Trabajo de Fin de Grado

Editor de mecánicas de juego

Game mechanics editor

Laura Fariña Rodríguez

La Laguna, 5 de Julio de 2015

D. **Pedro A. Toledo Delgado**, con N.I.F. 45725874-B profesor ayudante adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

“Editor de mecánicas de juego”

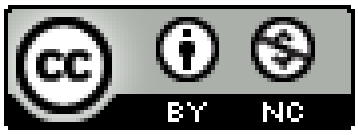
ha sido realizada bajo su dirección por Dña. **Laura Fariña Rodríguez**, con N.I.F. 79083369-P.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de Julio de 2015.

Agradecimientos

Gracias a mi tutor Pedro Toledo por ayudarme y guiarme por el proyecto, y a Alberto Erice por ayudarme con mi aprendizaje de Unity3D.

Licencia



Resumen

El objetivo de este trabajo ha sido la creación de un editor de mecánicas de juego mediante una interfaz sencilla, donde el usuario puede definir los elementos que componen el videojuego, con sus propiedades y el conjunto de reglas que van a definir la mecánica del juego.

Para ello se ha decidido dividir el trabajo en tres partes: análisis, programación del editor de mecánicas, y validación mediante la programación de un ejecutor.

La finalidad del trabajo es que cualquiera pueda crear unas reglas para definir el funcionamiento de un videojuego, aunque no tenga nociones de programación. Esto es útil debido a que, en la amplia comunidad de amantes de los videojuegos, es una minoría la que sabe programar.

Este programa tiene como meta a alcanzar formar parte del software "Mundo Isla", desarrollado por la Universidad de La Laguna, pudiéndose integrar como generador de historias individuales dentro del juego.

Palabras clave: editor, mecánicas, videojuegos

Abstract

The main objective of this project is the implementation of a game mechanics editor that uses a simple GUI where the user can define the elements that take part of the videogame, as well as their properties (position, etc.) and the rules that define their behavior.

In order to achieve the main objective, the project has been divided into three stages: analysis, mechanics designer implementation and validation with an mechanics executor.

The motivation of this project is to allow people who cannot program to "create" a videogame mechanics, by defining the elements and the behavior rules that the game should implement in a high level intuitive way.

It is planned to incorporate the results of this project into the software "Mundo Isla", a MMOG that has been developed by the iTED research group of the University of La Laguna.

Keywords: *videogame, generator*

Índice General

Capítulo 1. Introducción

Con el crecimiento de la informática estas últimas décadas, el mercado de los videojuegos, así como la comunidad de “gamers” o jugadores, ha aumentado de forma considerable. Desde juegos clásicos como el PacMan, se ha evolucionado en poco tiempo a videojuegos con gráficos casi realistas, como el GTAV, o con mecánicas de interacción más naturales como los juegos de la Wii.

Debido a esta evolución y a la diversificación de los videojuegos, cada vez son más las personas que se consideran “fans” de los mismos, o de alguno en concreto. Muchos son los que desearían crear sus propios videojuegos, pero muy pocos tienen las nociones necesarias, ya que se requiere de una formación que no todos quieren o son capaces de afrontar.

Con esta idea en mente, se ha decidido facilitar las cosas a esta clase de personas mediante la generación de un programa fácil de utilizar, y por el cual el usuario puede definir las mecánicas del videojuego que tiene en mente sin necesidad de ningún concepto de programación.

Una herramienta de autor es un programa que facilita la creación de aplicaciones informáticas o material digital. El código de la aplicación que genera suele estar oculto para el usuario, mediante la implementación de una interfaz que actúa como mediador entre ambos.

En la actualidad hay herramientas de autor de muchos tipos, desde las más básicas, como el PowerPoint, hasta algunas algo más avanzadas, como es el caso de Flash.

Gracias a este tipo de herramientas se ha facilitado la creación de contenido digital, puesto que está al alcance del usuario promedio, sin necesidad de que tenga nociones de programación. Por lo tanto, el programa objetivo de este proyecto puede ser calificado como herramienta de autor.

1.1 Objetivos

1.1.1 Objetivo general

El objetivo de este proyecto consiste en la creación de un programa (editor de mecánicas) el cual, mediante una sencilla interfaz, pueda facilitar a los

usuarios la definición de una serie de elementos, propiedades y reglas que puedan ser pasadas a otro programa (ejecutor) capaz de ejecutarlas, para así generar un videojuego con las mecánicas definidas desde el editor de mecánicas.

Se pretende implementar el programa dentro del videojuego “Mundo Isla”, desarrollado por la Universidad de La Laguna. “Mundo Isla” es un mundo virtual por el cual los usuarios pueden moverse con su “avatar” o personaje, y en el cual existen varios minijuegos.

Mundo Isla está desarrollado con la herramienta Unity3D, y está dirigido principalmente a niños hospitalizados, para que puedan de este modo comunicarse con otros niños que se encuentren en una situación parecida. Los niños pueden hablar y trabajar juntos para superar de manera cooperativa las diferentes misiones que pueden encontrar por el mundo virtual.

Las misiones constan de diferentes minijuegos educativos o que ayuden con su rehabilitación. Además, son ampliables y flexibles, con la finalidad de ir añadiendo misiones para así enriquecer cada vez más la plataforma virtual. Al implementar el editor de mecánicas de juego, se pretende facilitar la colaboración de la comunidad en la creación de las mismas.

Se ha de poner énfasis en la sencillez de la interfaz, ya que el programa va principalmente enfocado a un grupo de personas que no tienen nociones de programación; por lo tanto se ha de evitar en todo lo posible el uso de elementos de una complejidad mayor a la que tienen las interfaces de usuario estándar.

1.1.2 Objetivos específicos

I.1.1.2.1 Funcionales del Editor de mecánicas

- Obtener y almacenar los datos introducidos por el usuario mediante la interfaz.
- Poder definir, distinguir y almacenar correctamente los diferentes elementos del juego.
    - Definir los elementos controlables: Personajes Jugadores, Personajes no Jugadores y Objetos.
    - Definir los eventos que funcionan como disparadores de reglas: Acciones.
- Cambiar el comportamiento del juego dependiendo del evento disparado.

- Ser capaz de cambiar las propiedades de los elementos: Posición, Color, etc.
- Volcar los datos a un fichero de texto.

I.1.1.2.2 Funcionales del Ejecutor

- Leer y almacenar datos desde un fichero de texto, distinguiendo y separando esos datos por tipos.
    - Elementos controlables: Personajes Jugadores, Personajes no Jugadores y Objetos.
    - Eventos que funcionan como disparadores de reglas: Acciones.
- Ser capaz de ejecutar el cambio de comportamiento del juego de acuerdo a los eventos que se activen.
- Ser capaz de recoger e implementar los cambios en las propiedades definidos por el editor de mecánicas.

I.1.1.2.3 Requisitos no funcionales

- **Facilidad de uso:** La utilización del programa debe resultarle sencilla al usuario.
- **Eficiencia:** El programa debe ejecutar las funciones de forma rápida y fluida.
- **Portabilidad:** El programa debe ser capaz de ser utilizado en varias plataformas o aplicaciones.
- **Modularidad:** El programa debe ser flexible a la hora de admitir cambios o adición de nuevos módulos.

Capítulo 2. Antecedentes y estado del arte

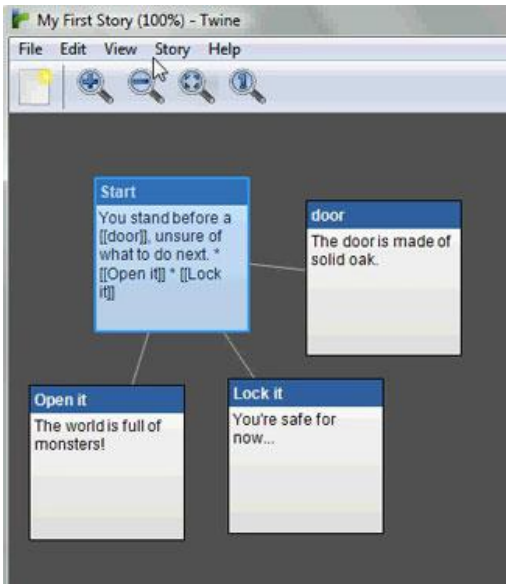
Actualmente existen herramientas con el propósito de facilitar la creación de videojuegos, como es el caso de RPGMaker, una plataforma para crear juegos RPG sin necesidad de programarlo, y de Unity3D, que combina interfaz con scripts para aumentar la potencia del editor.

Se ha comenzado por buscar programas con una finalidad parecida a la que se busca en este proyecto, con el objetivo de ver si resultan de utilidad para empezar a programar a partir de ellos.

2.1 Herramientas

Se han encontrado varias herramientas cuya finalidad es la generación de mecánicas de juego. Las principales han sido:

2.1.1 Twine



La mayoría de las herramientas encontradas han sido con orientación a grafos. Como en general tienen una mecánica similar, se ha optado por poner una de las más representativas: Twine.

