negación, en la evaluación de booleanos. Carmen le explica también la importancia de que estas expresiones estén bien construidas, para que no haya problemas de incompatibilidad de tipos.

A continuación tendrá que hacer el ***mini-juego de las expresiones,*** que consiste en una nueva pantalla en la que aparecerán distintos valores y operadores de comparación. El jugador tendrá que ir seleccionando dos valores de distintos tipos y un operador para construir una expresión. De esta manera el jugador puede construir expresiones ciertas, expresiones falsas o expresiones incorrectas. Cuando el jugador construya una expresión incorrecta, por ejemplo *5 AND true*, se verá un mensaje de

error. Para conseguir ganar el *mini-juego* el jugador deberá componer al menos 5 expresiones correctas. Si no, al pulsar el botón de salir, saldrá un mensaje de Carmen instándole a continuar.

6.1.7 If-Then-Else

Para explicar la estructura IF-THEN-ELSE, hemos utilizado circunstancias del propio juego, como por ejemplo la máquina de snack en la que Lucas consiguió la chocolatina para explicar la asignación. El ejemplo queda de la siguiente manera:

IF tienes dinero THEN
    Puedes comprar un snack en la máquina
ELSE
    No puedes comprar

Una vez terminado el diálogo Lucas tendrá que resolver el ***mini-juego de la balanza***. Éste consiste en una nueva pantalla en la cual aparece una balanza que tiene en su brazo derecho dos bloques, uno con un peso=10, y otro con un peso=2. En la parte inferior de la pantalla, aparecen más bloques de distintos pesos. El jugador debe seleccionar qué bloques quiere poner en el brazo izquierdo de la balanza. El juego termina cuando la balanza está equilibrada, según esta expresión if:

IF (pesoIzquierda == pesoDerecha) THEN juego ganado
ELSE juego perdido

6.1.8 While

Para explicar la estructura *WHILE (condición) DO* hemos creado el ***mini-juego del laberinto***. Lucas aparecerá en un laberinto, en el que hay una puerta y un botón. El jugador tendrá que pulsar este botón, lo que hará que la puerta se abra y comience un contador de cuenta atrás de quince segundos. Cuando este contador se ponga a cero, la puerta se cerrará del todo y Lucas tendrá que volver a pulsar el botón para reiniciar el contador y abrir la puerta de nuevo. La estructura *while* queda de esta manera:

```
temporizador = 15;
    WHILE (temporizador > 0) DO
        BEGIN
            cerrar la puerta 10 centímetros;
            temporizador := temporizador -1;
        END
```

6.1.9 For

Para enseñar el funcionamiento y el significado de la estructura **for**, primero se le ponen ejemplos de la vida real planteados como un for. Por ejemplo el proceso de afinar una guitarra:

```
FOR cuerda_x = 1 hasta cuerda_x = 6 BEGIN
    Afina cuerda_x;
END.
```

A continuación el jugador tendrá que realizar el ***mini-juego del for***. Carmen le dará unas cartulinas de colores. En la pantalla habrá unos cubos de basura, y un cartel en el que aparecerá lo siguiente:

```
FOR i=0 TO 6 DO{
    IF (i es PAR) { pegar cartulina i en cubo i; }
    ELSE { pegar cartulina negra sin numero en cubo i;}
```

Si Lucas intenta pegar un cartel incorrecto, o en un orden que no sea el correcto, saldrá un mensaje de error recordándole que debe cumplir el for del cartel en orden. De esta manera, el jugador se dará cuenta de cómo el for realiza una tarea concreta en cada pasada.

6.1.10 Vector

Para explicar los **vectores** usamos los mismos cubos que para el juego del for, sólo que en este caso los cubos podrán abrirse y cerrarse en un determinado orden. Primero, el jugador tendrá que pintar su nombre en los cubos de basura, una letra en cada cubo. Cuando el jugador termine de pintar las cinco letras de la palabra “Lucas” le explica cómo eso se asemeja a un vector de caracteres, quedando de esta manera:

Vector[0]=’L’
Vector[1]=’U’
Vector[2]=’C’
Vector[3]=’A’
Vector[4]=’S’

Una vez terminada esta parte, Carmen pedirá que introduzca las cartulinas utilizadas en el juego del for haciendo que el vector quede de ésta manera.

Vector[1]= 6;
Vector[3]= 3;
Vector[4]= 0;

Para introducir en los cubos las cartulinas correspondientes, deberá abrir previamente los cubos teniendo en cuenta que éstos sólo podrán abrirse si no tienen ningún cubo adyacente abierto de manera que le impida su apertura.

