



ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

**DISEÑO E IMPLEMENTACIÓN DE UN MOTOR PARA AVENTURAS
GRÁFICAS EN UNITY2D**

Alicia Guardeno Albertos

9 de septiembre de 2015



ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

DISEÑO E IMPLEMENTACIÓN DE UN MOTOR PARA AVENTURAS GRÁFICAS EN UNITY2D

- Departamento: Ingeniería Informática
- Director del proyecto: Manuel Palomo Duarte
- Autor del proyecto: Alicia Guardeño Albertos

Cádiz, 9 de septiembre de 2015

Fdo: Alicia Guardeño Albertos

Agradecimientos

Me gustaría agradecer y dedicar este texto a:

- Manuel Palomo por su labor de tutorización y consejos.
- Javier Cadenas por su asesoramiento respecto al diseño de videojuegos orientado al género de las aventuras gráficas.
- José Joaquín Rodríguez por sus consejos y sugerencias sobre cuales libros escoger para la documentación histórica.
- Celia Fermoselle, *boredBit*, Laura J. Torres, Daniel Brey y Encarnación M. R. por ofrecerme su colaboración para elaborar diversos recursos necesarios para el proyecto.
- Eduardo Garabito por sus sugerencias e ideas para mejorar el proyecto.
- Eric Juste por ser mi querido *betatester* y apoyo moral durante la realización del proyecto.
- Toda mi familia por su cariño incondicional durante todo este tiempo.
- La comunidad de Unity por ayudarme a resolver dudas y consejos sobre la implementación.
- La comunidad de Indieorama, ZehnGames, Games Tribune, Devuego y muchos más por su labor de difusión sobre este proyecto, sin ella no habría conseguido a los colaboradores que me ayudaron en su momento.

Licencia

Este documento ha sido liberado bajo Licencia GFDL 1.3 (GNU Free Documentation License). Se incluyen los términos de la licencia en inglés al final del mismo. Sin embargo, las imágenes de videojuegos comerciales incluidos están sujetos a copyright y no se distribuyen bajo licencia libre.

Copyright (c) 2015 Alicia Guardeño Albertos.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Notación y formato

Para poder mantener un estilo uniforme y legible, a lo largo de esta memoria de **Proyecto Final de Carrera** se ha utilizado la siguiente notación:

- Para referirnos a nombres de ficheros o funciones de un lenguaje usaremos: `unity.cs`.
- Cuando mencionemos el nombre de un programa se hará de la siguiente manera: *Unity*.
- Los nombres de los objetos importantes que hayamos creado en *Unity* se escriben con este formato: **GameController**.
- Los componentes de los diversos tipos de objetos que contiene *Unity*, excepto cuando se traten de *Scripts*, tendrán el siguiente formato: **Rigidbody**.
- En el caso de adjuntar un fragmento de código, utilizamos bloques como el siguiente:

```
1 //Ejemplo de programa en C#
2
3 using System;
4
5 namespace HelloNameSpace{
6
7     public class HelloWorld
8
9     {
10         static void Main(string[] args) {
11
12             Console.WriteLine("Hola Mundo!");
13         }
14     }
15 }
```


Índice general

| | |
|---|-----------|
| 1. Motivación y contexto del proyecto | 1 |
| 1.1. Introducción | 1 |
| 1.2. Conceptos y terminología previos a la realización del proyecto | 2 |
| 1.2.1. Historia del auge, caída, y resurgir de las aventuras gráficas | 2 |
| 1.2.2. Videojuegos en el ámbito educacional | 18 |
| 1.3. Motivaciones | 25 |
| 1.4. Objetivos | 26 |
| 1.4.1. Sobre este documento | 27 |
| 2. Organización temporal | 29 |
| 2.1. Planificación inicial | 29 |
| 2.1.1. Diagrama de Gantt | 29 |
| 2.1.2. Etapas planeadas inicialmente para el desarrollo del proyecto | 32 |
| 2.2. Aparición de problemas para seguir la planificación inicial | 35 |
| 2.3. Metodología de desarrollo del proyecto: SCRUM | 36 |
| 2.3.1. Búsqueda y elección del nuevo método | 36 |
| 2.3.2. Metodología SCRUM personalizada usando Trello | 38 |
| 2.3.3. Diagrama de Gantt final | 41 |
| 2.3.4. Etapas del desarrollo del proyecto seguidas de manera orientativa | 42 |
| 3. Desarrollo | 47 |
| 3.1. Metodología | 47 |
| 3.2. Documentación | 47 |
| 3.2.1. Documentación histórica | 47 |
| 3.2.2. Documentación de Unity y C#/.NET | 48 |
| 3.2.3. Documentación del Diseño del Juego | 49 |
| 3.2.4. Documentación del Puzzle | 49 |
| 3.3. Pasos previos | 50 |
| 3.3.1. Instalación de Unity | 50 |
| 3.3.2. Preparación del proyecto de Unity para ser subido a un repositorio | 51 |
| 3.3.3. Normas de estilo para código de los scripts en C#/.NET y Assets de Unity | 53 |
| 3.3.4. Organización de Assets | 54 |
| 3.3.5. Instalación del plugin de documentación Doxygen en Unity | 55 |
| 3.4. Análisis | 55 |
| 3.4.1. Especificación de requisitos del sistema | 55 |
| 3.4.2. Modelos de Casos de Uso | 71 |
| 3.5. Diseño | 82 |
| 3.6. Implementación | 86 |
| 3.6.1. Capas jerarquizadas para el orden de dibujado de los gráficos | 86 |

| | |
|--|------------|
| 3.6.2. Gestión de estados de juego | 88 |
| 3.6.3. Sistema de localización de textos | 91 |
| 3.6.4. Objetos interactivos | 97 |
| 3.6.5. Detección y gestión de colisiones | 101 |
| 3.6.6. Sistema de inventario | 105 |
| 3.6.7. Sistema de audio | 112 |
| 3.6.8. Interfaces hechas en Unity UI | 118 |
| 3.6.9. Integración de base de datos hecha en SQLite y Unity | 124 |
| 3.6.10. Búsqueda de caminos | 134 |
| 3.7. Pruebas | 142 |
| 3.7.1. Pruebas unitarias | 142 |
| 3.7.2. Pruebas de integración | 142 |
| 3.7.3. Pruebas de jugabilidad | 142 |
| 3.7.4. Pruebas de interfaz | 143 |
| 4. Conclusiones | 145 |
| 4.1. Conclusiones personales | 145 |
| 4.2. Conclusiones técnicas | 148 |
| 4.3. Trabajos futuros | 149 |
| 5. Software utilizado | 151 |
| 6. Manual de instalación | 161 |
| 7. Manual de usuario | 165 |
| 8. Manual básico para editar y traducir los textos de 1812: la aventura | 177 |
| 9. Manual de uso del Motor de Aventuras Gráficas 2D en Unity | 181 |
| 10. Comunidad y difusión | 225 |
| Bibliografía y referencias | 229 |
| GNU Free Documentation License | 237 |
| 1. APPLICABILITY AND DEFINITIONS | 237 |
| 2. VERBATIM COPYING | 238 |
| 3. COPYING IN QUANTITY | 238 |
| 4. MODIFICATIONS | 239 |
| 5. COMBINING DOCUMENTS | 240 |
| 6. COLLECTIONS OF DOCUMENTS | 241 |
| 7. AGGREGATION WITH INDEPENDENT WORKS | 241 |
| 8. TRANSLATION | 241 |
| 9. TERMINATION | 241 |
| 10. FUTURE REVISIONS OF THIS LICENSE | 242 |
| 11. RELICENSING | 242 |
| ADDENDUM: How to use this License for your documents | 242 |

Indice de figuras

| | | |
|-------|--|----|
| 1.1. | Colossal Cave Adventure (PDP-10, 1977), la primera aventura conversacional | 3 |
| 1.2. | Mystery House, la primera aventura con gráficos | 3 |
| 1.3. | Maniac Mansion (1987), con su interfaz exclusivamente <i>point-and-click</i> | 4 |
| 1.4. | Manhunter: New York (1989), con una interfaz <i>point-and-click</i> primitiva con teclas | 4 |
| 1.5. | Shadow of the Comet (MS-DOS, 1993), realizado por la compañía francesa Infogrames . | 5 |
| 1.6. | Policenauts (PlayStation, 1994) | 6 |
| 1.7. | Myst (Mac OS, 1993) | 6 |
| 1.8. | Indiana Jones and the Last Crusade: The Graphic Adventure (1989) | 7 |
| 1.9. | The Secret of Monkey Island (1990) | 7 |
| 1.10. | Day of the Tentacle (1993) | 8 |
| 1.11. | Leisure Suit Larry in the Land of the Lounge Lizards (1987, remake: 1991) | 8 |
| 1.12. | Broken Sword: The Shadow of the Templars (Microsoft Windows y Mac OS, 1996) . . | 9 |
| 1.13. | Grim Fandango (Microsoft Windows, 1998) | 10 |
| 1.14. | Hotel Dusk: Room 215 (Nintendo DS, 2006) | 11 |
| 1.15. | The Walking Dead (Xbox 360, Playstation 3, Microsoft Windows, Android, IOS, OS X, PlayStation 4 y PlayStation Vita, 2012) | 12 |
| 1.16. | Broken Age (Microsoft Windows, OS X, Linux, Ouya, iOS, Android, PlayStation 4 y PlayStation Vita, 2014) | 13 |
| 1.17. | Don Quijote (ZX Spectrum, Amstrad CPC, Commodore 64, MSX, Atari ST y MS-DOS , 1987) | 14 |
| 1.18. | Hollywood Monsters (PC, 1997) | 15 |
| 1.19. | Randal's Monday (PC, 2015) | 16 |
| 1.20. | Dead Synchronicity (PC, 2015) | 16 |
| 1.21. | Dead Synchronicity: The Longest Night (PC, 2015) | 17 |
| 1.22. | Caballeros Templarios jugando al ajedrez, según el Libro de los Juegos (1283) | 18 |
| 1.23. | Army Battlezone tenía unos gráficos superiores a su época al ser en un principio de uso militar | 19 |
| 1.24. | ¿Dónde está Carmen Sandiego en el mundo? (Apple II, 1985) | 20 |
| 1.25. | La saga Mario is Missing (MS-DOS, 1992) fue uno de los intentos fallidos por parte de Nintendo de realizar juegos educativos | 21 |
| 1.26. | Brain Training del Dr. Kawashima (Nintendo DS, 2005), un juego que estimulaba la mente para que pensáramos más rápido | 22 |
| 1.27. | Simulador de vuelo profesional para enseñar a los nuevos pilotos | 23 |
| 2.1. | Planificación del proyecto desde abril de 2014 hasta septiembre de 2014 | 31 |
| 2.2. | Diagrama explicativo del funcionamiento de Scrum | 37 |
| 2.3. | <i>Trello</i> en uso para hacer el seguimiento del desarrollo del proyecto | 40 |
| 2.4. | Diagrama de Gantt orientativo del desarrollo del proyecto desde octubre de 2014 a septiembre de 2015 | 42 |

| | | |
|-------|--|-----|
| 3.1. | Pantalla de creación de nuevo proyecto en <i>Unity</i> | 51 |
| 3.2. | Consola de comandos de GitHub For Windows | 52 |
| 3.3. | Diagrama de flujo de las pantallas de la demo técnica de 1812: La aventura | 57 |
| 3.4. | Boceto del Menú Principal | 58 |
| 3.5. | Boceto de la pantalla de Cargar Partida | 59 |
| 3.6. | Boceto de la pantalla de Créditos | 60 |
| 3.7. | Boceto de la pantalla del Menú de Opciones | 61 |
| 3.8. | Boceto de la pantalla Mapa | 62 |
| 3.9. | Boceto de la pantalla de Partidas | 63 |
| 3.10. | Boceto de la pantalla del Buscador de Gadipedia | 64 |
| 3.11. | Boceto de la pantalla de Resultados de Gadipedia | 65 |
| 3.12. | Boceto de la pantalla de Artículo de Gadipedia | 66 |
| 3.13. | Boceto de la pantalla de Lista de Notas | 67 |
| 3.14. | Boceto de la pantalla de Nota | 68 |
| 3.15. | Boceto de la pantalla de Ajustes | 69 |
| 3.16. | Diagrama de Casos de Uso | 72 |
| 3.17. | Diagrama de clases UML de la pequeña parte del motor de <i>Unity</i> que vamos a usar | 83 |
| 3.18. | Diagrama de clases UML del sistema de objetos interactivos de Motor de Aventuras Gráficas 2D de Unity | 85 |
| 3.19. | Zonas transitables y sus enlaces en un nivel de <i>The Secret of the Monkey Island</i> | 136 |
| 3.20. | Grafo calculado en la búsqueda de caminos en un nivel de <i>The Secret of the Monkey Island</i> | 137 |
| 5.1. | Interfaz de <i>Unity</i> para proyectos 2D | 152 |
| 5.2. | Interfaz de <i>Monodevelop</i> en uso durante la realización del proyecto | 153 |
| 5.3. | Ejemplo de <i>Pencil</i> con un boceto de la aplicación web de Twitter. | 158 |
| 5.4. | <i>OBS</i> capturando el vídeo y audio de una <i>build</i> de la demo de 1812: La aventura | 159 |
| 5.5. | Interfaz de <i>Lightworks</i> durante un tutorial sobre cómo aprender a usarlo | 160 |
| 7.1. | Menú principal | 166 |
| 7.2. | Cargar partida | 167 |
| 7.3. | Créditos | 167 |
| 7.4. | Introducción del juego, con el protagonista dirigiéndose al despacho del profesor | 168 |
| 7.5. | Introducción del juego, justo después de que se hayan caído las banderitas del mapa situado en el tablón de la pared | 168 |
| 7.6. | Después de la introducción, el <i>Jugador</i> retoma el control del juego | 169 |
| 7.7. | Controles básicos del juego | 169 |
| 7.8. | Usando el inventario dentro del juego | 170 |
| 7.9. | Icono del menú de opciones dentro del juego | 171 |
| 7.10. | Menú de opciones | 171 |
| 7.11. | Menú de notas | 172 |
| 7.12. | Buscador del menú de gadipedia | 172 |
| 7.13. | Resultados del menú de gadipedia | 173 |
| 7.14. | Menú de mapa | 173 |
| 7.15. | Menú de ajustes | 174 |
| 7.16. | Menú de partidas guardadas | 174 |
| 7.17. | Salir del juego | 175 |
| 8.1. | Bloque de información meta dentro del fichero <i>LocatedTexts.xml</i> | 178 |
| 8.2. | Fragmento de los bloques <i>string</i> dentro del fichero <i>LocatedTexts.xml</i> | 178 |

Capítulo 1

Motivación y contexto del proyecto

1.1. Introducción

Este documento representa la memoria del desarrollo seguido en el Proyecto Final de Carrera **Motor de Aventuras Gráficas 2D en Unity**.

Unity es un motor de desarrollo de videojuegos, siendo un videojuego o juego de vídeo un juego electrónico (o software) en el que una o más personas interactúan, por medio de un controlador, con un dispositivo dotado de imágenes de vídeo. Este dispositivo electrónico, conocido genéricamente como “plataforma”, puede ser un ordenador, una máquina arcade, una videoconsola o un dispositivo portátil (un teléfono móvil, por ejemplo). Los videojuegos son, hoy por hoy, una de las principales industrias del arte y el entretenimiento.

Dentro de los videojuegos, existen varios géneros según las acciones físicas o lógicas necesarias para ganar a un videojuego, uno de estos géneros es el de las aventuras, siendo uno de los géneros más antiguos que existen dado que se podían implementar sin necesidad de máquinas que soportasen gráficos ni periféricos de entrada específicos (por ejemplo, un ratón o un *joystick*), pues con un teclado para introducir los comandos para jugar era más que suficiente.

A partir de este género primitivo de aventuras, más conocidas como aventuras conversacionales (por lo de que el videojuego te va contestando según las acciones que vayas escribiendo que realice), fueron surgiendo las aventuras gráficas gracias al avance de las tecnologías, entre ellos la inclusión de visionado de gráficos en las pantallas y ratón como periférico en los ordenados. Ahora mediante el ratón se podía interactuar con los gráficos de un escenario, pero la base para resolverlos seguía siendo la misma: resolver puzzles, planteados como situaciones que se suceden en la historia, de manera lógica y ordenada.

Esto unido con el alzamiento de los juegos “serios” y/o educativos en popularidad y viendo sus beneficios en cualquier ámbito, hicieron que me plantease desarrollar como demostración técnica del **Motor de Aventuras Gráficas 2D de Unity** una pequeña aventura gráfica con tintes educativos dado que estos videojuegos se concentran en usar lógica y conocimientos para poder completarse.

Si bien el proyecto va a ser principalmente el desarrollo de un proyecto en *Unity* para crear aventuras gráficas 2D de corte clásico en los que tanto sus componentes como su código (en la forma de scripts)

puedan ser reutilizados, personalizados, o expandidos por terceras personas para crear sus propios proyectos; decidí que para llevar a cabo la demostración del sistema crearía lo que sería la demo de un videojuego llamado **1812: La aventura**. Dicho videojuego sería, obviamente, una aventura gráfica, ambientada en la Cádiz actual cuyo objetivo es tanto de entretenir al jugador y hacerle resolver un puzzle relacionado con Cádiz en los años que fue asediada y vieron nacer la Constitución de 1812.

En **1812: La aventura** controlaremos a un estudiante de la Licenciatura de Historia de la Universidad de Cádiz, que recientemente ha suspendido un examen y va al despacho a pedir una revisión para su nota. Sin embargo, no logra encontrar al profesor y se embarca en una pequeña aventura para descubrir el paradero de dicho profesor ayudándose de las pistas obtenidas al resolver el puzzle con datos relacionados de la Cádiz de 1812. La demo de **1812: La aventura** estará documentada con un documento de diseño como si fuera una aventura gráfica completa, pues la idea es ayudar a orientar con la documentación de este proyecto a posibles desarrolladores interesados en realizar aventuras gráficas 2D con *Unity*.

Se incluye como extra, la documentación de cómo se buscó información y el proceso para definir poco a poco el puzzle incluido en la demo de **1812: La aventura**. Queriendo demostrar con este puzzle, una pequeña parte del proceso que supondría realizar un videojuego para entornos educativos.

1.2. Conceptos y terminología previos a la realización del proyecto

En esta sección hablaremos largo y tendido sobre el género de las aventuras gráficas: cómo surgieron, su evolución tanto en mecánicas, tecnología y ventas; la importancia que tiene en estos últimos años los nuevos estudios independientes que han hecho que vuelva a resurgir el género y en última instancia, un breve repaso sobre las aventuras españolas. Además de eso, también comentaremos de forma breve la historia y el devenir de los juegos en el ámbito educacional.

1.2.1. Historia del auge, caída, y resurgir de las aventuras gráficas

La aventura gráfica es un subgénero de los videojuegos de aventura. Su mecánica consiste en ir avanzando por el mundo, escenario o juego a través de la resolución de diversos puzzles, planteados como situaciones que se suceden en la historia, interactuando con personajes y objetos a través de un menú de acciones o interfaz similar, utilizando un cursor para manejar al personaje y realizar las distintas acciones. En su concepción clásica, esta siempre incluía la visión de los personajes en tercera persona, aunque en varias ocasiones se planteasen en primera persona.

Este género se originó a partir de las aventuras conversacionales anteriores a los años 80. Era una época en la que los ordenadores personales aún carecían de gráficos y sólo se podía interactuar con ellos escribiendo líneas de comandos a lo que te contestaban en texto.

Como se estaban instaurando con éxito en los hogares, pronto hubo gente que se dedicó a buscar un nuevo abanico de entretenimiento y ocio basado en ellos. De ellos nacieron las aventuras conversacionales, la primera de ellas fue Colossal Cave Adventure [121] (véase la figura 1.1). En estos la acción se desarrollaba describiendo en un párrafo la situación actual del protagonista y abajo un cuadro de texto en el que había que escribir sencillas frases para interactuar con el entorno, del tipo “hablar con el anciano”, “usar llave”, “salir por la puerta” o indicando puntos cardinales para ir de un sitio a otro, como

por ejemplo “norte” o “sur”.

```
.RUN ADV11
WELCOME TO ADVENTURE!! WOULD YOU LIKE INSTRUCTIONS?

YES
SOMEWHERE NEARBY IS COLOSSAL CAVE, WHERE OTHERS HAVE FOUND
FORTUNES IN TREASURE AND GOLD, THOUGH IT IS RUMORED
THAT SOME WHO ENTER ARE NEVER SEEN AGAIN. MAGIC IS SAID
TO WORK IN THE CAVE. I WILL BE YOUR EYES AND HANDS. DIRECT
ME WITH COMMANDS OF 1 OR 2 WORDS.
(ERRORS, SUGGESTIONS, COMPLAINTS TO CROWTHER)
(IF STUCK TYPE HELP FOR SOME HINTS)

YOU ARE STANDING AT THE END OF A ROAD BEFORE A SMALL BRICK
BUILDING . AROUND YOU IS A FOREST. A SMALL
STREAM FLOWS OUT OF THE BUILDING AND DOWN A GULLY.

GO IN
YOU ARE INSIDE A BUILDING, A WELL HOUSE FOR A LARGE SPRING.

THERE ARE SOME KEYS ON THE GROUND HERE.

THERE IS A SHINY BRASS LAMP NEARBY.

THERE IS FOOD HERE.

THERE IS A BOTTLE OF WATER HERE.
```

Figura 1.1: Colossal Cave Adventure (PDP-10, 1977), la primera aventura conversacional

No fue hasta en 1980, en la que la compañía Online System (que más tarde pasó a llamarse Sierra Online [137]) creó la primera aventura gráfica propiamente dicha. Este juego con gráficos rudimentarios era Mystery House [130] (Apple II, 1980) (veáse la figura 1.2), luego le siguió Wizard and the Princess [143] (Apple II, 1980) ya con gráficos en color, y finalmente establecieron el género con King’s Quest [119] (Apple II, 1984) con su aparición en varios sistemas y asentándose en la industria con cada vez títulos más fuertes.



Figura 1.2: Mystery House, la primera aventura con gráficos

El lanzamiento del Apple Macintosh y su interfaz controlada por ratón supuso la creación de las aventuras gráficas *point-and-click*, introduciendo a más empresas a este mercado. LucasArts [124] fue una de ellas, la cual consiguió un éxito rotundo con su primer juego, Maniac Mansion (véase la figura 1.3) [126], con una interfaz íntegramente *point-and-click*, imponiéndose como otra gran empresa dentro de la industria. Mientras tanto, Sierra Online también se sumaría a este cambio pasando su sistema de introducir comandos por una rudimentaria interfaz *point-and-click* con Manhunter: New York [43] (véase la figura 1.4).



Figura 1.3: Maniac Mansion (1987), con su interfaz exclusivamente *point-and-click*



Figura 1.4: Manhunter: New York (1989), con una interfaz *point-and-click* primitiva con teclas

Sierra Online y LucasArts seguían un camino lleno de rivalidades la una con la otra, aunque cada una seguiría su propio camino: Sierra Online apostaba por las grandes sagas como King's Quest [120] o

Space Quest [152], por otro lado, LucasArts prefería las aventuras sin continuidad por estos años. Los juegos de Sierra Online mantenían un desarrollo no encadenado, no obligaban a realizar una acción en concreto para poder avanzar en el juego. En cambio, los de LucasArts tenían un planteamiento más lineal en el que sólo se podía avanzar hasta cierto límite sin realizar la acción precisa.

En relación al punto anterior, las aventuras de LucasArts eran mucho más sencillas de finalizar, y por extensión, gozaban de mayor popularidad. Sierra Online apostaba por una perspectiva quizás más adulta, con historias que rozaban la épica, como en King's Quest, mientras que LucasArts era similar a su matriz cinematográfica, con un tono más apto para todos los públicos al ser de tono aventurero y humorístico. Si bien estas fueron las dos desarrolladoras más destacadas en el género, hubo varios juegos de otras compañías que también son dignos de destacar. Entre otros, Policenauts [133] (1994) (véase la figura 1.6) de Hideo Kojima, u otras centradas en el terror, tales como Shadow of the Comet [136] (véase la figura 1.5), o como la saga Clock Tower [102] de Human Entertainment [115]. Myst [129] (1993) (véase la figura 1.7) de corte fantástico, que poseía una perspectiva en primera persona en contrapunto a la tercera persona que se estilaba, tuvo una gran recepción en el público.



Figura 1.5: Shadow of the Comet (MS-DOS, 1993), realizado por la compañía francesa Infogrames

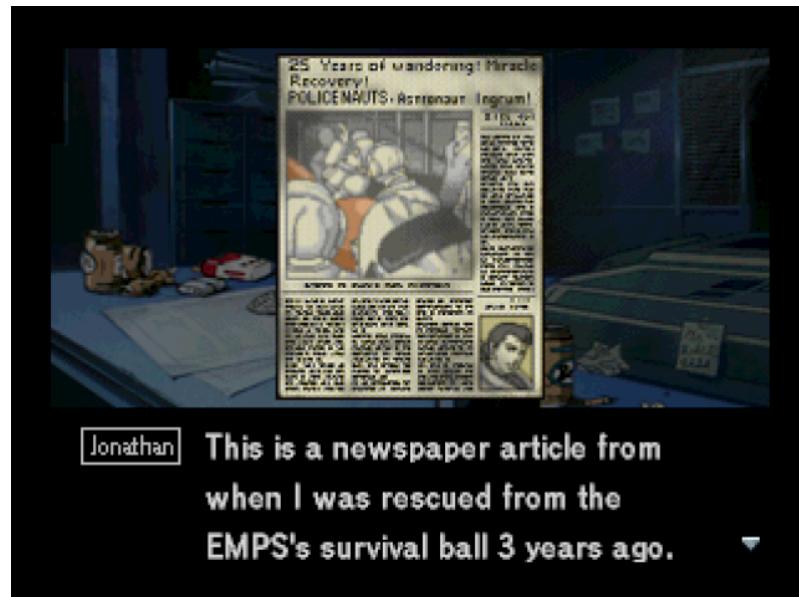


Figura 1.6: Policenauts (PlayStation, 1994)



Figura 1.7: Myst (Mac OS, 1993)

Sin embargo, no sería hasta 1989, con el lanzamiento de *Indiana Jones and the Last Crusade: The Graphic Adventure* [117] (véase la figura 1.8) de LucasArts, cuando se pondría este género de moda. Llegó la *Edad de Oro* de las aventuras gráficas. Con la aparición de los CD-ROM se podían crear aventuras cada vez más largas y con mejores gráficos, incluso algunos incorporaban elementos 3D pre-renderizados y vídeos de imagen real. Grandes aventuras, en su gran mayoría de LucasArts, marcaron este inicio de la década de los 90, tanto que algunas acabaron como iconografía de la cultura popular y en los anales de la historia de los videojuegos. Una de las más destacadas fue *The Secret of Monkey Island* [140] (1990) (véase la figura 1.9) al centrarse más en la exploración y que el protagonista no pueda morir. Otra fue *Day of the Tentacle* [103] (1993) (véase la figura 1.10), secuela de *Maniac Mansion*, que logró tal éxito que con las ganancias le permitió al director del apartado de diseño, Tim Schafer [142], producir el videojuego exitoso *Full Throttle* [109] incorporando las voces de Roy Conrad y Mark

Hamill.



Figura 1.8: Indiana Jones and the Last Crusade: The Graphic Adventure (1989)



Figura 1.9: The Secret of Monkey Island (1990)



Figura 1.10: Day of the Tentacle (1993)

Sierra Online, por su lado, seguía con sus sagas populares Space Quest y King's Quest. Les mejoró su interfaz *point-and-click* por una más amigable para el jugador, sin necesidad de introducir texto. No obstante, algunas de sus aventuras más reconocidas no llegaron a ser las de estas sagas, tal como pasó con Leisure Suit Larry in the Land of the Lounge Lizards [122] (veáse la figura 1.11) (1987, remake: 1991) que ganó en 1987 el premio al “Mejor Juego de Aventura” de la *Software Publishers Association*.

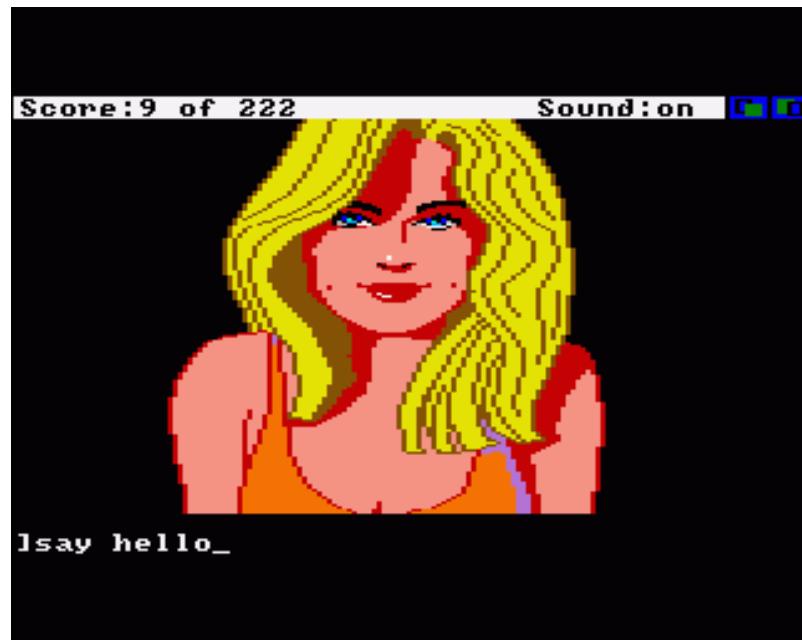


Figura 1.11: Leisure Suit Larry in the Land of the Lounge Lizards (1987, remake: 1991)

Lamentablemente, esta *Edad de Oro* duró pocos años. A finales de los años 90, el público empezó a desviar sus miradas con las mejoras gráficas y de jugabilidad en los ordenadores. Aparecieron juegos dedicados más a la acción como los shooters en primera persona, y con el asentamiento de internet en los primeros hogares, los juegos online. Las aventuras gráficas poco podían hacer con estos avances, pues su mecánica hace que sean irrelevantes, y su popularidad y ventas cayeron. Así que los editores fueron cada vez más reacios a financiar aventuras gráficas por miedo a malas ventas.

Tal fue la crisis, que Sierra Online casi cerró completamente y LucasArts dejó de publicar después del año 2000. Hubo unas cuantas perlas antes del declive, *Broken Sword: The Shadow of the Templars* [100] (1996) , una gran aventura gráfica en las que sus secuelas no estuvieron a su altura, (véase la figura 1.12) o *Grim Fandango* [110] hecho íntegramente en 3D sin la clásica interfaz *point-and-click* (véase la figura 1.13), siendo este último un fracaso en ventas a pesar de ser aclamado por la crítica.



