Escuela Politécnica Superior

Videojuego para que los niños aprendan a programar.

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Juan Esquerdo Roig

Tutor/es:
Rosana Satorre Cuerda

Carlos Villagrá Arnedo

Enero 2014

Videojuego para que los niños aprendan a programar.

UNIVERSIDAD DE ALICANTE

CURSO 2013-2014

Tutores

Rosana Satorre Cuerda

Carlos Villagrá Arnedo

Alumno

Juan Esquerdo Roig

Agradecimientos

En primer lugar, quiero agradecer a mis tutores **Carlos Villagrá y Rosana Satorre,** por ayudarme y guiarme en la correcta realización de este proyecto con tanta trascendencia académica como es un trabajo de fin de grado. Gracias a ellos y a la elección de este proyecto, se me han planteado situaciones que he tenido que resolver de forma ágil acorde a mis conocimientos adquiridos durante mis asignaturas cursadas, lo cual me ha ayudado a crecer personal y profesionalmente.

En segundo lugar a los **docentes encargados del Máster Tecnologías de la Informática**, por sus conocimientos y por la gran comprensión respecto a la situación en la que me encontraba; posponiendo la entrega de algunos trabajos debido a la total dedicación y entrega del proyecto.

Y, por último, pero no por ello menos importante, agradecer **a mi familia** por apoyarme y animarme en este proyecto que cierra mi etapa de formación como ingeniero informático.

Este trabajo de fin de grado es el punto final que muestra todos los conocimientos adquiridos durante estos cuatro años. Gracias a él, he conseguido poner en práctica mis habilidades y darme cuenta de todo lo que me ha enseñado esta carrera. Me siento orgulloso de haber podido sacar el proyecto adelante, pero sobre todo, me siento orgulloso de haber aprendido una profesión que me va a abrir un enorme abanico de posibilidades en el mercado laboral.

Índice general

Capítulo 1

Introducción

1.1. Aprender a programar

Los lenguajes de programación han formado parte de los ordenadores desde sus inicios, concretamente Fortran fue el primer lenguaje de programación ampliamente conocido y exitoso (desarrollado entre 1954 y 1957). Fortran es un lenguaje de programación de alto nivel de propósito general, procedimental e imperativo, que está adaptado al cálculo numérico y a la computación científica.

Más adelante nacerían distintos lenguajes de programación con distintas características. Pero todos ellos tienen la finalidad de crear programas o tareas que exhiban un comportamiento deseado; para por ejemplo, resolver problemas numéricos; de aquí la importancia de aprender a utilizarlos.

Aprender a programar requiere frecuentemente conocimientos en diversas áreas distintas, además del dominio del lenguaje a utilizar. Es por ello que aprender a programar puede ser una ardua tarea si no se dispone de los recursos correctos.

Actualmente se dispone de distintos recursos para el aprendizaje de la programación, se siguen consultando como apoyo los recursos tradicionales (libros, clases); pero el aprendizaje de los lenguajes de programación (y otras ramas) está tendiendo hacia recursos más interactivos; ya que si no tenemos metas concretas, o algún aliciente que nos anime a seguir, es posible que abandonemos el aprendizaje.

Podemos recurrir a métodos divertidos y estimulantes para aprender como por ejemplo aprender jugando. Y es que podemos encontrar juegos que nos permiten mejorar nuestras habilidades como programadores.

Una de las principales razones de la investigación y desarrollo de estos recursos interactivos (como por ejemplo los videojuegos) es debido a que desde una edad temprana puede resultar más fácil aprender el concepto abstracto de la programación; resultando para un niño más divertido el aprendizaje de los lenguajes de programación mediante un videojuego.

En la web podemos encontrar videos explicando distintos lenguajes de programación, páginas interactivas donde podremos comprobar las sentencias que estamos aprendiendo, cursos online y un largo etcétera. Pero estos recursos pueden llegar a resultar un poco tediosos y aburridos para un niño.

1.2. Objetivos del proyecto

Se desarrollará un videojuego para PC para que los niños aprendan a programar, y se centrará principalmente en el aprendizaje de diferentes estructuras de programación en diferentes partes del proyecto, aunque no es competencia de este proyecto el que el juego esté completo en su totalidad, sino que sea jugable y pueda mostrar los elementos que lo componen como la interfaz usuario y principalmente el aprendizaje de programación del juego, para lo que se usarán ciertos elementos como sonidos, música o fondos de pantallas que no son propiedad del autor de este trabajo de final de carrera sino que son de licencia “Creative Commons No Comercial” y que, en caso de usarse una parte o totalidad del presente proyecto en futuros proyectos comerciales, deberán ser removidos para evitar problemas legales.

Para lograr el objetivo, diseñará e implementará un sistema de interfaz GUI que permitirá al usuario interactuar con el videojuego y viceversa mediante sus elementos y navegar entre diferentes niveles y que será reutilizable en futuros proyectos en los que sea necesaria una interfaz de usuario en entornos gráficos.

Podremos elegir entre distintos tutoriales donde aprenderemos las sentencias básicas de programación y además un nivel final donde se aprenderán más características de las sentencias básicas; y la combinación de varias sentencias.

Habrá un personaje principal que controlará el usuario mediante las flechas del teclado y que al encontrarse con diferentes obstáculos, podrá eliminar los obstáculos escribiendo la sentencia de programación correcta para ese evento; pasando a la selección de un nuevo nivel o avanzando en el nivel final, creando una sensación de avance progresivo durante el transcurso de la partida; además de adquirir el conocimiento de la sentencia que necesitara para posteriores obstáculos.

Se comprueba la sentencia escrita por el usuario es correcta; independientemente de los espacios, tabulaciones o saltos de línea que el usuario escriba; ya que cada usuario tiene una manera personal de escribir el código. Además se avisa al usuario si ha cometido un error y se especifica qué error (si se ha olvidado un paréntesis o las llaves, etc.).

1.3. Metodología

El desarrollo de un videojuego engloba ciertas limitaciones en la planificación como definición de requisitos débiles o poco específicos, largos periodos de construcción del proyecto que dificultan la planificación temporal, la necesidad de mucho aprendizaje y la incertidumbre de si ciertas partes van a ser viables o no al inicio. Por todo esto se ha decidido usar como metodología de trabajo una metodología ágil ya que se ajusta con la mayoría de restricciones comentadas.

Dentro de las metodologías ágiles hay varias opciones disponibles como Scrum, xP, ASD, FDD y muchas otras, pero se ha seleccionado FDD (Feature-Driven Development), debido a que el juego dispone de muchos niveles distintos y se pueden ir añadiendo al sistema como *features* (Características de un elemento software) en cada iteración.

Hay que tener en cuenta que aunque se haya usado una metodología ágil, estas suelen estar preparadas para trabajo en equipo, y dado que este trabajo se ha desarrollado de manera individual no se han seguido todas las características de la metodología al pie de la letra.

La metodología es iterativa e incremental y se compone de 5 fases, 3 de ellas iniciales en las que se crea una visión general del proyecto y se obtiene una lista de funcionalidades ordenadas por prioridad y las dos últimas fases en las que se itera por cada una de las funcionalidades de la lista y en las que se diseña y se implementa esa funcionalidad.

Las reuniones con el cliente (en este caso se entiende como ‘cliente’ a los tutores del TFG) se han realizado en periodos de 1 semana en los que se mostraban los resultados finales de la funcionalidad programada para la iteración y se recogía el feedback para modificar si fuera necesario y posteriormente se iniciaba la siguiente iteración hasta que el producto estuviera terminado.