6.1.11 Repaso de conceptos

Una vez que se han explicado todos los conceptos, se realizan dos mini-juegos más. Uno a modo de repaso, y otro como final del juego.

Mini-juego del ascensor

A modo de repaso hemos creado este *mini-juego* en el que Lucas tendrá que asignar valores a unas variables utilizando unos switches y pulsando unos botones en una pantalla en la que aparecerá un cuadro de fusibles. Para saber qué valores deberá tener cada uno deberá consultar las siguientes instrucciones en las que se muestra el siguiente programa en *pseudo-código*.

¿Sabes como funciona un ascensor?

```
//Programa para encender el ascensor

While (AscensorFunciona= =false)
{
If (boton7pulsado = = true)
{
                    Boton7pulsado = false;
For (cont=1 to 8)
{
                                If (InterruptoresGrandes[cont] = =true)
{
                                Interruptores[cont] = false;
                                }
}

If (Interruptores[1] = = true  and  Interruptores[2] = = false and
          Interruptores[3] = = true  and  Interruptores[4] = = true and
          Interruptores[5] = = true  and  Interruptores[6] = = false and
          Interruptores[7] = = false and Interruptores[8] = = false)
{
Funcional = true;
}
}
If (Boton1pulsado = = true) {
If ((Funcional = = true)
AscensorFunciona = true;
}
}
```

Figura 6.1: Instrucciones del ascensor

Mini-juego final

Para terminar el juego, el jugador tendrá que enfrentarse a un test de conocimientos de programación. Éste test consistirá en preguntas y ejemplos sobre todos los conceptos abordados en el desarrollo del juego. Para ganar el juego, el jugador deberá acertar cinco preguntas consecutivas. Si fallase alguna, volvería a empezar de cero. El jugador podrá en todo momento salir del *mini-juego* y podrá volver a empezarlo

cuando desee simplemente hablando otra vez con Borja. Al igual que el *mini-juego de las asignaciones*, las preguntas se eligen de forma aleatoria de una batería de preguntas. De esta manera se evita que se convierta en una cosa lineal y que el jugador pueda aprenderse las preguntas de memoria al realizarse siempre en el mismo orden.

6.2 Teoría

Si bien el objetivo del juego no era que el jugador aprendiese ningún lenguaje de programación en particular, y por lo tanto no se hace demasiado hincapié en la sintaxis, sí se han explicado unos mínimos teóricos que facilitan el aprendizaje y la asimilación de los conceptos. Para hacerlo de una manera progresiva, a medida que el personaje va avanzando en el juego, va pudiendo acceder a un "libro" donde tiene unos resúmenes de los conceptos principales, así como ejemplos y sintaxis en pseudo código. Además, una vez que se desbloquea una parte de la teoría, siempre está disponible para que el jugador pueda repasarla y consultarla en un futuro a medida que vaya avanzando el juego y necesite conceptos explicados previamente. A continuación detallamos la teoría que aparece en dicho libro.

6.2.1 Algoritmo

Un *algoritmo* es un conjunto de pasos que, realizados en un cierto orden, llevan a un estado objetivo. Este estado, normalmente corresponde con la solución de un problema.

Es muy importante este concepto en programación, pues es la base de todos los programas.

Ejemplo: Si queremos freír un huevo frito, el "algoritmo" sería:
Echar el aceite.
Encender el fuego.
Echar el huevo.
Sacar el huevo.
Echar la sal.

Constantes y variables

Una *constante* en programación es algo que no cambia durante la ejecución del programa. Es muy útil para restringir los programas a los valores que aceptamos como válidos.

Ejemplo: si tengo un programa que gestiona una biblioteca, y todos los usuarios pueden tener 4 libros prestados como máximo, utilizo una constante MaximoPrestados = 4.

Una *variable*, a diferencia de una constante, puede cambiar de estado a lo largo de la ejecución del programa. Pueden ser de distintos tipos, y sirven para almacenar valores útiles para el programa.

Ejemplo práctico: el agua es un elemento variable. Tiene distintos estados, que pueden cambiar de uno a otro a lo largo del tiempo. Un ejemplo de una variable en programación puede ser el número de libros que tiene actualmente prestado un usuario de una biblioteca.

6.2.2 Asignación

Las variables cambian de estado, pero esto no lo hacen aleatoriamente. En programación, una variable mantiene el valor hasta que se hace una asignación que le asigne un nuevo valor del tipo de la variable (los tipos se estudiarán en la siguiente lección).

Ejemplo: el precio de un billete de autobús es una cosa variable, pues se puede cambiar. Sin embargo, el precio se mantiene siempre igual hasta que a principio de año se le da un nuevo valor.