Figura 1.12: *Broken Sword: The Shadow of the Templars* (Microsoft Windows y Mac OS, 1996)



Figura 1.13: Grim Fandango (Microsoft Windows, 1998)

Después, si bien de manera independiente o amateur se fueron haciendo pequeñas obras gracias al auge de Adobe Flash y su soporte en internet. El subgénero de “Escapar de la habitación” fue el que más juegos en su haber tuvo en estos años. Sin embargo, eran muy cortas, rudimentarias y repetitivas, no llegando casi nunca al público.

No fue hasta la llegada de otras nuevas tecnologías, un breve resurgimiento de este género. La Nintendo DS, y Wii, permitía interactuar con el juego de una manera similar a usar un ratón de ordenador. Como resultado, varios desarrolladores crearon nuevas aventuras gráficas para estas plataformas. Ejemplos de aventuras gráficas de estas plataformas incluyen Zack & Wiki: Quest for Barbaro's Treasure [47] (2007) para Wii, Hotel Dusk: Room 215 [114] (2006) para Nintendo DS que es una aventura gráfica estilada como si fuera una novela negra, y un port de Broken Sword: The Shadow of the Templars (2009) también para Nintendo DS.

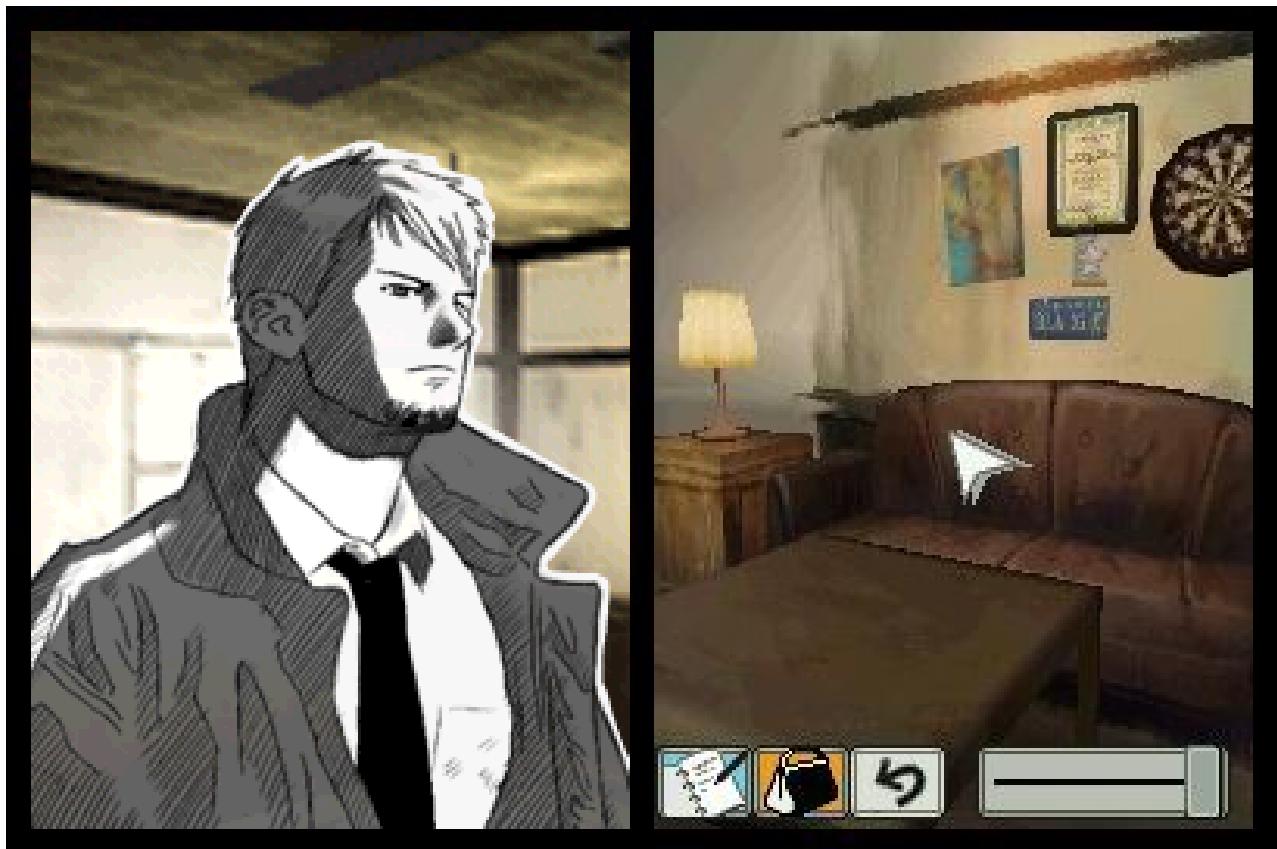


Figura 1.14: Hotel Dusk: Room 215 (Nintendo DS, 2006)

Pero sin duda, el verdadero renacer fue gracias a internet por asentarse totalmente en los hogares, y las mejoras de su velocidad a lo largo de estos años. Al fin era factible el poder promocionarte y distribuir tu juego sin costes intermedios, al menos dentro de un mercado de nicho. Una nueva compañía llamada Telltale Games [45], formada por antiguos miembros de LucasArts, empezó a producir nuevas aventuras gráficas para ordenador. Siguiendo una metodología de distribuir sus juegos de manera episódica, sus juegos incluyen Sam & Max Save the World [44] (2006), Strong Bad's Cool Game for Attractive People [138] (2008), el resurgir de Monkey Island con Tales of Monkey Island [139] (2009), Back to the Future: The Game [97] (2010).

Su ópera prima llegó con el videojuego The Walking Dead [141] de estética de cómic americano (véase la figura 1.15), aclamado tanto por la crítica como con el público. Su sistema de realizar decisiones difíciles en un momento crucial de la narrativa y ver como influían en los personajes impactó enormemente, tanto que mucha gente hizo videos en Youtube de sus reacciones y elecciones, fomentando enormemente su difusión y su venta tanto en ordenador como en consolas.



Figura 1.15: The Walking Dead (Xbox 360, Playstation 3, Microsoft Windows, Android, IOS, OS X, PlayStation 4 y PlayStation Vita, 2012)

No hay que olvidarse de otros muchos estudios independientes. Algunos optaron por seguir con las mecánicas clásicas, Machinarium [125] (2009) y Botanicula [98] (2012) de Amanita Designs son ejemplos de ello. Otras por mezclar mecánicas y nuevas tecnologías para reinventar las aventuras gráficas, como Dreamfall [107] (2006), Portal [134] (2007) y muchos otros juegos, borrando las líneas del género al mezclarlo con otros.

También podríamos incluir en este último, las películas interactivas por su semejanza con las aventuras gráficas, la más destacada Heavy Rain [111] (2010) que fue un éxito de ventas.

Actualmente, sobre todo estos últimos años, el género está gozando una *Edad de Plata*. Gracias a las redes nuevamente se han logrado realzar sistemas de financiación basados en el mecenazgo pero con tintes modernos, el micromecenazgo [128] o cómo es más conocido, **crowdfunding**. Este método de financiación se basa en que grandes cantidades de usuario pueden invertir dinero en cantías variables (poco o mucho, dependiendo de su nivel adquisitivo y/o lo que se quiera comprometer con el desarrollo del producto a financiar) directamente a los creadores del producto a desarrollar, sin necesidad de intermediarios o editores.

Además de eso, se afianzaron las plataformas de distribución de videojuegos digitales, reforzando la presencia de estudios independientes y la creación de juegos que antes no se hubieran podido permitir.

Una de las primeras en aprovecharse de este método, fue Double Fine Productions [106]. Junto con Tim Schafer, en febrero de 2012 lanzó una campaña en Kickstarter [118], la web de crowdfunding más famosa del mundo, para financiar Broken Age [99] (véase la figura 1.16). Causó un gran impacto,

recaudando hasta la inmensa cantidad de 3,45 millones de dólares, confirmando el establecimiento del crowdfunding como una alternativa viable para financiar proyectos.

Una vez allanado el camino, otros estudios independientes siguieron la estela de Double Fine. AI Lowe, original creador de Leisure Suit Larry, lanzó una campaña para financiar un remake completo de su primera obra [123].

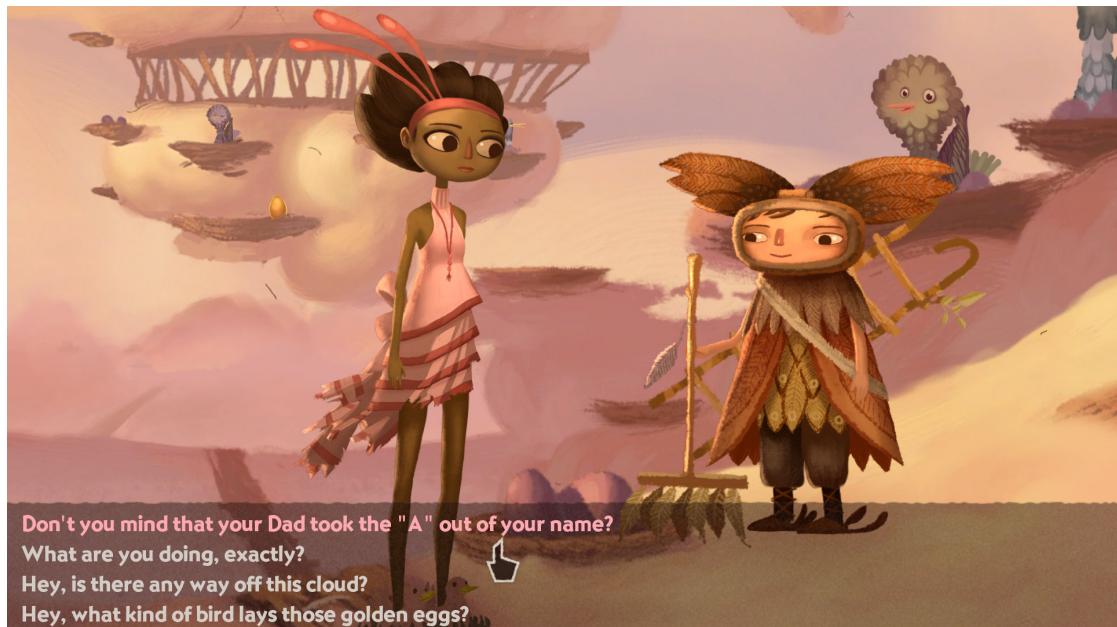


Figura 1.16: Broken Age (Microsoft Windows, OS X, Linux, Ouya, iOS, Android, PlayStation 4 y PlayStation Vita, 2014)

Día a día, más estudios y empresas independientes se animan a lanzar sus aventuras gráficas. Y así gracias a las nuevas tecnologías y los estudios independientes, hemos recuperado este género olvidado.

Aventuras de origen español

¿Y qué pasó durante todos estos largos años en España? Pues desde los albores del género de aventura, cuando aún sólo eran aventuras conversacionales, existió la desarrolladora Dinamic Software [104] que, si bien su repertorio de desarrollo fue amplio (aplicaciones y videojuegos de diversos géneros), se volcaron especialmente en el desarrollo de videojuegos de aventuras. Su debut como desarrolladora fue en 1983 con su primer juego Yength, una aventura conversacional para ZX Spectrum, pero no lograría su primer éxito comercial hasta el lanzamiento de Saimazoom, un juego del género arcade. Por supuesto, este triunfo en un género diferente al de aventuras no impidió que la desarrolladora creara más aventuras.

Cabe destacar el año 1987, en el que se lanzó la aventura conversacional española más exitosa de aquella época: Don Quijote [105] (véase la figura 1.17), la cual basada en la serie de animación de TVE, que a su vez está basada en el libro del mismo nombre, Don Quijote de La Mancha. Se lanzó bajo el sello interno llamado Aventuras Dinamic, que sería un departamento del estudio especializado en aventuras conversacionales.



Figura 1.17: Don Quijote (ZX Spectrum, Amstrad CPC, Commodore 64, MSX, Atari ST y MS-DOS , 1987)

No tardó mucho tiempo este departamento en separarse de Dinamic Software y formar un estudio semi-independiente llamado Aventuras AD [95] y Dinamic Software solamente se dedicaría a hacer de distribuidora. Entre los primeros proyectos de la nueva compañía estaba una versión propia de La Aventura Original [121] y la aventura La Diosa de Cozumel.

No obstante, con el cambio de década, la aparición de la tecnología de 16-bits y diversos problemas que fueron arrastrando algunas de las compañías de videojuegos de la época, las ventas de las compañías españolas de videojuegos fueron decreciendo cada vez más y más. Finalmente acabaron por cerrar la mayoría, dado que los costes de producción y de desarrollo para crear editores y herramientas eran mayores que los beneficios obtenidos por sus productos. O, en el peor de los casos, en ser absorbidas y despedazadas por otras empresas, como es el caso de Eber Soft [108].

En 1994 debutó Pendulo Studios [132] con la primera aventura gráfica española que usase la mecánica *point & click*, Igor Objetivo Uikokahonia [116]. En 1997, lograría su mayor renombre con Hollywood Monsters [112] (véase la figura 1.18). Durante muchos años, fueron el mayor exponente de desarrolladores del género de aventuras gráficas en España con la saga Runaway [135]. Más tarde, ya en 2011, ve la luz Hollywood Monsters 2 [113], realizada enteramente en alta definición. Mientras que todos estos títulos siguen el más puro estilo clásico del género, el 29 de marzo de 2012, Péndulo lanzó New York Crimes [131], una aventura mucho más oscura y adulta de lo que sigue siendo el estilo clásico de las aventuras gráficas, y la fusiona con la estética y composición propias del cómic.



Figura 1.18: Hollywood Monsters (PC, 1997)

No obstante, en estos años Pendulo Studios ha ido decayendo debido a las pocas ventas que han ido generando sus últimas obras y cada vez se le ha ido restando más y más importancia dentro de la industria española, por lo que no ha habido nadie en España que hiciera aventuras gráficas. Así ha sido hasta estos últimos años en los que, gracias a las facilidades actuales para crear estudios independientes, están empezando a surgir nuevos estudios que han comenzado a hacer pesquisas en este género otra vez.

En estos dos últimos años ha salido al mercado Randal's Monday [65] (véase la figura 1.19) de Nexus Games Studios, el cual tuvo un gran éxito en España al ser una aventura humorística con muchísimos cameos de la cultura pop, ciencia ficción, videojuegos, y doblado por remarcables dobladores españoles (aunque lamentablemente no cosecharía el mismo éxito en tierras extranjeras dado que ese tipo de humor no se apreció de la misma forma).



Figura 1.19: Randal's Monday (PC, 2015)

También salió Dead Synchronicity [31] (véase la figura 1.20) de Fictorama Studios en abril de 2014, el cual logró ser financiado con éxito gracias al crowdfunding mediante la web Kickstarter. Tras su salida después de un año de desarrollo, generaron buenas críticas en todos los medios, tanto españoles como extranjeros, gracias a su historia oscura post-apocalíptica con toques de ciencia ficción y el arte característico con el que se envuelve el juego, lo cuales lograron ciertamente llamar la atención de todos.



Figura 1.20: Dead Synchronicity (PC, 2015)

Meses más tarde, Fictorama Studios publicó una aventura conversacional con gráficos pixelados de manera gratuita llamada Dead Synchronicity: The Longest Night [32] (véase la figura 1.21) como precuela de Dead Synchronicity, dejando clara la determinación que hay en el estudio de realizar una segunda parte de este juego, en el que seguramente en estos próximos años se publique.



Figura 1.21: Dead Synchronicity: The Longest Night (PC, 2015)

1.2.2. Videojuegos en el ámbito educacional

Un juego educativo (o “juegos serios” como se llaman actualmente), tal y como su nombre indica, es un juego diseñado con propósitos educacionales o que, de forma incidental o secundaria, tiene valor educativo. Cualquier juego ya sea en forma de juego de mesa, de cartas, o ser un videojuego puede ser usado, con el enfoque adecuado, en un ambiente educacional. Un juego educativo es un juego diseñado para enseñar a los humanos sobre una materia específica o destreza. De hecho, cuando los educadores, gobiernos y padres se dieron cuenta de la necesidad psicológica y los beneficios de aprender jugando, esta herramienta educacional se extendió masivamente. Los juegos son una herramienta interactiva que, al utilizarlos, nos enseñan objetivos, reglas, resolución de problemas, interacción, todo representado como una historia.

El juego siempre ha sido una herramienta de aprendizaje para enseñar conceptos o habilidades nuevas. Antiguamente se usaban las parábolas y las fábulas para promover el cambio social, incluso en la Edad Media se enseñaba a jugar al ajedrez con el fin de enseñar a usar tácticas en las guerras (véase la figura 1.22). Pero lamentablemente, no existe mucha información concerniente a ello. No fue hasta el siglo XIX que el hombre empezó a tomarse en serio el uso de los juegos como una manera de enseñar con la creación de los jardines de infancia por Friedrich Fröbel, que basaba el aprendizaje mediante el juego. Los niños jugaban encantados con sus simples juegos educativos: bloques, kits de coser y materiales para tejer.

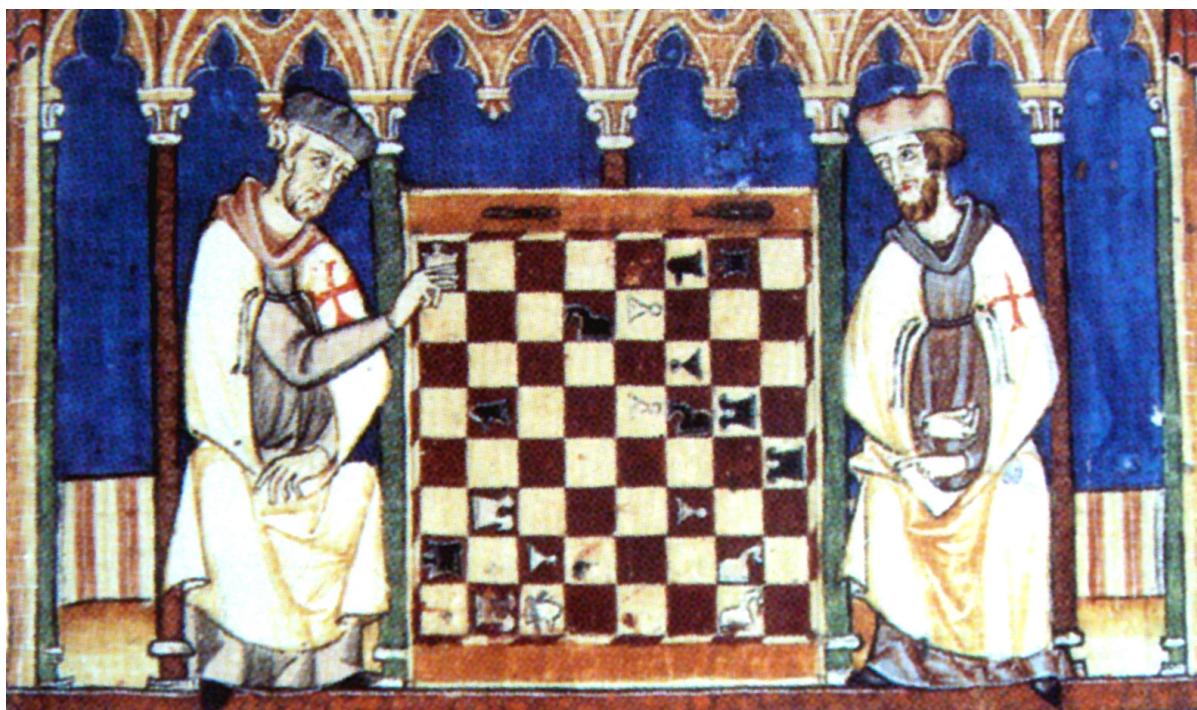


Figura 1.22: Caballeros Templarios jugando al ajedrez, según el Libro de los Juegos (1283)

Los juegos educativos fueron expandiéndose en diversas materias y creando juegos de mesas, cartas, etc. Pero no fue hasta la aparición de la tecnología en las casas, en la década de los 70, cuando Clark Abt propuso en su libro “Serious Games”^[1] una definición concreta de este tipo de juegos:

“Reducido a su esencia formal, un juego es una actividad entre dos o más personas con capacidad para tomar decisiones que buscan alcanzar unos objetivos dentro de un contexto limitado. Una definición más convencional es aquella en la que un juego es un contexto con reglas entre adversarios que intentan conseguir objetivos. Nos interesan los juegos serios porque tienen un propósito educativo explícito y cuidadosamente planeado, y porque no están pensados para ser jugados únicamente por diversión.”

Aparte de esta definición, también se incluyó términos tales como “juego educativo”, “simuladores”, “edutainment” (entretenimiento educativo), etc. que años más tarde, se pondrían en práctica no solo en los juegos de mesa y cartas, sino también en la incipiente industria de los videojuegos.

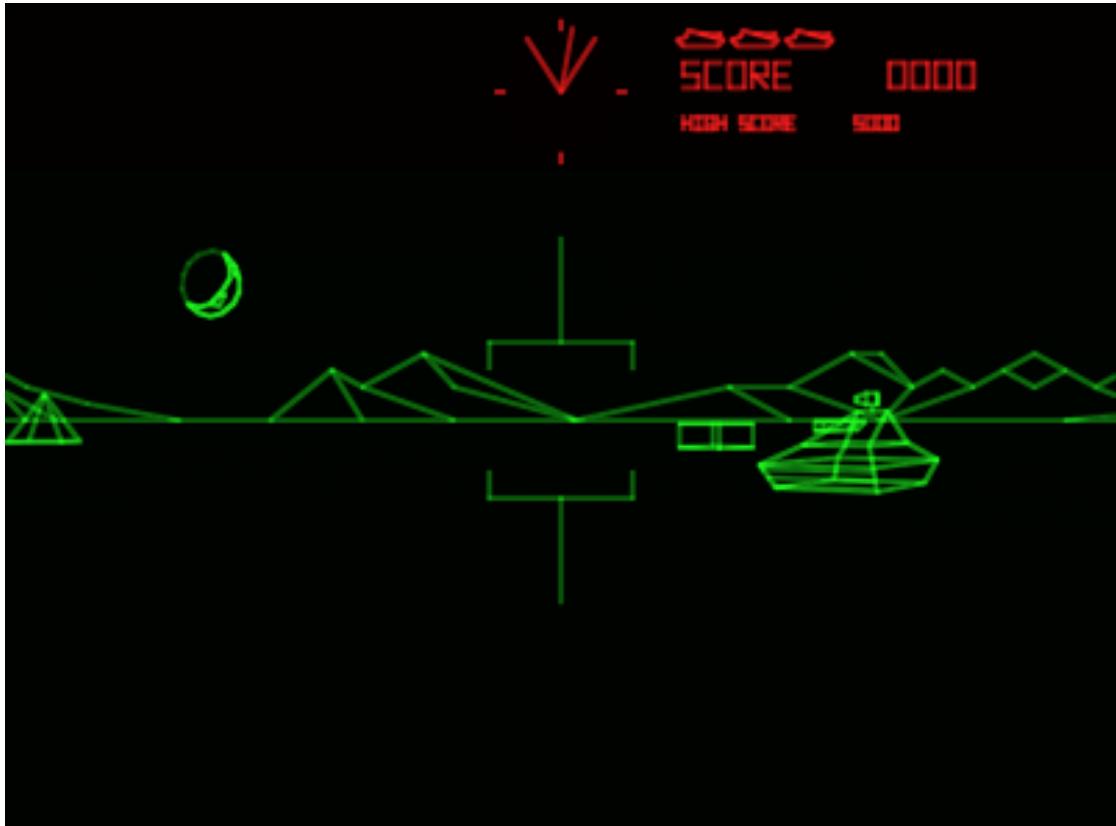


Figura 1.23: Army Battlezone tenía unos gráficos superiores a su época al ser en un principio de uso militar

Army Battlezone [96] se considera el primer videojuego dentro de la categoría de juegos serios, un proyecto fallido liderado por Atari en 1980, el cual fue diseñado para usar el videojuego arcade Battlezone como entrenamiento militar. Algunos juegos triunfaron también en el sector de los “edutainment” como fue el caso de ¿Dónde está Carmen Sandiego en el mundo? [101] (Apple II, 1985) (veáse la figura 1.24) que acabó convirtiéndose en los 90 en una franquicia de juegos, series de televisión y libros; o más tarde, la saga EcoQuest [42] de Sierra Online o el de la aventura gráfica de La Pantera Rosa en Misión Peligrosa [46] (PC, 1996).

Game Options Acme Dossiers



Figura 1.24: ¿Dónde está Carmen Sandiego en el mundo? (Apple II, 1985)

Pero con el cambio de las tecnologías y las formas de distribuir un juego, la cosa cambió. Desarrollar un juego “edutainment” era altamente costoso y largo de desarrollar, y la única manera de que saliera rentable era vendiendo los juegos a precios elevados (alrededor de 60\$) que en cartuchos de consolas o disquetes era un precio aceptable porque los costes de producción eran elevados en todos los tipos de videojuegos. Con la llegada del CD, esto cambió, los costes de producción bajaron enormemente y se podía distribuir mayores cantidades de un videojuego. Los precios de venta de la mayoría de los videojuegos bajaron gracias a esta tendencia (alrededor de 20\$), algo que la industria de los videojuegos educativos no podían permitirse. Tal y como se relata detalladamente en un reciente artículo en la página de Gamasutra de un antiguo trabajador en la industria de los “edutainment” [17].

Así que a pesar de los esfuerzos de muchas compañías, entre ellas Disney o Nintendo, la mayoría de estos juegos fueron un fracaso tras otro. Ejemplo de ello fue por ejemplo la saga Mario is Missing [127] (1992) (véase la figura 1.25) en un intento de repetir el éxito de la franquicia de Carmen Sandiego. Los juegos de entretenimiento educativo, definitivamente, no eran rentables.



Figura 1.25: La saga Mario is Missing (MS-DOS, 1992) fue uno de los intentos fallidos por parte de Nintendo de realizar juegos educativos

Así que según fueron creciendo las capacidades técnicas de los juegos para proporcionar escenarios realistas, el concepto de juegos serios tuvo que ser reexaminado a finales de la década de los 90 con el fin de reorientar el camino de los juegos educativos. Durante este tiempo, algunos estudiosos comenzaron a examinar la utilidad de los juegos para otros propósitos, contribuyendo al creciente interés por emplearlos con nuevos fines. Además, la capacidad de los juegos para contribuir a la formación se vio ampliada con el desarrollo de los juegos multijugador.

En 2002, el Centro Internacional para Académicos Woodrow Wilson creó la Serious Games Initiative [18] con el fin de fomentar el desarrollo de juegos sobre temas políticos y de gestión. Otros grupos más especializados aparecieron después en 2004, como por ejemplo Games for Change [4], centrado en temas sociales y en cambio social, y Games for Health, sobre aplicaciones relacionados con la asistencia sanitaria. Pero no se llegó a actualizar el término de juego serio.

Hasta 2005, no se abordó este término de una forma actualizada y lógica. Mike Zyda escribió artículo publicado en la revista “Computer” de la IEEE Computer Society que llevaba por título “From Visual Simulation to Virtual Reality to Games”[155]. Zyda define primero el término de qué es un juego y luego continúa a partir de aquí:

- **Juego:** una prueba física o mental, llevada a cabo de acuerdo con unas reglas específicas, cuyo objetivo es divertir o recompensar al participante.
- **Videojuego:** una prueba mental, llevada a cabo frente a una computadora de acuerdo con ciertas reglas, cuyo fin es la diversión o esparcimiento, o ganar una apuesta.
- **Juego serio:** una prueba mental, de acuerdo con unas reglas específicas, que usa la diversión como modo de formación gubernamental o corporativo, con objetivos en el ámbito de la educación, sanidad, política pública y comunicación estratégica.

Fue ampliamente aceptada por el público dicha definición, aunque no es la única para el término de "juego serio", pero se entiende que hace referencia a juegos usados en ámbitos como la formación, la publicidad, la simulación o la educación. Definiciones alternativas incluyen conceptos propios de los juegos y las tecnologías, así como nociones provenientes de aplicaciones no relacionadas con el entretenimiento. Los juegos serios empiezan a incluir también hardware específico para videojuegos. Ejemplos de ello son de los videojuegos para mejorar la salud y la forma física que creó Nintendo: Wii Fit [66] para mejorar la salud a base de ejercicios físicos diarios y Brain Training del Dr. Kawashima [144] que es un juego que estimulaba la mente para que pensáramos más rápido y fue un éxito de ventas para todas las edades (véase la figura 1.26).

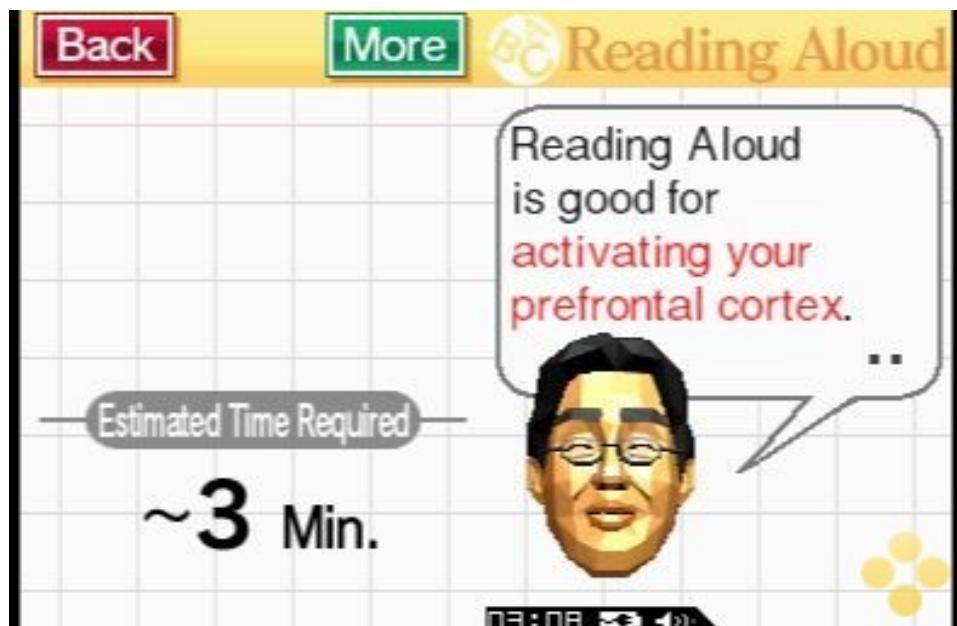


Figura 1.26: Brain Training del Dr. Kawashima (Nintendo DS, 2005), un juego que estimulaba la mente para que pensáramos más rápido

Los videojuegos son una herramienta a tener en cuenta en la estimulación cognitivo afectiva, que favorecen el aprendizaje, la autoestima, potencian la creatividad y las habilidades digitales, al mismo tiempo que generan motivación y entretenimiento. Los videojuegos suponen una modalidad de enseñanza que debe ser aprovechada por la comunidad educativa, por la cantidad de elementos emocionales que integran, su estimulación sensorial y la posibilidad de inmersión a través de los ambientes virtuales en los que se desenvuelven.