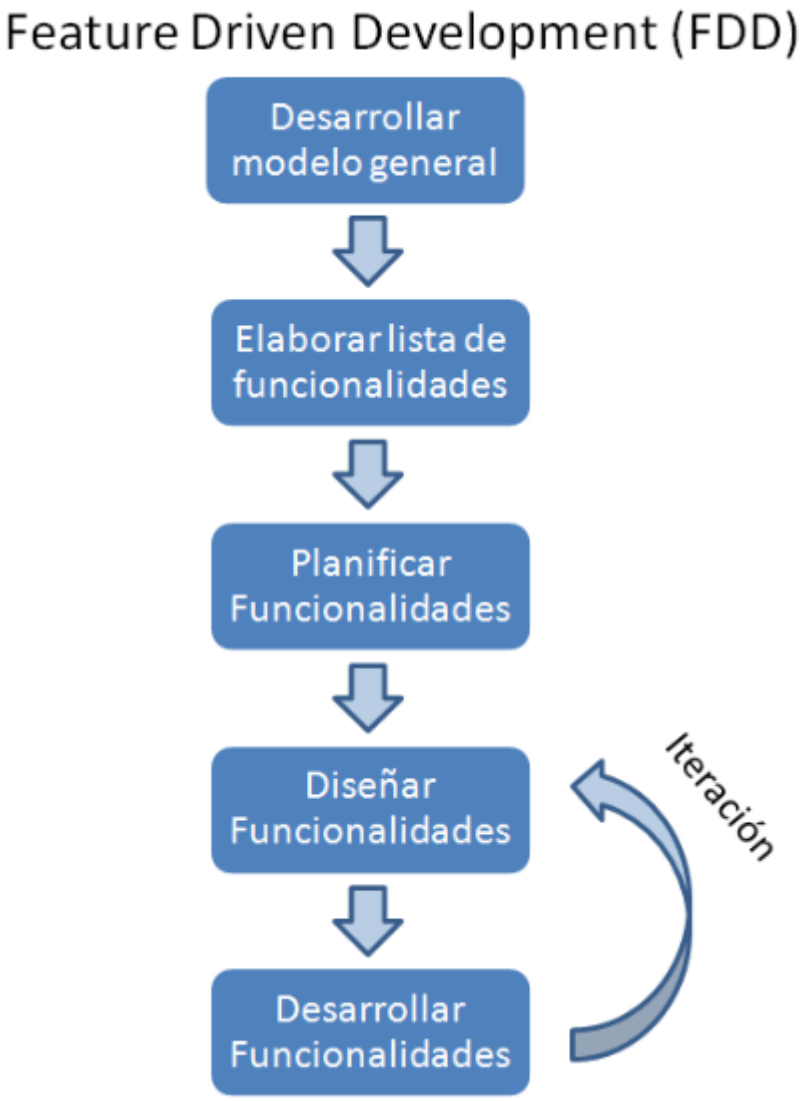

Figura 1.1: Fases de la metodología FDD

Capítulo 2

Estado del arte

2.1. Juegos para aprender a programar

Los videojuegos para aprender a programar podríamos incluirlos en el género de juegos educativos (especializados en la enseñanza de aprender a programar) y la idea principal es que el jugador avance en diferentes niveles, enfrentándose a obstáculos durante el camino para aprender de manera progresiva a programar.

Dentro de las características que definen con más detalle a este subgénero, se encuentran las siguientes:

- Diferentes niveles con diferentes obstáculos que tienen como finalidad que el usuario utilice la programación para pasar el nivel.

- La dificultad aumenta progresivamente debido a que cada vez se enseñan sentencias más complejas o la unión de varias sentencias.

- Son juegos para un solo jugador.

- La única manera de resolver el problema o sortear los obstáculos es introduciendo la sentencia correcta.

- Recibimos retroalimentación indicándonos si hemos sorteado correctamente el obstáculo o hemos fallado.

- Un nivel o varios niveles finales donde se intenta que el usuario aplique todos los conocimientos aprendidos durante el juego.

En las primeras versiones de los juegos para aprender a programar simplemente elegíamos un nivel; que representaba diferentes estructuras de programación, codificábamos; y el juego nos decía si la sentencia era correcta o si había algún tipo de error. Un ejemplo de este tipo de juego un poco más actual pero siguiendo este patrón se puede ver en el juego Check iO:

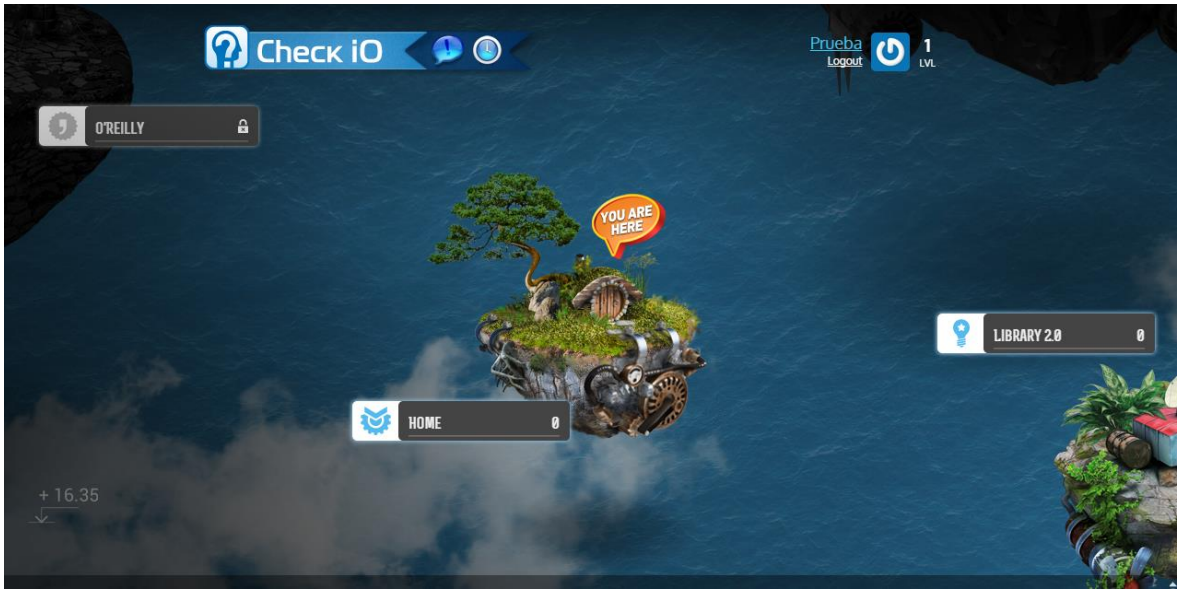

Figura 2.1: Captura de seleccionar nivel en Check iO

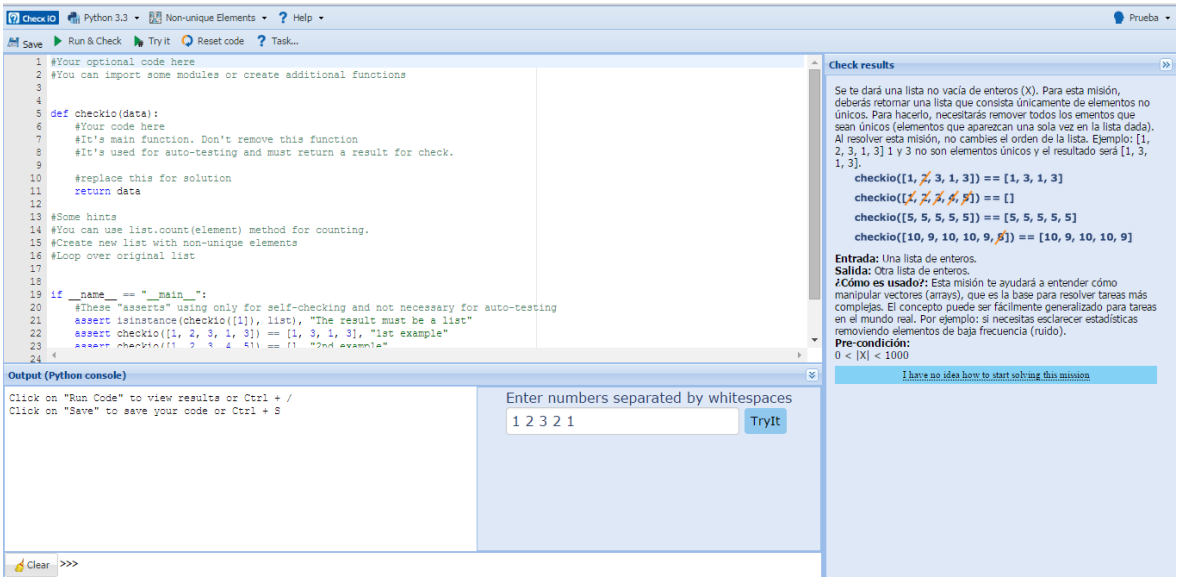

Figura 2.2: Captura de resolución de nivel en Check iO

Con el aumento constante en las mejoras de la tecnología, en las metodologías de la programación y los lenguajes de programación, se han implementado a lo largo de los años diferentes visualizaciones para el género, empezando por tener un personaje que mover; ya sea con comandos o controlándolo el usuario, como ocurre en FightCodegame:

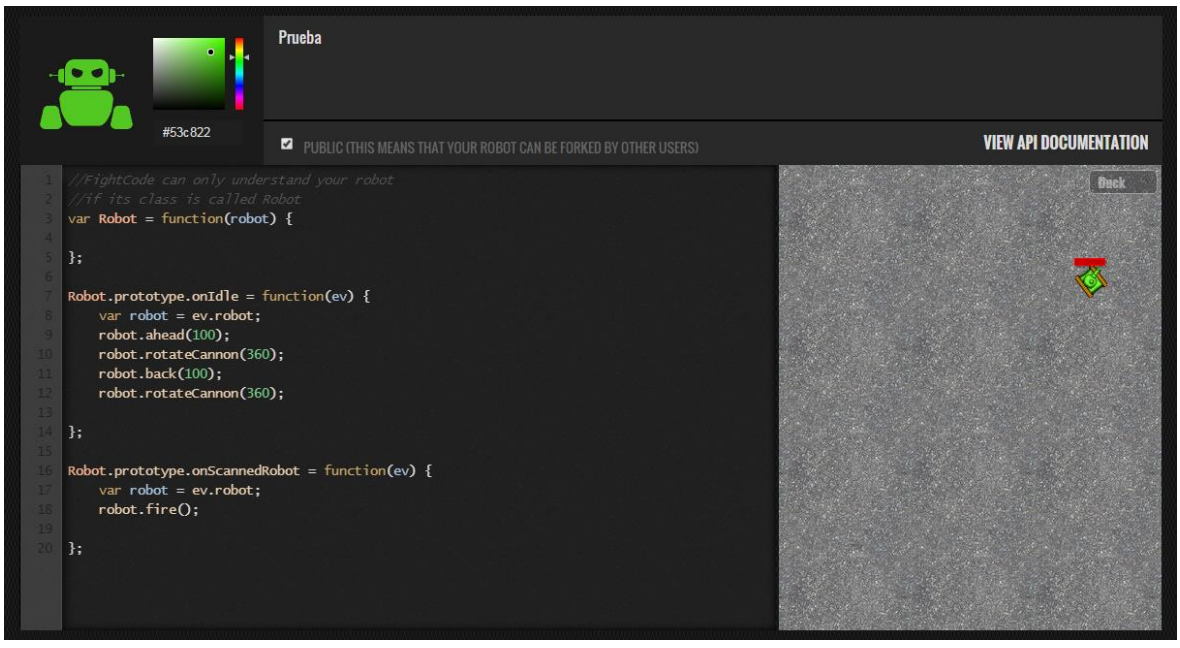

Figura 2.3: Captura de FightCodegame

Más adelante nacerían juegos en los que no solo importa aprender a programar, sino que también importa la temática, la historia, darle recompensas al jugador (más nivel, mejores objetos) y un larga lista de propiedades; dándole un aspecto al juego más logrado, captando así la atención de muchos jugadores; como por ejemplo CodeCombat, un juego de rol donde nos pondremos a los mandos de un mago que deberá enfrentarse a distintos retos realizando hechizos. Estos juegos empiezan por un nivel bastante básico; que va aumentando de forma progresiva, siendo cada vez más complejo:

Figura 2.4: Captura selección de nivel CodeCombat

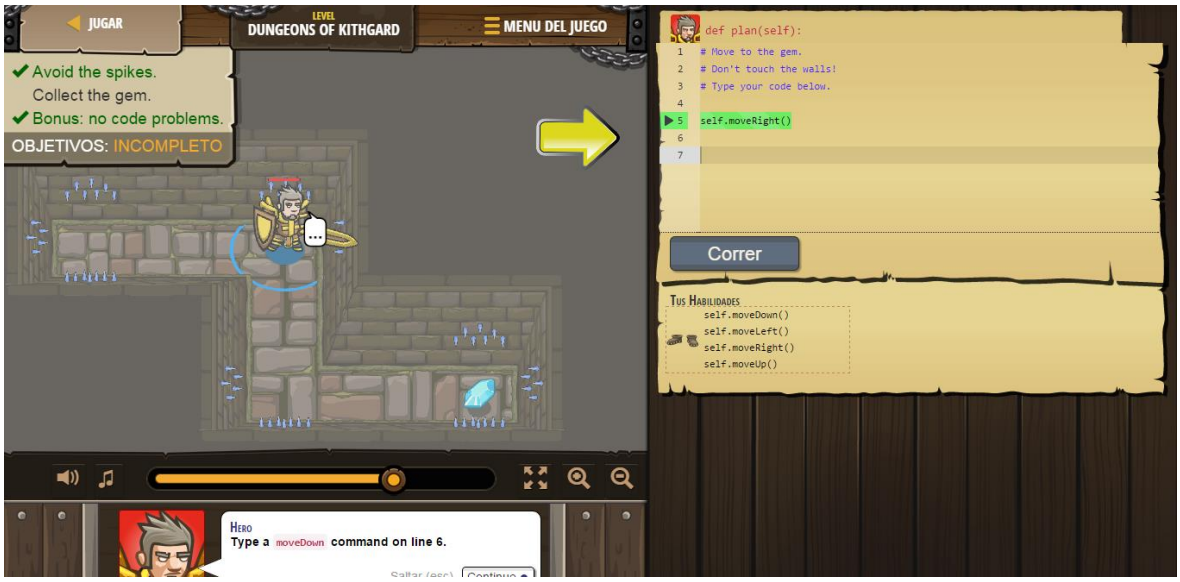

Figura 2.5: Captura resolución de nivel CodeCombat

Uno de los saltos evolutivos más notables en cuanto a representación en el género ha sido el poder jugar contra otros usuarios, además de la posibilidad de participar en torneos con distintos premios (como por ejemplo dinero) además de tener una comunidad y número de usuarios muy grande. Uno de los juegos que está marcando tendencia es JavaCup; organizado por JavaHispano, que nos permite crear nuestro propio equipo de fútbol con Java; donde podremos competir en un torneo que se realiza de forma anual, pudiendo obtener diferentes premios dependiendo de la posición en la que se quede en el torneo.

Figura 2.6: Captura de partido de JavaCup

2.2. Componentes

Una vez definido lo que es un juego para aprender a programar y mostrados algunos ejemplos de estos juegos a lo largo de los años, es hora de adentrarse un poco más en ellos y listar ciertos elementos que suelen aparecer en juegos de este estilo.

2.2.1 Escenario

Una de las partes que más llaman la atención y donde ocurrirán todos los eventos es el escenario. Es aquí donde nos encontraremos con los obstáculos, y ocurrirán los eventos pertinentes cuando resolvamos mediante la programación el obstáculo que se quiere evitar.