Twine es una herramienta Open Source para la creación de historias y la implementación de decisiones en ella. Su interfaz se basa en la creación de nodos que contendrán el texto de la historia, y una serie de aristas que conectarán un nodo con otro, de forma que se le pueden ofrecer varias decisiones al usuario.

Los usuarios de esta herramienta crean historias mediante la creación de nodos. Cada uno de esos nodos es editable, y el usuario puede definir opciones al final del mismo, escritas entre “[[ ]]”. Por cada opción se crea un nodo nuevo. De este modo, se puede crear una historia por la que se puede navegar a través de pulsar en las diferentes opciones.

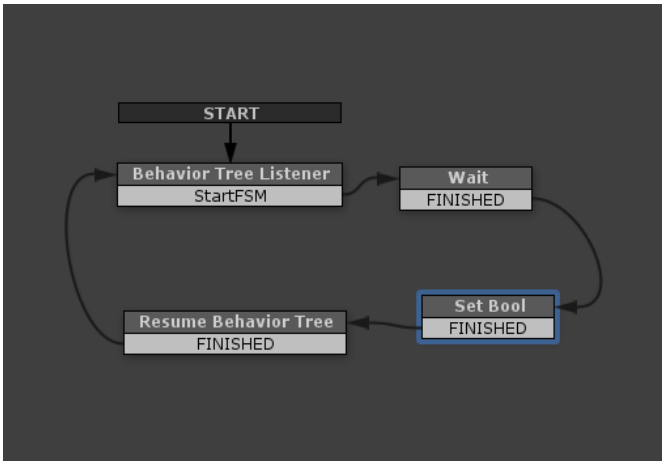
Por último, su licencia es GPL, por lo que los usuarios finales pueden utilizar, modificar o compartir el software como ellos deseen. Además, se puede descargar para Windows, OS X y Linux.

Se ha optado por no utilizar esta herramienta por varios motivos. El primero, es una herramienta orientada a historias. Es decir, resultaría muy útil para videojuegos basados enteramente en decisiones, pero para otro tipo de videojuegos más complejos se dificulta bastante su uso.

La segunda razón por la que se ha rechazado, es la razón por la que se ha decidido no utilizar ninguna herramienta similar: el uso de grafos. Aunque no sea muy complicado aprender a utilizar herramientas de ese estilo, se quiere buscar algo más amigable para el usuario. Los grafos son más abstractos de lo que el usuario medio está acostumbrado, siendo esto interfaces que utilizan paneles de texto, botones y demás elementos más comunes en los aparatos digitales.

En conclusión, aunque sea una herramienta que se acerca a lo que queremos (generación de historias y Open Source), posee limitaciones y puede resultar más complicado para el usuario que lo que se había decidido en un principio.

2.1.2 PlayMaker



PlayMaker se trata de un plugin para Unity3D para la creación de mecánicas de juego que facilita al usuario el uso de Unity3D, simplificando las acciones que se pueden realizar de un modo más amigable para el usuario.

Esta herramienta consta de una serie de estados, variables y eventos, todos ellos definidos por el usuario. En cada estado se definen los eventos que ocurren, y en qué orden. Estos eventos se basan en las funciones del editor de Unity3D, y pueden ser, por ejemplo, mover un objeto o eliminarlo. Los estados se pueden conectar entre ellos para pasar de uno a otro en el juego, dependiendo de los eventos que ocurran.

Los requisitos para poder utilizarla son tener Unity3D. Se trata de una herramienta de pago, y su licencia permite la utilización de la misma por el usuario que la compró, pero no permite su distribución.

Esta herramienta tiene muchas ventajas en cuanto a potencia, ya que gracias a que utiliza las clases de Unity3D, se pueden crear mecánicas con una complejidad bastante elevada. Sin embargo, requiere de una noción básica de Unity3D, y es poco intuitiva, por lo que el usuario debe documentarse antes de utilizarlo, y eso es algo que se pretende minimizar lo máximo posible en nuestro programa.

Como conclusión, debido al problema de la complejidad de aprendizaje, junto con el hecho de que se trata de una herramienta de pago, se ha decidido rechazar el uso de la misma.

2.1.3 Unity3D

Unity3D se trata de una herramienta de creación de videojuegos con una alta potencia. Posee una escena donde se pueden añadir objetos de forma gráfica, y soporta código escrito en tres tipos de lenguajes de programación orientados a scripts: C#, Boo y JavaScript.

A continuación se van a definir brevemente los principales conceptos de Unity3D que han sido de utilidad para el proyecto.

- **Editor gráfico**
  - **La escena:** Lo principal en Unity3D. Es donde se programa, como el escenario del juego, que va a contener todos los objetos que sean necesarios para el programa.
  - **Las cámaras:** Sin ellas, el usuario no podría ver la escena. Se tiene que utilizar una cámara principal que apunte al lugar de la escena que se quiera hacer visible para el usuario. También se pueden utilizar varias cámaras para poner varias visiones, o minimapas, pero en este proyecto en concreto solo se utilizará una.
  - **Las luces:** Se pueden obviar, pero con ellas se da un efecto 3D más realista. Para el editor de mecánicas no se utilizarán, puesto que se hará en 2D, pero como el ejecutor tendrá parte de 3D, las luces son un buen recurso.
  - **Los GameObject:** Son los “objetos” en Unity3D. Son entidades a las que les puedes asignar forma, malla, textura, material, propiedades y scripts, pudiendo crearse numerosas combinaciones. Todos los elementos en Unity3D que están dentro de la escena son GameObjects.
- **Estructura y programación**
  - **Las clases:** Unity3D posee varias clases predefinidas, debiéndose destacar la clase principal de la que suelen

heredar la mayoría de los scripts: MonoBehaviour. Dicha clase principal posee los métodos Start() y Update(), entre otros, siendo normal sobreescribir estos dos primeros, ya que poseen mucha utilidad. Start() es un método cuyo código es ejecutado una vez cargado el GameObject que contiene el script que lo implementa. Update(), por otra parte, es un método cuyo código se ejecuta constantemente, por lo que es muy útil para elementos que necesitan ser constantemente revisados y actualizados, por ejemplo.

- **La estructura de directorios:** Se debe seguir una estructura de directorios para una mayor facilidad a la hora de hacer el programa. La carpeta raíz es la de "Assets". Dentro de ahí, el convenio es crear una carpeta "Resources" en la cual han de ir todos los recursos que utilizas en tu programa: materiales, texturas, scripts, etc, cada uno en su respectiva carpeta, para un mayor orden. Además de la carpeta de Resources es recomendado crear una carpeta Scenes, ya que Unity3D se basa en escenas, siendo cada escena como una "pantalla" del programa.
- **Los materiales:** Para dar más realismo a los objetos de Unity, se les suele asignar un material. Este material puede tener física (elasticidad, rebote, etc), pero lo que más lo caracteriza es su textura.
- **Las texturas:** A cada material se le asigna una textura, las cuales constan de un color, o una imagen renderizada sobre dicho material y una luminosidad. Gracias a las texturas se le puede dar efectos "realistas" a los objetos.
- **Los prefabs:** Se trata de objetos que se han creado y para los cuales se ha definido una serie de propiedades, que se han guardado para usar varias copias del mismo tipo. De este modo si se va a usar muchas veces un objeto con las mismas características no se han de definir todas las propiedades de cero.

- **El 2D**
  - **El uso del 2D:** ya que Unity3D, como su propio nombre indica, es una plataforma mayormente orientada a la programación de videojuegos con gráficos de estilo 3D, pero con opciones especiales para el 2D. Al ser algo añadido posteriormente, cuenta con métodos y propiedades nuevas que se han de mirar.

- **La GUI:** La interfaz de usuario en Unity3D es relativamente nueva, ya que se ha introducido en una de las últimas versiones. En versiones anteriores existía algo parecido a una interfaz, pero era muy simple gráficamente y difícil de implementar, tanto visualmente como en temas de funcionalidad.

  Con la llegada de la nueva interfaz, la llamada NGUI, se ha facilitado mucho la programación en 2D con menús e interfaces de usuario. Hay varios prefabs ya definidos para varios elementos comunes en las interfaces, lo que ha sido de mucha utilidad para la implementación del editor de mecánicas.

Finalmente, tras esta búsqueda de herramientas, y al no encontrar ninguna que cumpliera con los requerimientos del proyecto, se ha decidido empezar el proyecto desde cero. Sin embargo, sí se utilizará la ayuda de la plataforma de creación de videojuegos Unity3D.

Se ha optado por dicha plataforma debido a que facilita mucho la creación de la interfaz debido a que se pueden añadir elementos de forma gráfica, y a estos elementos a su vez se le pueden adjuntar scripts para que realicen una función determinada. Además, uno de los principales objetivos era la integración en el programa “Mundo Isla”, que ha sido desarrollado utilizando esta misma herramienta.