Los juegos serios están dirigidos a una gran variedad de público, desde estudiantes de educación primaria y secundaria a profesionales y consumidores. Los juegos serios pueden ser de cualquier género, usar cualquier tecnología de juegos y estar desarrollados para cualquier plataforma. Algunos lo consideran un tipo de entretenimiento educativo, aunque el grueso de la comunidad se resiste a este término.

Un juego serio puede ser una simulación con la apariencia de un juego, pero está relacionado con acontecimientos o procesos que nada tienen que ver con los juegos, como pueden ser las operaciones militares o empresariales. Los juegos están hechos para proporcionar un contexto de entretenimiento y autofortalecimiento con el que motivar, educar y entrenar a los jugadores, como por ejemplo el entorno virtual en 3D VirtUAM [91]. Este entorno permite a los alumnos entrenar en sus habilidades lingüísticas en

alemán dentro de los distintos “mundos virtuales” que tiene programados, permitiendo almacenar y analizar el comportamiento y resultado de los alumnos [33]. Otro ejemplo es el videojuego de Omnitum Labs: U-Startup [68] que ayuda a aprender las herramientas básicas necesarias para poder emprender un nuevo negocio.

Otros objetivos de estos juegos son el marketing y la publicidad. Los grandes usuarios de los juegos serios parecen ser el gobierno de los Estados Unidos y los médicos. Otros sectores comerciales están también persiguiendo activamente el desarrollo de este tipo de herramientas.



Figura 1.27: Simulador de vuelo profesional para enseñar a los nuevos pilotos

Actualmente existe una clasificación, más o menos aceptada pues no existe consenso oficial, de los juegos serios de acuerdo a su propósito:

- **Advergames:** del inglés *advertising* y *game*, es decir, publicidad y juego, es la práctica de usar videojuegos para publicitar una marca, producto, organización o idea.
- **Edutainment:** este es un término que resulta de la unión de *education* y *entertainment*, es decir, educación y entretenimiento o diversión. Se aplica a los programas que enseñan mediante el uso de recursos lúdicos.
- **Aprendizaje basado en juegos:** del inglés *educational game*, estos juegos tienen como objetivo mejorar el aprendizaje. Están diseñados en general manteniendo un equilibrio entre, por un lado, la materia y, por otro, la jugabilidad y la capacidad del jugador para retener y aplicar dicha materia en el mundo real. Este último tipo de juegos se utilizan en el mundo empresarial para mejorar las capacidades de los empleados en temas, atención al público y negociaciones.
- **Edumarket Games:** cuando un juego serio combina varios aspectos (por ejemplo, los propios del advergaming y del edutainment u otros relacionados con la prensa y la persuasión), se dice que la aplicación es un juego de tipo edumarket, término que resulta de la unión de *education* y *marketing*.
- **News Games:** son juegos periodísticos (del inglés *news*, es decir, noticias) que informan sobre eventos recientes o expresan un comentario editorial.

- **Simuladores:** son juegos que se emplean para adquirir o ejercitarse distintas habilidades o para enseñar comportamientos eficaces en el contexto de situaciones o condiciones simuladas. En la práctica, son muy usados los simuladores de conducción de vehículos (coches, trenes, aviones, etc.), los simuladores de gestión de compañías y los simuladores sobre negocios en general, que ayudan a desarrollar el pensamiento estratégico y enseñan a los usuarios los principios de la micro y macroeconomía y de la administración de empresas.
- **Juegos persuasivos:** del inglés *persuasive games*, son juegos que se usan como tecnología de la persuasión para convencer a sus jugadores de que un concepto o idea está bien o mal.
- **Juegos organizativos dinámicos:** del inglés *organizational-dynamic games*, son juegos que enseñan y reflejan la dinámica de las organizaciones a tres niveles: individual, de grupo y cultural.
- **Juegos para la salud:** del inglés *games for health*, son juegos diseñados como terapia psicológica, o juegos para el entrenamiento cognitivo o la rehabilitación física.
- **Juegos artísticos:** del inglés *art games*, son juegos usados para expresar ideas artísticas, o arte creado, utilizando como medio los videojuegos.
- **Militainment:** es un término de la unión de *military* y *entertainment*, es decir, militar y entretenimiento o diversión. Son juegos financiados por el ejército o que, de lo contrario, reproducen operaciones militares con un alto grado de exactitud. Lamentablemente, en su mayoría son privados para el público general dado el secretismo militar sobre para qué lo usan y cómo lo usan.

Cómo conclusión, podemos decir que si actualmente los juegos serios se han separado bastante de los videojuegos como medio de entretenimiento, aún siguen perdurando juegos de la gama de edutainment que intentan aunar estos dos campos. Pero lo que si podemos decir con certeza es que los primeros videojuegos educativos eran en su mayoría aventuras gráficas tal y como se ha visto en los ejemplos dados. No es difícil de saber el por qué, las aventuras gráficas, tal y como se describen en la sección anterior, son juegos dedicados a resolver problemas de lógica. Al recurrir más al esfuerzo mental que otros juegos que buscan los reflejos motores en los jugadores, es fácil realizar conexiones lógicas con las que enseñar datos concretos de forma que acabemos asociándolos a la realidad y aprendamos.

1.3. Motivaciones

Con la crisis en España, la industria de los videojuegos ha sido una de las pocas que, no solo no se ha visto afectada, sino que ha ido a la alza, aumentando el número de desarrolladores independientes durante estos últimos años tal y como afirma la Asociación Española de Desarrolladores de Videojuegos (DEV) en su último libro blanco [5]. Siendo seguramente, una industria al alza para los próximos años y en la que el número de oportunidades de trabajo irán aumentando.

Cabe añadir que parte de esa democratización y aumento del desarrollo de los videojuegos es gracias a motores de videojuegos potentes y gratuitos como *Unity*, que es actualmente uno de los motores de videojuego con editor integrado con más éxito en el mercado. Gran parte de su éxito reside en varios motivos, los más destacables son:

- Rapidez de prototipado de videojuegos, permitiendo desarrollar pruebas de conceptos de nuevos videojuegos a bajo coste sin riesgos de grandes pérdidas.
- Versión gratuita del motor que permite exportar a varias plataformas con el mismo código: Linux, Mac OS X, Microsoft Windows, HTML5, Android, iOS, etc.
- Una gran comunidad de desarrolladores detrás de ella, tanto en foros y páginas oficiales como en no oficiales.
- Mercado de Assets enfocado sobretodo a desarrolladores independientes, para ahorrarles tiempo y esfuerzo a la hora de producir un videojuego.
- Una gran cantidad de documentación referente a las librerías de *Unity* y de tutoriales oficiales tanto para desarrolladores principiantes como veteranos.
- Por último, las oportunidades de trabajo que se están abriendo a los desarrolladores que saben *Unity*, no sólo ya para desarrollar videojuegos sino aplicaciones.

Además de eso, en *Unity* se pueden programar en los lenguajes Javascript y C#. Pero acabé eligiendo C# por encima de Javascript por ser ampliamente usado en el mundo de desarrollo de aplicaciones y videojuegos junto con las librerías .NET, y por su parecido con el lenguaje Java, que era enseñado en la asignatura *Programación Concurrente y Distribuida* en la carrera de *Ingeniería Técnica de Informática de Sistemas*.

Comenzar a usar una tecnología desconocida como *Unity*, aprender un nuevo lenguaje como C# junto con la API de *Unity* para este lenguaje, o cualquier otra, siempre entraña dificultad. Además, si le añadimos la documentación histórica para contrastar que toda la información que se presente tanto en esta memoria como en el resultado final de este desarrollo para que sea lo más verídica, lo dificulta aún más. La principal motivación para embarcarse en la confección de este proyecto, es la de la capacidad que existe en un videojuego tanto para entretener, transmitir sentimientos, ideas y emociones, como para ser una herramienta de apoyo a la enseñanza en prácticamente cualquier ámbito que se proponga la industria.

Diseño de Videojuegos era una asignatura optativa de tercer curso de la extinta titulación Ingeniería Técnica de Informática de Sistemas dentro de la Universidad de Cádiz. En dicha asignatura los alumnos se organizaban en grupos de tres alumnos con el objetivo de desarrollar un juego sencillo durante el cuatrimestre. Las únicas restricciones eran que el videojuego resultante debe ser completamente libre y que

se había de utilizar la forja de código de RedIRIS [70] junto a Subversion [19] como sistema de control de versiones. Los alumnos escogían bibliotecas, usualmente libres, para el desarrollo de aplicaciones multimedia en dos dimensiones como libSDL [54] o Gosu [153]. Toda la documentación estaba en inglés, pero por suerte ya se publicó una excelente documentación en castellano en formato wiki llamada Wikijuegos [3] para libSDL.

Dicha asignatura era casi totalmente enfocada a aprender a programar y el diseño era algo más secundario, limitándose a unas pocas páginas que definieron a grandes rasgos los elementos que iban a desarrollarse en el juego.

Lamentablemente mucha, prácticamente toda la documentación sobre el diseño de juegos con una complejidad alta o mediana, es escasa y en inglés en su práctica totalidad. Poniendo por ejemplo el caso de documentos de diseño de aventuras gráficas, las cuales son orientadas más al contenido narrativo y varían el diseño general por uno más específico para puzzles, solo se pueden encontrar liberados documentos de puzzles y/o diseño en inglés, siendo las más destacadas la de Grim Fandango [76], las aventuras clásicas de AI Lowe [2] y alguna que otra reciente como la de Resonance en los foros de AdventureGameStudio [94]. Con todo esto, una de mis principales motivaciones es la de crear un documento de diseño detallado en español, y aparte otro documento con el guión y el desarrollo seguido para la elaboración del puzzle realizado para este proyecto, todo esto en español y lo más completa posible, para que sirva de ejemplo para desarrolladores de videojuegos de habla hispana.

Por otra parte, dejando de lado el diseño, lo mismo se puede decir de ejemplos de código en español de aventuras gráficas. *Unity* es una herramienta ideal para ello, pues es la más usada actualmente para iniciarse en el mundo del desarrollo de videojuegos, además de ser ampliamente usada por estudios por su versión gratuita y la variedad de plataformas en las que se puede compilar con un único código. Así pues, una motivación es la de generar un juego completo con el código documentado dividido en bloques reutilizables.

La ubicación de la Universidad de Cádiz y el pasado histórico de la ciudad fue otra motivación, pues cuando empezaba con las dudas de sobre qué proyecto iba a realizar, se celebró el Bicentenario de la Constitución de 1812. Proporcionándome la idea de dar a conocer cómo es Cádiz y su historia del 1812, tanto para residentes de Cádiz, como españoles y extranjeros que nunca hayan visitado Cádiz.

Es posible resumir las motivaciones que me han llevado a desarrollar este proyecto en cubrir el vacío de documentación y la falta de proyectos completos en castellano, y el deseo personal de aprender a diseñar videojuegos.

1.4. Objetivos

El objetivo del proyecto viene extraído de forma lógica de las motivaciones (sección 1.3). Con **1812: La aventura** se pretende crear un juego ampliamente documentado y accesible que sirva de referencia para futuros desarrolladores de videojuegos españoles. Deberá cumplir con los siguientes requisitos:

- Servir de ejemplo lo más cercano a la realidad posible, de lo que es un diseño medianamente complejo de un videojuego. Es muy habitual terminar de leer documentación, realizar diseños sencillos para pequeños juegos pero que a la hora de embarcarse a un proyecto con más complejidad

dad es necesario organizarlo y esquematizarlo bien o acabaremos con un videojuego que no tiene ni pies ni cabeza. Para eso es imprescindible que se documente cada fase del desarrollo.

- Crear un sistema intermediario entre el videojuego y el texto de los diálogos, de forma que, al ser independiente el texto, se pueda modificar sin interferir con el código, e incluso poder crear traducciones sin afectar en nada al mismo. Este sistema debe ser reutilizable y estar bien documentado.
- Crear una Base de Datos con información histórica y un buscador dentro del juego para acceder a ella. El motivo de esto es el poder acceder a los datos de manera rápida y sencilla para poder seguir avanzando en el juego sin problemas.
- Demostrar que para desarrollar un videojuego, hacen falta conocimientos de diversas disciplinas, por lo tanto es necesario llegar a trabajar con un equipo multidisciplinar. En un videojuego profesional deben trabajar juntos desde unas pocas personas hasta varios centenares por lo que la coordinación entre profesionales de distintas áreas es muy importante. Será necesario encontrar artistas relacionados con la animación e ilustración, música y efectos de sonido.
- No sólo debe ser útil a los interesados en el desarrollo sino también de cara al usuario final. **1812: La aventura** debe de ser entretenido y potencialmente útil para ayudar a asimilar datos, en este caso históricos.
- Crear *scripts* y objetos en *Unity* que puedan ser reutilizables, e incluso que se pueda expandir y/o personalizar sus funcionalidades al heredar de los *scripts* y objetos mencionados. De esta manera, podemos asegurarnos que puedan ser utilizados no únicamente para este proyecto, sino para otros desarrolladores que deseen también realizar una aventura gráfica 2D en *Unity* ya sea con los mismos u otros fines que **1812: La aventura**.

1.4.1. Sobre este documento

La presente memoria de **Proyecto fin de Carrera** posee la siguiente estructura en capítulos y apéndices:

- En el capítulo 1 que actualmente nos ocupa, se hace una breve introducción sobre el contexto en el que nos situamos. Se ofrece una visión del mundo de la industria de videojuegos, principalmente independiente, se habla sobre las aventuras gráficas, que es el género en el que se va a basar **1812: La aventura** y la importancia de dicho género en el ámbito educativo. Finalmente se evidencia la necesidad de documentación en cuanto a diseño de videojuegos en castellano y se adjuntan el resto de motivaciones y objetivos relacionados con el proyecto.
- En el capítulo 2 se expone la organización temporal de todo el desarrollo a través de un diagrama de Gantt que pretendía seguirse en un principio, y luego un diagrama de Gantt que represente la realidad de cómo se ha ido desarrollando al final el proyecto. Conjuntamente se incluirá unos comentarios del por qué no pudo ser posible seguir la planificación inicial, y cómo se solventó la manera de organizar el desarrollo después. Más adelante se comenta brevemente cada una de las tareas que conforman la planificación.
- En el capítulo 3 se refleja el proceso de desarrollo para **1812: La aventura**. Comenzando con el documento de diseño, el documento con el guión y el puzzle, se sigue con la fase de análisis, diseño e implementación. Se hace especial hincapié en el sistema intermediario del texto con el juego, y la base de datos como buscador de información sencillo en el proyecto.

- En el capítulo 4 se hace una breve reflexión personal seguida de otra técnica tratando de aglutinar los conceptos aprendidos, la riqueza que ha proporcionado la experiencia y lo que podrá aportar tanto a usuarios como desarrolladores. Finalmente termina con un pequeño listado sobre las posibilidades de ampliación del proyecto.
- En el apéndice Software utilizado se hace un repaso por todas las herramientas empleadas a lo largo del desarrollo del proyecto. Asimismo, se adjuntan comentarios y razones de uso.
- En el apéndice Manual de usuario se incluye un completo manual de usuario para **1812: La aventura**. El documento contiene información sobre la introducción de la historia, personajes y lugares así como una completa guía de instalación en sistemas GNU/Linux y Microsoft Windows. Posteriormente se detallan las mecánicas de juego y los controles en detalle. Finalmente se añade una completa guía para añadir traducciones adicionales a **1812: La aventura** usando el lenguaje XML.

Capítulo 2

Organización temporal

En este capítulo se expone la planificación de tareas que se realizó inicialmente para lo que sería **1812: La aventura** como un videojuego completo. En su sección se adjunta el diagrama de Gantt inicial y completo, complementándolo con un breve comentario sobre cada tarea que debía realizarse.

Después, en la siguiente sección se pone en evidencia los fallos en las premisas de la planificación inicial, los problemas que surgieron e impidieron seguir dicha planificación. Y finalmente, la reorientación a un resultado más viable del proyecto de videojuego a demo técnica para probar el motor de aventuras gráficas en 2D programado en *Unity*.

Por último, explicaremos el por qué de la metodología finalmente escogida para organizar el desarrollo del proyecto, en qué consiste y en cómo se adaptó para las necesidades específicas para este tipo de proyectos. Como punto final, se incluye un diagrama de Gantt orientativo del desarrollo de las tareas más representativas e importantes y una breve explicación de cómo se realizaron.

2.1. Planificación inicial

En esta sección detallaremos la planificación inicialmente escogida, cuando este proyecto estaba orientado al desarrollo de un videojuego completo llamado **1812: La aventura**, mostraremos el diagrama de Gantt inicial que se hizo y comentaremos brevemente las tareas que se deberían haber realizado.

2.1.1. Diagrama de Gantt

En el diagrama de Gantt, en el apartado artístico de **1812: La aventura** participan varias personas aparte de mí. Esto se debe a que no poseía los conocimientos o destrezas requeridos para crear los gráficos y sus animaciones, esto incluía escenarios de la ciudad de Cádiz en su momento, todo con estética en *Pixel Art* [148]. Lo mismo ocurría con la parte sonora de **1812: La aventura**, había que crear piezas musicales y efectos de sonido. También alguien con la capacidad de escribir guiones y ayudarme a diseñar los puzzles, para que pudiera dedicar el máximo tiempo posible en el desarrollo de la parte de programación y documentación del proyecto.

Aparte de eso, hacía falta como mínimo, crear la traducción de un juego completo (diálogos, descripciones, interfaces...) de español al inglés, con lo que era una tarea ardua para una persona no habituada a traducir cantidades largas de texto de un idioma a otro de manera profesional, con lo que era necesario depender también de una persona con estudios y/o experiencia traduciendo videojuegos al inglés.

Por último, también hacía falta alguien con los conocimientos necesarios de la historia de Cádiz en la época de 1812, para evitar que apareciesen incongruencias históricas en los datos mostrados dentro de **1812: La aventura**.

Por ello, contacté con gente con experiencia en los temas expuestos y que pudiesen colaborar en el desarrollo de un videojuego completamente libre. Finalmente, los participantes que inicialmente decidieron colaborar fueron los siguientes:

- Celia Fermoselle: artista especializada en *Pixel Art* encargada de diseñar, dibujar, animar a los personajes, objetos y escenarios del juego.
- boredBit: grupo formado por Dmitry Jbanov y Manu Garrido que se encargan de componer y producir la banda sonora completa y efectos de sonido para el juego.
- Daniel Brey: encargado de escribir en su mayoría del guión de diálogos y descripciones del juego.
- Laura J. Torres: encargada en traducir el guión del juego del español al inglés.
- Encarnación M. R. : encargada en asesorar el contenido histórico del proyecto para evitar incongruencias.

El diagrama de Gantt sobre la planificación inicial del proyecto fue la siguiente:

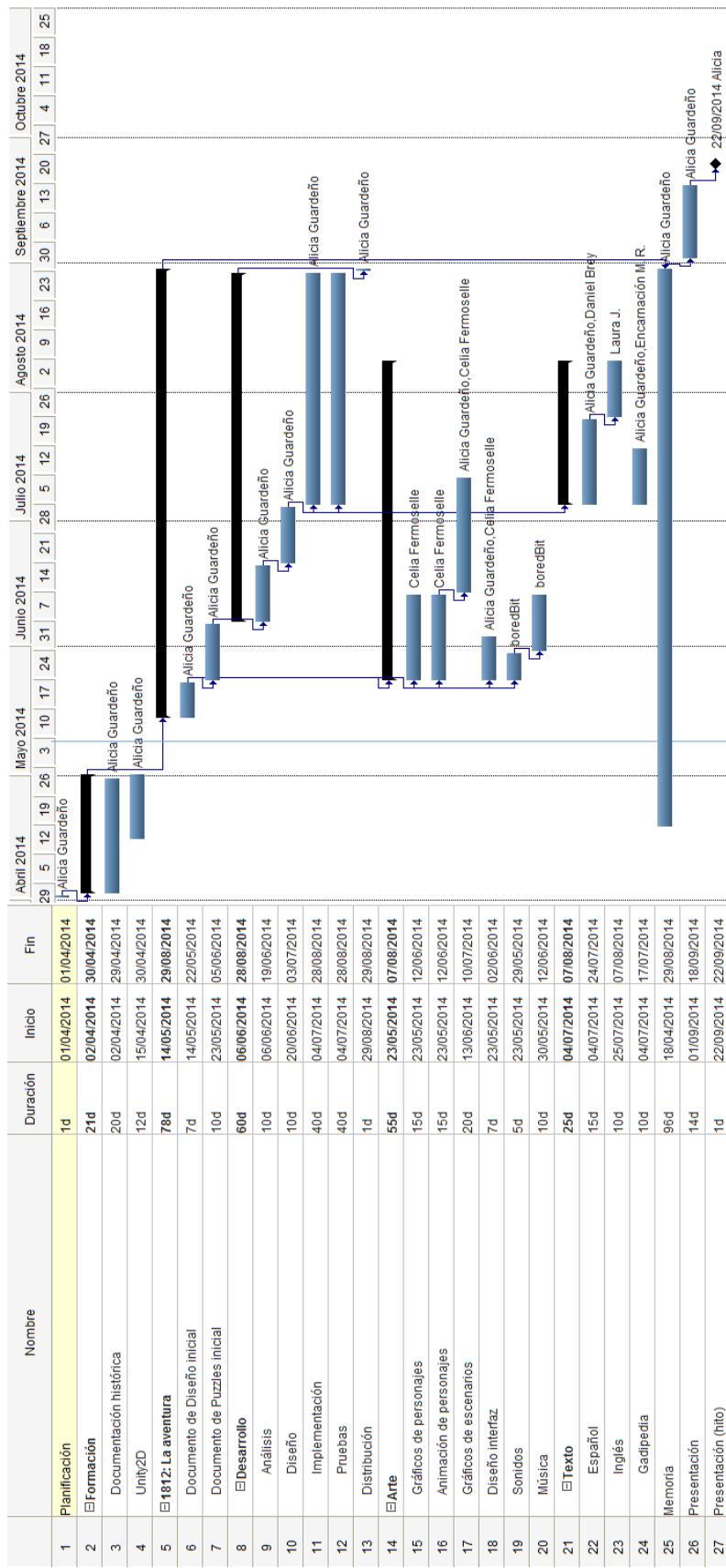


Figura 2.1: Planificación del proyecto desde abril de 2014 hasta septiembre de 2014

2.1.2. Etapas planeadas inicialmente para el desarrollo del proyecto

1. Planificación

Dada la envergadura del proyecto, era necesaria una etapa de planificación en la que se había que estudiar de forma cuidadosa el alcance del mismo y las posibles dificultades a encontrar durante el desarrollo.

2. Formación

Al comienzo del proyecto desconocía por completo cómo usar *Unity*, su API con sus clases y métodos específicos para escribir los *scripts*, y el lenguaje C#. Además de que carecía de conocimientos históricos profundos sobre el Cádiz en la época de 1812. Fue necesaria, por tanto, una larga etapa de formación personal utilizando varios recursos bibliográficos, tanto históricos como para desarrollo de videojuegos en *Unity* y el seguimiento de tutoriales para realizar pruebas básicas dentro del entorno de este programa.

3. 1812: La aventura

Tras el periodo de aprendizaje, se comenzaría con el videojuego **1812: La aventura**. El primer paso era crear una cuenta en la forja de *GitHub*. Dicha forja proporciona un repositorio *Git* y herramientas web que ayudan en la gestión de un proyecto: gestor de tareas, subida de ficheros, publicación de noticias, wiki, foros, etc.

a) Documento de diseño

En el documento de diseño de un videojuego se detallaban elementos como la historia, género, personajes, mecánicas de juego, objetos o escenarios entre otros muchos. En definitiva, era el documento que define de forma más o menos concisa cómo sería el videojuego. Se trataba de un escrito muy importante ya que ayudaba a que todo el equipo albergase la misma idea sobre el videojuego y pudiera trabajar de forma más compenetrada.

b) Documento de puzzles

El documento de puzzles de un videojuego, es un documento extra necesario para el desarrollo de una aventura gráfica, en ese se detallan los puzzles, la jerarquía de resolución de puzzles, escenarios y objetos involucrados en su resolución. Se trataba de un escrito muy importante ya que determinaba el flujo del desarrollo de una aventura gráfica, y la dificultad que podía suponer al jugador seguir avanzando o no.

c) Análisis

La fase de análisis tendría que haber dado comienzo después de la redacción y revisión de los documentos de diseño y puzzles. Se tendría que haber procedido con la toma de requisitos a partir de dicho documento, y a partir de estos confeccionar los casos de uso, el modelo conceptual de datos y detallar el modelo de comportamiento.

d) Diseño

Durante la fase de diseño, que siguió de forma inmediata al análisis, se elaboraron los diagramas de clases.

e) Implementación

La fase de implementación de **1812: La aventura** era, con diferencia, la más extendida de todo el desarrollo. Motivada por la inexperiencia del entorno de trabajo, el nuevo lenguaje de programación a usar y varios sistemas más. Entre los más importantes estaba el diseñar scripts encargados de trabajar con ficheros XML, de manera que se pudieran leer (en el caso de los textos de los diálogos y descripciones o los ficheros de guardado de partidas y ajustes) o grabar (solo para los ficheros de guardado de partida y ajustes), y aprender a diseñar una pequeña base de datos e integrarla en *Unity*.

f) Pruebas

Durante lo que hubiera sido la fase de implementación, se tendría que haber realizado pruebas de los módulos individuales (en este caso los objetos de *Unity* o sus *scripts*) y posteriormente ver si se integraban correctamente con los otros módulos. No sólo se tendría que haber trabajado para que el código fuese correcto, sino que **1812: La aventura** tenía que haber sido probado de forma extensiva por colaboradores distintos al desarrollador principal (siendo yo ese desarrollador) con el claro objetivo de pulir ciertos detalles relacionados como el balanceo de la dificultad de los puzzles (que no fueran ni fáciles ni difíciles), y el correcto control del personaje entre otras cosas.

g) Distribución

Al finalizar la implementación, se tendría que crear y publicar paquetes descargables para los sistemas GNU/Linux y Windows. Por suerte *Unity* es un editor que cuenta con un compilador multiplataforma, con lo que no tomaba mucho tiempo incluso si queríamos incluir para GNU/Linux scripts para instalar y desinstalar el juego automáticamente.

h) Arte

El proceso de creación de arte en cuanto a los primeros gráficos, sonidos y músicas necesarios para comenzar con **1812: La aventura** empezaron nada más acabar la redacción del primer borrador del documento de diseño. Con este borrador, ya se tenía claro el estilo visual del juego, los escenarios, personajes, música y sonidos iniciales y prioritarios para el desarrollo. Exceptuando el guión que dependía de la finalización del documento de puzzles. Por la complejidad que suponía ir manteniendo y actualizando el documento de diseño según iría avanzando el proyecto, el trabajo en este apartado se hubiera tenido que extender prácticamente durante todo el desarrollo. Además, al ser la única parte del desarrollo en la que participaban colaboradores externos, hubiera sido necesario realizar labores de supervisión y coordinación: estilo, formato de entrega, corrección de fallos, etc.

1) Gráficos de personajes

Los gráficos los iba a realizar Celia Fermoselle, mediante la técnica de *pixel art* dado que era como mejor y más rápido trabajaba. La coordinación en este punto fue crítica pues hubo que discutir varias veces el tipo de estética a conseguir, que simulara ser antiguo pero sin llegar a aumentar la dificultad de creación de los gráficos, la resolución y tamaño de los gráficos, etc.

2) Animación de personajes

Una vez finalizados los gráficos de los personajes, Celia hizo la animación por cuadros (más conocidos como *frames*) del personaje principal y la bibliotecaria al principio, y más adelante debería haber hecho la de los demás personajes. La técnica usada para animar que ella usaba, se basa en modificar los gráficos originales en sucesivos cuadros, que al repetirse rápidamente dan ilusión de movimiento.

3) Gráficos de escenarios

Aquí el trabajo fue realizado entre Celia y yo para los dos primeros escenarios, yo le pasé los diseños básicos de los escenarios con los elementos clave según el documento de diseño inicial y ella los ilustraba después mediante el uso del *pixel art*. Después de Luego una vez hubiera ido avanzando el documento de diseño, el trabajo conjunto de las dos debería haber continuado de la misma manera según se fueran perfilando los demás escenarios a incluir.

4) Diseño de la interfaz

A partir del documento de diseño, la interfaz debería haberse realizado por Celia Fermoselle.

5) Efectos de sonido

Una vez detallados los sonidos requeridos en el documento de diseño, estos deberían haber sido realizados por el equipo de músicos boredBit.

6) *BSO*

Una vez acabados los sonidos principales del juego, boredBit se tendría que haber dispuesto a realizar la música del juego. Para ello una mínima coordinación hubiera sido necesaria, dado que al igual que con los gráficos, se habrían tenido que definir el tipo de melodía y sensaciones a evocar por estas según la situación y el momento del juego.

7) *Texto*

Debiendo haber sido **1812: La aventura** una aventura gráfica, la calidad de los diálogos y descripciones era necesaria para que pudiera ser disfrutado por el público y haber ayudado a la resolución de puzzles. Daniel Brey se prestó en un principio a colaborar conmigo para la realización de estos fijándonos en el documento de diseño y de puzzles. Lo mismo que Encarnación M. R. que inicialmente me asesoró en los asuntos iniciales incluidos en el documento de diseño, para que no hubiera incongruencias en los contenidos relacionados a añadir a **1812: La aventura** con el Cádiz de 1812.

a' *Español*

Tal y como se ha dicho en el apartado anterior, Daniel Brey tendría que haber realizado el guión del videojuego en español sirviéndose del documento de diseño y puzzles finales, habiendo contado con mi ayuda y la de Encarnación M. R. (en temas históricos).

b' *Inglés*

Una vez se hubieran acabado de escribir los textos en español, Laura J. Torres tendría que haber procedido a traducirlos al inglés. Con que durante la realización de la traducción, hubiera sido necesario coordinarse con ella para que no hubiese malentendidos al traducir ciertas informaciones que pudieran llegar a ser más ambiguas. Y por último, haberla ayudado en la realización de un documento que dictase los pasos a seguir para modificar los textos o traducirlos a otros idiomas más adelante.

c' *Gadipedia*

Gadipedia es el nombre que le puse a lo que sería el pequeño buscador de información sobre el Cádiz de 1812 que incluiría el juego **1812: La aventura**. Ahora bien, si la parte de la implementación corrió a cuenta mía, la de la generación y supervisión de los textos fue gracias a Encarnación M. R.

4. Memoria

Antes, durante y después del desarrollo de **1812: La aventura** se tendría que haber procedido a la redacción de un documento igual al presente documento que incluiría apéndices adicionales como el manual de usuario del videojuego.

5. Presentación

Como última tarea en la organización temporal debería figurar la elaboración de la exposición de cara a la presentación del **Proyecto fin de Carrera**. Tratando de plasmar el trabajo que se tendría que haber realizado y los objetivos conseguidos con el desarrollo de lo que sería inicialmente este proyecto.

6. Comunidad

Una de las partes que más me hubiera gustado fomentar de la planificación inicial del proyecto era su objetivo de servir a la comunidad. Hasta el momento había (y sigue habiendo) una escasa documentación sobre cómo diseñar un videojuego en español, más concretamente diseño de las aventuras gráficas. Por lo que a lo largo de todo el desarrollo se hubiera intentado interactuar con diversas comunidades mediante foros, redes sociales y el blog del proyecto.