Algunos ejemplos de asignación:

x := 5 //Le asigna el valor 5 a la variable x.
exito := true //Le asigna el valor *cierto* a la variable éxito.

6.2.3 Tipos

En programación hay distintos tipos de datos predefinidos. Éstos son los que se corresponden con los principales datos con los que un programa trabaja. Los más importantes son:

- **Enteros (int, integer):** engloban los números enteros. Ejemplos: *-5, -4, …, 1, 2, … 100*
- **Reales (real, float):** engloban los números reales. Ejemplos: *3.5, 2.46, -0.001*.
- **Booleanos (boolean, bool):** tienen dos posibles valores: cierto (*true*) y falso (*false*).
- **Carácter (char):** se corresponden con un carácter, que es cualquier dígito, letra o carácter que esté dentro de la codificación elegida (normalmente la ASCII). Es importante resaltar que, para diferenciarlos de los números enteros, los caracteres se representan entre comillas simples. Ejemplos: *'1', 'H', 'h', '@', '%'*.
- **cadena de caracteres (String):** a diferencia del tipo carácter este engloba varios caracteres. Suelen ir entre comillas dobles "". Ejemplos*: "Hola" "Ejemplo de frase". "10000"*.

Además de los tipos predefinidos, el programador puede definir otros tipos. Esto lo veremos más adelante.

Las constantes y las variables son siempre de algún tipo. Al declararlas es necesario indicar a qué tipo pertenecen. El tipo de una constante o variable se mantiene invariable durante toda la ejecución del programa, y las posibles asignaciones deben hacerse con valores de tipos compatibles.

Ejemplos:

```
VAR
    x: integer;
    exito: boolean;
    ch: char;
```

BEGIN

x:= 5; //Asignaría el valor 5 a la variable x

exito:=true; //Asignaría el valor true a la variable éxito

e*xito= 6;* //Daría fallo pues 6 no es de tipo boolean

ch:= 'h'; //Asignaría 'h' a la variable ch

ch:= "Hola"; //Daría fallo pues "Hola" no es de tipo char.

END;

6.2.4 Vectores

Un vector es una estructura de datos que consta de n "*contenedores*" que van uno detrás de otro, ocupando posiciones contiguas.

Es posible definir vectores de cualquier tipo. Por ejemplo podemos definir un vector que contenga números enteros, caracteres… Pero todos los elementos del vector han de ser del mismo tipo. Es decir, si defino un vector de enteros, todas las posiciones del vector han de almacenar valores enteros.

Ejemplo práctico: un tren de mercancías con 6 vagones que transporten líquidos.

Para definir el vector, se usa la siguiente sintaxis:
mi_vector: ARRAY (1..tam) of TIPO

(1..tam) establece el tamaño (tam) del vector. En este caso el vector tiene tam elementos. Si se define desde 0..tam tendría tam+1 elementos pues habría un elemento en la posición 0.

Para acceder a los datos de un vector, se usa el nombre del vector, y entre corchetes el número de posición.

Ejemplo:

Para guardar en un vector de tipo *char* la palabra “Hola” se declararía un vector de esta manera:

mi_vector: ARRAY (1..4) of CHAR;

y se harían las siguientes asignaciones.

mi_vector[1]:=’H’;

mi_vector[2]:=’o’;

mi_vector[3]:=’l’;

mi_vector[4]:=’a’;

6.2.5 If-Then-Else

A la hora de programar es muy útil poder evaluar condiciones y, en función de estas, realizar una u otra opción.

Una de las más utilizadas es la cláusula IF. Su forma de uso es:

IF (condicion) THEN

BEGIN

Secuencia de instrucciones1

END

ELSE

BEGIN

Secuencia de instrucciones2

END.

Si la *condición* resulta cierta se ejecuta la Secuencia de instrucciones1. Si por el contrario, la condición es falsa, se ejecuta la Secuencia de instrucciones2.

Ejemplo práctico:

Si tengo una linterna, veo el fondo de un pozo.

Sino, no veo nada.

Ejemplo: la sintaxis para calcular el máximo de dos enteros sería:

```
BEGIN
    if (a > b) then mayor = a
    else   mayor = b
END;
```

6.2.6 While

Aparentemente, la cláusula WHILE es parecida al IF, sin embargo tiene una utilidad muy diferente pues permite repetir muchas veces una misma secuencia de instrucciones.

Su sintaxis es: *WHILE (condición) DO*

```
BEGIN
    …
END;
```

Mientras condición sea cierto, se ejecutan las instrucciones

Ejemplo práctico:

Mientras (no esté saciado)

```
BEGIN
    Comer;
    Beber;
END
```

Ejemplo: *WHILE (x<100) DO*

```
    BEGIN
        X := x+5;
    END.
END; //del while
```

NOTA: Un riesgo de la condición while es hacer por error “Bucles infinitos” que se producen cuando la condición del while no se hace nunca falsa.