Como lenguaje de programación, se ha optado por C#, ya que es el preferido por la comunidad de Unity3D debido a su potencia y su variedad de clases.

Capítulo 3. Planteamiento general

La idea general del proyecto consiste en la creación de un programa capaz de generar mecánicas de juego definidas por el usuario. Para ello se debe definir una estructura de reglas que servirán de base para la creación de dichas mecánicas.

El usuario hará uso de cuatro tipos de elementos diferentes: Los personajes, los NPC (personajes no jugadores), las acciones y los objetos. Dichos elementos definirán los componentes del videojuego, y serán utilizados para la generación de reglas.

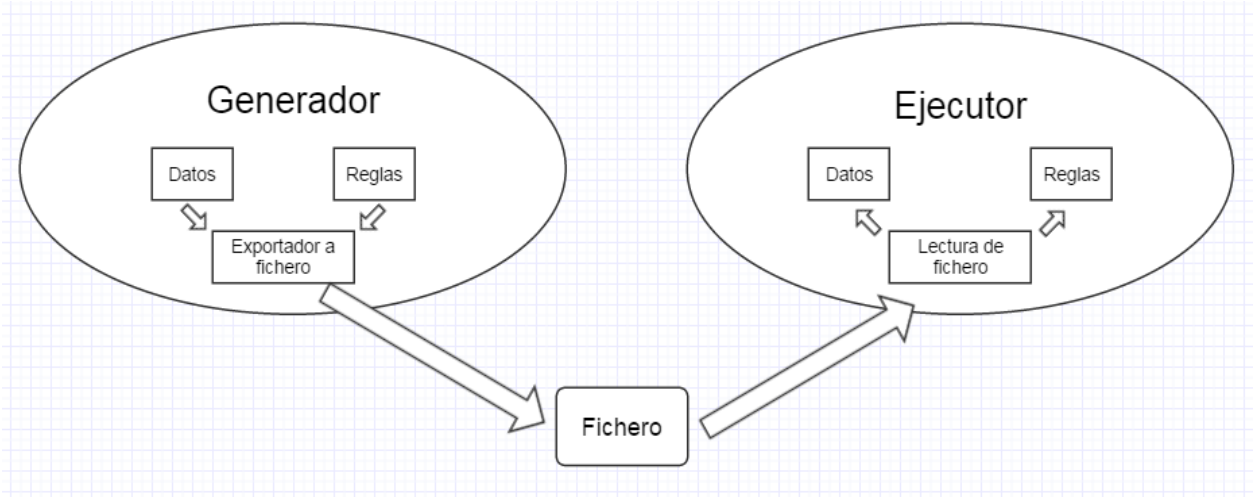
Las reglas son lo que realmente define la mecánica de juego. Cada regla especifica qué acción ejecutar cuando ocurre un determinado hecho o evento. Una regla puede definir el comportamiento de un determinado elemento ante una situación en concreto, por lo que el conjunto de reglas es capaz de determinar la mecánica completa del videojuego.

Las reglas pueden ser definidas completamente por el usuario, siendo uno de los formatos más comunes en el antecedente el siguiente: “Elemento – Acción – Elemento”. Denotamos como elemento a cualquier tipo de componente del juego. Es decir, “Personaje – Acción – Personaje” o “Personaje – Acción – Objeto”, por ejemplo.

La tripleta anteriormente mencionada quiere decir “Si ‘Personaje’ realiza ‘Acción’ sobre ‘Objeto’. Se trata del evento que se ha producido. La otra parte de la regla es la que definiría el comportamiento frente a ese evento. Por ejemplo: ‘Objeto’ realiza ‘Acción’.

Un ejemplo más claro sería la regla con antecedente “Jugador – Mata – Enemigo” y con consecuente “Enemigo – Desaparece”. Lo que está definiendo dicha regla es lo siguiente: Si Jugador mata a Enemigo, Enemigo desaparece. Está definiendo el comportamiento que tendrá Enemigo si se produce el evento de que Jugador lo mata.

Una vez introducidos los datos y las reglas necesarias, se exportaran a un fichero que luego será leído por el ejecutor.



El ejecutor, por su parte, se encarga de interpretar esos datos del fichero que le pasa el editor de mecánicas. Almacena todos esos elementos en sus propias estructuras, y actúa de acuerdo a las reglas que se le indican.

Para que esto sea posible, el editor de mecánicas y el ejecutor deben seguir un convenio de palabras clave para determinados elemento, como pueden ser las acciones. De este modo, la acción de colisión entre dos objetos debe usar el mismo nombre siempre para evitar confusiones entre editor de mecánicas y ejecutor. Un ejemplo es “Personaje – Chocar – Enemigo”. En esta tripleta, se utiliza la palabra “Chocar” para la acción de colisión entre dos elementos. Es importante que siempre se utilice la misma, ya que sería muy complicado para el ejecutor determinar en tiempo de ejecución qué palabra está utilizando el usuario para cada acción. Por lo tanto, las acciones estarían programadas con un nombre en concreto, que se deberá dar a saber al usuario para que utilice ese mismo nombre para esa acción.

La forma que tiene el ejecutor de interpretar las reglas, es la siguiente:

Primero, define unas palabras clave para las acciones que puede realizar, y para los eventos que se pueden producir. Seguidamente, extrae las reglas del fichero que le pasa el editor de mecánicas, y recorre los antecedentes de cada una. Si encuentra una palabra clave relacionada con una acción, almacena los elementos que hacen dicha acción posible. Esto depende de la acción, ya que, por ejemplo, para la acción “chocar” se almacenarían los elementos anterior y posterior a “chocar”, ya que son los que realizan la acción. Sin embargo, si la acción es “Morir”, tan solo se almacenaría el elemento anterior, que es el que realiza la acción.

Además de los elementos que producen la acción, se almacena también el consecuente, para saber qué ocurre cuando se produce dicha acción.

Cada vez que se produce un evento relevante en el juego, el choque de dos personajes, la muerte de uno, etc., el ejecutor revisa los elementos que tiene almacenados, y los compara con los que produjeron la acción. Si no

existe ninguno que coincida, o ningún par que coincida en el caso que haya dos elementos por acción (chocar, por ejemplo), sigue la ejecución del programa como si nada hubiera sucedido. Sin embargo, si encuentra una coincidencia, busca el consecuente almacenado junto al dato que coincidió, y lo ejecuta.

A la hora de ejecutar el consecuente, realiza una búsqueda de palabras clave, tal y como hizo en el antecedente. Debe tener ciertas palabras reservadas para las diferentes acciones que pueda realizar. Si encuentra alguna de ellas, se ejecuta la función correspondiente a dicha acción, con el elemento o los elementos que se vean afectados. Por ejemplo, si el consecuente es “Personaje – Morir”, Morir debería ser una palabra reservada para el acto de que un elemento desaparezca. Además, solo debe afectar a un elemento, puesto que reglas con el formato “Personaje – Morir – Enemigo” no tendrían sentido. Por lo tanto, el programa encontraría la palabra reservada Morir, y ejecutaría sobre Personaje la función que el programador ha decidido para la acción de morir, como puede ser destruir el elemento, o hacerlo invisible para que desaparezca del mapa.

Capítulo 4.
Editor de mecánicas

El editor de mecánicas es el programa encargado de proporcionar al usuario un entorno cómodo mediante el cual pueda definir su mecánica de juego, creando los elementos que lo componen y las reglas que lo definen.

4.1 Interfaz

En el menú principal el usuario decide qué tipo de elemento añadir, y pulsa el botón correspondiente a su elección. Dicha acción le llevará a una nueva ventana desde la cual puede añadir o borrar tantos elementos de ese tipo como desee.

Una vez el usuario ha introducido todos los datos y reglas necesarias para que su mecánica de juego quede definida por completo, desde el menú principal puede exportar dichas mecánicas a un fichero de texto.

4.2 Funcionamiento

El usuario empieza en el menú principal, donde hay varios botones para seleccionar el tipo de elemento a añadir. Una vez pulsa uno de esos botones, se le lleva a otra pantalla donde puede añadir varios elementos de ese tipo, así como todas las propiedades que se quieran añadir a cada uno de los

elementos. Para añadir un nuevo elemento, escribe el nombre del elemento en el cuadro de texto y pulsa el botón de añadir.

Las propiedades son atributos que el usuario define para cada elemento. Cuando son creados, los elementos aparecen sin ninguna propiedad visible. Para añadir más funcionalidad a las mecánicas de juego, el usuario es libre de agregar todas las propiedades que crea conveniente, definiendo su nombre y su valor. Por ejemplo, la posición de un elemento en el mapa o el color del mismo.

Para añadir una propiedad, el usuario debe pulsar con el ratón en el botón de propiedades que hay en cada pantalla de adición de elementos. Esta acción lo lleva a una nueva pantalla desde la cual selecciona el elemento al que quiere añadir la propiedad. Una vez seleccionado se muestra la lista de propiedades actuales y se le da al usuario la opción de agregar más. Para que una propiedad pueda ser añadida necesita un nombre y un valor, los cuales el usuario puede escribir en los dos cuadros de texto correspondientes. Una vez los haya escrito, pulsa el botón de añadir, y dicha propiedad queda agregada al elemento que primeramente seleccionó.