7. Presentación (hito)

Finalmente, para Septiembre de 2014 se hubiera entregado toda la documentación del proyecto y se hubiera preparado la presentación final que tendría lugar unas semanas después.

2.2. Aparición de problemas para seguir la planificación inicial

Si bien la planificación inicial era concienzuda y bastante razonable en los tiempos dedicados a cada apartado del desarrollo de **1812: La aventura**. Esta planificación fue hecha teniendo basada en las siguientes premisas:

- Que no hubiera ningún tipo de problemas de salud o incapacidad durante todo el desarrollo del proyecto.
- Que a razón de la anterior premisa, se dispusieran de más de seis horas al día para trabajar en el proyecto.
- No hubiese problemas organizativos entre los colaboradores y estos no dejases a medias su trabajo.
- No surgiera ningún trabajo profesional, con lo cuál el tiempo a dedicar al proyecto se vería disminuido al tener que compaginarlo con otro proyecto.

Cómo se puede presuponer por las fechas de entrega de este proyecto, comparado con las fechas del diagrama de Gantt original (véase la figura 2.1), son un año después. Las premisas con las que se quiso realizar **1812: La aventura** no se pudieron cumplir.

Los factores por los que no se pudieron cumplir con ellas son variables, y algunos incluso de ámbito personal. No obstante, los hechos más relevantes por los que no se pudieron cumplir estos plazos pueden ser relatados.

En primer lugar, no tuve en cuenta mis limitaciones físicas y dependencia de una tercera persona para la realización de la mayoría de las actividades cotidianas, con lo cual el tiempo que disponía cada día para trabajar era enormemente variable dependiendo de la disponibilidad de la tercera persona y el cansancio y/o dolores físicos del día. Un día podía dedicarle seis u ocho horas como lo previsto, y en otros casi no podía dedicarle ninguna hora de trabajo.

También influyó la mala suerte de que a mediados de 2014 apareció un agravamiento de una enfermedad que padezco, dicha enfermedad afectaba enormemente el tiempo disponible que podía estar trabajando sentada y mermaba mi capacidad de concentración. Hasta el invierno de 2014, los médicos especializados en dicha enfermedad no lograron encontrarme un tratamiento que estabilizara mi condición.

Ante tanto tiempo sin poder trabajar, el contacto con los colaboradores fue mínimo y con algunos de ellos se perdió. En el caso de Celia Fermoselle, solo me mandó los encargos iniciales tras varias conversaciones de cómo íbamos a orientar la estética del proyecto, la resolución de la pantalla, la cantidad de gráficos y animaciones. Pero, al intentar retomar el contacto con ella para continuar con los gráficos que hacían falta, resultó que Celia había cambiado de paradigma de artista y ahora solo se dedicaba al modelado en 3D, con lo que ella desistió en seguir trabajando en **1812: La aventura**.

Con boredBit ocurrió algo parecido sin llegar a ser tan problemático como con Celia, me mandaron los primeros sonidos y canciones para el juego antes de perder el contacto con ellos. Tuvimos contacto un par de veces al poder yo retomar al fin el proyecto y me hicieron las modificaciones pertinentes de los sonidos, pero fue transcurriendo 2015, perdí definitivamente el contacto con ellos sin más. Además de que tanto como sus páginas web y redes sociales fueron borradas, con lo que perdí la comunicación definitivamente con ellos.

Siendo los dos apartados más importantes del juego, aparte de la programación y el diseño, acortados drásticamente, no tuve más remedio que reorientar el proyecto y reducirlo de tamaño. **1812: La aventura** pasó de ser un juego completo a una demo técnica con componentes reutilizables por si en un futuro, una vez presentado este proyecto, se continuara con **1812: La aventura** para convertirlo en lo que originalmente debió ser. O en cualquier caso, para una aventura gráfica en *Unity* totalmente nueva ya sea por mí u otros desarrolladores.

Dada la nueva situación y el acortamiento del proyecto a una demo con un puzzle únicamente, la colaboración con Laura J. Torres para traducir los textos, las discusiones con Daniel Brey sobre los distintos puzzles a implementar, y el asesoramiento histórico de Encarnación M. R. ya no resultaba su colaboración. Aunque les estaré siempre agradecida por querer colaborar con el proyecto a pesar de no haber tenido apenas fruto sus colaboraciones.

Finalmente, me surgió la ocasión de trabajar en un grupo de Cádiz (que actualmente conforman la empresa Mad Gear Games [58]) que estaban desarrollando su propio videojuego: A Hole New World [53], así que aproveché dicha experiencia para aprender como trabajaban y organizaban ellos entorno a un videojuego independiente real. Si bien dicha experiencia fue fructífera y aprendí bastante, aunque no requería mucho tiempo de trabajo por mi parte (entre 5 y 20 horas semanales, dependiendo de la carga de trabajo semanal), tuve que compaginarlo con el desarrollo del proyecto. Lo cual unido a mi gran variabilidad de tiempo disponible a diario, hizo que el desarrollo de la demo de **1812: La aventura** se dilatara aún más en el tiempo.

Así que en resumen, el proyecto tuvo que reorientarse y se retrasó un año más de lo planeado debido a muchos factores que no se tuvieron en cuenta a la hora de la planificación inicial.

2.3. Metodología de desarrollo del proyecto: SCRUM

Con el cambio del planteamiento del proyecto, el tiempo tan variable que disponía cada día para trabajar en él, y el tener que compaginarlo con un trabajo, hacía que seguir una metodología clásica de desarrollo de software como la del Modelo en Cascada [146] fuera totalmente descartable.

2.3.1. Búsqueda y elección del nuevo método

La solución a mi problema de planificación al final vino de observar cómo se organizaban al trabajar en Mad Gear Games. Usaban la metodología ágil SCRUM [149] mediante la herramienta de planificación Trello [79].

La base de toda metodología ágil, es que las tareas se van iterando, de forma que varias etapas del

desarrollo pueden solaparse a la vez o retomar una tarea anterior para volver a iterar sobre ella para añadir cambios que hayan surgido desde la vez anterior. Además, en el caso de SCRUM, se definen bloques de tareas según las prioridades del proyecto en dicho momento de desarrollo que se eligen entre todos los participantes del proyecto y liderados por el *Scrum Master*, luego a base de revisiones diarias, se van eliminando o añadiendo nuevas tareas según prioridades, se vayan haciendo o se necesite reiterar un trabajo en concreto. Después, dichos bloques se tienen que realizar en períodos de tiempo concreto a definir antes de empezar el proyecto (normalmente una o dos semanas, o un mes) llamados *sprints*. Y así sucesivamente hasta que se haya dado por finalizado el desarrollo del proyecto.

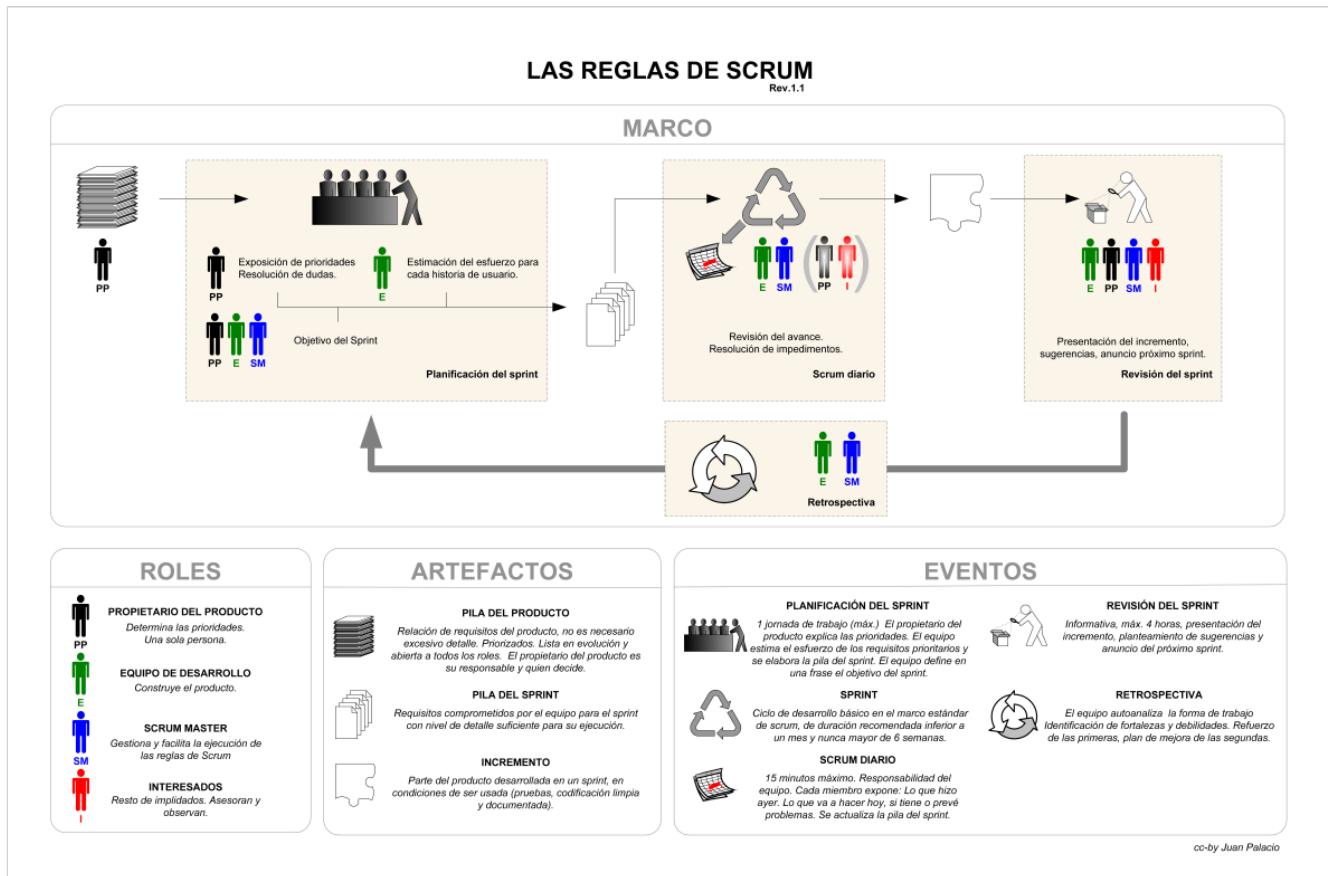


Figura 2.2: Diagrama explicativo del funcionamiento de Scrum

Resultaba ser un método altamente flexible, con el que se podían marcar metas pequeñas en períodos cortos, pero a su vez mantener una visión global de los avances en el proyecto en general. Además de poder seguir marcando fechas específicas como plazos límite o *hito* para no acabar cayendo en la dejadez y vagancia en la realización de las tareas.

2.3.2. Metodología SCRUM personalizada usando Trello

Viendo que la metodología SCRUM era adecuada para mi situación. Aunque no supiera cada día cuento tiempo iba a disponer, al tener metas cortas en periodos de una semana, lograba centrarme en las prioridades del proyecto en cada momento, y en caso de no poder terminar alguna tarea del sprint, se dejaba para el sprint siguiente.

Para poder hacer el seguimiento de las tareas para cada día, empecé a usar *Trello* [79]. Es una aplicación web en la que la base de su funcionamiento trata en crear listas con el nombre que queramos, y estas a su vez se pueden llenar de tarjetas, cada una con una tarea específica. Dichas tarjetas pueden ser asignadas a distintos integrantes del proyecto, ponerles etiquetas según el tipo de tarea que le pongamos, tener fechas límite, sublistas de tarea dentro de una tarea, añadirles imágenes y comentarios... Una aplicación que parte de una base muy sencilla pero que puede ser altamente personalizable y útil a la hora de crear listas de tareas y poder moverlas de un lado a otro.

Adaptar *Trello* a la metodología SCRUM es bastante sencillo. Partiendo de la simplificación máxima de lo que compone un *sprint* en cualquier proyecto, podemos partir de estas listas básicas:

- *DOING* (haciendo): En esta lista pondremos todas las tareas que vayamos a hacer durante el *sprint*.
- *TO DO* (por hacer): aquí iremos colocando las tareas que vayan surgiendo diariamente que se necesiten hacer para poder colocarlas en la lista *DOING* en próximos *sprint*.
- *DONE* (hecho): Lista donde iremos colocando las tareas realizadas, para llevar un seguimiento de todas las tareas que hemos hecho del proyecto y tener una visión de lo que falta aún por hacer.

Con estas tres listas, ya podemos hacer un SCRUM básico para cualquier tipo de proyecto. No obstante, dada la naturaleza del proyecto, el desarrollo de una aplicación multimedia (específicamente un videojuego), creé otras tres listas con propósitos concretos para ajustarme a las necesidades del proyecto:

- *BUGS* (fallos): Es una lista en las que voy reuniendo los fallos al jugar a **1812: La aventura** que van saliendo según avanza el desarrollo o se descubren. Con lo que a veces se van creando nuevas tareas en esta lista, siendo etiquetadas a veces según la prioridad o dificultad que suponen el arreglar dicho bug del juego.
- *BACKLOG* (sugerencias y/o ideas): Esta lista esta creada para ir recopilando las sugerencias o ideas que han ido surgiendo tanto más como de gente que ha ido jugando **1812: La aventura**. Las sugerencias pueden ir desde cambios o mejoras de mecánicas implementadas hasta cambios y/o optimizaciones de algunos scripts del proyecto. Dependiendo de la dificultad y/o la necesidad de implementar alguna de estas mejoras, se van añadiendo a la lista *TO DO*.
- *TRASH* (basura): En esta lista se van colocando las tareas desecharadas, ya sea porque no se pueden implementar o porque ya no es necesario realizarlas. Normalmente cuando se deja una tarea en esta lista, dejo un comentario explicando en cada tarea explicando el por qué no se ha hecho.

Además de las listas, algunas veces me resultaba necesario el poder diferenciar las prioridades de las tareas y el tipo de tarea para cada *sprint*. Para solventar este problema de diferenciar el tipo de tareas y

sus prioridades de realización durante en *sprint*, creé una serie de etiquetas de colores, cada una con su significado:

- **ROJO**: Tareas con máxima prioridad.
- **CIAN**: Tareas con baja prioridad y/o son optativas de realizar.
- **AZUL**: Tareas relacionadas con la programación del proyecto.
- **MORADO**: Tareas relacionadas con el arreglo de bugs aparecidos en el proyecto.
- **VERDE**: Tareas relacionadas con la realización de gráficos y/o audio.
- **NARANJA**: Tareas relacionadas con la elaboración del GDD (documento de diseño de **1812: La aventura**).
- **AMARILLO**: Tareas relacionadas con la elaboración de la memoria del proyecto.
- **ROSA**: Tareas relacionadas con la elaboración de entradas en el blog de seguimiento del desarrollo de **1812: La aventura** .

Esto unido con la posibilidad de crear listas de sub-tareas en las tarjetas de tareas y poder ir marcándolas según se van realizando, nos da un completo seguimiento del tipo de tarea, prioridad y sub-tareas por cada tarea que estemos realizando.

Teniendo todas estas listas creadas, la categorización personalizada de las tareas para el proyecto en *Trello* y las bases de SCRUM, el funcionamiento de mi SCRUM personalizado fue el siguiente:

1. Realización de *sprints* semanales.
2. Cada día iba añadiendo nuevas tareas a la lista *TO DO* o *BUGS* según iban surgiendo nuevas tareas.
3. Cada semana se iban colocando a la lista *DOING* las tareas a realizar para esa semana.
4. Cada vez que se terminaba una tarea, se colocaba en la lista *DONE*
5. Si una tarea no se podía realizar por algún motivo en específico, se colocaba en la lista *TRASH* explicando el por qué.
6. En el caso de quedarse sin tareas en la lista *DOING* antes de finalizar el *sprint*, se podían colocar nuevas tareas en la lista *DOING* que se hubieran ido acumulando a lo largo de la semana las listas de *TO DO*, *BACKLOG* o *BUGS*. Siempre y cuando se considerase que se pudieran completar a tiempo.

A pesar de la simplicidad de estas directrices, los resultados de seguir esta metodología fueron satisfactorios. En todo momento sabía lo que tenía que hacer, cuánto había hecho, y cómo priorizar las tareas para el próximo *sprint*.

Adjunto como ejemplo, una captura del *Trello* usado durante los *sprint* finales del proyecto:



Figura 2.3: *Trello* en uso para hacer el seguimiento del desarrollo del proyecto

2.3.3. Diagrama de Gantt final

Aunque no se llevase al final una metología clásica de desarrollo y con la metodología SCRUM se podía dar marcha atrás en las etapas de desarrollo o volver a reiterar sobre una etapa en concreto, para dar una idea generalizada de cómo fue el desarrollo durante el periodo de todo este año de la demo de **1812: La aventura** y cuánto tiempo se dedicó a cada apartado del desarrollo, se adjunta un diagrama de Gantt orientativo.

Con la reorientación del proyecto debido a los problemas surgidos comentados anteriormente (véase la sección [2.2](#)), al final tuve que prescindir de la mayoría de colaboradores y solo conté con los trabajos artísticos que inicialmente me mandaron para poder desarrollar la base de la demo de **1812: La aventura** y a partir de ahí, yo me hice los recursos artísticos necesarios que pudieran faltarme para terminar el proyecto. Por ello, incluyo en el diagrama de Gantt orientativo a los colaboradores que llegaron a aportar inicialmente algún tipo de trabajo para el proyecto, ellos son los siguientes:

- Celia Fermoselle: que se encargó de diseñar en *Pixel Art* dos escenarios, dos personajes con sus respectivas animaciones y la interfaz del inventario.
- boredBit: grupo formado por Dmitry Jbanov y Manu Garrido que se encargaron de componer y producir tres canciones y un par de sonidos para el juego.

Dicho diagrama, en el cuál se han dejado la mayoría de la duración de los días con el símbolo ‘?’ para que quede constancia de que son aproximaciones, es el siguiente:

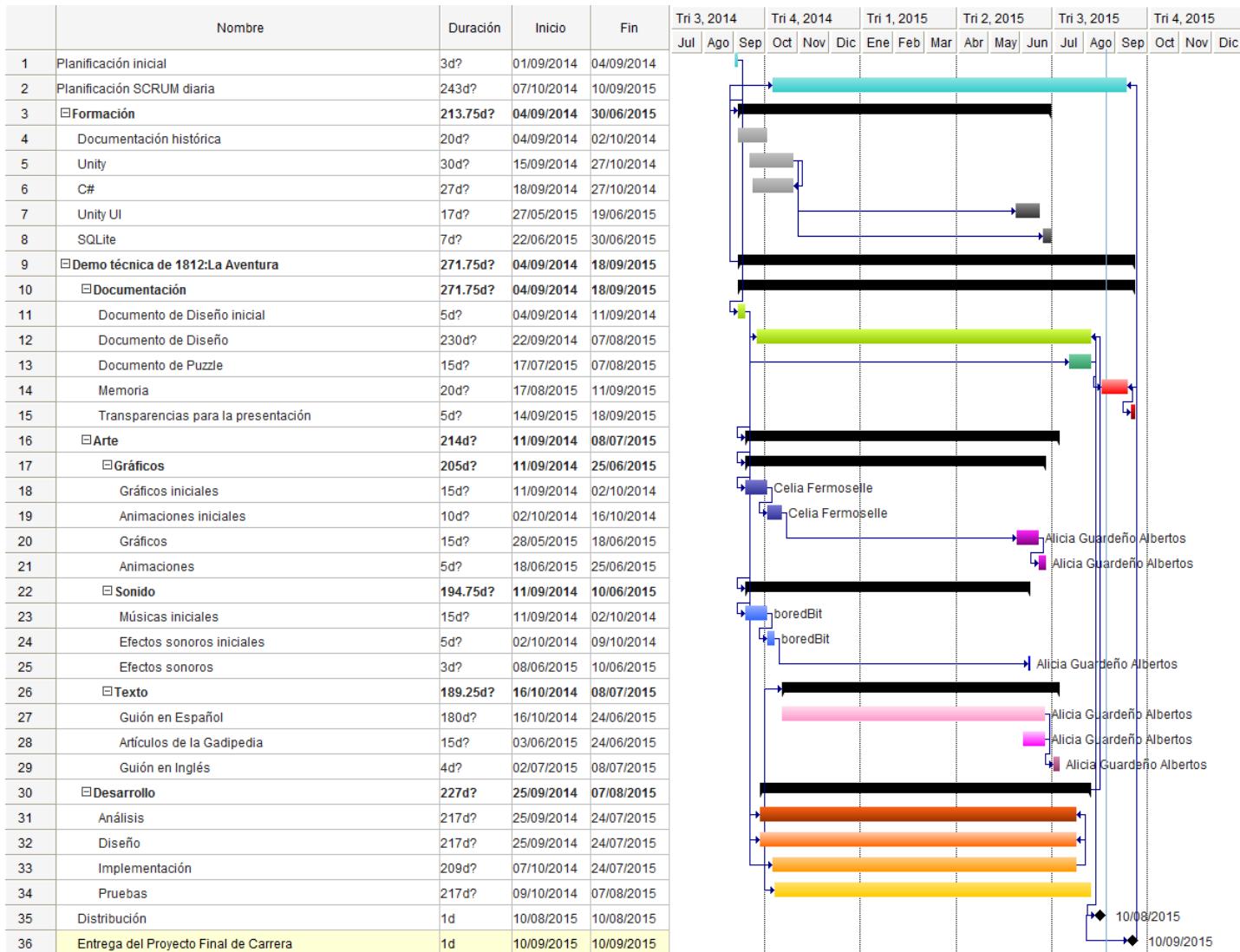


Figura 2.4: Diagrama de Gantt orientativo del desarrollo del proyecto desde octubre de 2014 a septiembre de 2015

2.3.4. Etapas del desarrollo del proyecto seguidas de manera orientativa

1. Planificación inicial:

Aunque finalmente decidiera usar la metodología SCRUM para desarrollar el proyecto, era necesario una primera planificación sobre las distintas partes que iban a componer el proyecto y desglosar a *grosso modo* los grupos de tareas más importantes para cada parte. Por suerte no hubo muchos problemas ni me ocupó mucho tiempo este aspecto ya que me valió gran parte de la planificación que había hecho previamente antes de la reorientación del proyecto.

2. Planificación SCRUM diaria:

Cada lunes de una nueva semana realizaba un nuevo *sprint* con las nuevas tareas a realizar dicha semana, pero cada día tenía que invertir un tiempo en decidir en qué tarea (o tareas) tenía que dedicarme ese día. Y si se daba el caso, ir añadiendo nuevas tarjetas en las listas de **BUGS** o

BACKLOG, o ir añadiendo nuevas tareas o subtareas no previstas en la lista **TO DO** para el siguiente *sprint*.

3. Formación:

Gran parte de la formación, sobretodo en el apartado histórico, la hice antes de la reorientación del proyecto y la nueva planificación. Pero fue necesario seguir invirtiendo tiempo en formación durante el proyecto, dado que algunos aspectos necesitaban ser estudiados más en profundidad para hacer funcionar correctamente el proyecto o porque fueron añadidas nuevas funcionalidades a *Unity* a *posteriori*.

a) Documentación histórica:

Hubo que seguir realizando documentación histórica para poder diseñar correctamente el puzzle del mapa incluido en la demo técnica de **1812: La aventura**. En este caso concreto, tuve que indagar información sobre la cartografía de la época y de edificaciones históricas concretas que tuvieran relevancia.

b) Unity:

Aún poseyendo previamente conocimientos de *Unity* antes de empezar el proyecto, fue necesario a lo largo del proyecto ir aprendiendo más profundamente sobre aspectos más difíciles de la API. Donde se requiso documentarse más fue en cuanto al uso de las *Coroutines*, para lograr la sincronización de las acciones del *Jugador* correctamente.

c) C# y .NET:

Lo mismo que pasó con *Unity*, tuve que aprender algunas características más específicas de .NET como el uso de la librería *Linq*.

d) Unity UI:

Durante el desarrollo de la demo técnica de **1812: La aventura**, *Unity* sacó nuevas versiones de su software, algunas con cambios bastante significativos. El más importante fue el cambio del sistema para implementar interfaces gráficas en *Unity*, el nuevo sistema se llamaba *Unity UI*. Como el proyecto ya había empezado a programar algunas partes de la interfaz gráfica con el antiguo sistema, tuve que sopesar si cambiar al nuevo sistema o no. Al final decidí cambiar a *Unity UI* por sus facilidades a la hora de mantener la composición de la interfaz al cambiar de resolución, por lo tanto tuve que desechar lo hecho y documentarme sobre cómo usar este nuevo sistema.

e) SQLite:

En la etapa final de la implementación del proyecto, uno de los objetivos era incluir una base de datos funcional que funcionase como buscador rudimentario dentro de *Unity*. Así que fue necesario aprender como integrar *Unity* con *SQLite*, y posteriormente como programar la conexión y el uso de la base de datos con C# y .NET.

4. Demo técnica de 1812: La aventura :

La mayor parte del aprendizaje se realizó antes de reanudar con el proyecto reorientado, y se reutilizó el repositorio creado en la forja de *GitHub*.

a) Documentación:

1) Documento de Diseño inicial:

Este documento se mantuvo casi idéntico al documento de diseño inicial realizado antes de reorientar el proyecto, solo tuve que eliminar la información referente a los escenarios, personajes y *Assets* que no iban a ser incluidos en la demo técnica de **1812: La aventura**.

2) *Documento de Diseño:*

A partir del Documento de Diseño inicial, se fueron modificando algunos detalles extra a lo largo del desarrollo por sugerencia de las personas que fueron probando las *builds* de la demo técnica de **1812: La aventura**, entre ellas los controles para el juego y algunos *Assets* extra que no se tuvieron en cuenta al principio.

3) *Documento de Puzzle:*

Con la base del Documento de Diseño, realicé en un documento aparte en donde explicaba todo el proceso realizado para diseñar el puzzle incluido en la demo técnica de **1812: La aventura**: desde la documentación histórica hasta las pistas finales, pasando por las explicaciones del por qué de las decisiones tomadas en cuanto al diseño. Además de eso, se incluye un diagrama donde se detallan el flujo de acciones posibles que puede realizar el *Jugador* dentro de la demo, destacando el camino crítico de las acciones mínimas y necesarias para completarla.

4) *Memoria:*

Al terminar con el proceso de implementación de la demo técnica de **1812: La aventura**, se realizó esta memoria que detalla todo el proceso de desarrollo seguido de este proyecto.

5) *Transparencias para la presentación:*

Después de entregar el proyecto junto con la memoria, se realizarán las transparencias

b) *Arte:*

La mayoría de los gráficos, sonidos y música necesarios para la demo técnica de **1812: La aventura** se realizaron antes de la reorientación del proyecto gracias a los colaboradores. Sin embargo, fue necesario después realizar nuevos gráficos y sonidos que o bien faltaban o no fueron previstos en el Documento de Diseño inicial.

1) *Gráficos:*

a' Gráficos iniciales:

Estos fueron los gráficos que realizó Celia Fermoselle mediante la técnica de *Pixel Art*. Realizó una serie de gráficos de escenarios, personajes y objetos y posteriormente no continuó con su trabajo dado que volcó su carrera artística a aprender 3D.

b' Animaciones iniciales:

Aparte de los gráficos iniciales, Celia Fermoselle también hizo las animaciones del personaje principal y la bibliotecaria. Posteriormente no continuó por los motivos comentados.

c' Gráficos:

Los gráficos que no realizó Celia Fermoselle los realice yo mediante la misma técnica artística (*Pixel Art*). Estos fueron los distintos cursores que aparecen en el juego, todos los gráficos de las interfaces gráficas excepto el inventario, y retoques a los gráficos de Celia que contenían algunos fallos.

d' Animaciones:

Al igual que con los gráficos, las animaciones faltantes las realicé yo. Estas animaciones fueron el viento de la ventana y las banderitas cayéndose del mapa, que las realicé gracias al editor de imágenes para *Pixel Art GraphicsGale*.

2) *Sonido:*

a' Músicas iniciales:

Estas fueron las músicas realizadas por *boredBit*. Por suerte con las tres canciones

que hicieron, fue más que suficiente para el contenido que iba a ofrecer la demo técnica de **1812: La aventura** .

b' Efectos sonoros iniciales:

Estos fueron los sonidos realizados por *boredBit*. Pero al contrario que con las músicas, solo se hicieron unos pocos sonidos de los que al final solo se pudieron usar un par: el de pulsar botón y el de abrir y cerrar puerta y/o ventana.

c' Efectos sonoros:

Al juego le hacia falta por lo menos un sonido más: el del viento soplando a través de la ventana. Como *boredBit* había desaparecido tuve que apañarmelas sola para crear el sonido aunque nunca lo hubiera hecho antes. Por suerte encontré el programa *Bfxr* con el que jugando con sus ajustes logré crear un ruido blanco convincente para el juego, al que luego lo edité con *Audacity* para regularle los niveles de volumen y ajustar el *loop* del sonido.

3) *Texto*:

Con la reorientación del proyecto de videojuego a **Motor de Aventuras Gráficas 2D de Unity** , decidí que no era necesario recurrir a la ayuda de colaboradores dado que la cantidad de diálogos y textos a incluir en la demo no iba a ser muy grande.

a' Guión en español:

El guión se fue escribiendo a medida que se iban implementando los sistemas con los que crear nuevos objetos interactivos, personajes, escenas cinematáticas e interfaces gráficas.

b' Artículos de la Gadipedia:

Finalizando el guión, me dispuse a ir escribiendo la información que faltaba para los artículos de la Gadipedia. Al ser solo una demo técnica, solo escribí unos pocos artículos con información histórica sobre edificaciones que fueron de relevancia durante el sitio de Cádiz de 1810 a 1814. De esta forma que servían de apoyo para la resolución del puzzle contenido dentro del juego si con las pistas no era suficiente.

c' Guión y artículos en inglés:

Una vez acabado de escribir por completo el guión y los artículos de la Gadipedia de la demo técnica de **1812: La aventura** , me dispuse a realizar la traducción completa de los textos en inglés. Así durante la demo técnica se podría ver con más claridad la independencia de los ficheros de texto en *.xml* que contienen los nombres y diálogos del código del juego gracias al sistema de lectura de ficheros implementado.

c) *Desarrollo*:

En este periodo de tiempo se comprende todo el proceso de desarrollo para la implementación del **Motor de Aventuras Gráficas 2D de Unity** junto con la demo técnica de **1812: La aventura** .

1) *Análisis*:

La fase de análisis fue la que más relevancia tuvo en las primeras semanas del proyecto. Pero como fueron surgiendo cambios en el diseño del juego, del puzzle, e imprevistos a la hora de la implementación de alguna parte del código, en algunos *sprints* se tuvo que volver a la fase de análisis para volver a recalcular algunos aspectos. Por estos motivos en el diagrama de gantt orientativo decidí poner que esta fase duró todo el tiempo que se estuvo implementando el proyecto.