Figura 2.7: Escenario en CodeCombat

2.2.2 Editor de texto

Otra parte muy importante es el editor de texto donde programaremos las sentencias, debe permitir tabular, espaciar y los saltos de líneas. Algunos muestran el número de líneas y otros no, además tenemos que tener algún botón que nos permita hacer la comprobación de que el código es correcto.

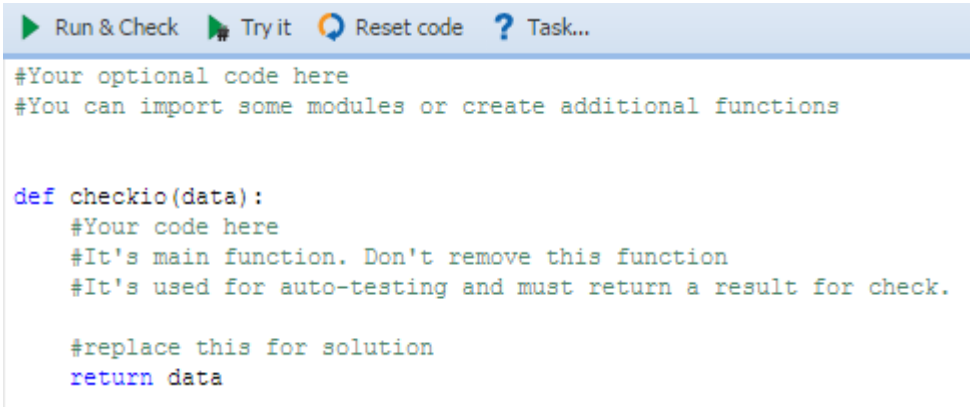

Figura 2.8: Editor de texto en Check iO

2.2.3 Instrucciones

Antes de empezar el nivel o paralelamente cuando se está programando, deben aparecer las instrucciones de lo que el usuario debe programar; explicando el uso de la sentencia; además de dar las pistas necesarias para la resolución del obstáculo o nivel.

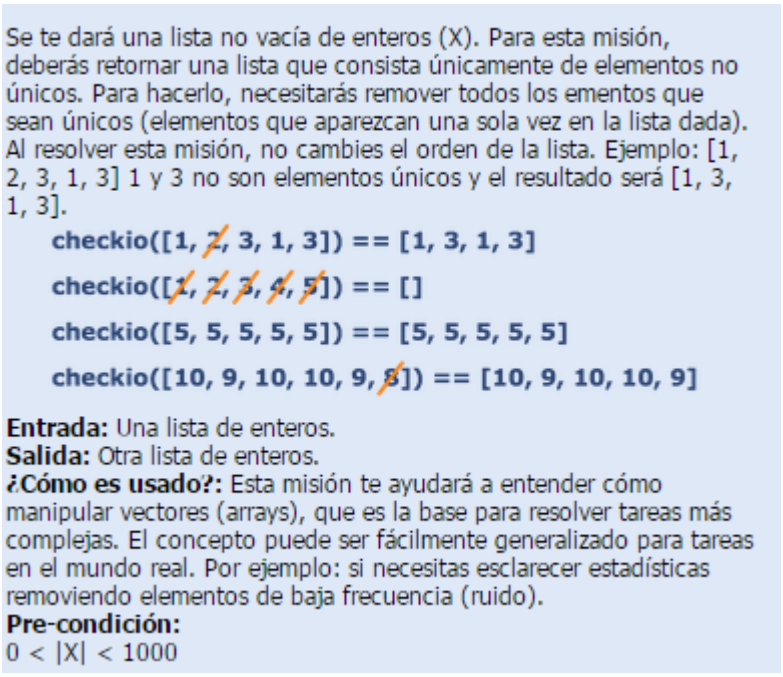

Figura 2.9: Instrucciones en Check iO

2.2.4 Retroalimentación

Una vez ejecutada la sentencia si la instrucción es correcta se debe retroalimentar al usuario para que sepa que la instrucción es correcta; en caso contrario se notificara al usuario que su instrucción es incorrecta.

Si la instrucción es incorrecta la retroalimentación debe especificar en la medida de lo posible donde se ha cometido el fallo para que el usuario pueda corregir la instrucción y así pasar de nivel.

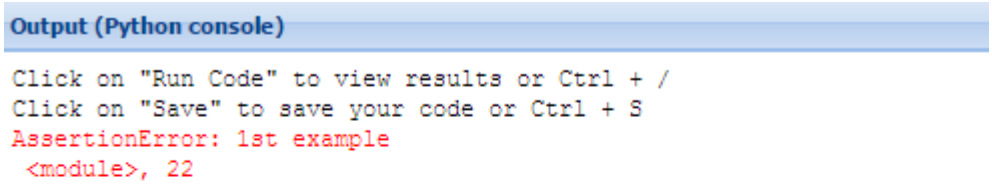

Figura 2.10: Retroalimentación en Check iO

2.3. Programación

La programación es la parte más importante en este tipo de juegos, pero hay dos factores muy importantes a tener en cuenta sobre ella.

Lo primero a tener en cuenta es el lenguaje de programación escogido, no hay ningún lenguaje que predomine, sino que el lenguaje escogido es el que más se adapte para las funcionalidades del juego, además también se tiene en cuenta el número de usuarios y la relevancia del lenguaje en concreto.

Lo segundo a tener en cuenta son las sentencias que queramos que el usuario queremos que aprenda, ya que dependiendo de las sentencias; la temática y los eventos que puede encontrar el jugador, deberán ir acorde con las sentencias que el usuario debe aprender.

Más adelante (concretamente en el capítulo 3) haremos referencia al lenguaje escogido y las sentencias utilizadas para este proyecto.

Capítulo 3

Lenguaje de programación a aprender

Hay un extenso mundo de lenguajes de programación que se pueden clasificar en tres paradigmas principales: imperativos, declarativos y orientación a objetos; por lo que primero el creador del videojuego deberá decantarse por un paradigma y luego inspeccionar los distintos lenguajes que cumplan ese paradigma; seleccionando el lenguaje de programación que mejor se adapte.

Aunque normalmente es subjetiva la elección de un lenguaje, ya que muchas veces; puede ser subjetivo que lenguaje puede ser mejor o peor para aprender a programar, además se suma que cada uno tiene unas preferencias más fuertes o menos fuertes hacia ciertos lenguajes.

En nuestro caso la elección de lenguaje de programación para el aprendizaje del jugador ha sido bastante clara; ya que en la universidad de Alicante el lenguaje con el que nos inicializamos es con C; es por ello que haciendo referencia a este hecho hemos elegido C.

Se han elegido las sentencias más básicas del lenguaje, que además son imprescindibles para cualquier otro lenguaje. Es por ello que hemos querido dar más énfasis en las sentencias básicas, para que así en un futuro;

independientemente del lenguaje que escojamos, tener una noción clara de las sentencias básicas.

Las sentencias básicas que explicamos en este juego son: Mostrar por pantalla, aumento y decremento de variables, condiciones, bucles while y bucles for; estas sentencias se enseñan en los niveles más básicos (en los tutoriales).

Para añadir una dificultad progresiva se ha añadido un nivel final donde encontraremos varios obstáculos en el mismo nivel; la finalidad de este nivel es de aplicar los conocimientos ya aprendidos en los anteriores tutoriales.

Además en el nivel final también se añaden nuevas sentencias que ahora enumeramos: operadores lógicos “AND” y “OR”, condición sino, bucles anidados y bucles con condiciones. Añadiendo así sentencias u operadores imprescindibles para las sentencias básicas.

Se deja para una versión futura el tratamiento de sentencias más complejas como por ejemplo el uso de vectores, la llamada y creación de funciones, el sistema de tipos y un largo etcétera.

Estas nuevas sentencias podrían añadirse en futuros niveles o incluirse en el nivel final; pudiendo acabar el juego y tener un conocimiento muy avanzado sobre el lenguaje de programación.