Ejemplo: *WHILE (1>0) DO*

BEGIN

Secuencia de instrucciones

END;

1 siempre es mayor que 0, así que el bucle nunca acaba.

7 Conclusiones y trabajo futuro

Para terminar la presente documentación queremos mencionar nuestras conclusiones sobre el trabajo realizado a lo largo de 10 meses y lo que nos ha aportado la realización del proyecto.

En primer lugar hemos vivido las diferentes etapas de un proyecto de comienzo a fin. Desde la preproducción, teniendo que indagar con el fin de encontrar un software adecuado para el desarrollo del proyecto, valorando las diferentes opciones que encontramos, cada una con sus ventajas y desventajas y, una vez tomada la decisión de tomar AGS como referencia, tuvimos que investigar a fondo sobre una herramienta totalmente nueva para nosotros. También ha sido necesario adquirir nociones de instrumentos de apoyo, como el software cEditor para la realización de conversaciones o editores fotográficos.

Como se habrá podido observar, el proyecto ha sido multidisciplinar, en el sentido de que ha conllevado una fase artística como es el desarrollo del guión, la realización de las fotografías y la manipulación de estas para utilizarlas en el juego de forma adecuada, y por otro lado la implementación del juego en sí mismo. Estas diferentes facetas nos han enseñado como un proyecto, que en principio se podría considerar de “Desarrollo de software” tiene muchas y muy diversas facetas, que pese a que no son tareas propias de un ingeniero de software, sí tiene que tener conciencia y unos conocimientos mínimos sobre ellas.

Por otro lado, decir que en nuestro caso, por desgracia, hemos vivido en nuestras carnes la gestión de riesgos de un proyecto, ya que una vez comenzado el proyecto tuvimos una baja en el equipo, disminuyéndose la fuerza de trabajo en un 33%. Este contratiempo se ha intentado suplir modificando ciertas partes del alcance del proyecto y aumentando nuestra implicación en el mismo.

Por último, hemos llevado a cabo gestión de recursos (tomando como recurso el tiempo). La principal dificultad ha sido saber distribuir correctamente grandes cargas de

trabajo a largo y medio plazo, siendo conscientes de que hay partes del año en las que se puede dedicar más tiempo al desarrollo del proyecto y otras en las que menos.

Respecto al estado del proyecto, consideramos que está completo en sí mismo, puesto que tiene un guión elaborado con un comienzo, un desarrollo y un desenlace. Además se cumplen los objetivos inicialmente propuestos. No obstante, creemos que también se proporciona una buena base para crear un proyecto mayor, en el que se añadan conceptos nuevos (como por ejemplo los procedimientos, funciones y punteros), y principalmente el gran reto sería ampliar el número de pruebas y ejercicios para reforzar los conceptos ya explicados. Teniendo en cuenta esto, y otros factores mejorables del software desarrollado, creemos que algunas de las tareas a desarrollar en un futuro serían:

- Crear tramas secundarias, paralelas a la principal, en las que se realicen nuevas pruebas y se traten nuevos conceptos.
- Mejorar la calidad de algunos aspectos gráficos y sprites.
- Añadir sonidos y doblaje de voces a los personajes.
- Mejorar el formato de los diálogos que contienen código para hacerlo más legible.

8 Bibliografía

- Adventure Game Studio Forums. Foros de programadores AGS. Disponible en: *http://www.bigbluecup.com/yabb/*

- Adventure Game Studio Official Webpage. Página web oficial de AGS. Disponible en: *http://www.adventuregamestudio.co.uk/*

- Wikipedia, la enciclopedia Libre. Artículo sobre AGS. Disponible en: *http://es.wikipedia.org/wiki/Adventure_Game_Studio*

- Joyanes Aguilar, L. (1997). *Turbo/Borland Pascal 7*. Madrid, Mc Graw Hill.

- AGS Resources. Página web con recursos (CEditor) para programación en AGS. Disponible en: http://www.americangirlscouts.org/agsresources/

- Ayuda interna de AGS versiones 2.7 y 3.0.

Autorización

Los autores del proyecto: Antonio De Miguel Vicenti y Borja Gómez Gómez, autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos y de investigación, no comerciales, la presente memoria, así como el código, el software y el resto de material que componen este proyecto.

Firmado en Madrid, a 1 de julio de 2008

Borja Gómez Gómez
DNI 2.669.085

Antonio de Miguel Vicenti
DNI 50.739.634