2) *Diseño*:

Al igual que con la fase de análisis, se tenía que volver a esta fase cada vez que se

cambiaba algo en el proceso de análisis del proyecto. Así que por los mismos motivos citados en la anterior fase, decidí poner que esta fase duró todo el tiempo que se estuvo implementando el proyecto. Cabe mencionar que al poco tiempo de empezar con *Unity*, descubrí que realizar diagramas de clases UML iba a causar muchos problemas, al final se decidió no realizarlos tal y como se detalla en la sección de Diseño 3.5.

3) *Implementación:*

La fase de implementación del **Motor de Aventuras Gráficas 2D de Unity** y la demo técnica de **1812: La aventura** fue, con diferencia, la más extendida de todo el desarrollo. En gran medida debido a la inexperiencia del entorno de trabajo, el nuevo lenguaje de programación y sus librerías, y otros sistemas necesarios como *SQLite*. Todo esto añadido a que ha sido la primera vez que he implementado un videojuego con mediaña complejidad, fueron surgiendo numerosos imprevistos a lo largo del proyecto que prolongaron el tiempo de desarrollo aún más. Pero se llegaron a implementar todos los objetivos marcados en el proyecto, a destacar: los *scripts* encargados de trabajar con ficheros .xml y el del sistema de diálogos independiente del código.

4) *Pruebas:*

Durante la fase de implementación, se fueron realizando pruebas de cada *script* individual (con sus correspondiente objeto en *Unity*) y posteriormente ver si se integraban con los demás sistemas correctamente. Aparte de esto, la demo técnica de **1812: La aventura** fue probado de forma extensiva por gente voluntaria de las redes sociales y conocidos con *builds* (ejecutables del proyecto sin acabar) cada vez que lograba implementar con éxito un sistema nuevo dentro del juego. Esto me ayudó a pulir detalles como el cambiar los controles a unos más cómodos, corregir fallos y balancear la dificultad del puzzle del mapa.

5. **Distribución (hito):**

Una vez finalizada la implementación del proyecto y corregido los fallos más graves que fueron presentándose a lo largo de las pruebas finales de la demo, se distribuyó el juego mediante el repositorio de *GitHub*, el blog del proyecto, la web de *GameJolt* y las redes sociales.

6. **Entrega del Proyecto Final de Carrera (hito):**

Finalmente, para Septiembre de 2015 se entregó toda la documentación del proyecto y se preparó la presentación final que tuvo lugar unas semanas después.

Capítulo 3

Desarrollo

3.1. Metodología

1812: La aventura es un sistema software complejo y para su desarrollo se ha decidido emplear la metodología SCRUM [149] simplificada. Así mismo, se ha escogido la notificación *UML* con modificaciones (para poder hacerla compatible con *Unity*, al ser un motor de videojuegos orientado a **Objetos** y componentes) como la notación para modelar nuestro sistema.

Mediante la metodología SCRUM podemos realizar el desarrollo de un software orientado a objetos de forma iterativa y reiterativa. Las ventajas que nos proporciona el paradigma de la orientación a objetos consisten en una mayor reusabilidad (el objetivo principal de la demo técnica de **1812: La aventura**), escalabilidad, legibilidad y sencillez de mantenimiento.

3.2. Documentación

En esta sección detallaremos cómo ha sido el proceso de documentación para crear el Motor de Aventuras Gráficas 2D en *Unity* y la demo técnica de **1812: La aventura** para probarlo. Entre otras cosas, hablaremos de cómo se realizó la búsqueda y recopilación de información histórica necesaria para luego poder diseñar luego la documentación de Diseño del Juego y la del Puzzle incluido en la demo. También comentar brevemente cómo se aprendió a manejar el editor de *Unity* conjuntamente con su API y finalmente a escribir en C# y usar algunas librerías .NET .

3.2.1. Documentación histórica

Cuando el objetivo de este proyecto aún estaba enfocado al desarrollo de una aventura gráfica completa de nombre **1812: La aventura** , la información histórica que pudiera recopilar era sumamente importante. Pues iba a usarse para contextualizar el argumento del juego, los diálogos y en última instancia (y más importante) en el diseño de los puzzles que iba a contener.

Sin embargo, al reorientar este proyecto únicamente al de desarrollar un Motor de Aventuras Gráficas 2D en *Unity*, se redujo notablemente la cantidad de información histórica a recabar. No obstante, comentaremos de forma breve el como enfoqué la búsqueda en su momento, la bibliografía examinada, y por último, los elementos históricos usados para el puzzle que aparece en la demo técnica de **1812: La aventura**.

Al principio del todo, al carecer de bastantes conocimientos históricos pues solo contaba con lo que aprendí de Historia general en bachillerato, y tampoco ningún campo de la carrera tocaba ese ámbito. Por suerte, decidí contactar con un antiguo conocido llamado *José Joaquín Rodríguez* de varias actividades culturales en las que habíamos participado los dos. En esos momentos José Joaquín estaba ocupado terminando su tesis doctoral del *Doctorado en Historia*, con lo que me mandó una lista bibliográfica escueta pero bastante útil. Aparte de eso, decidí por mi cuenta buscar en el catálogo de la **biblioteca de la UCA** [92] y encontré los **Recursos Doceañistas** [93] de los que disponían las bibliotecas de la UCA.

Al final entre las recomendaciones y los recursos disponibles, acabé decidiéndome por un libro generalista centrado en diversos aspectos temáticos (viviendas y barrios, la burguesía ilustrada y sus cafés, periodismo, plagas, defensas y tipos de batallones, etc) en la ciudad de Cádiz en la época de 1812 que uno centrado únicamente en la Constitución. Este libro es el de *El Cádiz de las Cortes : la vida cotidiana en la ciudad en los años 1810 a 1813* de *Ramón Solís* [69] y acabó para darme una idea general de la situación económica, política, social y sanitaria de la época que me sirvió para diseñar el Documento de Diseño de Juego inicial.

Siendo ese libro el que más usé, también cabe mencionar los recursos bibliográficos de la *Junta de Andalucía* [51] para el ámbito educacional, cómo los de las webs especializadas que salieron con el Bicentenario del *Ayuntamiento de Cádiz* [8] y nuevamente de la *Junta de Andalucía* [52].

Lamentablemente, gran parte de la información recopilada se tuvo que descartar con la reorientación del proyecto a únicamente desarrollar una demo técnica para probar el Motor de Aventuras Gráficas 2D en *Unity* pero igualmente me fue de utilidad para diseñar el puzzle contenido en la demo técnica. Esto y otros recursos como material cartográfico que detallaremos luego en la sección de la **Documentación del Puzzle 3.2.4**.

3.2.2. Documentación de Unity y C#/.NET

No sólo de la Historia de Cádiz me faltaban conocimientos, también me faltaban de *Unity* y C#/.NET pues cuando empecé con el proyecto nunca había trabajado ni con el motor ni con el lenguaje. Por suerte, la búsqueda de información en este campo me resultaba harto fácil comparado con la histórica.

C# era un lenguaje nuevo para mí, pero al saber C++ gracias a la carrera y ver que su estructura de Orientación a Objectos era bastante similar a la de Java (lenguaje enseñado en la asignatura de *Programación Concurrente y Distribuida*) me fue fácil y rápido adaptarme, además de que en caso de duda siempre podía echar un vistazo a los tutoriales oficiales de C# y .NET [60] de Microsoft. Sólo me fue necesario ir consultando la documentación de referencia de .NET [59] de vez en cuando sobre las estructuras de .NET (sobre todo System.Collections.Generic, System.IO, System.Xml y System.Linq) que me eran desconocidas por completo.

En cuanto a *Unity*, por suerte tiene una gran colección de tutoriales oficiales [86] de diversa índole con las que empezar a poder crear rápidamente los primeros prototipos, y por si fuera poco cuenta con una

gran comunidad, así que para dudas más especializadas que no vengan cubiertas en la página oficial de *Unity* podía consultar **Unity Answers** [84] o **StackOverflow** [72]. Mención aparte de todo esto, también cabe mencionar el libro **Unity 2D Game Development** [24] lo usé como referencia para aprender a usar las *Máquinas de Estado Deterministas* mediante *scripts* para la gestión del nuevo sistema de animaciones 2D en *Unity* por aquél entonces.

3.2.3. Documentación del Diseño del Juego

Para crear el GDD, al ser un tema del que hay poca información al respecto sobre cómo realizar un documento de este tipo y viendo que los diseñadores de videojuego actuales tienen cada uno su propia manera particular para desarrollarlos. Ante tal situación, al final acabé creando mi documento de diseño del juego para demo técnica de **1812: La aventura** a partir de los documentos liberados de aventuras gráficas ya citadas como el de *Grim Fandango* [76] o *Leisure Suit Larry* [2], también me basé en el GDD de Sion Tower [25], proyecto final de carrera del antiguo compañero de la universidad *David Saltares Márquez*, al que espero que le depare un gran porvenir dentro de las empresas grandes de desarrollo de videojuegos.

El contenido del Documento de Diseño del Juego resultante que realicé se puede leer en su totalidad en la siguiente dirección:

<https://github.com/Firenz/1812/blob/master/documentacion/documentacion-dise~no-juego.pdf>

3.2.4. Documentación del Puzzle

Si bien el puzzle implementado en la demo técnica de **1812: La aventura** tiene su propio Documento de Diseño de Puzzle, en esta sección repasaremos la documentación histórica que se tuvo que hacer para poder desarrollar el puzzle. Cuando empecé a pensar sobre qué tipo de puzzle implementar, quería que fuera sobre un tema importante que todo estudiante debería saber pero a la vez que fuera algo generalista, que no fuera de un tema muy específico que sólo pudiera interesar a un grupo de estudiantes de especialidades concretas, así que al final opté por hacer un puzzle que enseñase los emplazamientos más importantes del asedio de Cádiz entre 1810 y 1814: un mapa al que hubiera que colocar qué tipo de emplazamiento era cada cual, dado que si lo hubiera dividido simplemente por cuáles zonas eran francesas o españolas habría sido demasiado fácil.

Una vez decidido el tipo de puzzle a implementar, me dispuse a documentarme sobre la cartografía de Cádiz sobre los años 1790 hasta el 1820. Para acceder a ese tipo de documentación, tuve que recurrir al **Banco de Documentación Hispánica** [62], en la que después de varios intentos de búsqueda en ella, logré acotar los resultados a las pedanías de Cádiz en dicha época [10].

Al final, los mapas más reveladores fueron:

- *Plano de la Ciudad de Cádiz en 1812* [13]
- *Plano de la Plaza de Cadiz y Fuertes dependyentes de ella hasta la cortadura de S. Fernando* [15]

- *Map of the Country round Cadiz* [11]
- *Plano de la bahía de Cádiz y sus contornos* [14]
- *Vista de Cádiz y sus contornos* [16]
- *Plan de Cadiz et de ses environs* [12]

Con esta cantidad de mapas fue más que suficiente como para obtener los emplazamientos más importantes y también varios secundarios que podían llamar la atención. Por último, me faltaba obtener información de estos lugares para poder elaborar las pistas a encontrar durante la demo técnica de **1812: La aventura**, así tuve que recabar más información histórica sobre las estructuras más importantes volví a recurrir a los **Recursos Doceañistas de la biblioteca de la UCA**: en este caso recurrí a dos libros centrados en los lugares, se llamaban *Cádiz y los lugares del Doce : de la Puerta de Tierra a La Caleta* [40] y *Cádiz y los lugares del Doce : de la Puerta de Tierra a Cortadura* [39]. Una vez ya con toda esta información, terminé de diseñar el nivel del *Despacho del profesor* en la demo y las pistas para poder completar el puzzle del mapa.

Por lo demás, el contenido del Documento del Puzzle resultante (que incluye el proceso de creación del puzzle del mapa paso a paso) que realicé se puede leer en su totalidad en la siguiente dirección:

<https://github.com/Firenz/1812/blob/master/documentacion/documentacion-puzzle.pdf>

3.3. Pasos previos

En esta sección hablaremos de los pasos que se han tenido que realizar antes de poder empezar a desarrollar el proyecto en sí.

3.3.1. Instalación de Unity

Para poder instalar *Unity*, primero hacía falta un sistema operativo Windows u OS X, en mi caso preferí optar por el de Windows ya que es el sistema operativo que uso diariamente. Después de haber elegido el sistema operativo, tendremos que registrarnos en la página web oficial de *Unity* y después elegir la versión entre la edición Personal o Profesional, en mi caso fue la edición Personal que es la versión gratuita del motor.

Una vez descargado e instalado *Unity*, lo ejecutamos y tendremos que hacer login con nuestra cuenta creada antes y ya nos saldrá la pantalla de inicio de *Unity*. Le damos al botón de *New Project* y elegimos nombre del proyecto, el directorio de destino donde se creará, si es un proyecto con gráficos 3D o 2D y finalmente si le vamos a incluir algún *Asset* preinstalado (en este caso ninguno). Pulsamos finalmente el botón *Create Project*, se nos generará un proyecto vacío y se mostrará el editor del motor con un escenario vacío.

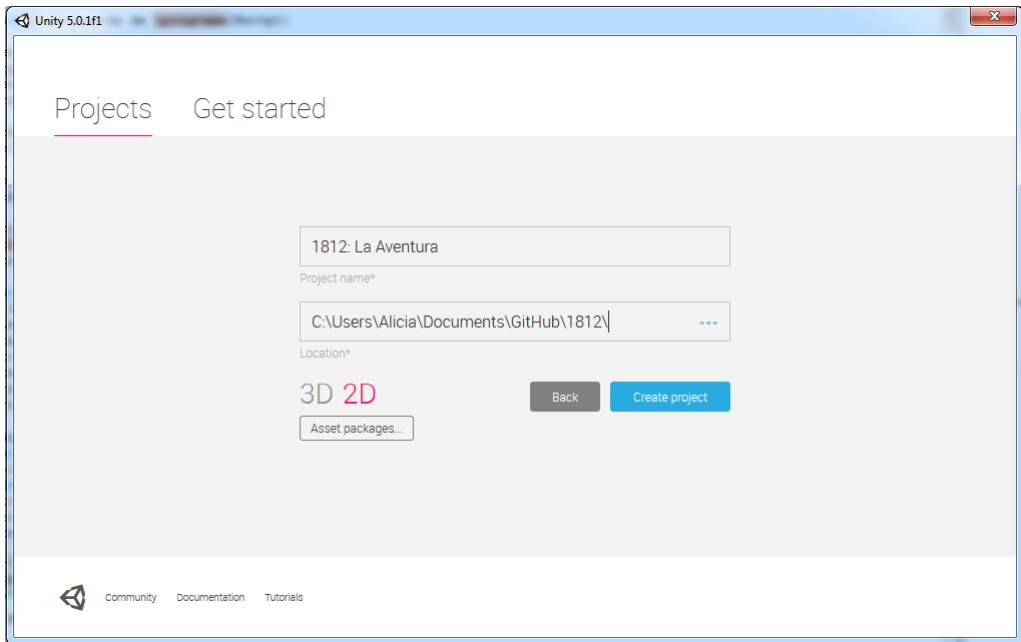


Figura 3.1: Pantalla de creación de nuevo proyecto en *Unity*

Como mención especial, decir que hace escasos días *Unity* lanzó una versión de prueba para GNU/Linux orientado a Debian [85]. Con suerte dentro de un año ya será estable para GNU/Linux y no habrá que depender de sistemas operativos propietarios para usar el motor y el editor.

3.3.2. Preparación del proyecto de Unity para ser subido a un repositorio

Dado que había que trabajar en Windows por no existir ninguna versión de *Unity* para GNU/Linux y no poseer un Mac OS X, tuve que buscar software de Control de Versiones basados en la tecnología Git con aplicación para este sistema. Algunos bastante completos tales como *SourceTree* [7] o la oficial de programaGit para Windows [57]. Aunque finalmente opté por *GitHub For Windows* [34] por su simplicidad, incluye tanto una versión con interfaz como una por comandos. En mi caso preferí la consola de comandos para así ir aprendiendo y memorizando los comandos de Git más comunes a usar, ganando experiencia en esta tecnología para cuando tenga que afrontar futuros proyectos que la usen.

```

posh~git ~ 1812 [master]
a--- 12/07/2014 22:22 1990 PlayerStateListener.cs
a--- 12/07/2014 18:23 19768 RedSquare.psd
a--- 10/07/2014 16:23 626 ssh-agent.exe.stackdump
a--- 13/07/2014 23:40 576654 TitleCard.png
a--- 13/07/2014 23:47 5880126 TitleCard.psd
a--- 13/07/2014 17:22 20644 WhiteSquare.psd

C:\Users\Alicia\Documents\GitHub> cd 1812
C:\Users\Alicia\Documents\GitHub\1812 [master +1 ~4 -0 !]> ls

    Directorio: C:\Users\Alicia\Documents\GitHub\1812

Mode           LastWriteTime      Length Name
----           -----          -----
d---           10/07/2014 20:25            documentacion
d---           15/07/2014 12:49            project
-a--           15/07/2014 19:11        631 .gitignore
-a--           10/07/2014 20:25      35293 LICENSE
-a--           10/07/2014 20:25      1051 README.md

C:\Users\Alicia\Documents\GitHub\1812 [master +1 ~4 -0 !]> _

```

Figura 3.2: Consola de comandos de GitHub For Windows

Pero eso no era todo lo que había que hacer para que *Unity* funcionase con repositorios. Normalmente en otro tipo de proyectos que solo entrañan código y librerías, con excepción de algún archivo de gráficos y de audio, no generan demasiados problemas el subirlos a un repositorio. Pero con *Unity* la cosa difería bastante ya que los proyectos realizados con este software, generan archivos auxiliares y metadatos. Si bien estos archivos normalmente son inocuos a la hora de subirlos al repositorio para mí como desarrolladora, si se hubiera dado el caso de que algún interesado se bajase el proyecto y lo ejecutara, podría tener la mala suerte de que no pudiera abrir el proyecto debido a problemas de mal enlazamiento entre los ficheros auxiliares del proyecto porque estos tienen referenciado carpetas de mi ordenador, no del suyo.

Para evitar este problema, hubo que modificar algunos parámetros del proyecto en *Unity*.

1. Para hacer que se muestren los ficheros auxiliares de *Unity* de cara al software de Control de Versiones:

Edit->Project Settings->Editor->Version Control Mode = Meta Files

2. Para que *Git* pudiera usar bien su diferenciador de código entre distintas versiones de los ficheros:

Edit->Project Settings->Editor->Asset Serialization Mode = Force Text

3. Por último y ya teniendo instalada la aplicación para trabajar con *Git* junto con una copia del repositorio para trabajar en el ordenador, en el directorio donde alojamos el repositorio, nos sale un fichero de nombre `.gitignore`. Se le hace click derecho y se elige editar con un editor de texto, en este caso usé *Geany* [29] y escribí los directorios, ficheros, y extensiones de ficheros que quería evitar subir al repositorio. En este caso concreto, el fichero se llenó con lo siguiente:

```

# ===== #
# Unity generated #
# ===== #
Temp/
Library/
# ===== #
# Visual Studio / MonoDevelop generated #
# ===== #
ExportedObj/
obj/
*.svd
*.userprefs
/*.csproj
*.pidb
*.suo
/*.sln
*.user
*.unityproj
*.booproj
# ===== #
# OS generated #
# ===== #
.DS_Store
.DS_Store?
._*
.Spotlight-V100
.Trashes
ehthumbs.db
/*Thumbs.db

```

Cada línea de este fichero es un archivo o (directorio si contiene "/") a ignorar cada vez que se subiera una nueva versión del proyecto al repositorio, aliviando la carga de escrutinizar los ficheros a subir para evitar la subida de archivos indeseados. Como nota final, los asteriscos sirven para indicar que nos da igual el nombre, solo los caracteres que lo acompañan a la hora de filtrar ficheros y directorios y también que si escribimos una línea empezando por "#" es un comentario y por tanto no será leído por *Git* a la hora de hacer uso de *.gitignore*.

Una vez hecho estos pasos ya se pudo sin problemas subir el proyecto de *Unity* a un repositorio basado en *Git*.

3.3.3. Normas de estilo para código de los scripts en C#/NET y Assets de Unity

Si bien en este caso en particular, al ser yo la única programadora del proyecto, no es tan necesario contar con unas normas de estilo estrictas al desarrollar el proyecto, me propuse el seguir las. La decisión de hacerlo fue para facilitar la legibilidad del código para que lo pudiesen leer con claridad otros desarrolladores que entraran en el repositorio del proyecto a ver el código, además de que saber seguir normas de estilo es algo que resulta esencial aprender para poder trabajar en proyectos de desarrollo de software de más de una persona.

Si bien no hay unas normas de estilo oficiales ni para C# o *Unity*, existen varias guías de estilo recomendadas, en mi caso opté por la que hay definida en la wiki oficial de *Unity*: concretamente la página [CSharp Coding Guidelines](#) [90].

3.3.4. Organización de Assets

Aparte de las normas de estilo, también era necesario llegar a un consenso sobre cómo organizar los directorios de los *Assets* dentro del proyecto de *Unity*. En este caso el consenso general entre los desarrolladores de *Unity* de cómo hacerlo es más ambiguo aún que con las normas de estilo, dado que cada proyecto hecho en *Unity* difiere en los tipos de *Assets* a organizar dependiendo enormemente del tipo de juego y estilo que tenga.

Teniendo en cuenta al final que necesitaba un directorio para los datos a leer y/o escribir en archivos .xml, otro para colocar las librerías de terceros para poder trabajar con bases de datos y plugin de *Doxygen* para generar la documentación hicieran, los diversos recursos a los que la mayoría tendrá que acceder mediante *scripts*, las escenas de la demo técnica y finalmente uno para organizar los *scripts* que fuera a usar en los objetos de *Unity*; al final opté por la siguiente distribución de directorios para los *Assets*:

- *Assets/*
 - Data/
 - DB/
 - Localization/
 - Saves/
 - Plugins/
 - Doxygen/
 - x86/
 - x64/
 - Resources/
 - Animations/
 - Fonts/
 - Graphics/
 - Prefabs/
 - Sound/
 - Scenes/
 - Menus/
 - Levels/
 - Scripts/
 - System/
 - Files/
 - GUI/
 - Navigation/

- Actors/
- InteractiveElements/
- PuzzleElements/
- Inventory/

3.3.5. Instalación del plugin de documentación Doxygen en Unity

Para poder documentar debidamente el código de los *scripts* del proyecto en *Unity*, no es tan sencillo por su orientación a **GameObjects** y componentes como en un proyecto que únicamente dependa de archivos de código y los recursos gráficos y de audio. Decidí usar *Doxygen* para documentar todo el código y a partir de sus resultados generar los diagramas UML gracias a *Graphviz*, y así después que hubiera una referencia para que desarrolladores externos pudieran usar al querer usar el Motor de Aventuras Gráficas de *Unity*.

Para poder integrar *Unity* con *Doxygen* era necesario utilizar un *plugin*, lamentablemente no disponía del tiempo ni de los conocimientos para hacerme uno propio, así que opté por usar uno ya hecho: **Jacob Pennock's Unity Automatic Documentation Generation** [49]. Este *plugin* es gratuito y con sólo seguir unos cuantos pasos explicados en el vídeo del tutorial, ya podremos hacer uso de *Doxygen* dentro de *Unity* para generar la documentación.

3.4. Análisis

3.4.1. Especificación de requisitos del sistema

En esta sección haremos un recorrido por los requisitos que debe cumplir la demo técnica de **1812: La aventura** tanto en el plano de interfaces, como de rendimiento o de funcionalidades entre otros.

Requisitos de interfaces externas

En este apartado describiremos los requisitos relacionados con la conexión entre el hardware y el software además de la interfaz con el usuario. En este caso, *Unity* es un motor de videojuegos completo y por lo tanto, distintos elementos que lo componen se encargan de realizarla estas gestiones:

- Para gestionar la captura de entrada del usuario, tenemos la clase `Input` [81].
- Para el renderizado de gráficos 2D, cuenta con varios tipos de componentes, los más importantes son `SpriteRenderer` [83] y `Sprite`. Se pueden gestionar en los *scripts* con las clases del mismo nombre.
- Para el audio, tenemos los componentes `AudioSource`, `AudioListener` y `AudioMixer` [89]. Se pueden gestionar en los *scripts* con las clases del mismo nombre.

- Para crear interfaces, está el nuevo sistema *UI* [87] compuesta de diversos componentes para generar una interfaz. Para poder acceder a *UI* y sus clases desde *scripts* hay que importar la librería `UnityEngine.UI`.

La demo técnica de **1812: La aventura** deberá soportar varias resoluciones de pantalla siempre que se mantenga su resolución de aspecto 4:3. Además, debe ser posible establecer un modo a pantalla completa si el usuario lo desea. Varias resoluciones aceptadas por el videojuego deben ser:

- 640 x 480
- 800 x 600
- 960 x 720
- 1024 x 768
- 1152 x 864
- 1280 x 960

La navegación por los menús debe ser intuitiva y se realizará utilizando el ratón. En la figura 3.3 pueden observarse todas las pantallas de la demo técnica de **1812: La aventura** dispuestas en un diagrama de flujo para mostrar su navegación:

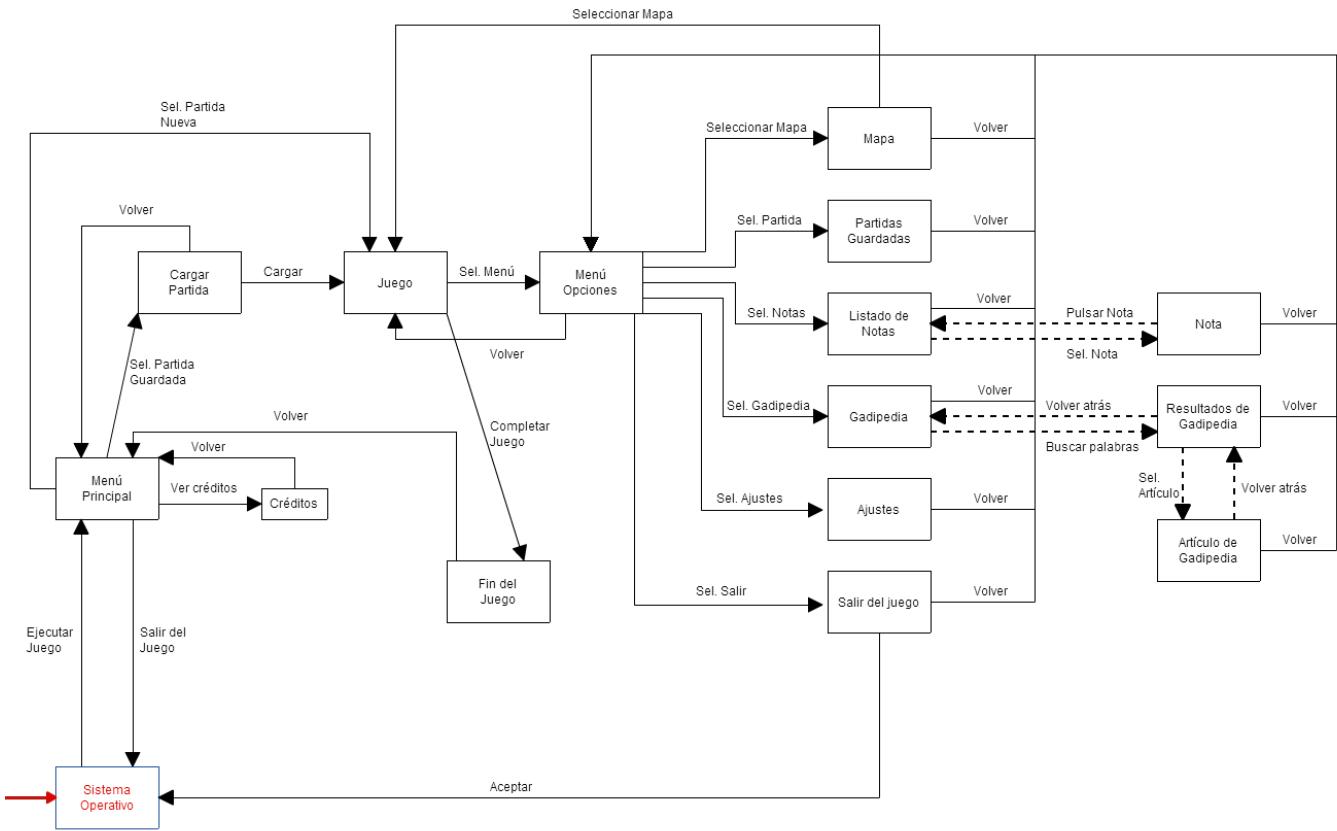


Figura 3.3: Diagrama de flujo de las pantallas de la demo técnica de **1812: La aventura**

Describimos de manera individual todas las pantallas que hacen aparición en **1812: La aventura**.

A continuación el boceto de la pantalla de *Menú Principal*:



Figura 3.4: Boceto del Menú Principal

Lista y descripción de todos sus componentes:

- **Botón Nueva Partida:** al pulsarlo lleva a la pantalla de *Crear Partida*.
- **Botón Cargar Partida:** al pulsarlo lleva a la pantalla de *Cargar Partida*.
- **Botón Créditos:** al pulsarlo nos lleva a la pantalla *Créditos*.
- **Botón Salir:** al pulsarlo nos lleva de vuelta al Sistema Operativo.
- **Imagen de un logo antiguo de la Guerra de la Independencia:** Un logo antiguo que se ha pixelado y cambiado el texto que contenía por **1812: La aventura**, el nombre del juego de la demo técnica.

A continuación el boceto de la pantalla de *Cargar Partida* :

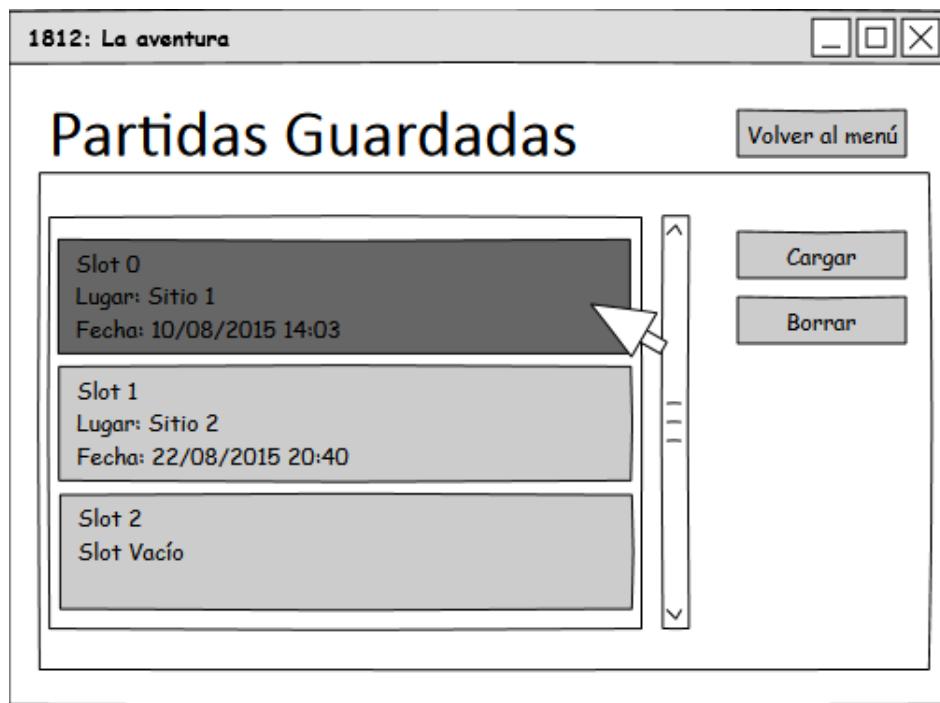


Figura 3.5: Boceto de la pantalla de Cargar Partida

Lista y descripción de todos sus componentes:

- **Lista de partidas:** lista con las partidas guardadas hasta el momento. Se muestra el día y la hora de la última actualización de esa partida.
- **Botón Cargar:** al pulsarlo coge la partida seleccionada (se pondrá de color oscuro más oscuro el bloque escogido) y carga la pantalla de juego tal y como la dejamos.
- **Botón Borrar:** al pulsarlo nos dará un mensaje de aviso de si verdaderamente queremos borrar la partida del bloque seleccionado, si le damos a sí nos borrará la partida, si elegimos la otra opción desaparecerá el mensaje de aviso sin realizar ningún borrado.
- **Botón Volver al menú:** al pulsarlo volvemos al *Menú Principal*.