Capítulo 4

Análisis

4.1 Requisitos funcionales

Los requisitos funcionales nos definen las funciones del sistema de software o de sus componentes, donde cada función es un conjunto de entradas, comportamientos y salidas que define una funcionalidad específica que el software debe cumplir. A continuación se listan los requisitos funcionales de este proyecto:

- Debe existir un menú principal desde donde el usuario podrá acceder al menú de selección de niveles.

- En la sección de niveles, el usuario podrá elegir el nivel que desee jugar.

- El juego constara de cinco tutoriales y un nivel final.

- El primer tutorial definirá la sentencia “cout” para imprimir por pantalla.

- El segundo tutorial explicará el incremento y decremento de variables.

- El tercer tutorial describirá las condiciones, para ello el tutorial definirá la sentencia “if”.

- El cuarto tutorial enseñara las sentencias iterativas, concretamente la sentencia “while”.

- El último tutorial aclarará las sentencias iterativas, utilizando para ello la sentencia iterativa “for”.

- Una vez pasados los niveles, podrán repetirse los niveles las veces que el jugador desee.

- Debe existir una dificultad progresiva.

- El personaje debe poder moverse libremente por todo el nivel hasta encontrar el obstáculo; en cuyo caso se activara el evento para que el usuario empiece a programar.

- Al activar el evento debe aparecer unas instrucciones y un editor de texto.

- El usuario debe poder utilizar el número de tabulaciones, espacios y saltos de línea que desee, sin afectar a que la sentencia sea correcta o no.

- Para algunas sentencias, debe permitirse la resolución del obstáculo con pequeñas modificaciones.

- Debe existir retroalimentación tanto si la sentencia es correcta como sino.

- Si la retroalimentación es incorrecta entonces se ha de mostrar en la medida de lo posible el error que produce que esa sentencia sea incorrecta.

- Debe existir un nivel final donde se utilicen las sentencias aprendidas en los tutoriales.

- El juego debe reproducir música de fondo.

4.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que no afectan a la funcionalidad del programa pero que son exigidos por las especificaciones. En el caso de este proyecto, los requisitos no funcionales son los siguientes:

- El juego será uniplataforma; debe ser ejecutable y funcional en Windows 7 y versiones superiores.

- El idioma del juego será el castellano en su totalidad, no se contempla multi-idioma en este proyecto; a excepción de las sentencias de programación.

- El juego debe poder ser ejecutado de manera fluida en al menos un procesador Intel Core Duo a 2.1 GHz con 4 GB de RAM y tarjeta gráfica GEFORCE G210M.

- El juego se ejecutará en la resolución gráfica 800x600 aunque se podrá ejecutar en otras resoluciones.

La selección de requisitos hardware y software se ha realizado en base al equipo portátil que se ha utilizado para realizar las presentaciones del proyecto, mientras que las de idioma se basan en que, a pesar de que el multi-idioma es una característica interesante, sale de los objetivos del presente

proyecto de final de grado y por lo tanto se deja como tarea futura en caso de proseguir con el proyecto tras la entrega.

4.3 Herramientas y tecnologías

En esta sección se muestra las herramientas y tecnologías que se han empleado para la creación del proyecto, así como la justificación de las elecciones.

Motor de videojuego: Unity

Este motor de videojuegos para el manejo de elementos multimedia como fuentes, imágenes, sonido, música, modelos 3D, etc. Proporciona una capa de abstracción que permite usar estos componentes de manera más sencilla y no nos obliga a implementar muchas de las funcionalidades (como por ejemplo la colisión entre dos objetos) para poder desarrollar el proyecto.

Otra de las grandes ventajas por las que se escogió este motor es porque su curva de aprendizaje es bastante baja, en poco tiempo se puede tener un domino bastante avanzado de Unity, permitiendo así una rápida ejecución del proyecto; una de las cosas que permite esto es la gran cantidad de tutoriales y proyectos de ejemplo que podemos encontrar en la propia página. Además cuenta con una gran comunidad detrás donde la mayoría de las dudas que pueden surgir a los programadores más novatos están resueltas. Sumados todos estos factores obligan a cualquier programador novato de videojuegos a ser una herramienta a tener muy en cuenta.

Por último Unity es multiplataforma, no solo permite programar para Windows, Linux y Mac; sino que también permite crear videojuegos para móviles (Ya sean Android, Windows Phone o iPhone), o videojuegos que funcionen en el navegador; algo que hace mucho más atractivo Unity respecto a otros motores o librerías.

Lenguaje de programación: C#

Si bien es cierto que la mayoría de juegos en el mercado están programados en C++, se ha de tener en cuenta que este proyecto no es llevado por una gran compañía de videojuegos que se puede permitir el tiempo, personal y dinero necesarios para la creación del videojuego usando C/C++ y que, las ventajas de velocidad y rendimiento que se obtendrían no compensan la inversión sobre todo temporal, ya que este proyecto debe entregarse en tiempo finito.

Finalmente se ha seleccionado C# como lenguaje a usar en el proyecto, debido a la rapidez con la que permite la creación de código y que va a permitir centrarme más en los objetivos del proyecto que en batallas con el código, teniendo en cuenta además que la eficiencia va a ser menor que una compilación en C++.

Editor de código: MonoDevelop

La elección de MonoDevelop es natural, debido al lenguaje de programación y motor de videojuego seleccionados y además, se han tenido en cuenta las facilidades que este editor aporta para facilitar la tarea de creación de código (como por ejemplo su autocompletado); además respecto a su gran competidor (Visual Studio) MonoDevelop es un entorno de desarrollo integrado, libre y gratuito.

Editor 2D: Gimp 2

En el caso de manejo de imágenes 2D tales como fondos de pantalla, iconos, objetos de interfaz, etc. Ha sido necesario el uso de una herramienta de diseño gráfico además esta herramienta es gratuita respecto a sus otros competidores; y que además, es suficiente para obtener el resultado esperado.

Control de versiones: Git

Se ha utilizado como herramienta de control de versiones Git por su rapidez, sencillez, dominio y efectividad, además se ha subido el repositorio a github para que cualquier persona tenga acceso al proyecto.

Capítulo 5

Diseño del sistema

A la hora de diseñar el videojuego, se ha tenido en cuenta el modo en que Unity nos permite trabajar. Gracias a este software, la parte de dividir el diseño del videojuego queda más clara, además de que el trabajo es más modularizable gracias al diseño basado en componentes de Unity.

Cuando hablamos de un desarrollo de software basado en componentes, hablamos de crear diferentes trozos o módulos de código que puedan ser reutilizados en otras partes del videojuego o en otros videojuegos, esto nos permite reducir el tiempo de desarrollo y disminuir costes gracias a requerir menos tiempo de desarrollo. Además se mejora la calidad del videojuego ya que si un módulo está probado y su calidad es óptima, no es necesario tener que volver a implementarlo y basta con añadirlo para hacer uso del mismo.

Cuando trabajamos con Unity en realidad trabajamos con objetos vacíos a los cuales les añadimos módulos o componentes, por ejemplo para que una cámara renderice dicho objeto tendremos que añadirle al mismo un componente de render. Este funcionamiento es el mismo si queremos añadir nuestros scripts, colisionadores (colliders), cuerpos rígidos, etcétera. Mostramos un ejemplo de los componentes del protagonista en nuestro proyecto:

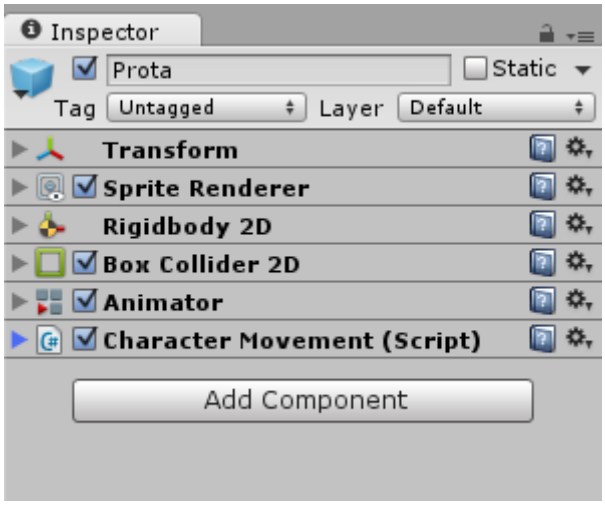

Figura 5.1: Componentes del protagonista.