Una vez el usuario ha añadido los elementos deseados, desde el menú principal puede ir a la pantalla de reglas, donde, para cada nueva regla, selecciona varios de los elementos ya introducidos de modo que se creen uno o varios antecedentes, así como un consecuente. Por ejemplo, con los elementos “protagonista”, “enemigo”, “chocar” y “morir”, se puede crear una regla para la cual si dos personajes chocan, uno de ellos muere.

La pantalla de añadir reglas consta de una serie de listas expandibles las cuales contienen los elementos añadidos por el usuario. Los antecedentes y consecuentes se formarían a partir de los elementos de esa lista que el usuario seleccione. Una vez ha decidido los componentes de la regla, la añadirá al programa con el botón de añadir.

Una vez terminadas las reglas, el usuario puede exportar lo que ha generado. Esto lo hace el programa mediante una librería de Unity3D que facilita el trato de ficheros.

Capítulo 5. Ejecutor

El ejecutor es el encargado de interpretar los datos creados por el editor de mecánicas, y este sí que tiene que ser realizado por un programador, por lo que el usuario final no debe realizar nada en este apartado. Debido a la gran variedad de videojuegos que se pueden crear, es muy complicado programar un ejecutor general que pueda ejecutar cualquier tipo de fichero de datos, por lo que cada juego o serie de juegos es recomendable que tenga su propio editor.

El ejecutor debe definir una serie de datos clave, que suelen ser nombres de elementos o propiedades que hacen que se modifique el videojuego. Por ejemplo, el programador puede reservar la palabra “color” como propiedad dentro de un personaje, para que este cambie el color. Dicha información debe ser accesible por el usuario para que sea consciente de las palabras que debe utilizar para determinados elementos o propiedades.

En el caso de Mundo Isla, por ejemplo, el propio Mundo Isla podría funcionar como programa ejecutor. Necesitaría implementar la funcionalidad de leer e interpretar el fichero, y las palabras clave que el usuario debe usar en el editor de mecánicas podrían ser los nombres que ya tienen los elementos en Mundo Isla, o las IDs que poseen.

5.1 Interpretación de los datos

El ejecutor debe interpretar el XML que ha salido como resultado en el editor de mecánicas. Para ello, se utiliza la clase de Unity3D específica para el trato de XML, y se almacena cada elemento en las estructuras correspondientes. De este modo, todo queda igual que como se almacenaba en el editor de mecánicas, así como las reglas pertinentes.

A la hora de recoger las reglas, además de almacenarlas, se analizan a ver si contienen las palabras clave que se han definido para los posibles eventos dentro del juego.

Las palabras clave son acciones o propiedades a las que se les da un determinado nombre. Dichos nombres determinarán esa acción o propiedad tanto en el editor de mecánicas como en el ejecutor.

El ejecutor creará una estructura por cada palabra clave asociada a una acción. En dichas estructuras, se almacenarán los elementos que, realizando dicha acción, causen que una regla se active.

De ese modo, cada vez que ocurre un evento asociado a una acción, el programa revisa esas estructuras en busca del elemento que causó el evento, para activar o no la regla correspondiente.

Por ejemplo, si el programa leyera la regla “personaje choca enemigo -> consecuente”, se almacenarían en la estructura de “chocar” los elementos personaje y enemigo. De este modo, si estos elementos chocan, se dispara el evento asociado a “chocar” con los activadores “personaje” y “enemigo”, y se activará la regla anteriormente mencionada, puesto que habrá encontrado que esos elementos efectivamente se encuentran en la estructura de “chocar”, lo que significa que existe una regla con ese antecedente.

Capítulo 6.
Posibles implementaciones en familias de videojuegos

6.1 Rol

En este tipo de juegos, el jugador maneja uno o varios personajes, cada uno con sus características, y posee un inventario donde guardar los objetos que va encontrando en su viaje.

En esta clase de juegos suele haber una misión principal que lleva al usuario por numerosas pantallas en las cuales se enfrenta a enemigos para mejorar las características de sus personajes.

Una posible implementación en este género de videojuegos sería a la hora de crear las misiones, ya que suelen basarse en una serie de condiciones que se han de cumplir. Por lo tanto, en el editor de mecánicas se crearían los diálogos, objetos, acciones y situaciones necesarias para dicha misión, y con las reglas se comprobaría que se cumplen las condiciones.

Algunos ejemplos de reglas serían meter objetos en el inventario si el personaje los encuentra, comprobar qué objetos tiene el personaje al hablar con otro para decidir qué diálogo escoger, etc.

6.2 Aventura gráfica

Este tipo de juegos se basa principalmente en el diálogo, ya que se trata básicamente de una historia narrada en la cual el usuario decide qué decisiones tomar. También suele haber un inventario para ir almacenando los objetos que encuentra el usuario durante la aventura.

La implementación en este caso sería de diálogos, y las reglas mostrar un diálogo u otro dependiendo de las acciones que toma el usuario y/o de si tiene o no tiene cierto objeto en el inventario.

6.3 Arcade

Los videojuegos arcade se basan en las máquinas recreativas. Son simples, y suelen carecer de historia. Se suelen basar en un personaje que

se mueve y tiene que realizar una serie de acciones básicas para ganar la partida. Ejemplos de este tipo de videojuegos son el PacMan o el Snake.

En estos casos, la implementación se realizaría definiendo los elementos del juego, y el tipo de movimiento que tiene cada uno, así como las acciones básicas que suelen ser “chocar”, “morir”, etc.

La implementación cambiaría dependiendo del juego, pero no serían cambios significativos debido a la simplicidad que caracteriza este género de videojuegos.

6.4 Tower Defense

Este género de videojuegos es relativamente reciente, y se basa en la colocación estratégica de elementos en un terreno con el objetivo de destruir las hordas enemigas que aparecen en un cierto intervalo de tiempo.

Una posible implementación sería definir cada uno de los tipos de “torres” o elementos que defenderían el terreno, así como cada uno de los tipos de enemigos. En las propiedades se definiría la vida de cada elemento, así como el rango y el ataque, por ejemplo.

En cuanto a las reglas, se definirían para cambiar el comportamiento de un elemento dependiendo del tipo que sea, por ejemplo, o de sus características.

6.5 Educativos o juegos serios

Este género de videojuegos tiene por finalidad el aprendizaje en una o varias materias determinadas del usuario. Suele estar orientado a público infantil o juvenil, aunque existen juegos de este tipo para toda franja de edades.

La implementación en esta clase de videojuegos se basaría, por ejemplo, en la generación de preguntas y respuestas, y se aplicaría una regla u otra en función de si la respuesta que ha elegido el usuario es correcta o no.

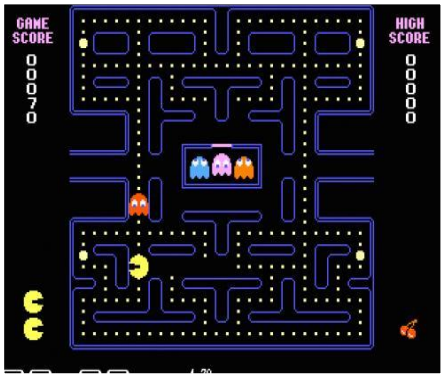
Capítulo 7. Validaciones

Para comprobar que el editor de mecánicas funciona correctamente, se ha realizado una serie de validaciones utilizando cuatro ficheros generados por el programa.

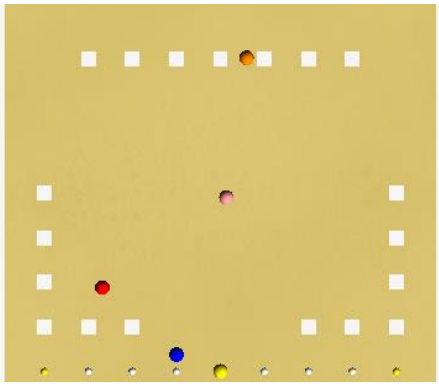
7.1 PacMan

El PacMan o ComeCocos es un juego arcade clásico en el que el usuario controla a una bola amarilla en un escenario cerrado, y su finalidad es terminar con todos los “cocos” o bolitas blancas que hay en el escenario, pasando por encima. Hay cuatro enemigos, que tratan de cazar al usuario, y si lo consiguen, este muere. Sin embargo hay una serie de “cocos” especiales que son más grandes, que si son tocados por el personaje, se vuelve invulnerable e incluso mata a los enemigos si choca con ellos.