A continuación el boceto de la pantalla de *Créditos* :



Figura 3.6: Boceto de la pantalla de Créditos

Lista y descripción de todos sus componentes:

- **Panel:** texto con los componentes del equipo de desarrollo.
- **Botón Volver al menú:** al pulsarlo volvemos al *Menú Principal*.

A continuación el boceto de la pantalla del *Menú de Opciones* :

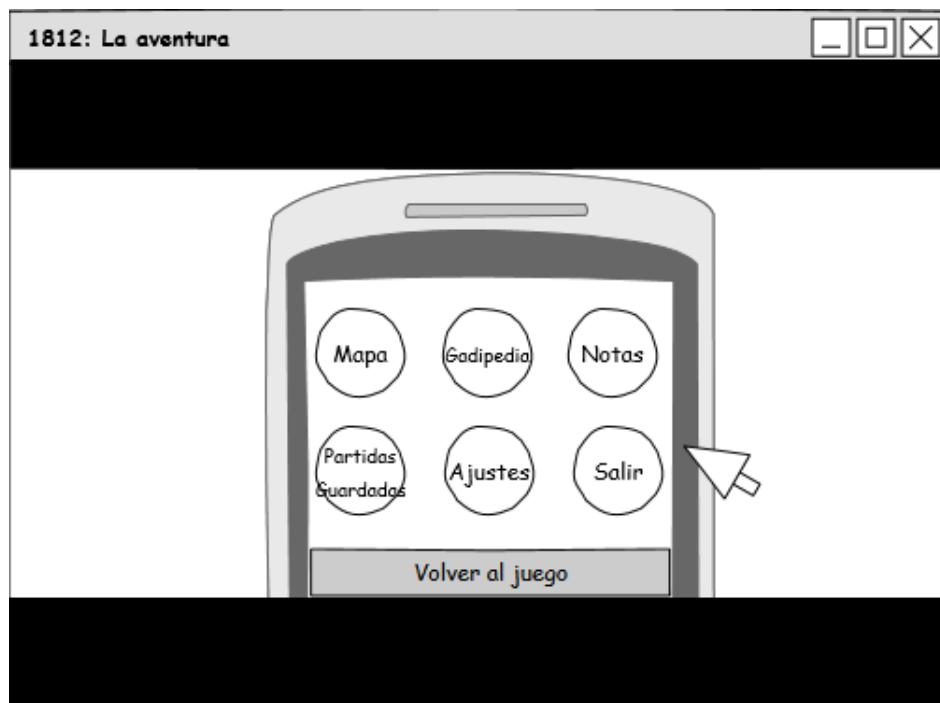


Figura 3.7: Boceto de la pantalla del Menú de Opciones

Lista y descripción de todos sus componentes:

- **Icono Mapa:** al pulsarlo vamos a la pantalla *Mapa*.
- **Icono Gadipedia:** al pulsarlo vamos a la pantalla *Gadipedia*.
- **Icono Notas:** al pulsarlo vamos a la pantalla *Notas*.
- **Icono Partidas Guardadas:** al pulsarlo vamos a la pantalla *Partidas Guardadas*.
- **Icono Mapa:** al pulsarlo vamos a la pantalla *Ajustes*.
- **Icono Salir:** al pulsarlo nos saldrá un mensaje que nos preguntará si queremos salir del juego y que todo lo que no haya sido guardado se perderá, si le damos a *sí* cerrará el juego, si elegimos la otra opción el mensaje desaparecerá sin realizar ninguna acción.
- **Botón Volver al juego:** al pulsarlo volvemos a la pantalla de juego.
- **Imagen smartphone:** el Menú de Opciones simula ser el *smartphone* del protagonista, y las “aplicaciones” son los distintos submenús que hay dentro de la demo técnica. Para acceder a un submenú solo hay que hacer **click izquierdo** sobre uno de los botones que simulan ser aplicaciones.

A continuación el boceto de la pantalla *Mapa* :

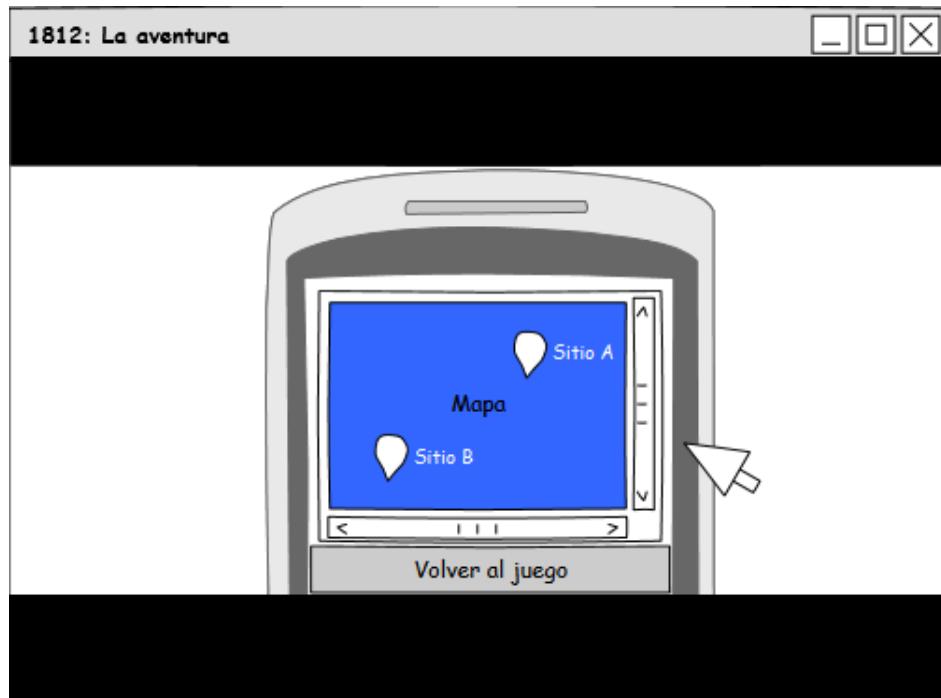


Figura 3.8: Boceto de la pantalla Mapa

Lista y descripción de todos sus componentes:

- **Panel Mapa:** Panel con una imagen de los pasillos de la universidad.
- **Iconos de Sitio:** Muestran un lugar al que se puede visitar dentro del juego, al pasar el cursor por encima nos saldrá un texto diciendo el nombre del sitio al que nos puede trasladar. Al pulsarlo con **click izquierdo** saldrá un mensaje preguntándonos si queremos ir a esa localización, si le decimos que sí volveremos a la pantalla de juego pero en la ubicación especificada, si elegimos la otra opción el mensaje desaparecerá sin realizar ninguna acción.
- **Icono Volver al Menú:** al pulsarlo volvemos a la pantalla *Menú Opciones*.

A continuación el boceto de la pantalla de *Partidas Guardadas* dentro del juego :

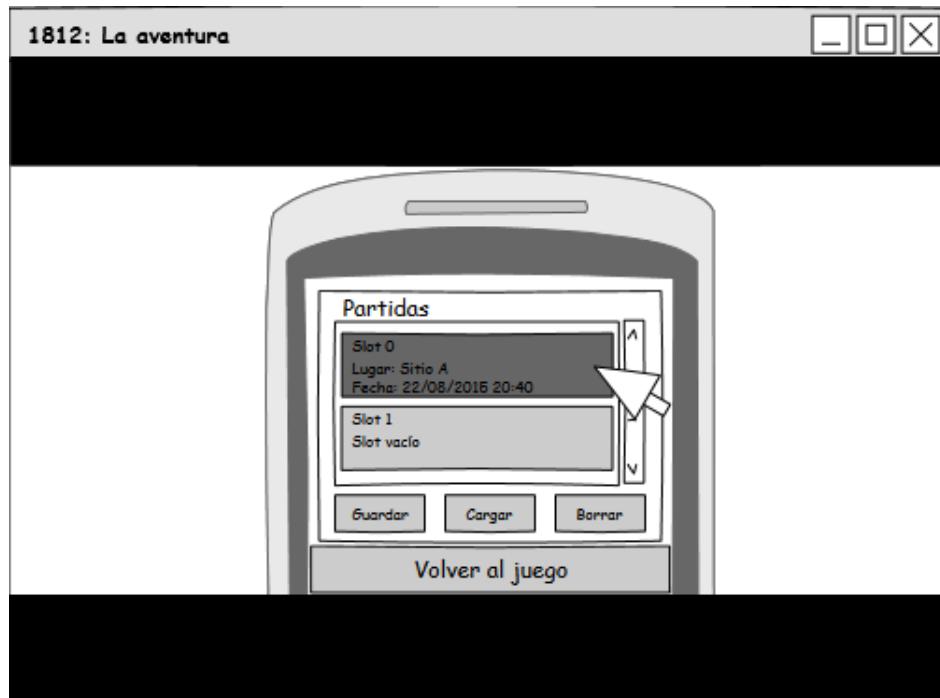


Figura 3.9: Boceto de la pantalla de Partidas

Lista y descripción de todos sus componentes:

- **Botón Guardar:** guarda el estado actual de la partida en el bloque que tengamos seleccionado (si no hay ninguno seleccionado, no hará nada). Nos saldrá una ventana que nos preguntará si queremos guardar la partida, si le damos a sí, se guardará en el bloque seleccionado con los datos del último lugar visitado y la fecha de la última vez que se jugó a esa partida guardada. En caso de indicar que no a la ventana, no pasará nada y se deseleccionará el bloque seleccionado automáticamente.
- **Botón Cargar:** al pulsarlo coge la partida seleccionada (se pondrá de color oscuro más oscuro el bloque escogido) y carga la pantalla de juego tal y como la dejamos.
- **Botón Borrar:** al pulsarlo nos dará un mensaje de aviso de si verdaderamente queremos borrar la partida del bloque seleccionado, si le damos a sí nos borrará la partida, si elegimos la otra opción desaparecerá el mensaje de aviso sin realizar ningún borrado.
- **Lista de bloques de partida:** lista de un total de 6 bloques, contabilizados de 0 a 5, en el que puede haber tanto bloques vacíos (indicados por la frase *Slot vacío*) como otros ocupados con las partidas guardadas hasta el momento. Se muestra el último lugar visitado, el día y la hora de la última vez que se jugó a esa partida.
- **Icono Volver al Menú:** al pulsarlo volvemos a la pantalla *Menú Opciones*.

La *Gadipedia* está compuesto de varios submenús:

Boceto del submenú del Buscador de *Gadipedia*:

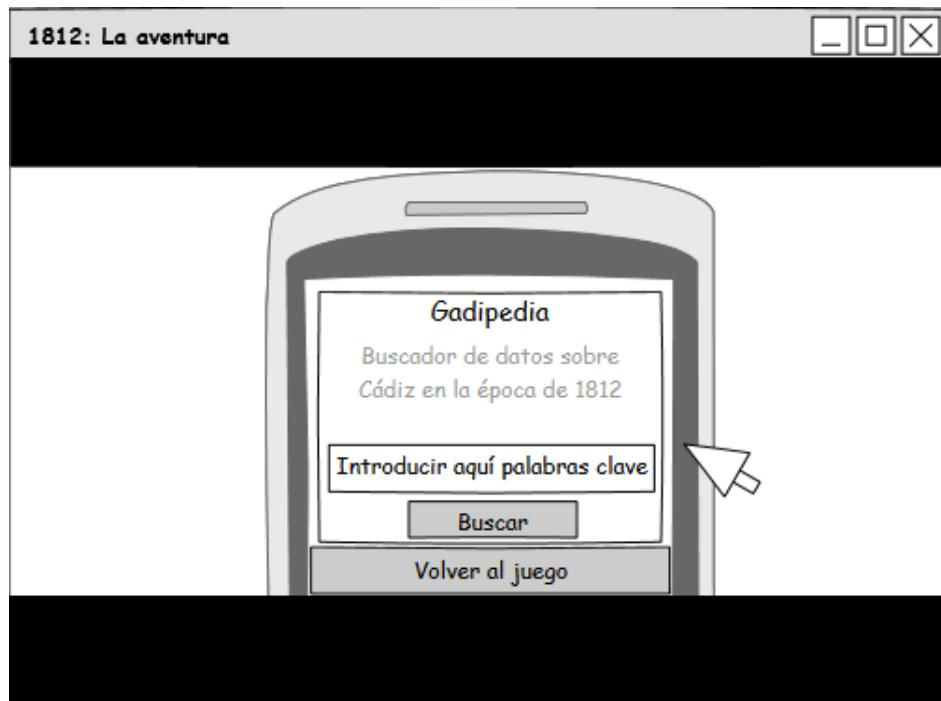


Figura 3.10: Boceto de la pantalla del Buscador de *Gadipedia*

Lista y descripción de todos sus componentes:

- **Texto explicativo:** texto que explica para que sirve la *Gadipedia* y como usarlo.
- **Caja de texto:** caja de texto donde introducimos las palabras clave de la información que queremos saber de la historia de Cádiz de 1812.
- **Botón Buscar:** al pulsarlo cotejará las palabras clave escritas en la caja de texto con la base de datos de artículos relacionados por si están relacionadas con dichas palabras clave. El resultado del cotejamiento sale después en el submenú Resultados de la *Gadipedia*.
- **Icono Volver al Menú:** al pulsarlo volvemos a la pantalla *Menú Opciones*.

Boceto del submenú de Resultados de Gadipedia:

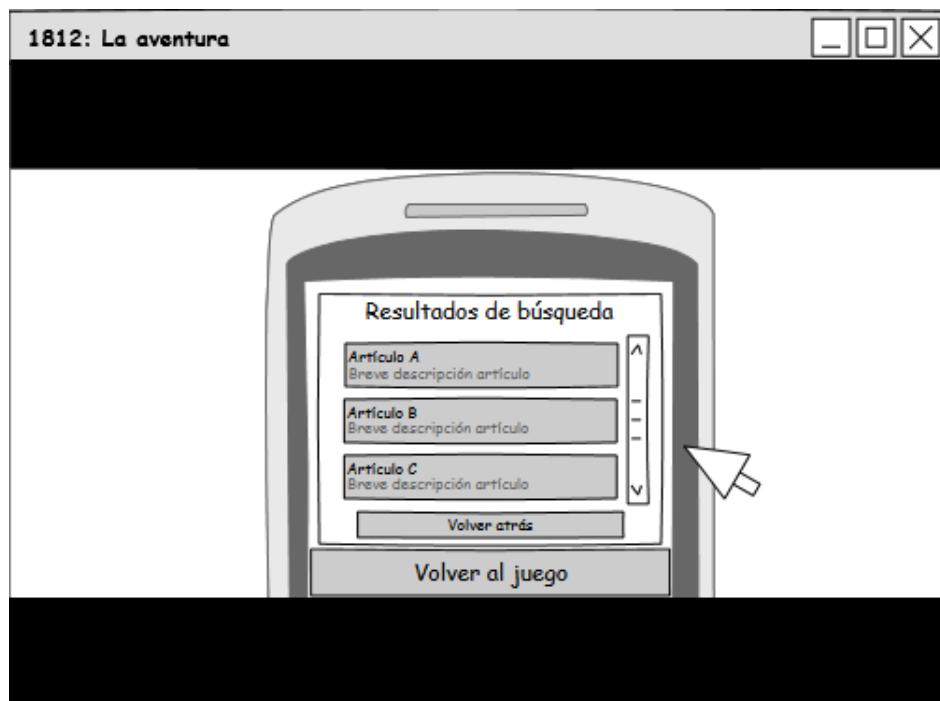


Figura 3.11: Boceto de la pantalla de Resultados de Gadipedia

Lista y descripción de todos sus componentes:

- **Lista de resultados:** lista con los resultados de la búsqueda de las palabras clave en caso de que si se correspondiesen con artículos en la base de datos, en caso contrario saldría un texto diciendo que no ha habido resultados. Cada resultado contiene el nombre del artículo y una breve descripción de este, si hacemos **click izquierdo** en un resultado iremos al submenú de Artículo de *Gadipedia* que contendrá la información completa del resultado clickado.
- **Botón Volver atrás:** al pulsarlo nos lleva de nuevo al submenú de *Búsqueda de Gadipedia*.
- **Icono Volver al Menú:** al pulsarlo volvemos a la pantalla *Menú Opciones*.

Boceto del submenú de Artículo de Gadipedia:

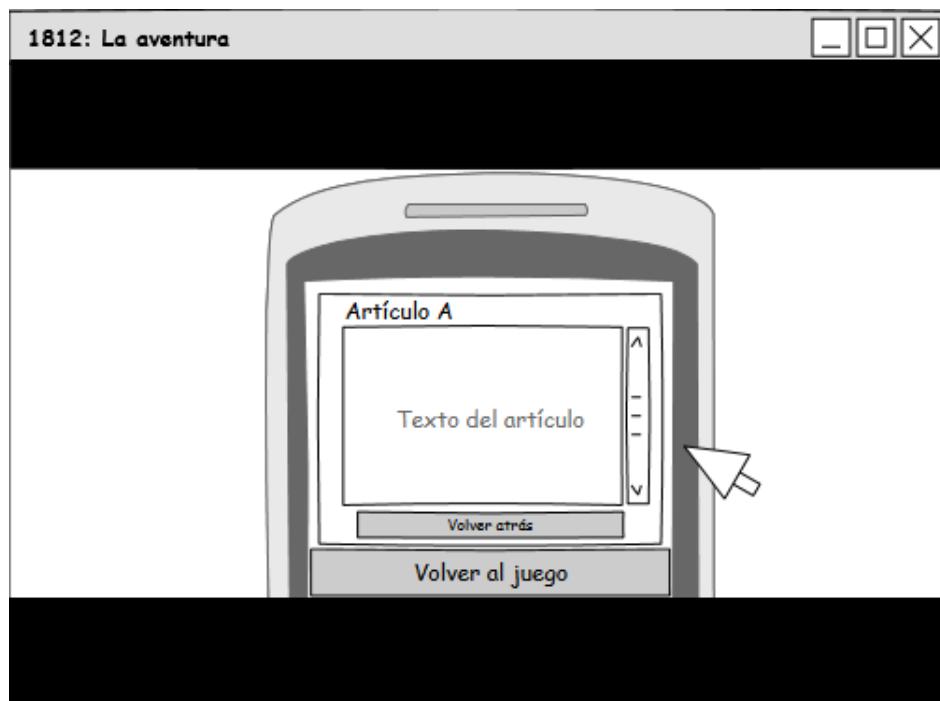


Figura 3.12: Boceto de la pantalla de Artículo de Gadipedia

Lista y descripción de todos sus componentes:

- **Nombre del artículo:** nombre del artículo del resultado que hemos clickado en el submenú de *Resultados de Gadipedia*.
- **Texto del artículo:** contenido completo del resultado que hemos clickado en el submenú de *Resultados de Gadipedia*.
- **Botón Volver atrás:** al pulsarlo nos lleva de nuevo al submenú de *Resultados de Gadipedia*.
- **Icono Volver al Menú:** al pulsarlo volvemos a la pantalla *Menú Opciones*.

El menú de *Notas* está compuesto de dos submenús:

A continuación el boceto de la pantalla Lista de Notas :

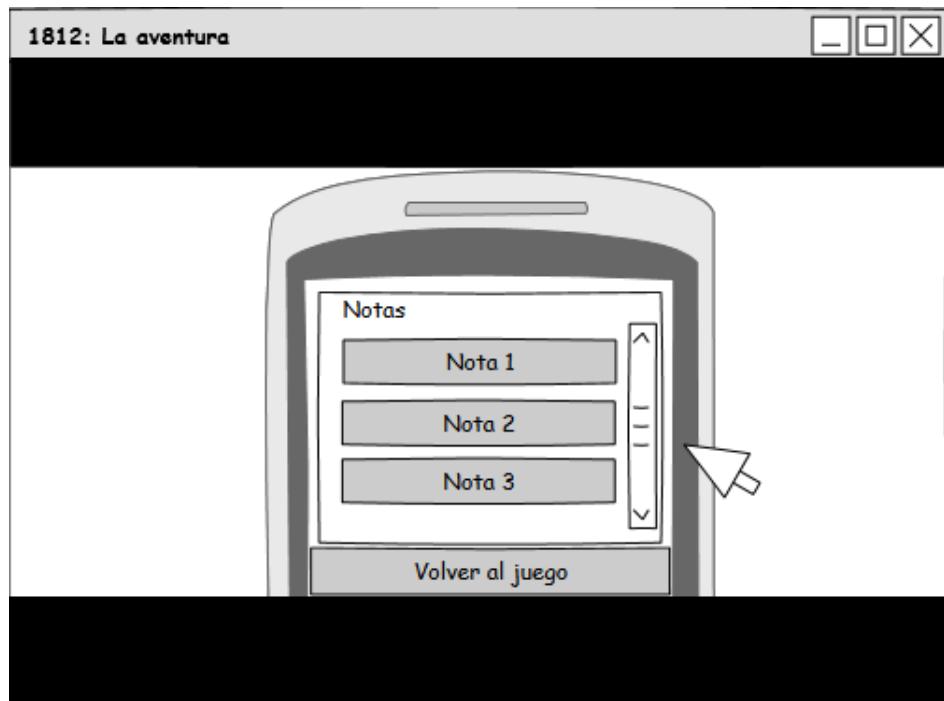


Figura 3.13: Boceto de la pantalla de Lista de Notas

Lista y descripción de todos sus componentes:

- **Lista de notas:** lista con las notas que relatan los sucesos importantes que han ocurrido en la partida hasta el momento. Se muestra el nombre de la nota, que suele es una frase corta resumiendo el suceso.
- **Bloque de texto:** bloque de texto informativo más descriptivo sobre el suceso importante.
- **Icono Volver al Menú:** al pulsarlo volvemos a la pantalla *Menú Opciones*.

A continuación el boceto de la pantalla del submenú Nota:

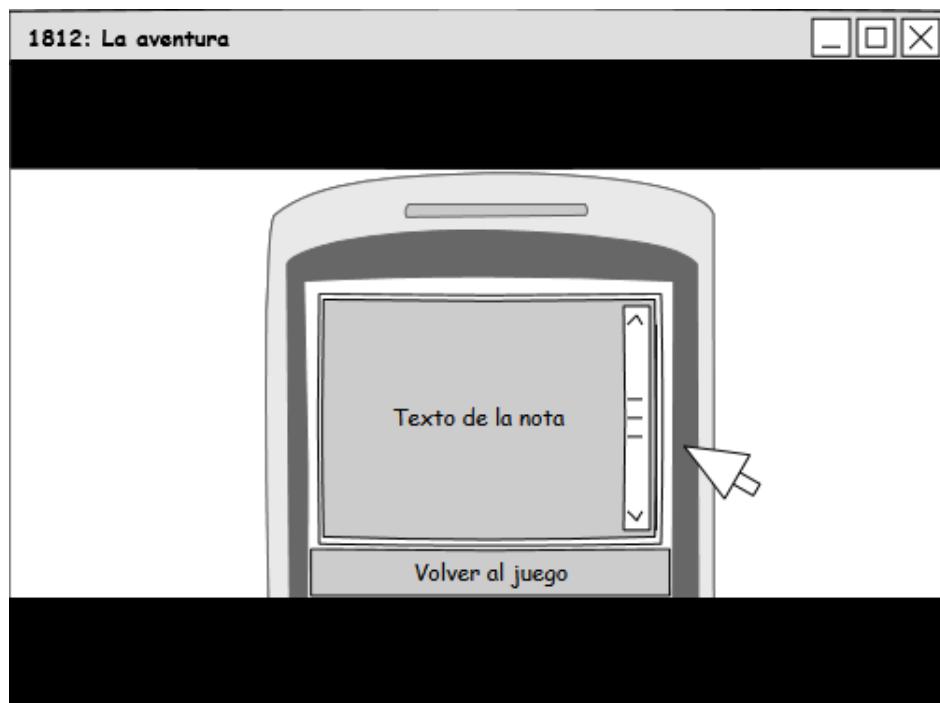


Figura 3.14: Boceto de la pantalla de Nota

Lista y descripción de todos sus componentes:

- **Texto de la Nota:** bloque de texto informativo más descriptivo sobre el suceso importante. Si hacemos **click izquierdo** encima de él, nos lleva al submenú de *Lista de Notas*.
- **Icono Volver al Menú:** al pulsarlo volvemos a la pantalla *Menú Opciones*.

A continuación el boceto de la pantalla *Ajustes* :

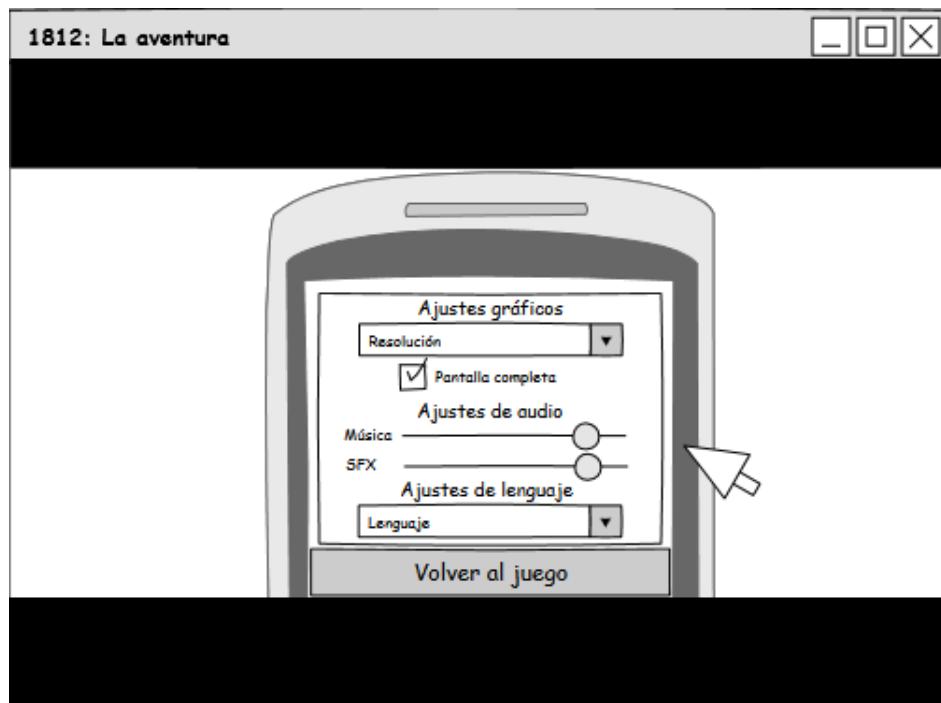


Figura 3.15: Boceto de la pantalla de Ajustes

Lista y descripción de todos sus componentes:

- **Lista despegable de resolución:** lista despegable con las resoluciones de pantalla disponibles (de formato 4:3) para el juego. Por defecto la resolución del juego es 800x600.
- **Casilla de pantalla completa:** si pulsamos la casilla el juego se pondrá en modo pantalla completa, si la desmarcamos se pondrá en modo ventana. Por defecto está desactivada.
- **Barra de volumen de música:** barra ajustable para regular el volumen de música, solo hay cinco niveles de volumen en el juego que van aumentando de 25 en 25 hasta 100.
- **Barra de volumen de efectos de sonido:** barra ajustable para regular el volumen de los efectos de sonido, solo hay cinco niveles de volumen en el juego que van aumentando de 25 en 25 hasta 100.
- **Lista despegable de idiomas:** lista despegable con los idiomas disponibles para el juego.
- **Icono Volver al Menú:** al pulsarlo volvemos a la pantalla *Menú Opciones*.

Requisitos funcionales

La demo técnica de **1812: La aventura** cuenta con los siguientes requisitos funcionales:

- Salir de la aplicación cerrando la ventana en cualquier momento.
- Mover al personaje.
- Interacción entre los elementos del escenario y el personaje. Esto incluye interacciones automatizadas no controladas por el *Jugador*.
- Gestión de partidas guardadas desde el menú principal, cada una con su propio estado del juego y avances desbloqueados. Se permite la selección, creación y eliminación de partidas guardadas.
- Acceder al menú de opciones mientras se está jugando.
- Usar el menú de notas dentro del menú de opciones. Dentro de este menú tenemos dos submenús: la lista de notas y la vista de una nota en particular.
- Usar el menú de gadipedia dentro del menú de opciones. Dentro de este menú tenemos tres submenús: buscar en la gadipedia, resultados obtenidos en la gadipedia y vista de un artículo en particular de la gadipedia.
- Usar el menú de mapas dentro del menú de opciones y trasladar el personaje al mismo u a otro nivel del juego.
- Usar el menú de ajustes dentro del menú de opciones.
- Gestión de partidas guardadas dentro del menú de opciones, cada una con su propio estado del juego y avances desbloqueados. Se permite la selección, creación y eliminación de partidas guardadas.

Requisitos de rendimiento

Para disfrutar de la demo técnica de **1812: La aventura** de forma satisfactoria será necesaria, al menos, una resolución de 800 x 600 y una tasa de fotogramas por segundo igual o superior a 30. Por lo general, el contenido desarrollado con *Unity* puede ejecutarse bastante bien en todas partes [82]. Qué tan bien se ejecuta depende de la complejidad del proyecto, que en este caso al ser una demo técnica de un juego en 2D, no son demasiados. Unos requisitos mínimos para poder jugar más detallados son estos:

- **Sistema Operativo:** Windows XP+, Mac OS X 10.7+, Ubuntu 12.04+, SteamOS+
- **Procesador:** cualquier procesador compatible con el conjunto de instrucciones SSE2.
- **Tarjeta de vídeo:** capaz de mover DX9 con shaders modelo 2.0. Por lo general, cualquier modelo de tarjeta de vídeo fabricada desde 2004 debería funcionar.
- **Espacio en disco:** 100MB.
- **Periféricos de entrada:** Ratón y teclado.