Unity lleva un paso más allá el concepto de componentes y permite crear lo que denomina "prefab". Los "prefabs" son entidades creadas previamente, es decir, prefabricadas. Podemos crear objetos con los componentes y atributos que creamos oportunos y crear un "prefab" con ellos, a partir de entonces podemos emplear dicho "prefab" para crear una instancia de ese objeto siempre que queramos, además si hacemos un cambio en él podemos actualizar los objetos y viceversa.

Por ejemplo podríamos crear el protagonista como un "prefab" y copiarlo tantas veces como queramos, pero todavía más, podemos crear otra escena y añadirlo en ella. De hecho Unity nos permite incluso exportar nuestros "prefabs" e importarlos en otro proyecto, este es el verdadero potencial de Unity.

Capítulo 6

Cuerpo del trabajo

En este apartado queremos demostrar los resultados de nuestro proyecto. Para ello lo haremos en forma de demostración con algunas capturas de pantalla de lo que ha sido el proyecto. Cabe destacar que para no extendernos mucho con las imágenes vamos a hacer una demostración resumida de las partes más importantes.

Descripción del juego:

1. Empezaremos en la pantalla de inicio, donde pulsaremos el botón de empezar el juego; que nos redirigirá a la pantalla de selección de nivel.

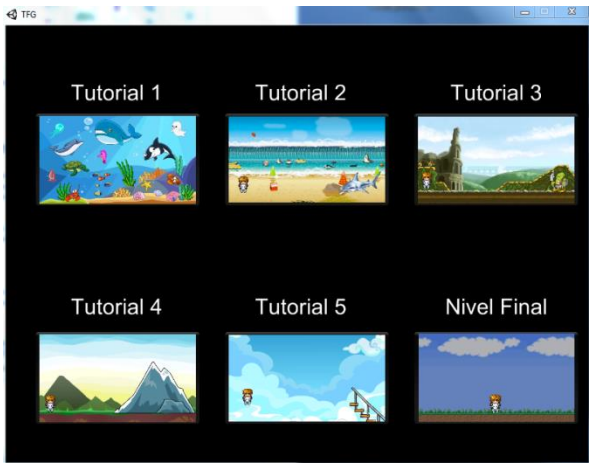

Figura 6.1: Pantalla de selección de nivel

2. Seleccionaremos el tutorial 3 por poner un ejemplo.

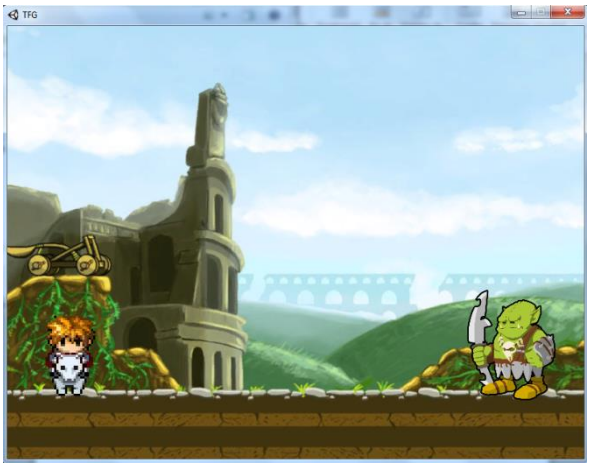

Figura 6.2: Tutorial 3

En este juego se han introducido 5 tutoriales que hacen referencia a las instrucciones más básicas de un lenguaje de programación; más concretamente, por orden de tutoriales, nos encontramos con las sentencias cout, aumento y decremento de variables, if, while y for.

Todos los tutoriales siguen el mismo patrón, tenemos en el escenario nuestro personaje y un obstáculo o enemigo. Podremos movernos por el escenario libremente, al colisionar con el obstáculo aparecerá el editor de texto y las instrucciones para sortear el obstáculo; si introducimos correctamente la sentencia entonces pasaremos de nivel; y volveremos a la selección de niveles.

3. Chocamos contra el obstáculo (En este caso el orco)

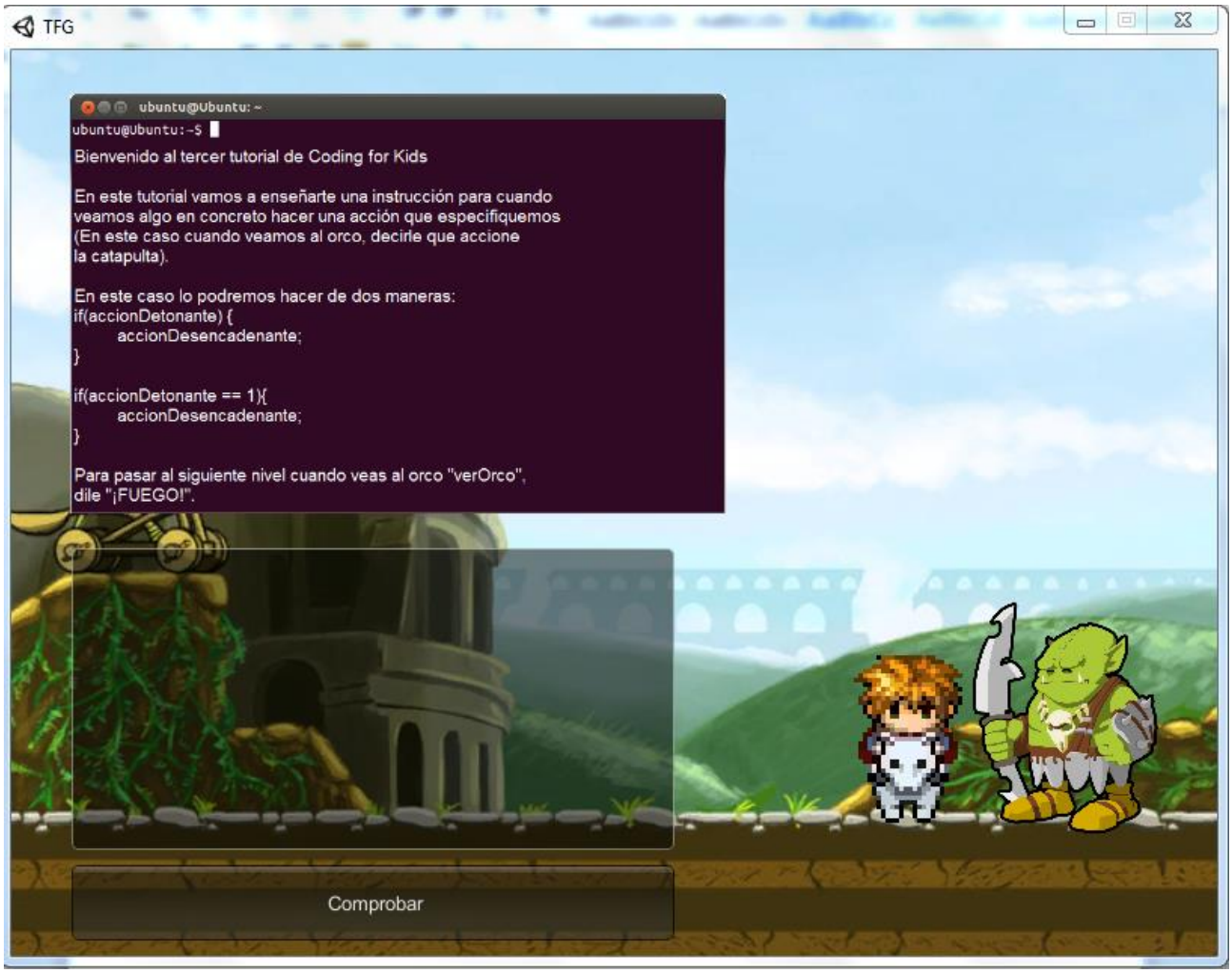

Figura 6.3: Instrucciones y editor Tutorial 3

4. Escribimos la sentencia

En caso de que la sentencia sea incorrecta nos encontraremos ante la siguiente pantalla:

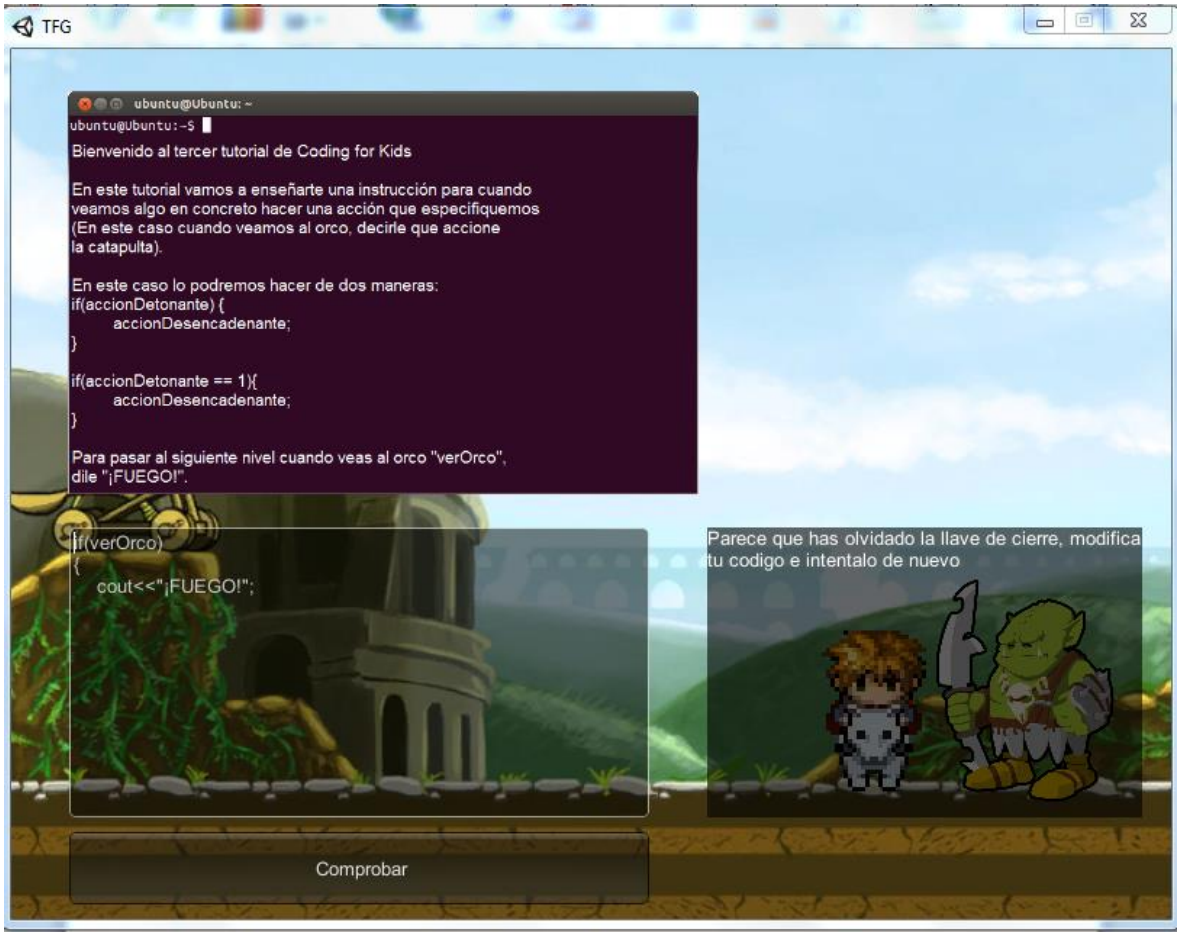

Figura 6.4: Retroalimentación cuando la sentencia es incorrecta

Cuando la sentencia es incorrecta la retroalimentación nos avisa de que hay algo incorrecto en nuestro código, en este ejemplo como podemos ver en la imagen hemos olvidado poner el cierre de llave de la sentencia if; si olvidáramos el punto y coma, las comillas o incluso si escribimos incorrectamente “cout”, el juego con la retroalimentación; nos avisaría de que hemos olvidado o debemos corregir nuestro código.

Una vez corregido nuestro código e introducido la sentencia correcta el juego mostrará que hemos pasado de nivel y nos llevará a la pantalla de selección de nivel.

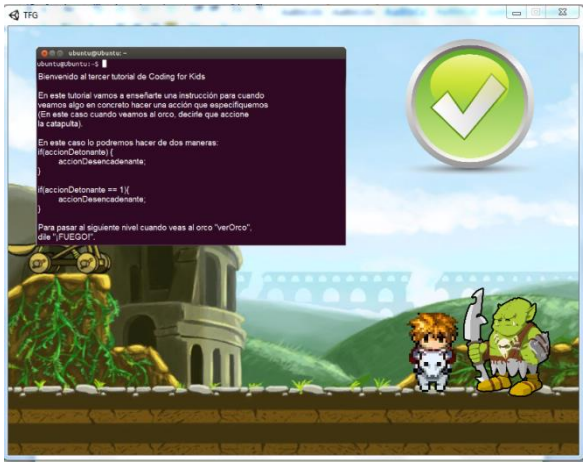

Figura 6.5: Retroalimentación cuando la sentencia es correcta

5. Selección de nivel final

A diferencia de los tutoriales el nivel final permite movernos por el escenario libremente y la cámara seguirá al protagonista; además no cuenta con un solo obstáculo sino que contamos con cinco obstáculos; obteniendo así una sensación de encontrarnos en una aventura en la que lidiaremos con varios obstáculos que deberemos sortear para finalizar la aventura.

El procedimiento es el mismo chocamos contra el obstáculo, si introducimos la sentencia correcta el obstáculo desaparece, permitiéndonos avanzar y encontrarnos con otros obstáculos; y si no mediante la retroalimentación el juego nos avisará en la medida de lo posible donde hemos cometido el error; para así corregirlo y continuar con la aventura.

Imágenes del último nivel:

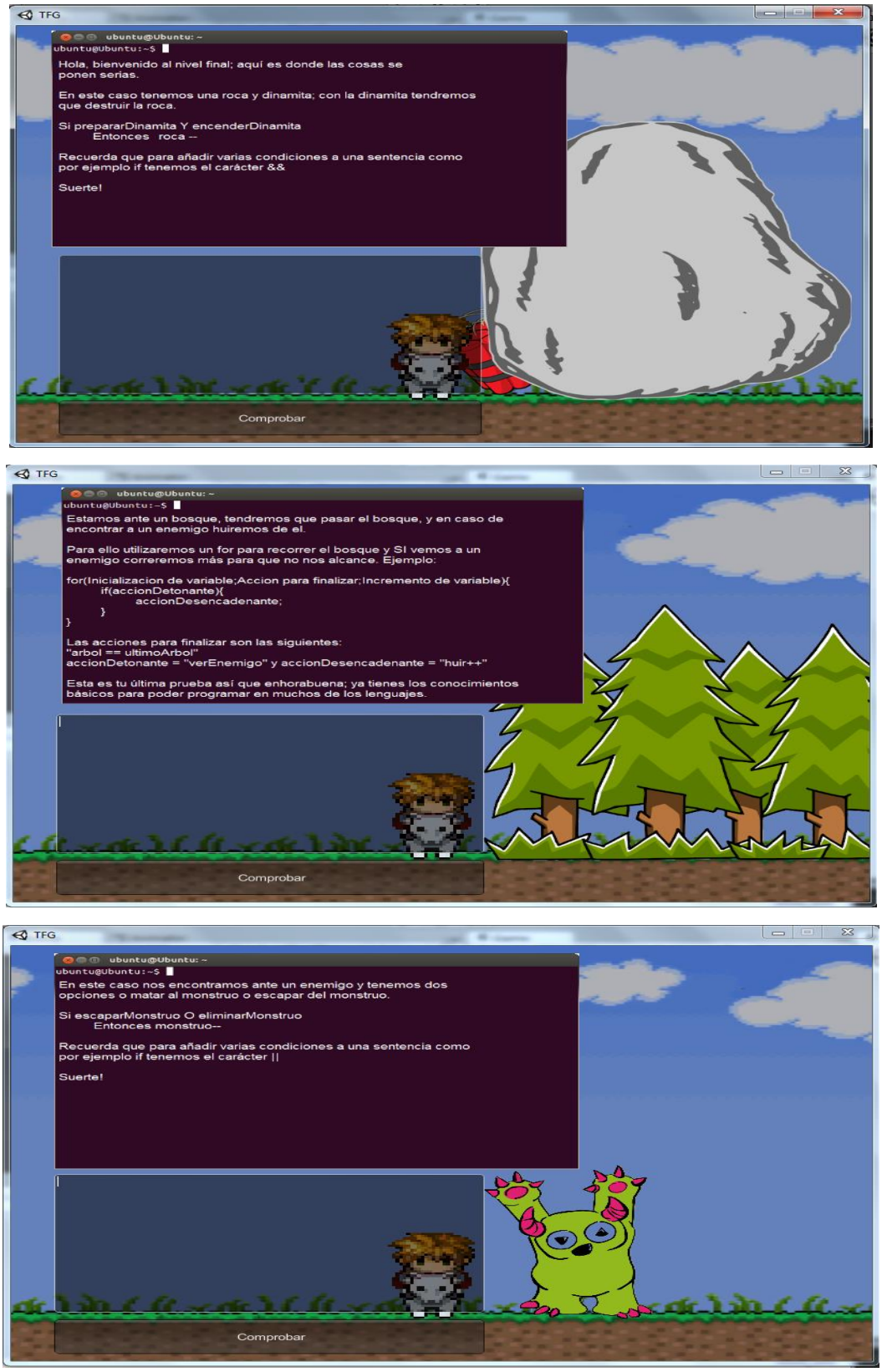

Figura 6.6: Imágenes de último nivel.

Capítulo 6

Conclusiones y futuros

6.1 Conclusiones

Con la información obtenida en la fase de investigación y siguiendo la planificación establecida, se ha llegado a obtener un resultado final jugable, como estaba estipulado en los objetivos.

En cuanto a la programación, se ha realizado un extenso esfuerzo en este apartado, y finalmente se ha conseguido por una parte crear distintas instrucciones claras y sencillas para el aprendizaje de los niños. Y por otra parte, el total uso de todas las sentencias sencillas con pequeñas variaciones de un lenguaje de programación.

La progresión en el avance de la aventura se ha conseguido en gran medida a la división por tutoriales donde se explican las sentencias más sencillas, y un nivel final; donde tendremos que poner en práctica todo lo aprendido.

Se ha implementado el control del personaje mediante el uso del teclado (concretamente las flechas) además de la parte de escritura del código mediante el teclado.

La visualización usada ha sido 2D debido a que los aspectos del juego no requerían una visualización 3D; ya que la parte más importante del videojuego es la jugabilidad y la retroalimentación; y para ello no es necesaria una perspectiva 3D.

En cuanto a la retrospectiva del juego, también se ha realizado un extenso esfuerzo, porque en caso de no escribir la sentencia correcta, se muestre en la medida de lo posible la causa del error; para así que el usuario pueda corregir su error intuitivamente al leer la causa del error.

6.2. Opinión personal

Estoy satisfecho con el resultado ya que se ha conseguido terminar con un producto jugable de una complejidad elevada; recorriendo todas las sentencias sencillas que encontramos en los lenguajes de programación.

Por otro lado, el poder completar un proyecto de estas características, con un ritmo frenético solventando problema tras problema, me ha motivado a seguir desarrollando videojuego en un futuro gracias a la experiencia obtenida y sobre todo a que gracias a este TFG ya no veo la programación de videojuegos como una utopía inalcanzable.

El mismo hecho de que realizar un videojuego de cierta complejidad es una tarea ardua y que requiere de gran cantidad de tiempo, explica por qué las grandes compañías de videojuegos disponen de grandes grupos de trabajo y no dejan la tarea a una sola persona.

Se ha podido comprobar que a pesar de que la programación del videojuego ha llevado gran cantidad de tiempo, se invierte mucho más tiempo en la parte de análisis y diseño lo que evita en gran medida que haya que refactorizar grandes porciones de código, o incluso que se deban reescribir o descartar. La búsqueda de información previa como las propiedades de los objetos ya existentes para evitar la reinvención de la rueda; también suma coste temporal, pero al final reduce el tiempo total de desarrollo del proyecto.

En resumen y para concluir, este trabajo de final de grado ha sido toda una experiencia positiva de la que espero sacar partido en un futuro.

6.3. Futuros

Dentro de objetivos futuros hay diferentes opciones viables a las que atenerse, primeramente la comprobación de si la sentencia es correcta, la retrospectiva o el editor de texto, se podrían reutilizar dichas partes para la realización de otros proyectos de videojuegos y dependiendo de la similitud al proyecto presentando en el presente documento se podrían reutilizar más o menos partes.

Otra de las posibles opciones es incorporar un compilador de C en el juego, ya que la comparación de cadenas podría mejorarse con la información de un compilador cuando se quieren abordar sentencias más complicadas, permitiendo una visión más real de los errores que encontrara cuando programe el usuario, en la retrospectiva que se le ofrece al jugador.

Para poder terminar este proyecto habría que incluir elementos multimedia propios como sonidos, músicas, gráficos o animaciones; un menú de opciones, la personalización del personaje, guardar los progresos, optimizar y depurar, etc. Plataformas como Steam soportan e incentivan el desarrollo de videojuegos indie; por lo que la comercialización del videojuego podría ser viable.

Finalmente, la última opción que se baraja es la más costosa, que sería adaptar los conocimientos adquiridos durante la realización del presente proyecto a otras plataformas móviles como iOS o Android o incluso navegador web; ya que aunque Unity es multiplataforma, y se puede reutilizar el código sin apenas modificaciones; algunas de las propiedades como por ejemplo mover el personaje con la pantalla táctil de un dispositivo móvil deberán añadirse al proyecto; teniendo versiones diferentes del proyecto para cada plataforma.

Sea cual fuere la decisión final en cuanto al futuro del presente proyecto, hay que recalcar la experiencia obtenida durante todo el proceso de realización del videojuego, donde se ha aprendido a crear un videojuego y sobre todo se han superado los constantes, diferentes y en algunos casos difíciles problemas que aparecen en la creación de un proyecto de esta envergadura.

Bibliografía

[1] Brian Schwab. *AI Game Engine Programming*. Thomson Learning, 2004.

[2] Steven John Metsker. *Design Patterns In C#.* Addison-Wesley, 2004.

[3] Roger S.Pressman. *Software Engineering 7ed*. McGraw Hill, 2010.

[4] Ian Sommerville. *Software Engineering 9ed* . 2011

[5] Ryan Henson Creighton. *Unity 4.x Game Development by Examples Beginner's Guide.* Packt Publishing 2013.

[6] Yasmin B. Kafai. *Minds in Play: Computer Game Design As A Context for Children's Learning.* Routledge 2009

Unity Documentation: https://unity3d.com/es/learn/documentation

StackOverflow: http://stackoverflow.com/