Original



Implementado



Se ha realizado una implementación de las mecánicas de juego de este videojuego en concreto. Para ello, se han utilizado los siguientes elementos:

- Personajes:
    - PacMan: Será el protagonista, por lo que se le añadirá una propiedad llamada “rol” con valor “protagonista”. Además tendrá las propiedades de color y posición en el mapa. También tendrá una propiedad especial llamada “movimiento” que definirá el tipo de movimiento que seguirá.
- Enemigos:

    - Blinky, Pinky, Inky y Clyde: Son los enemigos. Sus propiedades son las mismas, pero cada uno es de un color. Dichas propiedades son: movimiento, rol (en este caso es “enemigo”) y posición.

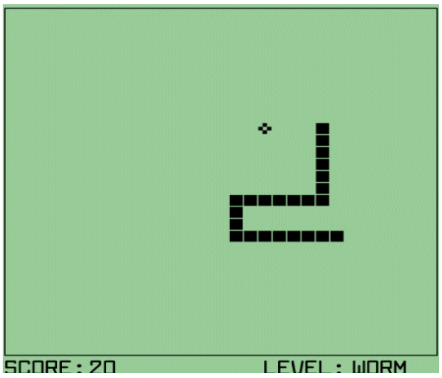
- Objetos:
    - Obstáculos: Son simples obstáculos. No tienen más propiedades que la posición y el rol (“obstáculo”), puesto que no realizan ninguna acción en concreto.
    - Cocos: Son los objetos que PacMan come. Tienen una posición y un rol “comida”.
    - SuperCocos: Son como los cocos, pero con un color diferente, para distinguirlos. Harán el papel de cocos especiales.
- Acciones:
    - Chocar: Es la acción predefinida para el evento de un elemento colisionando con otro.
    - Morir: Es la acción predefinida para el evento de la destrucción de un elemento.
    - Berserker: Es la acción predefinida para el estado de invencibilidad que adquiere PacMan cuando come un coco especial.
- Reglas:
    - Si PacMan choca con algún enemigo y PacMan no está en modo invencible, PacMan muere.
    - Si PacMan choca con algún enemigo y Pacman está en modo berserker, dicho enemigo muere.
    - Si PacMan choca con un coco, el coco muere.
    - Si PacMan choca con un supercoco, el supercoco muere.
    - Si PacMan choca con un supercoco, PacMan entra en modo berserker.

7.2 Snake

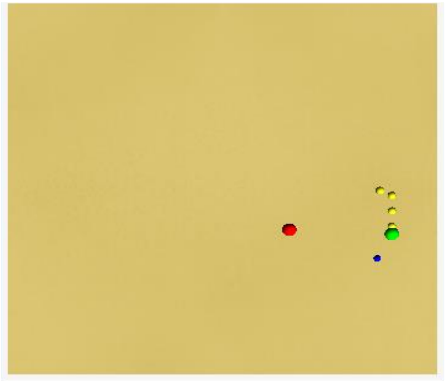
El Snake es un videojuego arcade clásico en el que el usuario es una serpiente, que empieza representada como una esfera, y cuyo objetivo es hacerse lo más grande posible. Para ello, tiene que ir comiendo las diferentes comidas que van apareciendo en el escenario. Cuando come una, crece su cola en una unidad. De este modo, se va haciendo cada vez mayor.

Cuanto más come el usuario, más puntos consigue. Sin embargo, si choca contra las paredes del escenario o contra su propia cola, muere y se acaba la partida.

Original



Implementado



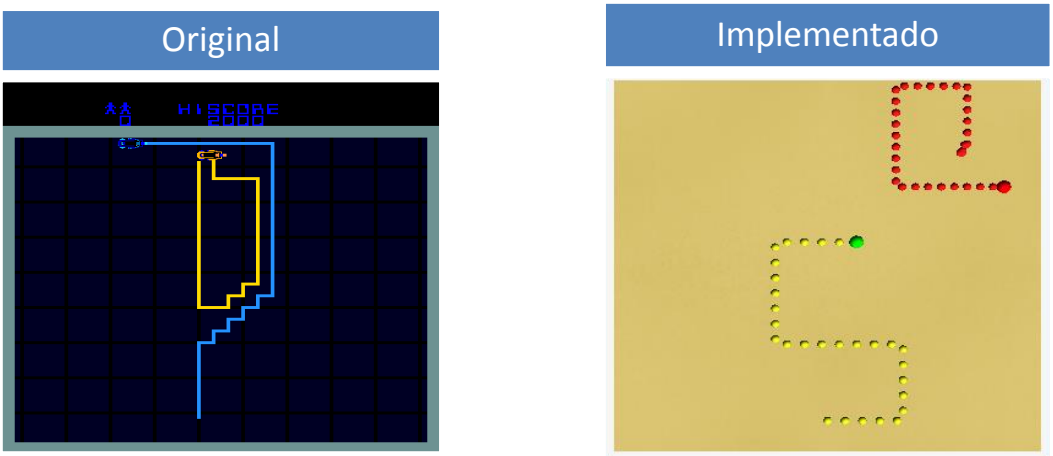
Para la implementación del Snake, se ha decidido hacer una pequeña variación e incluir un enemigo. De este modo, ha quedado la siguiente estructura:

- Personajes:
  - Snake: Será el protagonista, por lo que se le añadirá una propiedad llamada “rol” con valor “protagonista”. Además tendrá las propiedades de color y posición en el mapa. También tendrá una propiedad especial llamada “movimiento” que definirá el tipo de movimiento que seguirá.
- Enemigos:
  - SnakeEnemiga: Será el personaje enemigo, y contará con el rol “enemigo”, y una posición y un color en el mapa, además de la propiedad de movimiento.
- Objetos:
  - Comida: Es el objeto que Snake come. Tiene una posición aleatoria y un rol “comida”. Además, tiene la propiedad “reaparece”, ya que cuando Snake se lo come tiene que volver a aparecer para que el juego continúe.
- Acciones:
  - Chocar: Es la acción predefinida para el evento de un elemento colisionando con otro.
  - Morir: Es la acción predefinida para el evento de la destrucción de un elemento.

- Reglas:
    - Si Snake choca con SnakeEnemiga, Snake muere.
    - Si Snake choca con Snake, Snake muere.
    - Si SnakeEnemiga choca con Snake, SnakeEnemiga muere.
    - Si SnakeEnemiga choca con SnakeEnemiga, SnakeEnemiga muere.
    - Si Snake choca con comida, comida muere.
    - Si Snake choca con comida, snake crece.
    - Si SnakeEnemiga choca con comida, comida muere.
    - Si SnakeEnemiga choca con comida, SnakeEnemiga crece.

7.3 Tron

Tron es un videojuego en el cual el usuario se enfrenta a un enemigo en un escenario cerrado. Cada personaje deja un rastro por donde pasa, y el objetivo es hacer que el otro personaje se choque con cualquiera de los rastros. Si alguno se choca con los rastros o con las paredes muere.



Se ha realizado una implementación de las mecánicas de este juego utilizando los siguientes elementos:

- Personajes:
    - Tron: Será el protagonista, por lo que se le añadirá una propiedad llamada “rol” con valor “protagonista”. Además tendrá las propiedades de color y posición en el mapa. También tendrá una propiedad especial llamada “movimiento” que definirá el tipo de movimiento que seguirá, y una propiedad llamada “cola”, que indica que va dejando un rastro a medida que se mueve.

- Enemigos:
    - Enemigo: Será el personaje enemigo, y contará con el rol “enemigo”, y una posición y un color en el mapa, además de la cola y el movimiento.
- Objetos:

  Ninguno, en este caso.
- Acciones:
    - Chocar: Es la acción predefinida para el evento de un elemento colisionando con otro.
    - Morir: Es la acción predefinida para el evento de la destrucción de un elemento.
- Reglas:
    - Si Tron choca con Enemigo, Tron muere.
    - Si Enemigo choca con Tron, Enemigo muere.
    - Si Tron choca con Tron, Tron muere.
    - Si Enemigo choca con Enemigo, Enemigo muere.

7.4 Rol

Los juegos de Rol se basan en misiones que el protagonista ha de cumplir mediante una serie de objetivos a realizar. Normalmente estos objetivos son matar a algún enemigo, hablar con algún personaje o conseguir algún objeto.

Se ha realizado una implementación de una misión de un juego de rol en la que un personaje no jugador (NPC) necesita un determinado objeto. La estructura resultante ha sido:

- Personajes:
    - Hans: Será el protagonista, por lo que se le añadirá una propiedad llamada “rol” con valor “protagonista”. Además tendrá las propiedades de color y posición en el mapa. También tendrá una propiedad especial llamada “movimiento” que definirá el tipo de movimiento que seguirá, así como un inventario de objetos.
- NPCs:
    - Nadia: Es el personaje que nos dará la misión. Tiene el rol de “aliado”, y las propiedades de posición y color.