Además, la tasa de cuadros por segundo debe mantenerse estable. Es preferible contar con una tasa media estable que con una alta que oscile en gran medida.

Requisitos del Sistema de Software

El sistema software deberá adherirse a los siguientes requisitos:

- Todo el código debe ser multiplataforma ofreciendo exactamente el mismo comportamiento en todos los sistemas operativos para los que se compile.
- Se hará uso del ratón y el teclado.
- El control deberá ser intuitivo, el jugador debería poder jugar a la demo técnica de **1812: La aventura** sin necesidad de haber acudido al manual de usuario. La interfaz también debe presentar un manejo lógico y sencillo.
- La demo técnica de **1812: La aventura** debe ser fácilmente ampliable con nuevos niveles y elementos, permitiendo así que se pueda convertir en un juego completo o en otra aventura gráfica completa.

3.4.2. Modelos de Casos de Uso

Los pasos para obtener el modelo de casos de uso han sido los siguientes:

1. Identificar a todos los posibles usuarios del sistema y sus roles.
2. Para cada rol, determinar todas las maneras posibles que éste cuenta para interactuar con el sistema.
3. Creación de un caso de uso por cada objetivo que deba cumplir el sistema.
4. Estructurar todos los casos de uso, por ejemplo, empleando relaciones de inclusión o extensión.

Diagrama de Casos de Uso

En la figura 3.16 se muestra el diagrama de casos de uso para la demo técnica de **1812: La aventura**.

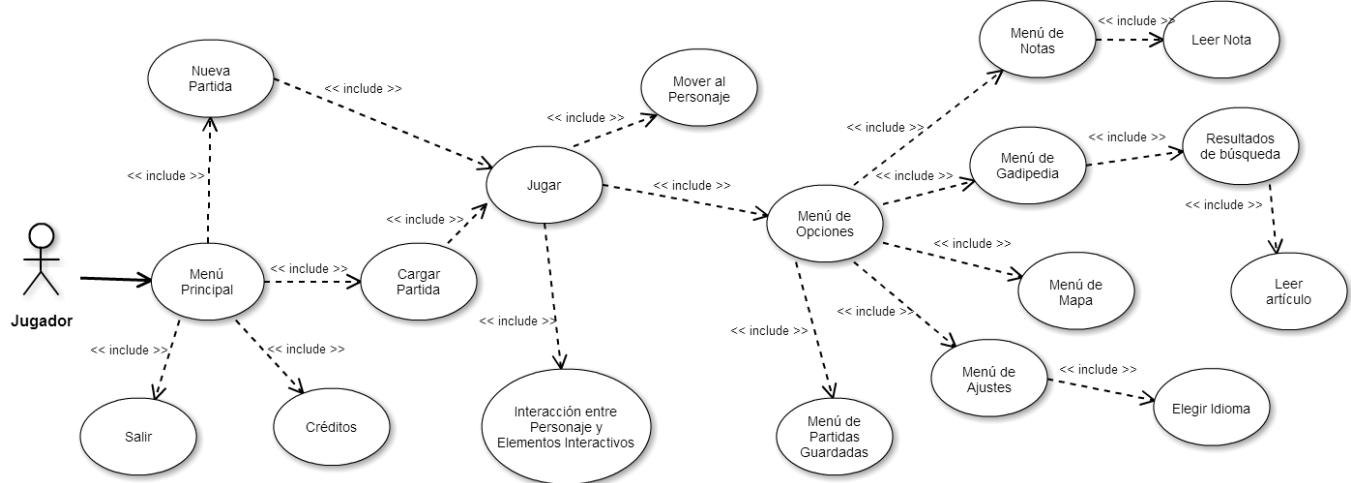


Figura 3.16: Diagrama de Casos de Uso

Descripción de los Casos de Uso

En esta sección adjuntaremos las descripciones de todos los casos de uso anteriormente expuestos. Para ello emplearemos una notación en texto utilizando un formato completo con plantilla. Se pretende que su lectura sea sencilla y resulte accesible a la vez que directo.

Caso de uso: Menú principal

Caso de uso Menú principal

Descripción Se le muestra al jugador el menú principal desde el cual es capaz de crear una partida nueva, acceder al menú de Cargar partida, acceder a los créditos o salir del juego.

Actores *Jugador*.

Precondiciones Ninguna.

Postcondiciones Ninguna.

Escenario principal

1. El *Jugador* inicia la aplicación.
2. El *Sistema* inicia el motor del juego y muestra el menú principal en la pantalla.
3. El *Jugador* inicia una Nueva Partida.
4. El *Sistema* carga una Nueva Partida.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

3a El *Jugador* selecciona la opción Cargar Partida.

1. El *Sistema* accede a la pantalla de Cargar Partida donde se muestran las partidas que el Jugador partida que hubiera guardado previamente.

3b El *Jugador* selecciona la opción Créditos.

1. El *Sistema* accede a la pantalla de Créditos.

3c El *Jugador* selecciona la opción Salir.

1. El *Sistema* libera los recursos y sale de la aplicación.

Caso de uso: Créditos

Caso de uso Créditos

Descripción Se muestra la pantalla de créditos con todos los que participaron en la creación del juego.

Actores *Jugador*.

Precondiciones Ninguna.

Postcondiciones Ninguna.

Escenario principal

1. El *Sistema* muestra la pantalla de créditos.
2. El *Jugador* selecciona la opción de volver.
3. El *Sistema* vuelve al menú principal.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

Caso de uso: Menú de Cargar Partida

Caso de uso Menú de Cargar Partida

Descripción El Sistema carga la pantalla de Cargar partida y el Jugador puede seleccionar una de entre las partidas disponibles para cargarla o borrarla. También puede seleccionar la opción de volver al Menú principal.

Actores *Jugador*.

Precondiciones Ninguna.

Postcondiciones Se carga o borra la partida seleccionada, de haber alguna que seleccionar.

Escenario principal

1. El *Sistema* carga la pantalla de Cargar Partida y las partidas guardadas que hubieran.

2. El *Jugador* selecciona una de las partidas guardadas y selecciona la opción de Cargar Partida.
3. El *Sistema* carga la partida seleccionada. Entonces el *Jugador* ya puede empezar a Jugar.

Extensiones – flujo alternativo

- ***a** El *Jugador* cierra la ventana.
 1. El *Sistema* libera los recursos y sale de la aplicación.
- 2a** El *Jugador* selecciona la opción de volver.
 1. El *Sistema* vuelve al Menú Principal.
- 2b** El *Jugador* selecciona una de las partidas guardadas y selecciona la opción de Borrar.
 1. El *Sistema* borra la partida seleccionada.

Caso de uso: Nueva Partida

Caso de uso Nueva Partida

Descripción El *Sistema* carga el juego en su estado inicial, en el cual el *Jugador* puede ver el inicio de la historia.

Actores *Jugador*.

Precondiciones Se ha seleccionado la opción Nueva Partida en el Menú Principal.

Postcondiciones El *Jugador* podrá empezar a Jugar.

Escenario principal

1. El *Sistema* carga el estado inicial del juego, incluyendo escenarios, música y sprites.
2. Durante los siguientes minutos, el *Sistema* muestra al *Jugador* la historia inicial del juego.
3. Una vez termina la presentación de la historia, el *Jugador* puede empezar a Jugar.

Extensiones – flujo alternativo

- ***a** El *Jugador* cierra la ventana.
 1. El *Sistema* libera los recursos y sale de la aplicación.
- 2a** El *Jugador* hace click en cualquier lugar de la pantalla.
 1. El *Sistema* se salta el diálogo que esté teniendo lugar en ese momento, lo cual acelera la presentación de la historia inicial del juego.

Caso de uso: Jugar

Caso de uso Jugar

Descripción El *Jugador* comienza a jugar, ya sea una nueva partida o una que hubiese guardado anteriormente.

Actores *Jugador*.

Precondiciones Se ha elegido Nueva Partida o Cargar Partida.

Postcondiciones Ninguna.

Escenario principal

1. El *Sistema* carga el nivel, escenario, música y sprites (suponiendo que no los hubiera cargado ya en Nueva Partida).
2. El *Jugador* y el *Sistema* interactúan durante la partida.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

2a El *Jugador* hace click en algún lugar del escenario en el que no haya un elemento interactivo.

1. El *Sistema* traza la ruta más corta entre el personaje principal y el lugar accesible donde el *Jugador* ha hecho click y mueve al personaje principal a ese punto (si procede).

2b El *Jugador* hace click en el botón de Menú de Opciones.

1. El *Sistema* accede a la pantalla de Menú de Opciones.

2c El *Jugador* hace click en algún lugar del escenario en el que hay un elemento interactivo.

1. El *Sistema* traza la ruta más corta entre el personaje principal y el lugar donde el *Jugador* ha hecho click, mueve al personaje principal a ese punto (si procede) y entonces el personaje principal interactúa con el elemento interactivo.

Caso de uso: Mover al personaje

Caso de uso Mover al personaje

Descripción El personaje principal se mueve en la dirección trazada por el Sistema para acabar lo más cerca posible del punto que seleccionó el *Jugador*.

Actores *Jugador*.

Precondiciones Estar jugando y que el *Jugador* haya hecho click en algún lugar accesible del escenario en el que no hubieran elementos interactivos.

Postcondiciones El personaje principal estará en una posición diferente en la que estaba antes de que el *Jugador* hiciera click.

Escenario principal

1. El *Sistema* traza la ruta más corta entre el personaje principal y el lugar donde el *Jugador* ha hecho click.
2. El *Sistema* mueve al personaje principal al punto más cercano utilizando la ruta calculada, si procede (es posible que no haga falta mover al personaje).
3. Tras terminar de moverse, el Juego continua.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

2a El *Jugador* hace click en otro lugar del escenario accesible o en otro elemento interactivo.

1. El *Sistema* comienza a Mover al personaje como si se tratara de una orden completamente nueva (anulando la anterior) desde la posición en la que se encuentre el jugador principal en el momento en que se hizo click.

2b El *Jugador* hace click en el botón de Menú de Opciones.

1. El *Sistema* accede a la pantalla de Menú de Opciones.

Caso de uso: Interacción entre personajes y elementos interactivos

Caso de uso Interacción entre personajes y elementos interactivos

Descripción El personaje principal interactúa con el personaje o elemento interactivo en el que el *Jugador* haya hecho click.

Actores *Jugador*.

Precondiciones Estar jugando y que el *Jugador* haya hecho click en algún personaje o elemento interactivo del escenario.

Postcondiciones Tras la interacción, es posible que el escenario o las opciones disponibles cambien, lo cual puede implicar una carga de datos y/o elementos por parte del *Sistema*.

Escenario principal

1. El *Sistema* lleva la interacción a cabo.
2. Tras la interacción, el *Sistema* carga (si debe) datos y/o elementos.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

Caso de uso: Menú de Opciones

Caso de uso Menú de Opciones

Descripción El *Sistema* carga la pantalla de Menú de Opciones y el *Jugador* puede seleccionar de entre las múltiples opciones disponibles. También puede seleccionar la opción de volver al Juego.

Actores *Jugador*.

Precondiciones Estar jugando y que el *Jugador* haya hecho click en el botón de Menú de Opciones.

Postcondiciones Dependiendo de la opción elegida, el jugador puede abandonar el juego, jugar otra partida diferente o acceder a otro menú.

Escenario principal

1. El *Sistema* carga la pantalla de Menú de Opciones.
2. El *Jugador* selecciona una de las opciones disponibles, como por ejemplo la opción de Salir.
3. El *Sistema* libera los recursos y sale de la aplicación.

Extensiones – flujo alternativo

- ***a** El *Jugador* cierra la ventana.
 - 1. El *Sistema* libera los recursos y sale de la aplicación.
- 2a** El *Jugador* selecciona la opción de Menú de Notas.
 - 1. El *Sistema* carga la pantalla del Menú de Notas.
- 2b** El *Jugador* selecciona la opción de Menú de Gadipedia.
 - 1. El *Sistema* carga la pantalla del Menú de Gadipedia.
- 2c** El *Jugador* selecciona la opción de Menú de Mapa.
 - 1. El *Sistema* carga la pantalla del Menú de Mapa.
- 2d** El *Jugador* selecciona la opción de Menú de Ajustes.
 - 1. El *Sistema* carga la pantalla del Menú de Ajustes.
- 2e** El *Jugador* selecciona la opción de Menú de Partidas Guardadas.
 - 1. El *Sistema* carga la pantalla del Menú de Partidas Guardadas.
- 2f** El *Jugador* selecciona la opción de volver al Juego.
 - 1. El *Sistema* carga el Juego tal y como estaba cuando el *Jugador* hizo click en el botón de Menú de Opciones dentro del Juego.

Caso de uso: Menú de Notas

Caso de uso Menú de Notas

Descripción El *Sistema* carga la pantalla del Menú de Notas y muestra las notas existentes.

Actores *Jugador*.

Precondiciones Se tiene que haber elegido la opción Menú de Notas en el Menú de Opciones.

Postcondiciones Ninguna.

Escenario principal

- 1. El *Sistema* carga la pantalla del Menú de Notas y muestra las notas existentes.
- 2. El *Jugador* elige una de las notas.
- 3. El *Sistema* carga la pantalla de Leer Nota y muestra la nota elegida.

Extensiones – flujo alternativo

- ***a** El *Jugador* cierra la ventana.
 - 1. El *Sistema* libera los recursos y sale de la aplicación.
- 2a** El *Jugador* selecciona la opción de volver al Menú de Opciones.
 - 1. El *Sistema* carga la pantalla de Menú de Opciones.

Caso de uso: Leer Nota

Caso de uso Leer Nota

Descripción Pantalla que muestra el texto de la nota seleccionada.

Actores *Jugador.*

Precondiciones Haber elegido una nota del Menú de Notas.

Postcondiciones Ninguna.

Escenario principal

1. El *Sistema* carga la pantalla del Leer Nota y muestra la nota seleccionada.
2. El *Jugador* hace click en la nota.
3. El *Sistema* carga la pantalla de Menú de Notas y muestra el listado de notas.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

2a El *Jugador* selecciona la opción de volver al Menú de Opciones.

1. El *Sistema* carga la pantalla de Menú de Opciones.

Caso de uso: Menú de Gadipedia

Caso de uso Menú de Gadipedia

Descripción Menú en el que el *Jugador* puede buscar artículos de información escribiendo palabras clave.

Actores *Jugador.*

Precondiciones Se tiene que haber elegido la opción Menú de Gadipedia en el Menú de Opciones.

Postcondiciones Ninguna.

Escenario principal

1. El *Sistema* carga la pantalla del Menú de Gadipedia.
2. El *Jugador* escribe una palabra clave y hace click en el botón de Buscar.
3. El *Sistema* carga la pantalla de Resultados de búsqueda.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

2a El *Jugador* selecciona la opción de volver al Menú de Opciones.

1. El *Sistema* carga la pantalla de Menú de Opciones.

Caso de uso: Resultados de búsqueda

Caso de uso Resultados de búsqueda

Descripción Pantalla que muestra los resultados de la búsqueda seleccionada (de haberlos).

Actores *Jugador.*

Precondiciones Haber elegido buscar una palabra en el Menú de Gadipedia.

Postcondiciones Ninguna.

Escenario principal

1. El *Sistema* carga la pantalla de Resultados de búsqueda, con los resultados de la búsqueda.
2. El *Jugador* elige cual de los resultados quiere leer en detalle.
3. El *Sistema* carga la pantalla Leer Artículo.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

2a El *Jugador* selecciona la opción de volver al Menú de Opciones.

1. El *Sistema* carga la pantalla de Menú de Opciones.

2b El *Sistema* no carga ningún resultado para la palabra escogida.

1. El *Jugador* selecciona el botón Volver atrás.

2. El *Sistema* carga la pantalla de Menú de Gadipedia.

2c El *Sistema* no carga ningún resultado para la palabra escogida.

1. El *Jugador* selecciona la opción de Volver al menú.

2. El *Sistema* carga la pantalla de Menú de Opciones.

Caso de uso: Leer Artículo

Caso de uso Leer Artículo

Descripción Pantalla que muestra el texto del artículo seleccionado.

Actores *Jugador.*

Precondiciones Haber elegido un resultado desde Resultados de Búsqueda.

Postcondiciones Ninguna.

Escenario principal

1. El *Sistema* carga la pantalla del Leer Artículo y muestra el artículo seleccionado.
2. El *Jugador* hace click en el botón Volver atrás.
3. El *Sistema* carga la pantalla de Resultados de Búsqueda y muestra el listado de resultados obtenidos de antes de cargar el artículo seleccionado.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

2a El *Jugador* selecciona la opción de volver al Menú de Opciones.

1. El *Sistema* carga la pantalla de Menú de Opciones.

Caso de uso: Menú de Mapa

Caso de uso Menú de Mapa

Descripción Menú en el que el *Jugador* puede elegir a qué escenario viajar.

Actores *Jugador*.

Precondiciones Se tiene que haber elegido la opción de Menú de Mapa en el Menú de Opciones.

Postcondiciones Ninguna.

Escenario principal

1. El *Sistema* carga la pantalla del Menú de Mapa.
2. El *Jugador* elige el destino al que quiere viajar.
3. El *Sistema* carga el nuevo nivel, escenario, música y sprites.
4. El *Sistema* deja al *Jugador* Jugar.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

2a El *Jugador* selecciona la opción de volver al Menú de Opciones.

1. El *Sistema* carga la pantalla de Menú de Opciones.

Caso de uso: Menú de Ajustes

Caso de uso Menú de Ajustes

Descripción Menú en el que el *Jugador* puede cambiar la configuración del juego.

Actores *Jugador*.

Precondiciones Se tiene que haber elegido la opción de Menú de Ajustes en el Menú de Opciones.

Postcondiciones Ninguna.

Escenario principal

1. El *Sistema* carga la pantalla del Menú de Ajustes.
2. El *Jugador* elige cambiar la resolución del juego a otra.
3. El *Sistema* cambia la resolución del juego.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

2a El *Jugador* selecciona la opción de volver al Menú de Opciones.

1. El *Sistema* carga la pantalla de Menú de Opciones.

2b El *Jugador* selecciona la opción de Pantalla Completa.

1. El *Sistema* carga el juego a pantalla completa o a modo ventana, dependiendo de como estuviera.

2c El *Jugador* modifica el volumen de la Música.

1. El *Sistema* cambia el volumen de la música acorde a lo que el jugador ha configurado.

2d El *Jugador* modifica el volumen de los Efectos de Sonido.

1. El *Sistema* cambia el volumen de los efectos de sonido acorde a lo que el jugador ha configurado.

2a El *Jugador* pulsa el botón de Cambiar lenguaje.

1. El *Sistema* carga la pantalla de Elegir Idioma.

Caso de uso: Elegir Idioma

Caso de uso Elegir Idioma

Descripción Pantalla en el que el *Jugador* puede elegir el idioma con el que quiera jugar en el Juego.

Actores *Jugador*.

Precondiciones Se tiene que haber elegido la opción de Cambiar Idioma en el Menú de Ajustes.

Postcondiciones Ninguna.

Escenario principal

1. El *Sistema* carga la pantalla del Elegir Idioma.
2. El *Jugador* elige el idioma que desea.
3. El *Sistema* cambia el idioma de todos los textos al que el jugador ha seleccionado.

Extensiones – flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

2a El *Jugador* selecciona la opción de volver al Menú de Opciones.

1. El *Sistema* carga la pantalla de Menú de Opciones.

Caso de uso: Menú de Partidas Guardadas

Caso de uso Menú de Partidas Guardadas.

Descripción El *Sistema* carga la pantalla de Menú de Partidas Guardadas y el *Jugador* puede seleccionar una de las partidas para guardar, cargar o borrarla.

Actores *Jugador*.

Precondiciones Se tiene que haber elegido la opción de Menú de Partidas Guardadas en el Menú de Opciones.

Postcondiciones Se carga, guarda o borra la partida seleccionada, de haber alguna que seleccionar.

Escenario principal

1. El *Sistema* carga la pantalla de Menú de Partidas Guardadas y las partidas guardadas que hubieran.
2. El *Jugador* selecciona una de las partidas guardadas (o uno de los huecos disponibles) y selecciona la opción de Guardar Partida.
3. El *Sistema* guarda el estado actual en la partida (o hueco) seleccionada.

Extensiones – flujo alternativo

- ***a** El *Jugador* cierra la ventana.
 1. El *Sistema* libera los recursos y sale de la aplicación.
- 2a** El *Jugador* selecciona la opción de volver al Menú de Opciones.
 1. El *Sistema* carga la pantalla de Menú de Opciones.
- 2b** El *Jugador* selecciona una de las partidas guardadas y selecciona la opción de Borrar.
 1. El *Sistema* borra la partida seleccionada.
- 2c** El *Jugador* selecciona una de las partidas guardadas y selecciona la opción de Cargar.
 1. El *Sistema* carga la partida seleccionada. Entonces el *Jugador* ya puede empezar a Jugar.

3.5. Diseño

A diferencia de otros productos software que son desarrollados solamente mediante librerías y código (por lo tanto la orientación de su diseño es a clases), en *Unity* su desarrollo es orientado a objetos y componentes, siguiendo la siguiente estructura:

UnityEngine.Object → UnityEngine.ScriptableObject → Component → Behaviour → MonoBehaviour

A partir de esta estructura, el resto de objetos se extienden de ellas. Todos los objetos de la jerarquía de *Unity* son **GameObjects** (es decir, derivan de `UnityEngine.Object`, ya sean Components o MonoBehaviours), y estos **GameObjects** pueden a su vez tener muchos componentes (y cada componente está asociado a un único **GameObject**). Por ejemplo, las Clases casi siempre derivan de `MonoBehaviour`, y todas las funciones y propiedades que aparecen en el `Inspector` son o derivan de `Component`.

Por lo tanto, aunque quisieramos hacer un diagrama de clases UML con las clases contenidas en el código de los *scripts* contenidos en los objetos, sería imposible ver todas las dependencias y componentes a los que puede acceder un *script*, por no mencionar que acabaría siendo más confuso que útil. Por lo tanto hacer un diagrama de clases en UML que pueda mostrarnos todo el diseño del proyecto resultaría impracticable.

No obstante, por poner un par de ejemplos para demostrar estas palabras, en primer lugar he hecho un diagrama de clases UML de las clases de *Unity* que más relevancia han tenido para este proyecto.

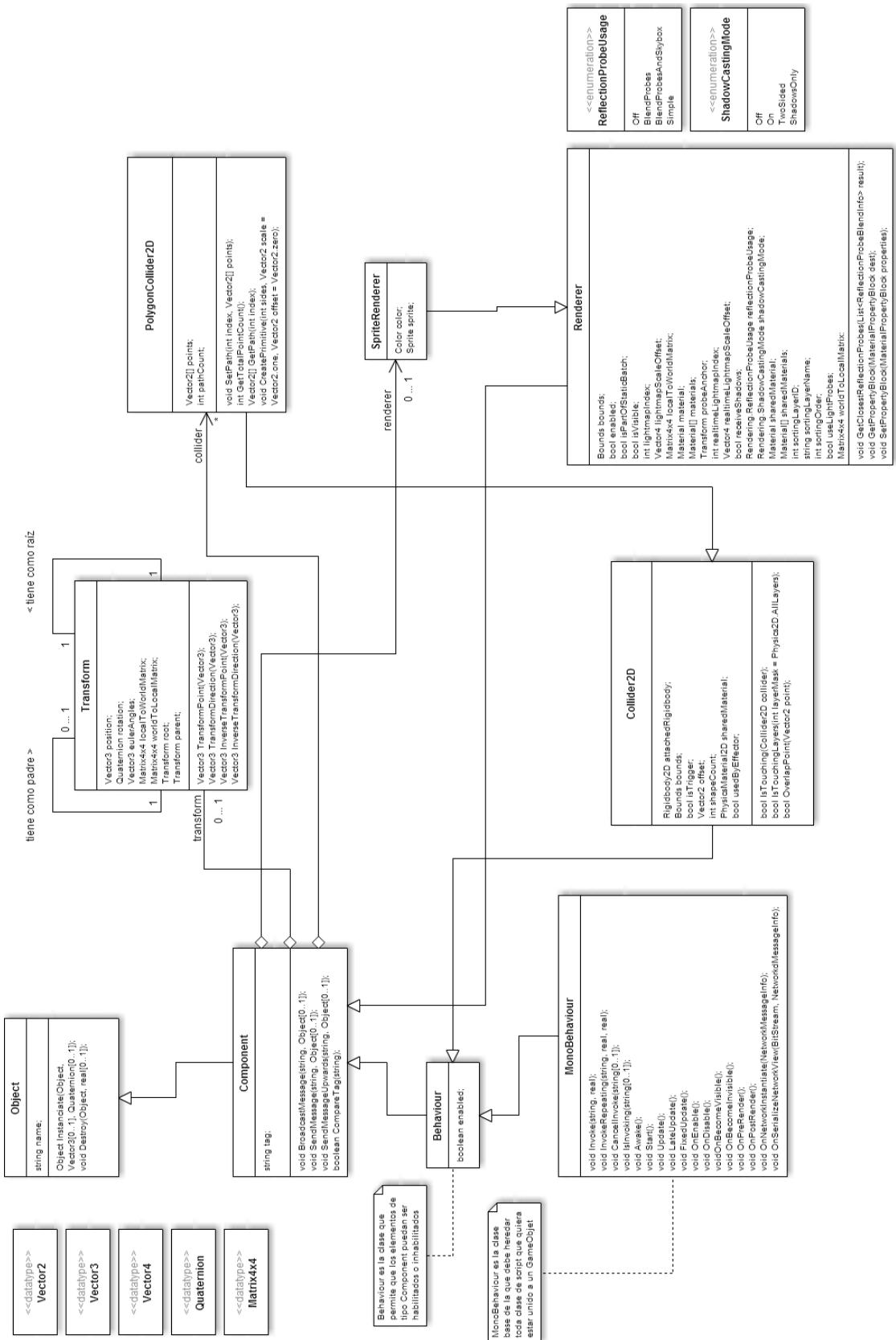


Figura 3.17: Diagrama de clases UML de la pequeña parte del motor de *Unity* que vamos a usar

No se han mostrado todas las clases ni tipos de datos que conforman *Unity* porque son inabarcables en un solo diagrama y pocas de ellas son utilizadas en el proyecto.

Y por segundo ejemplo, he hecho un diagrama de clases UML de una parte del proyecto, el sistema de objetos interactivos e inventario. He elegido esta parte porque es el conjunto de *scripts* donde mejor se ven las relaciones que hay entre ellos de entre todos los del proyecto.



Figura 3.18: Diagrama de clases UML del sistema de objetos interactivos de **Motor de Aventuras Gráficas 2D de Unity**

A pesar de que haya relaciones claramente definidas entre estos *scripts*, hay muchas más que no aparecen en el diagrama por el mero hecho de que tendríamos entonces que hacer otros diagramas con respecto la jerarquía de objetos y sus componentes, pues varios de los *scripts* citados en el último diagrama UML dependen de varios objetos a la vez de los que no hay constancia en estos diagramas.

3.6. Implementación

A lo largo de toda la fase de implementación se han ido encontrando diversos obstáculos en distintos subsistemas. Estos han surgido bien por desconocimiento de la materia o por la dificultad que entraña la misma. En cualquier caso, en este capítulo haremos un repaso por los detalles más interesantes de la implementación del motor de aventuras gráficas 2D para *Unity* que luego se probará en la demo técnica de **1812: La aventura**. En cada uno de ellos se expondrá el problema a resolver, las dificultades encontradas y la solución propuesta adjuntando si es necesario pequeños fragmentos de código. Para consultar el código fuente completo del juego, lo mejor es acudir al repositorio Git [56] de la forja de GitHub [35] en la siguiente dirección.

<https://github.com/Firenz/1812/tree/demo>

La documentación del código generada con Doxygen [26] facilitará en gran medida la lectura del código. Dicha documentación complementaria puede ser accedida desde la siguiente dirección web.

<https://github.com/Firenz/1812/tree/demo/documentacion/doxygen>

3.6.1. Capas jerarquizadas para el orden de dibujado de los gráficos

Para que puedan dibujarse los gráficos (o *sprites* de ahora en adelante) de los objetos de un videojuego en *Unity*, este ofrece el componente *SpriteRenderer* al que le podemos poner en el parámetro *Sprite* el gráfico que deseemos para que lo dibuje en el nivel. En un principio con esto sería más que suficiente, pero el problema viene cuando múltiples *sprites* se solapan en un mismo punto del escenario.

Dado que con *Unity* en el modo 2D no sirve variar la coordenada *z* (la de profundidad) en el componente *Transform* para cambiar el orden de dibujado, ni podemos acceder a los métodos internos de dibujado del motor de *Unity* para cambiarlos a nuestro antojo. La solución es usar las *Sorting Layer*, dentro de los componentes *SpriteRenderer*, para indicar a *Unity* en cuál capa se va a dibujar el *sprite* conjuntamente con *Sorting Order*, que nos indica el orden de dibujado dentro de la *Sorting Layer* en la que se va a dibujar el *sprite*.

En el caso de la demo técnica de **1812: La aventura**, estas fueron las *Sorting Layer* creadas para organizar el dibujado de los *sprites*:

BackgroundScenery Esta es la capa más baja en la jerarquía de dibujado por pantalla, en ella pondremos los *sprites* de los fondos de los niveles.

BackgroundInteractiveObjects Esta capa se dibuja justo encima de **BackgroundScenery**, en ella colocaremos los objetos interactivos que se vayan a dibujar detrás del personaje

principal, osease el objeto **Player**, y los personajes no controlables.

Actors Esta capa se dibuja justo encima de `BackgroundInteractiveObjects`, en ella colocaremos los personajes no controlables y a **Player**.

FrontlineScenery Esta capa se dibuja justo encima de `Actors`, en ella colocaremos tanto los objetos interactivos, personajes no controlables y gráficos de escenario, que vayan a verse por encima de todos los demás *sprites* de los niveles.

GUI Esta capa se dibuja justo encima de `FrontlineScenery`, en ella colocaremos solamente los *sprites* de los elementos de la interfaz del juego: los distintos cursores del ratón, el inventario y los objetos del inventario.

Con estas capas tendremos controlado más que suficientemente el orden de dibujado de los gráficos, pero si se diera el caso de que aún en la misma `Sorting Layer` se pudieran solapar varios *sprites*, también tenemos el parámetro `Sorting Order` de `SpriteRenderer`. Cuando mayor sea el número que le pongamos al parámetro, más alto en la jerarquía de dibujado estará (se verá por delante de los demás gráficos).

Como mención aparte, hay que decir que todo gráfico dibujado en *Unity* mediante el sistema UI (para el dibujado de interfaces gráficas), siempre se dibujarán encima de cualquier `SpriteRenderer` tenga la `Sorting Layer` que tenga.