- Objetos:
    - Amuleto: Es el objeto que tenemos que recuperar. Tiene el rol de “comida” (ya que en cierto modo es un “consumible” puesto que lo cogemos y desaparece del mapa), y una posición aleatoria en el mapa.
    - Diálogos: Se trata de objetos especiales para este tipo de videojuego. Se meterían dentro de un cuadro de texto. Tienen el rol “dialogo”, una posición dentro del cuadro de texto, y una propiedad “pulsable”, para ver si se puede pulsar o no el texto. La propiedad más característica es la de “texto”, ya que ahí almacenan el texto que caracteriza a cada diálogo.
- Acciones:
    - Chocar: Es la acción predefinida para el evento de un elemento colisionando con otro.
    - Desaparecer: Es la acción predefinida para el evento de la destrucción de un elemento.
    - Pulsar: Es la acción predefinida para el acto de pulsar sobre un diálogo pulsable.
    - Coger: Es la acción predefinida para el hecho de añadir un objeto al inventario.
    - Tener: Es la acción predefinida para el hecho de tener un objeto en concreto en el inventario.
    - Terminar: Es la acción predefinida para el acto de terminación de un determinado diálogo.
    - Mostrar: Es la acción predefinida para el hecho de mostrar un determinado diálogo.
- Reglas:
    - Si Hans choca con Nadia y Hans no tiene Amuleto, mostrar Diálogo1 (donde le pide que busque el amuleto).
    - Si Hans choca con Nadia y Hans tiene Amuleto, mostrar Diálogo2 (donde le pide que le entregue el amuleto).
    - Si Diálogo2 termina, mostrar DiálogoSi.
    - Si Diálogo2 termina, mostrar DiálogoNo.
    - Si pulsas DiálogoSi, mostrar Diálogo3 (donde da las gracias).
    - Si pulsas DiálogoNo, mostrar Diálogo4 (donde se queja).
    - Si Hans choca con Amuleto, Hans coge Amuleto.
    - Si Hans choca con Amuleto, Amuleto desaparece.

7.5 El ejecutor

El ejecutor debe ser capaz de leer y ejecutar un videojuego con las mecánicas definidas por el usuario mediante el editor de mecánicas. Para ello, prepara una escena 3D, pero vista desde arriba por una cámara central, lo que da un efecto 2D. Al encontrarse dicha escena en un plano tridimensional, se le pueden añadir luces, por lo que se ha añadido una luz global para un mejor efecto artístico. El escenario consta de una superficie y cuatro paredes, de modo que los elementos que se coloquen en él no podrán salirse de esa superficie.

Lo primero que hace el ejecutor es leer el fichero de datos que se le introduce e inicializar los elementos. Para ello los almacena en sus estructuras y busca en las propiedades alguna que haga alusión a la posición en el mapa, así como al color y cualquier otra que afecte al diseño, para instanciar un objeto en la escena de Unity3D con la forma, color, posición y otras características definidas en el fichero.

Una vez inicializada la escena, el ejecutor procede al almacenamiento de reglas, utilizando las estructuras para cada palabra clave asociada a cada acción, para que el hecho de que ocurra dicha acción dispare la regla.

A la hora de inicializar cada elemento, dependiendo de su rol le añade un script u otro. Por ejemplo, el script asociado al protagonista se lo añade al elemento con la propiedad de nombre “rol” y de valor “Protagonista”. En estos scripts se define el comportamiento de dichos roles. Por ejemplo, los scripts del protagonista y de los enemigos reflejan su movimiento en escena, dependiendo del valor que tengan. Así, para el caso de PacMan, así como para el de Snake, se moverá de manera continuada hacia la dirección que haya pulsado el usuario. En el caso de Tron el movimiento consta únicamente de izquierda y derecha, ya que el movimiento en este tipo de juegos es de giro hacia una de las dos direcciones, tomando como referencia la dirección actual a la que mira el personaje. Por último, en el caso del juego de rol, solo caminará el personaje mientras el usuario tenga la tecla pulsada.

Todo esto se ve reflejado en el código, mediante condiciones. Así como la propiedad “movimiento” puede hacer variar el tipo de comportamiento del elemento, existen otras que también lo hacen, como la propiedad “cola” o la propiedad “reaparece”.

Capítulo 8.
Conclusiones y líneas futuras

Como conclusión, debe destacarse que se ha programado un editor de mecánicas de juego que, mediante su sencilla interfaz, permite al usuario definir las mecánicas de su propio videojuego sin necesidad de que tenga nociones de programación.

Se ha realizado la implementación desde cero, utilizando la plataforma Unity3D, la cual ha facilitado mucho el proceso de creación de la interfaz, así como la asignación de funcionalidad a cada uno de los elementos de la misma, y la definición de un conjunto de reglas formadas por elementos, mediante las cuales se pueden generar numerosas mecánicas de juego,

Asimismo, se ha hecho un proceso de validación, mediante la programación de un ejecutor capaz de recibir y ejecutar cuatro tipos de mecánicas de juego diferentes creadas a partir del editor anteriormente mencionado.

Por lo tanto, se ha conseguido el objetivo principal del proyecto, que era crear un editor de mecánicas de juego funcional y sencillo, mediante el cual cualquier persona pudiera crear, a base de mecánicas, su propio videojuego.

En cuanto al futuro del proyecto, se podría integrar de manera más sencilla en videojuegos creados con Unity3D, como es el caso del proyecto “Mundo Isla”, el cual se pretende que haga uso del editor en un futuro. Para ello, se podría convertir el programa en un plugin de dicha plataforma, para poder usarlo dentro del propio Unity3D.

También se pueden realizar algunas mejoras en el proyecto, como por ejemplo, que el usuario defina las propiedades clave para que aparezcan ya en todos los elementos y solo quede añadirles un valor, como es el caso de la posición.

El objetivo a largo plazo de la creación del proyecto es que pueda ser integrado en los videojuegos que lo deseen. De este modo, los programadores de dichos videojuegos podrán contar con la ayuda de la comunidad de jugadores para añadir nuevas partes al código del videojuego, como historias o misiones aparte. También podría usarse como un creador de niveles, donde los usuarios aportan sus creaciones, y otros usuarios pueden probarlas.

Capítulo 9.
Summary and Conclusions

In conclusion, it has been programmed a game mechanics editor that allows people who cannot program to create their own videogame by defining its mechanic.

The implementation of this editor has been made using Unity3D, that has been very helpful in the creation of the GUI, and the functionality of its elements.

There has also been made an executor capable of execute four kinds of game mechanics that has been generated by the editor.

Thus, the main objective of this project, the creation of a simple editor that anyone can use, has been accomplished.

Looking ahead, the project can be integrated into Unity3D through implementing it as a plugin. By doing this, the project could be used into Unity3D, and the videogames created using this platform, like “Mundo Isla”, can use the editor easily.

There also can be future improvements in the project. For example, taking the reserved words for properties and insert them into the elements, so the user only have to give them the value of each one, instead of create the same property name for every element.

The future objective of this project is that it would be integrated in any kind of videogame, so the programmers can ask players for new functionality or missions for their games. It also can be using as a kind of level editor, so players can create their own levels and show them to the community.

Capítulo 10. Presupuesto

Ya que todo lo que se ha utilizado para realizar el proyecto ha sido de código abierto y solo se ha utilizado un ordenador, el presupuesto estaría basado en la amortización de dicho ordenador y en el coste del trabajo por hora.

Costes personales	
Coste por hora	15€
Horas trabajadas	260h
Total:	3.900€

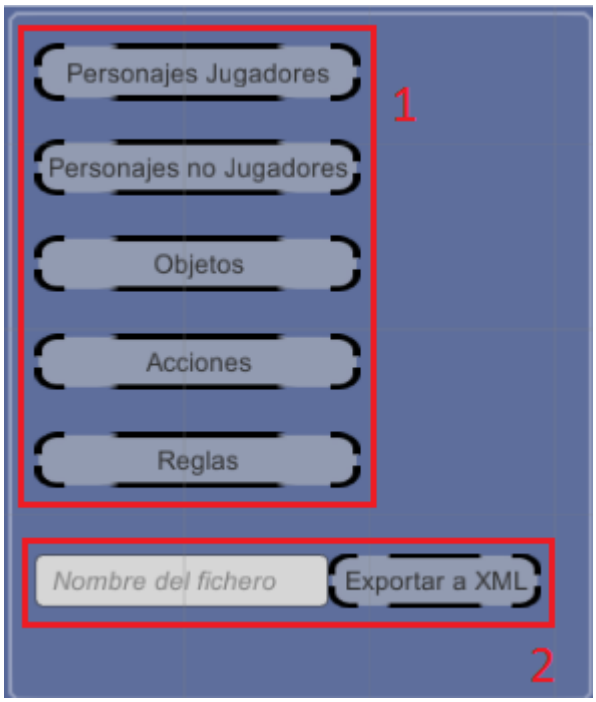
Costes totales	
Costes personales	3.900€
Amortización	50€
Total:	3.800€

Cabe destacar que no se obtiene ningún tipo de compensación económica por el proyecto en sí debido a que se trata de un proyecto desarrollado bajo una licencia Creative Commons.

Apéndice A.
Manual de usuario.

En este apartado se procederá a ilustrar con imágenes el exacto de cada parte de la interfaz del programa, con el objetivo de esclarecer posibles dudas a los usuarios.

Menú principal



Se trata del menú principal del juego; la pantalla del inicio del programa y desde la cual se accede al resto de pantallas.

1–Botones de cambio de pantalla: Estos botones llevarán a las respectivas pantallas de cada tipo de elemento: jugadores, NPCs, objetos, acciones y reglas.

2–Botón de exportar a XML: El usuario escribe el nombre con el que desea llamar a su fichero, y al pulsar el botón, se genera un XML con ese nombre, conteniendo todos los elementos que se han definido hasta entonces.

Pantalla de elementos

Desde esta pantalla, se pueden manejar los elementos y sus propiedades. Se ha puesto como ejemplo la de personajes jugadores, teniendo las de personajes no jugadores, objetos y acciones un comportamiento similar.

1-Botón de nuevo personaje: Sirve para agregar personajes a la lista correspondiente. El usuario escribe el nombre del personaje y pulsa el botón añadir. El programa le asigna un ID, y lo almacena en el programa con dicha ID y el nombre asignado por el usuario.

2-Botón de volver: Su finalidad es devolver al usuario al menú principal.

3-Botón de propiedades: Traslada al usuario hacia la pantalla de propiedades.

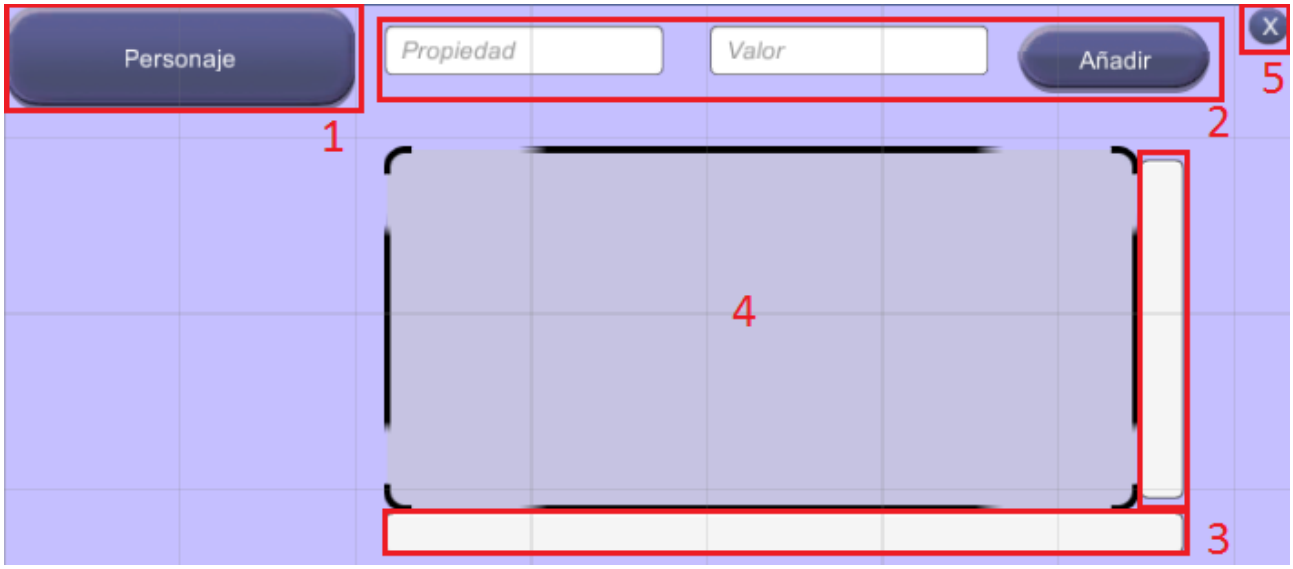
4-Botón de borrar personaje: Si el usuario desea borrar un personaje, escribe la ID del personaje en el recuadro de la izquierda del botón, y lo pulsa. Si existe un personaje con esa ID, queda borrado de la lista del programa.

5-ScrollBars: Son barras que sirven para mover el contenido dentro del panel de personajes. Se activan si el contenido es demasiado grande para ser mostrado todo de una vez. También se puede mover por el panel

haciendo click con el ratón encima del mismo y arrastrando hacia la dirección opuesta a donde se quiere mirar.

6-Panel de personajes: Muestra el contenido de la lista de personajes del programa; es decir, los personajes creados hasta el momento. Cada personaje saldría con su nombre y su ID, y se actualiza cada vez que se añade o borra un personaje.

Pantalla de propiedades



Esta es la pantalla desde la cual gestionamos las propiedades de cada elemento. Como en la pantalla anterior, se ha cogido como ejemplo la correspondiente a los personajes, siendo las demás de funcionamiento similar.

1-Botón de selección de personaje: Al pulsar este botón, se mostrará una lista seleccionable de los personajes que tenemos hasta el momento. Si seleccionamos uno de ellos, podemos acceder a sus propiedades.

2-Botón de añadir propiedad: Una vez seleccionado un personaje, se pueden añadir o editar propiedades. Para ello, existen dos recuadros de texto: uno para la propiedad, y otro para el valor de dicha propiedad. Se escribe lo que se desea y se pulsa el botón de añadir. Si ninguno de los dos valores es nulo, se generará una nueva propiedad con el nombre introducido en el primer recuadro, y el valor introducido en el segundo recuadro.

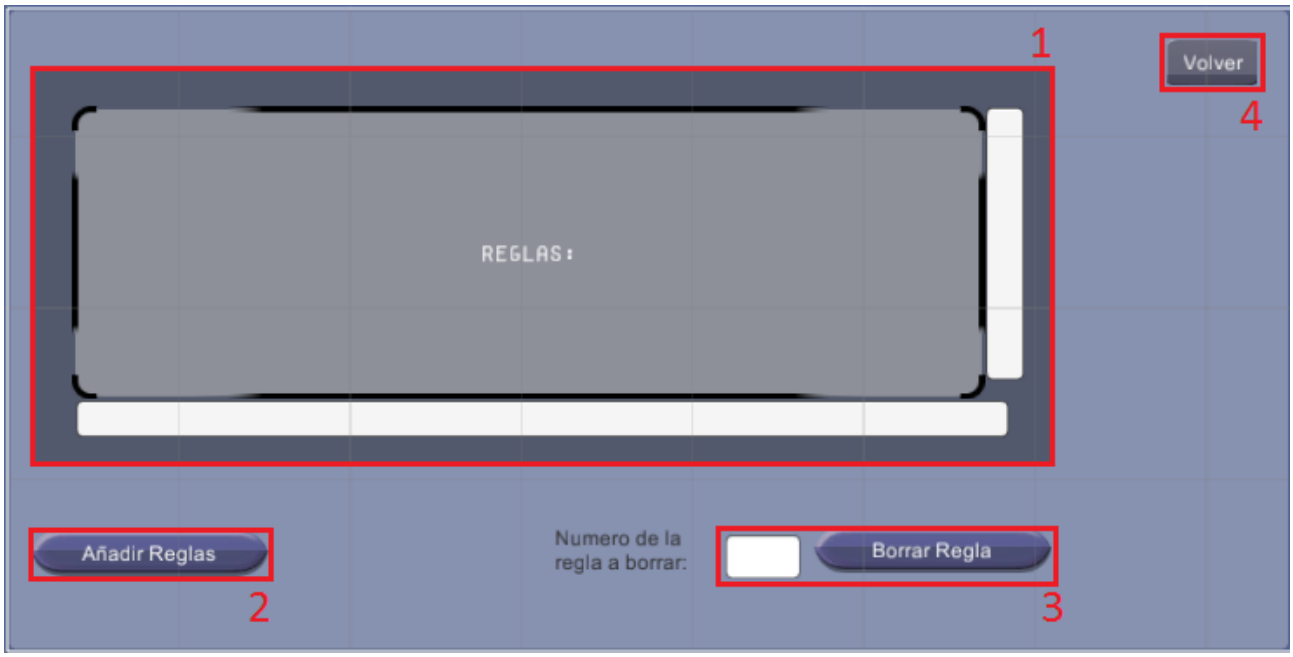
Para editar una propiedad ya creada, simplemente se añade la propiedad de nuevo, con el mismo nombre, pero un valor diferente. Haciendo esto se reemplaza el valor de dicha propiedad.

3-ScrollBars: Al igual que en las pantallas de manejar elementos, sirven para navegar por el contenido dentro del panel si este no cabe en el mismo.

4-Panel de propiedades: En este panel se visualizan las propiedades que tiene cada personaje. Se actualiza cada vez que se añade o cambia una propiedad, y su sintaxis es propiedad: valor.

5-Botón de cerrar pantalla: Es el botón que sirve para volver a la pantalla en la que se estaba anteriormente. En este caso, como es la pantalla de propiedades de personajes, llevará al usuario a la pantalla de personajes.

Pantalla de reglas



La pantalla de reglas es algo diferente a la del resto de elementos. Desde esta pantalla se pueden ver las reglas que tenemos hasta el momento.

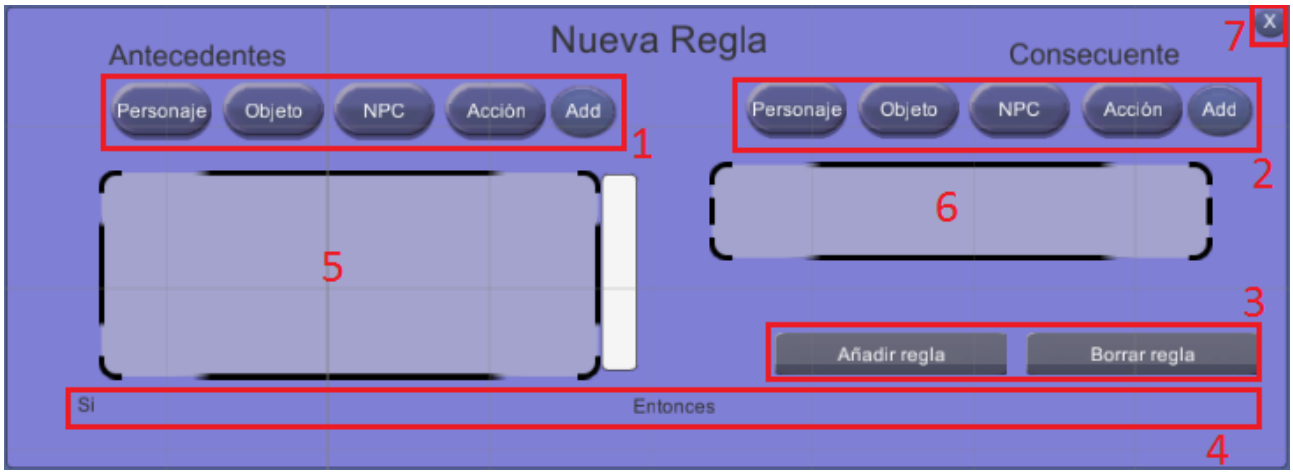
1-Panel de reglas: En este panel aparecerán las reglas definidas hasta el momento. Su sintaxis es: número de regla – antecedentes “entonces” consecuente. Al igual que el resto de paneles, se puede navegar por todo su contenido mediante las barras o haciendo click y arrastrando. Se actualiza cada vez que se añade o se borra una regla.

2-Botón de añadir reglas: Lleva al usuario a la pantalla de añadir reglas.

3-Botón de borrar reglas: A cada regla se le asocia un número de regla. Si se quiere borrar una regla, el usuario escribe el número correspondiente en la casilla a la izquierda del botón y lo pulsa. El programa entonces buscará la regla con ese número y la eliminará.

4-Botón de volver: Devuelve al usuario al menú principal.

Pantalla de añadir reglas



Al pulsar el botón de añadir regla, el usuario es llevado a esta pantalla, desde la cual puede agregar las reglas que desee.

1-Antecedentes: El usuario puede añadir aquí los antecedentes de la regla. Para ello se le proporcionan cuatro botones de selección, uno para cada tipo de elemento. Cada botón se expandirá en una lista seleccionable de todos los elementos de ese tipo que existen hasta el momento. De este modo, el usuario puede seleccionar los elementos que desee de los tipos que quiera para su antecedente. Una vez lo tenga completo, pulsará el botón Add para añadir dicho antecedente a la regla.

2-Consecuente: Al igual que con los antecedentes, se proporciona al usuario una serie de botones para seleccionar los elementos del antecedente de su regla. Sin embargo, solo puede haber un antecedente por regla. Si se intenta añadir otro, el programa no se lo permitirá.

3-Botones de añadir y borrar regla: Con estos botones, el usuario puede añadir definitivamente la regla a la lista de reglas cuando haya definido sus antecedentes y consecuente. Si quiere limpiar la lista de antecedentes y consecuente, pulsará el botón de borrar regla, lo que hará que los paneles se vacíen.

4-Previsualizadores: Cada vez que se añade un elemento al antecedente o al consecuente, aparecerá debajo dicha regla previsualizada. Los antecedentes se pondrán al lado del “Si” y el consecuente al lado del “Entonces”. Los elementos se quedarán en los previsualizadores hasta que el usuario pulse el botón de Add correspondiente. En ese momento, desaparecerán del mismo para pasar al panel que le corresponde.

5-Panel de antecedentes: Muestra los antecedentes que tiene la regla actual hasta el momento. Se va actualizando conforme se añaden antecedentes. Se puede navegar por el mismo con la barra vertical de su derecha.

6-Panel de consecuente: Muestra el consecuente que hemos elegido para la regla actual.

7-Botón de cerrar pantalla: Es el botón que sirve para volver a la pantalla en la que se encontraba el usuario anteriormente. En este caso llevará al usuario a la pantalla de reglas.

Apéndice B.
Manual del desarrollador.

La estructura consta principalmente de una serie de listas, una por cada tipo de elemento. El elemento “Regla” consta de una lista de antecedentes, y un consecuente, y el resto de elementos de un nombre, una lista de propiedades, y un ID que el programa genera automáticamente.

Las listas generadas por el programa serán lo que al final se pase al XML, y posteriormente el ejecutor será el encargado de leerlo y almacenar de nuevo dichas listas para su posterior uso en el programa. De este modo, los elementos serán agrupados por tipo, lo que facilita su interpretación en el programa.

Para coordinar el manejo de información durante todo el programa, se han creado dos controladores. Uno de la información propiamente dicha, y otro de las escenas.

-Controlador de escenas: Lo utilizan todas las pantallas de Unity, y cuenta con las funciones necesarias para cambiar de pantalla. Como es accesible para todos los elementos, se puede referenciar a la función que se desea desde un objeto de tipo botón, para que al ser pulsado se ejecute dicha función.

-Controlador de la información: Posee una clase con una instancia a si misma estática. De este modo se puede acceder a sus variables desde cualquier script del modo Controller.instancia.variable. Posee el contenedor con las listas de elementos, también estático para que prevalezca la información entre escenas. El controlador posee también los métodos necesarios para añadir o borrar un tipo de elemento de su lista correspondiente.

Por otra parte, la estructura de los contenedores de información, así como de los elementos, es la siguiente:

ListaClass: Es la clase que contendrá las listas para cada uno de los elementos. El Controlador de la información instancia de manera estática un

objeto de esta clase, que será el contenedor único global de toda la información del juego.

ListaClass posee como atributo una lista dinámica para cada tipo de elemento, así como los métodos para:

- Añadir un elemento a su lista correspondiente.

- Borrar un elemento de su lista correspondiente.

- Pasar cada lista a string. Estos métodos los utilizan constantemente los paneles dentro de la interfaz de usuario que contienen la información de cada lista.

- Getters y setters correspondientes, tanto para obtener un elemento de la lista correspondiente, como para devolver la lista en sí, y también para devolver el tamaño de cada lista.

Elemento: Se trata de una clase abstracta. Todos los tipos de elementos (excepto las reglas, por tratarse de un elemento especial) heredarán de ella.

Atributos:

- Nombre: Lo da el usuario.

- ID: Lo genera el programa automáticamente.

- Propiedades: Se trata de una lista dinámica de pares clave – valor, generada casi en su totalidad por el usuario mediante la interfaz correspondiente. Cada elemento tiene su propia lista de pares.

Métodos:

- AddPropiedad(string clave, string valor): Para añadir una propiedad nueva a la lista de propiedades.

- GetPropiedad(string clave): Devuelve el valor asociado a la clave, si esta existe.

Personaje: Clase que hereda de Elemento. En su constructor se le añade la propiedad "_TIPO" con valor "Personaje"

NPC: Clase que hereda de Elemento. En su constructor se le añade la propiedad "_TIPO" con valor "NPC"

Objeto: Clase que hereda de Elemento. En su constructor se le añade la propiedad "_TIPO" con valor "Objeto"

Accion: Clase que hereda de Elemento. En su constructor se le añade la propiedad “_TIPO” con valor “Accion”

Regla: Se trata de un elemento especial, ya que no posee la misma estructura que el resto. Sus métodos son su constructor y el paso a string, y consta de dos atributos:

-Antecedentes: Se trata de una lista de listas de elementos. Es decir, cada regla puede tener varios antecedentes. Estos antecedentes, a su vez, están formados por una lista de elementos. Por lo tanto, un antecedente es una lista de elementos, y el atributo “Antecedentes” es una lista de antecedentes, por lo tanto es una lista de listas de elementos.

-Consecuente: Se trata de una lista de elementos, ya que a un objeto de la clase Regla no se le está permitido tener más de un consecuente.

Bibliografía

[1] Twine. http://twinery.org/.

[2] PlayMaker. http://hutonggames.com/.

[3] Unity3D. https://unity3d.com/.

[4] Generos de videojuegos: https://es.wikipedia.org/wiki/G%C3%A9nero_de_videojuegos.