Aún así, puede darse el caso de que necesitamos que el *sprite* de algún objeto, dependiendo de la posición en el eje Y de **Player**, se dibuje por atrás o por delante de este. Para controlar esto, lo que hacemos es añadirle al objeto con el componente `SpriteRenderer` en cuestión, el script `SortingOrderLayerController.cs`.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class SortingOrderLayerController : MonoBehaviour {
5      public float changeAtPlayerPositionY;
6      public string sortingLayerNameAfterChange;
7      public int sortingLayerPositionAfterChange;
8
9      private int previousSortingPosition;
10     private string previousSortingLayerName;
11
12     void Start () {
13         previousSortingLayerName = this.gameObject.GetComponent<
14             Renderer>().sortingLayerName;
15         previousSortingPosition = this.gameObject.GetComponent<
16             Renderer>().sortingOrder;
17     }
18
19     void Update () {
20         if(Player.Instance.transform.position.y >
21             changeAtPlayerPositionY){
22             this.gameObject.GetComponent<Renderer>().
23                 sortingLayerName = sortingLayerNameAfterChange;
24             this.gameObject.GetComponent<Renderer>().
25                 sortingOrder = sortingLayerPositionAfterChange;
```

```

21     }
22     else{
23         this.gameObject.GetComponent<Renderer>().
24             sortingLayerName = previousSortingLayerName;
25         this.gameObject.GetComponent<Renderer>().
26             sortingOrder = previousSortingPosition;
27     }
}

```

3.6.2. Gestión de estados de juego

Una de las características principales de *Unity* es que los juegos realizados con ellos se generan mediante escenas en su editor. Para cambiar de escenas, solo es necesario usar el método `Application.LoadLevel(string nameScene)` y no es necesario gestionar ninguna pila de estados de las escenas del juego en todo momento, pues *Unity* lo hace todo solo internamente.

De lo único que no se encarga es de gestionar el guardado de datos al cambiar de escenas, que si bien en videojuegos que basan su jugabilidad en niveles autoconclusivos, en el género de las aventuras gráficas es al revés. Todos los niveles suelen estar interconectados, por lo tanto hay que guardar el estado de los objetos interactivos de las escenas, los objetos que hay en el inventario, la posición del personaje principal, y los eventos que han ocurrido.

La solución que encontré para solventar el guardado de datos entre escenas fue crear un *script* con una clase estática llamada `GameState.cs`, al ser estática y no depender de `MonoBehaviour`, sus datos no son borrados al salir de los niveles y siempre permanecen en memoria. Luego para los objetos interactivos y/o NPCs que necesitasen guardar su estado entre niveles, lo que hice fue crear *scripts* que heredasen de las clases `InteractiveElement.cs`, `PickableElement.cs`, `WarperElement` o `Actor.cs` que en sus métodos sobrecargados de inicialización, usaran los atributos de `GameState.cs` para verificar el estado en el que tienen que estar.

Por poner un ejemplo práctico, aquí tenemos un fragmento del objeto interactivo **Window** con el *script* `Windows.cs`:

```

1  using UnityEngine;
2  using System;
3  using System.Collections;
4
5  public sealed class Window : PickableElement {
6
7      //.....
8
9      //Método que se ejecuta al inicializarse el objeto interactivo
10     protected override void InitializePickableInformation() {
11         windAnimator = this.transform.FindChild("Wind").
12             GetComponent<Animator>();
13         isDestroyedAfterBeingPicked = false;
14
15         //Al iniciarse el objeto cuando se entra en el escenario,

```

```

    comprueba si la ventana se había dejado abierta o
    cerrada la última vez que se estuvo en el nivel del
    despacho del profesor
15   if(GameState.LevelProfessorOfficeData.isWindowOpened) {
16       this.Open();
17   }
18   else{
19       this.Close();
20   }
21 }
22
23 //.....
24
25 private void Close(){
26     //Luego cuando la ventana se cierra, se actualiza el
27     //atributo de GameState
28     GameState.LevelProfessorOfficeData.isWindowOpened = false;
29
30     //....
31 }
32
33 private void Open(){
34     //Lo mismo para cuando se cierra
35     GameState.LevelProfessorOfficeData.isWindowOpened = true;
36
37     //...
38 }
39
40 //.....
41 }
```

De esta manera mantenemos la gran mayoría de los datos de estado de los objetos aunque se cambien de nivel y se borren. Sin embargo, a pesar del guardado de los datos, en algunos casos necesitaremos que algunos objetos no se destruyan al cambiar de nivel, como es en el caso del objeto **Inventario** y los objetos de inventario que contenga.

Para solventar estos casos especiales, lo que hice fue crear un *script* de clase genérica llamada `PersistentSingleton.cs`. Tal como indica su nombre, lo que hace es que solo haya siempre una única instancia del objeto que tenga un *script* que herede de este, evitando duplicados de objetos únicos que puedan llevar al malfuncionamiento del juego. Y además de eso, durante la inicialización del objeto al que esté unido, le indica que no sea destruido al cambiarse de nivel mediante el método `MonoBehaviour.DontDestroyOnLoad(GameObject gameobject)`. A continuación explicamos el contenido del *script* `PersistentSingleton.cs`:

```

1 using UnityEngine;
2 using System.Collections;
3
4 //Clase abstracta y genérica
5 public abstract class PersistentSingleton<T> : MonoBehaviour
6 where T : Component{
7     //Atributo estático que contiene la instancia del script único
8     protected static T instance;
```

```

10     //Atributo público que se encarga de pasar en modo lectura el
11     //atributo instance protegido, evitando así que código externo
12     //pueda sobreescribirlo
13     public static T Instance{
14         get{
15             return instance;
16         }
17     }
18
19     //Si se destruye el objeto y la instancia que había inicializada en
20     //el atributo era de este objeto en concreto, indicamos que ya no
21     //hay ningún script instanciado mediante la asignación a null de
22     //la instancia
23     protected void OnDestroy() {
24         if (instance == this) {
25             instance = null;
26         }
27     }
28
29     //Si salimos de la aplicación, asignamos la instancia a null para
30     //que el colector de basura elimine la instancia
31     protected void OnApplicationQuit(){
32         instance = null;
33     }
34
35     //En el despertar del objeto, y de concretamente del script, se
36     //ejecuta este método
37     protected void Awake(){
38         //Comprobamos que ya no había un objeto de este tipo ya
39         //instanciado
40         if(instance == null){
41             //Si no había ninguno instanciado, asignado al
42             //atributo instance este objeto como el único
43             instance = this as T;
44             //Indicamos que el objeto que contiene el script
45             //instanciado, no se destruya al cambiar de nivel
46             DontDestroyOnLoad(instance.gameObject);
47             InitializeOnAwake();
48         }
49         //Si ya había un objeto instanciado, destruimos el nuevo
50         else{
51             Destroy(this.gameObject);
52         }
53     }
54
55     //Los scripts que hereden de este tendrán que sobrecargar este mé
56     //todo y usarlo para inicializar sus correspondientes atributos y
57     //métodos
58     protected abstract void InitializeOnAwake();
59 }
60 contenidos...

```

3.6.3. Sistema de localización de textos

Uno de los objetivos de **Motor de Aventuras Gráficas 2D de Unity** era crear un sistema de localización de textos desde 0, independiente del código del proyecto, que fuera sencillo para que personas sin apenas conocimientos de informática pudieran coger los ficheros de los textos y pudieran traducir la demo técnica de **1812: La aventura** a partir de ellos, pero que a la vez no fuera un sistema demasiado complicado de programar debido a mi inexperiencia al trabajar con ficheros. Al final el método que se me ocurrió fue el de usar ficheros .xml debido a:

- Se pueden crear grupos de etiquetas con nombres y atributos personalizados.
- Hay multitud de librerías que facilitan la búsqueda de información dentro del formato XML mediante el uso de los nodos. Entre ellas librerías nativas de .NET, con lo que no era necesario usar librerías de terceros.
- Gracias al uso de los nodos para buscar nodos hijos o el texto contenido entre etiquetas, podía modificar el texto contenido sin tener que modificar la ruta de búsqueda del nodo dentro del código del proyecto.
- El formato XML me era el único que me resultaba familiar gracias a la asignatura *Programación en Internet*.

Una vez decidida a usar el formato XML para la localización de textos, implementé una plantilla con la nomenclatura y la profundidad que habría como máximo, para que así todos los métodos del juego que tuviesen texto localizado, partiesen de la misma base. Al final la nomenclatura elegida fue que el ID de cada lenguaje fuese el nombre de la carpeta que contuviese el archivo .xml con las traducciones, dicho fichero tendría siempre de nombre LocatedTexts.xml y tendría siempre tres tipos de etiquetas anidadas de forma jerárquica. Las etiquetas decididas fueron:

group La etiqueta más alta en la jerarquía, en ella se engloban las etiquetas de los elementos de un mismo escenario o conjunto (como el de las interfaces gráficas o personajes no controlables). A cada etiqueta se le pone un atributo id con el ID del grupo en particular.

element Esta etiqueta se corresponde con un elemento en concreto del juego. A cada etiqueta se le pone un atributo id con el ID del objeto en particular.

string Esta etiqueta se corresponde a cada texto diferente que puede contener un elemento concreto, de un grupo concreto, dentro del juego. A cada etiqueta se le pone un atributo id con el ID del texto en particular.

Una vez pensado el sistema, realicé una clase estática que fuera una colección de métodos del manejo de ficheros en formatos .xml llamado XMLFileReader.cs. Después, los métodos de esta clase fueron usados por el sistema de localización de textos que implementé en otra clase estática llamada LocatedTextManager.cs. El por qué no quise juntar las dos clases en una, fue por mantener la independencia entre ellas y porque más tarde XMLFileReader.cs sería usado también para crear los ficheros .xml para las partidas guardadas.

Las dos funcionalidades más importante que posee la clase LocatedTextManager son:

- Mediante el paso de un ID de grupo, otro de elemento y otro de string, obtiene una List<string> que contiene los textos localizados de un archivo LocatedTexts.xml.

```
1 public static List<string> GetLocatedText(string groupId, string
2   elementId, string textId) {
3     //Asignamos a una cadena la ruta de nodos en el archivo XML
4     string _xmlPathToNode = "/locatedtexts/group[@id='"
5       + groupId + "']/element[@id='"
6       + elementId + "']/string[@id='"
7       + textId + "']";
8     //Llama a un método de XMLFileReader que devuelve el texto
9     //contenido en la ruta de nodos que le vamos a pasar como par
10    ámetro del documento XML que esté abierto en memoria
11    string _rawText = XMLFileReader.GetNodeInfo(locatedTextDoc,
12      _xmlPathToNode);
13
14    //Formateamos la cadena de texto recibida de forma que le
15    //quitamos tabulaciones, retornos de carro, y espacios y
16    //saltos de línea que haya al inicio y al final
17    _rawText = _rawText.Replace("\t", "");
18    _rawText = _rawText.Replace("\r", "");
19    _rawText = _rawText.TrimStart(' ', '\n');
20    _rawText = _rawText.TrimEnd(' ', '\n');
21    //A partir de cada salto de línea que haya en el interior de
22    //la cadena de texto, la dividimos en subcadenas
23    List<string> _locatedText = _rawText.Split('\n').Select(p =>
24      p
25        .Trim()).ToList();
26
27    return _locatedText;
28  }
29 contenidos...
```

- Mediante el paso de un ID de grupo, otro de elemento y otro de string, obtiene una List<string> que contiene los textos localizados de un archivo LocatedTexts.xml. Pero esta vez formateada de manera especial para los diálogos de los personajes de forma que entren en una caja de texto con un número máximo de caracteres por línea y de líneas por caja.

```
1 //Al método le pasamos como parámetros el id del grupo,
2 //elemento y cadena del que queremos obtener el texto
3 //localizado, y aparte, también el máximo de carácteres y de
4 //líneas por cuadro
5
6 public static List<string> GetLocatedTextFormatted(string groupID,
7     string elementID, string textID, int maxCharsPerLine = 0, int
8     maxDisplayingLines = 0){
9     //Obtenemos el texto
10    List<string> _locatedText = GetLocatedText(groupID, elementID,
11        textID);
12    //Inicializamos las variables a usar durante el método
13    List<string> _resultingText = new List<string>();
14    string _resultingString = "";
15
16    //Si resulta que no se ha pasado ningún parámetro para
17    //formatear los textos, se devuelve el texto localizado tal y
18    //como está
19
20    if(maxCharsPerLine == 0 && maxDisplayingLines == 0) {
```

```

11         _resultingText = _locatedText;
12         return _resultingText;
13     }
14
15     //Si no es así, por cada cadena de texto obtenida la dividimos
16     //en palabras que luego juntamos debidamente para que entren
17     //en las cajas de los diálogos
18     for(int i = 0; i < _locatedText.Count; i++){
19         //Obtenemos la cadena siguiente a formatear en uno o
20         //varios cuadros
21         string _currentLine = _locatedText[i];
22         //Formateamos la cadena
23         _currentLine = _currentLine.Trim(' ', '\r','\t', '\n')
24         ;
25         //La dividimos en palabras
26         List<string> _wordsInLine = _currentLine.Split(' ').
27             ToList();
28         //Inicializamos las variables que van a llevar la
29         //cuenta del tamaño del cuadro de texto actual
30         int _currentWordsLength = 0;
31         int _nextWordLength = 0;
32         int _current.NewLine = 0;
33         int _lastIndexInLine = _wordsInLine.Count -1;
34         _resultingString = "";
35
36         //Por cada palabra que hay en cada cadena de texto
37         for(int j = 0; j < _wordsInLine.Count; j++){
38             //Obtenemos la siguiente palabra a formatear
39             string _nextWord = _wordsInLine[j];
40             //Obtenemos la longitud de la palabra a
41             //formatear
42             _nextWordLength = _nextWord.Length;
43
44             //Comprobamos que la línea del cuadro actual
45             //no haya llegado al máximo número de líneas
46             //a mostrar
47             if(_current.NewLine < maxDisplayingLines){
48                 //si no se ha superado el máximo nú
49                 //mero de líneas a mostrar en la lí
50                 //nea actual, comprobamos que la
51                 //palabra actual a introducir no
52                 //superá el límite de caracteres por
53                 //línea del cuadro actual
54                 if((_currentWordsLength +
55                     _nextWordLength) <= maxCharsPerLine
56                 ){
57                     //Si no es la última palabra
58                     //de la cadena, añadimos un
59                     //espacio a la cadena del
60                     //cuadro actual y
61                     //actualizamos los caracteres
62                     //que hay en la línea actual
63                     if(j != _lastIndexInLine){
64                         _resultingString +=
65                             _nextWord + " ";
66                         _currentWordsLength +=
67                     }
68                 }
69             }
70         }
71     }
72
73     return _resultingString;
74 }
```



```

68         palabra, añadimos
69         un salto de línea,
70         añadimos la palabra
71         a la cadena del
72         cuadro actual,
73         también la añadimos
74         a la lista de
75         textos formateados
76         _resultingString += "\n"
77         + _nextWord;
78         _resultingText.Add(
79             _resultingString);
80         //Reseteamos el cuadro
81         //actual
82         _currentWordsLength =
83             0;
84         _currentNewLine = 0;
85         _resultingString = "";
86     }
87     else{
88         //Si se ha llegado al máximo de líneas
89         //por cuadro pero no se ha superado
90         //el máximo de caracteres de la ú
91         //ltima linea
92         if((_currentWordsLength +
93             _nextWordLength) <= maxCharsPerLine
94             ) {
95             if(j != _lastIndexInLine){
96                 //Si no es la última
97                 //palabra de la
98                 //cadena, añadimos un
99                 //espacio a la
100                //cadena del cuadro
101                //actual y
102                //actualizamos los
103                //caracteres que hay
104                //en la linea actual
105                _resultingString +=
106                    _nextWord + " ";
107                _currentWordsLength +=
108                    _nextWordLength +
109                    1;
110            }
111            else{
112                //En caso de que sea
113                //la última palabra a
114                //añadir, la añ
115                //adimos a la cadena
116                //del cuadro actual
117                _resultingString +=
118                    _nextWord;
119                //Añadimos el cuadro
120                //de texto actual a
121                //la lista de textos

```

```

89                               formateados
90                               _resultingText.Add(
91                               _resultingString);
92                               //Reseteamos el cuadro
93                               actual
94                               _currentNewLine = 0;
95                               _currentWordsLength =
96                               0;
97                               _resultingString = "";
98                           }
99
100                          }
101
102                         //Si se ha llegado al máximo
103                         //de líneas por cuadro y se
104                         //ha superado también el má
105                         //ximo de caracteres de la ú
106                         //ltima línea
107                         if(j != _lastIndexInLine){
108                             //Si no es la última
109                             //palabra de la
110                             //cadena, añadimos
111                             //primero la cadena
112                             //del cuadro actual a
113                             //la lista de
114                             //cadenas formateadas
115                             _resultingText.Add(
116                             _resultingString);
117                             //E iniciamos un nuevo
118                             //cuadro de texto a
119                             //partir de la
120                             //palabra actual
121                             _resultingString =
122                             _nextWord + " ";
123                             _currentWordsLength =
124                             _nextWordLength +
125                             1;
126                             _currentNewLine = 0;
127                         }
128                         else{
129                             //Si es la última
130                             //palabra de la
131                             //cadena, añadimos
132                             //primero la cadena
133                             //del cuadro actual a
134                             //la lista de
135                             //cadenas ya
136                             //formateadas
137                             _resultingText.Add(
138                             _resultingString);
139                             //Luego añadimos la
140                             //palabra actual
141                             //también a la lista
142                             //de cadenas
143                             //formateadas
144                             _resultingText.Add(
145                             _nextWord);

```

```

111                                     //Reseteamos el cuadro
112                                         actual
113                                         _resultingString = "";
114                                         _currentWordsLength =
115                                         0;
116                                         _currentNewLine = 0;
117                                         }
118                                         }
119
120                                         //Limpiamos el contenido de la lista que contiene las
121                                         //palabras de la cadena, así puede ser usado para la
122                                         //siguiente cadena
123                                         _wordsInLine.Clear();
124
125                                         //Devolvemos la lista de cadenas formateadas según el tamaño
126                                         //de los cuadros de diálogos dado
127                                         return _resultingText;
128
129

```

3.6.4. Objetos interactivos

Toda aventura gráfica que se precie, hace uso de los objetos interactivos del escenario para poder ir avanzando el argumento del videojuego mediante las pistas que ofrecen sus descripciones, o la manipulación de estos para resolver puzzles o que ocurran eventos en los niveles. Obviamente en **Motor de Aventuras Gráficas 2D de Unity** era necesaria su implementación, por lo que había que tener claro los tipos de objetos interactivos básicos que hay en una aventura gráfica para su posterior implementación:

Objeto descriptivo El objeto interactivo más básico que puede haber. No se puede manipular, sirve para ser inspeccionado y descubrir pistas sobre el entorno y/o puzzles, o para ambientar el nivel actual.

Objeto manipulable/cogible Objeto que al ser inspeccionado, es manipulado automáticamente, de esta manera, se puede cambiar el entorno o coger objetos para el inventario.

Objeto transportador Objeto que al ser inspeccionado, nos transporta automáticamente a otro nivel.

Teniendo claro la existencia de estos tres tipos de objetos interactivos, lo siguiente era saber que componentes iban a tener que tener los objetos interactivos para poder ser detectados. Este paso fue fácil: todos los objetos interactivos iba a necesitar tener el componente PolygonCollider2D para que pudiera ser detectado posteriormente por el cursor si pasaba por encima y se obtenía que el Tag del objeto interactivo era InteractivePoint.

Otro punto importante era que todos los objetos interactivos necesitaban un objeto hijo que se pudiera colocar fácilmente en la escena mediante el panel Scene y sus controles. Este objeto hijo se llamaría WalkingPoint y serviría para indicar la posición en la que el personaje principal tendría que dirigirse dentro del escenario, para poder interactuar con el objeto interactivo.

A partir de ahí, hice el *script* llamado `InteractiveElement.cs` para los objetos interactivos básicos (los descriptivos) pero de forma que pudiera ser luego fácilmente ampliado. Las funciones principales de `InteractiveElement.cs` son:

- Método que se ejecuta cuando el *Jugador* hace click derecho sobre el objeto interactivo. La función por defecto de este método es devolver una descripción del objeto. Puede ser sobrecargado por si se quiere cambiar o expandir el método por defecto.

```
1 //Método que se llama cuando el ratón hace click izquierdo encima de
2     un objeto interactivo
3 public virtual void LeftClickAction(){
4     //Si Player no está ejecutando ninguna acción, o se está
5         reproduciendo una escena cinematográfica, ejecuta la corutina
6         con las acciones a realizar de hacer click izquierdo
7     if(!Player.Instance.isDoingAction && !CutScenesManager.
8         IsPlaying()){
9         StartCoroutine(WaitForLeftClickAction());
10    }
11 }
12
13 //Corrutina con las acciones a ejecutar por Player y el objeto
14     interactivo al hacerle click izquierdo
15 protected virtual IEnumerator WaitForLeftClickAction(){
16     //Calculamos la distancia que hay entre el objeto interactivo
17         y Player, si la distancia es mayor que la distancia mínima,
18         hacemos que Player vaya a la posición del objeto
19     float _distanceBetweenActorAndInteractivePosition = Mathf.Abs(
20         Vector2.Distance(Player.Instance.currentPosition,
21             interactivePosition));
22     if (_distanceBetweenActorAndInteractivePosition >= minDistance)
23     {
24         Player.Instance.GoTo(interactivePosition);
25         do{
26             yield return null;
27             //Esperamos que Player termine de caminar
28             }while(Player.Instance.isWalking());
29     }
30
31     //Nos aseguramos que Player no haya cambiado de ruta al dejar
32         de caminar
33     if(Player.Instance.originalTargetedPosition ==
34         interactivePosition){
35         //Llamamos al evento para indicar que Player está
36             realizando una acción
37         BeginAction();
38
39         //Hacemos que Player mire hacia el objeto
40         Player.Instance.LookAtPosition(this.transform.position
41             );
42
43         //Hacemos que Player haga una inspección profunda (
44             hablada) del objeto interactivo
45         Player.Instance.Speak(groupID, nameID, "INTERACTION");
46         do{
```

```

32             yield return null;
33             //Esperamos que termine de hablar
34         }while(Player.Instance.isSpeaking);
35
36         //Mandamos el evento para indicar que Player ha
37         //finalizado de realizar la acción
38         EndAction();
39     }

```

- Método que se ejecuta cuando el *Jugador* hace click izquierdo sobre el objeto interactivo. La función por defecto es la de realizar una descripción más elaborada. Puede ser sobrecargado por si se quiere cambiar o expandir el método por defecto.

```

1  //Método que se llama cuando el ratón hace click derecho encima de un
2  //objeto interactivo
3  public virtual void RightClickAction(){
4      //Si Player no está ejecutando ninguna acción, o se está
5      //reproducido una escena cinematográfica, ejecuta la corutina
6      //con las acciones a realizar de hacer click derecho
7      if(!Player.Instance.isDoingAction && !CutScenesManager.
8          IsPlaying()){
9              StartCoroutine(WaitForRightClickAction());
10         }
11
12     //Corrutina con las acciones a ejecutar por Player y el objeto
13     //interactivo al hacerle click derecho
14     protected virtual IEnumerator WaitForRightClickAction(){
15         //Calculamos la distancia que hay entre el objeto interactivo
16         //y Player, si la distancia es mayor que la distancia mínima,
17         //hacemos que Player vaya a la posición del objeto
18         float _distanceBetweenActorAndInteractivePosition = Mathf.Abs(
19             Vector2.Distance(Player.Instance.currentPosition,
20                 interactivePosition));
21         if(_distanceBetweenActorAndInteractivePosition >= minDistance)
22         {
23             Player.Instance.GoTo(interactivePosition);
24
25             do{
26                 yield return null;
27                 //Esperamos que Player termine de caminar
28             }while(Player.Instance.isWalking());
29         }
30
31         //Nos aseguramos que Player no haya cambiado de ruta al dejar
32         //de caminar
33         if(Player.Instance.originalTargetedPosition ==
34             interactivePosition){
35             //Llamamos al evento para indicar que Player está
36             //realizando una acción
37             BeginAction();
38
39             //Hacemos que Player mire hacia el objeto

```

```

28         Player.Instance.LookAtPosition(this.transform.position
29             );
30
30         //Hacemos que Player diga la descripción del objeto
31         //interactivo
31         Player.Instance.Speak(groupID, nameID, "DESCRIPTION");
32         do{
33             yield return null;
34             //Esperamos que termine de hablar
35         }while(Player.Instance.isSpeaking);
36
37         //Mandamos el evento para indicar que Player ha
38         //finalizado de realizar la acción
38         EndAction();
39     }
40 }
```

- Método que define el gráfico del cursor cuando este le pasa por encima. También puede ser sobre-cargado para que se muestre otro gráfico del cursor.

```

1 public virtual void ChangeCursorOnMouseOver(){
2     CustomCursorController.Instance.
3         ChangeCursorOverInteractiveElement();
```

- Método vacío listo para sobrecargar, para que objetos del inventario puedan ser usados encima de los objetos interactivos (funcione o no).

```

1 public virtual void ActionOnItemInventoryUsed(GameObject itemInventory
2     ){
3         /* Como usarlo:
4         if(!Player.Instance.isDoingAction && !CutScenesManager.
5             IsPlaying()){
6                 switch(itemInventory.name){
7                     case "name-1":
8                         itemInventory.GetComponent<ItemInventory>().
9                             Unselect();
10                        //Acción para el objeto name-1....
11                        break;
12                        case "name-2":
13                            itemInventory.GetComponent<ItemInventory>().
14                                Unselect();
15                                //Acción para el objeto name-2....
16                                break;
17                                //...
18                }
19            }
20        */
21    }
```

Posteriormente, a partir de `InteractiveElement.cs`, se crearon *scripts* que ampliaban, mediante la sobre carga de los cuatro métodos mencionados, estas funcionalidades básicas para lograr los otros

dos tipos de objetos interactivos básicos `PickableElement.cs` y `WarperElement.cs`.

3.6.5. Detección y gestión de colisiones

En el género de las aventuras gráficas, las colisiones de los objetos tiene pocas utilidades debido a que no existe ningún tipo de físicas en este tipo de videojuegos. De hecho, sus únicas funciones dentro de este **Motor de Aventuras Gráficas 2D de Unity** son:

- Saber si el ratón está pasando por encima de un objeto interactivo.
- Saber si el ratón está pasando por encima de un personaje no controlable (NPC).
- Saber si el ratón está pasando por encima de un objeto del inventario.
- Saber si al hacer click con el ratón por un sitio del escenario sin objetos interactivos ni NPCs, es una zona transitable.
- Saber si el ratón está pasando por encima de un botón de la interfaz.

Para obtener el tipo de objetos con el que está colisionando el ratón en todo momento, *Unity* cuenta con la librería `Physics2D` que proporciona el método `Raycast` (y sus derivados y otros muchos más métodos distintos) que lo que hace es mandar un rayo desde la posición, dirección y longitud indicadas y devolver el primer objeto con `PolygonCollider2D` con el que colisiona. Del objeto que ha colisionado, podíamos obtener su nombre, `Tag`, componentes, `scripts` ... En este caso el más importante era `Tag`, ya que dependiendo del valor que tenga, lo identificamos con un tipo de objeto distinto. En **Motor de Aventuras Gráficas 2D de Unity** los `Tag` para las colisiones son:

InteractivePoint Nos dice que el objeto con el que el rayo ha colisionado es un objeto interactivo.

NPC Nos dice que el objeto con el que el rayo ha colisionado es un personaje no controlable.

ItemInventory Nos dice que el objeto con el que el rayo ha colisionado es un objeto del inventario.

NavigationCollider Nos dice que con el objeto que ha colisionado es una zona transitable del nivel.

Sabiendo ya los tipos de objetos y sus `Tag`, se creó el `script` `MouseClickHandler` de forma que en todo momento mediante el método `Update` detectaba el objeto por el que estaba pasando el cursor por encima, y si se hacía click, dependiendo del `Tag` del objeto, invocaba una función u otra del objeto que definía las acciones a realizar por **Player** para que interactúase con este.

```
1 private void Update() {
2     //Si no se está ejecutando ninguna escena cinematográfica y no está
3     //activo ningún sistema de elección de respuesta en una conversación
4     if (!CutScenesManager.isPlaying() && MultipleChoiceManager.Instance.
5         isSelectionEnded) {
```

```
4 //Si el cursor estaba oculto, lo volvemos visible
5 if(CustomCursorController.Instance.isCursorHidden) {
6     CustomCursorController.Instance.UnhideCursor();
7 }
8
9 //Reinicializamos los parámetros que cambian en cada
10 //iteración de Update
11
12 //Reinicializamos el texto de la interfaz gráfica que
13 //muestra el nombre del objeto por el que pasa el cursor a
14 //vacío
15 DisplayNameText.text = "";
16 //Cambiamos el gráfico del cursor al gráfico por defecto
17 CustomCursorController.Instance.ChangeCursorToDefault();
18
19 //Obtenemos el objeto más al frente de la pantalla por el
20 //que pasa el cursor
21 GameObject _firstGameObjectHit = FirstObjectOnMouseOver();
22
23 //Creamos una variable string que contendrá el campo Tag
24 //del objeto colisionado
25 string _tagFirstGO;
26
27 //Intentamos obtener el parámetro tag del objeto, si ocurre
28 //la excepción significará que el cursor no ha pasado por
29 //ningún objeto
30 try{
31     _tagFirstGO = _firstGameObjectHit.tag;
32 }
33 catch(NullReferenceException) {
34     _tagFirstGO = "";
35 }
36
37 //Si el cursor ha pasado por encima de un objeto
38 if(_firstGameObjectHit != null){
39     //Dependiendo del Tag del objeto colisionado
40     switch(_tagFirstGO){
41         //Si es una zona transitable
42         case "NavigationCollider":
43             if(Player.Instance.isUsingItemInventory){
44                 DisplayNameText.text = useText +
45                     " " + lastInventoryItemDisplayName +
46                     " " + withText;
47             }
48             CustomCursorController.Instance.
49                 ChangeCursorToDefault();
50             break;
51
52         //Si es un objeto interactivo
53         case "NPC": case "InteractivePoint": case "
54             ItemInventory":
55             string _nameInteractiveElement =
56                 _firstGameObjectHit.GetComponent<
57                 InteractiveElement>().displayName;
58
59             if(Player.Instance.isUsingItemInventory) {
```

```

47             DisplayNameText.text = useText + " "
48                     + lastInventoryItemDisplayName
49                     + " " + withText + " " +
50                     _nameInteractiveElement;
51         }
52     }
53     else{
54
55         DisplayNameText.text =
56             _nameInteractiveElement;
57         _firstGameObjectHit.GetComponent<
58             InteractiveElement>().
59             ChangeCursorOnMouseOver();
60     }
61     break;
62
63
64
65
66     else{
67
68         //Si es un elemento de la interfaz gráfica
69         case "GUI":
70             CustomCursorController.Instance.
71                 ChangeCursorOverUIElement();
72             break;
73         default:
74             if(Player.Instance.isUsingItemInventory){
75                 DisplayNameText.text = useText + " "
76                     + lastInventoryItemDisplayName
77                     + " " + withText;
78             }
79         }
80
81         //En caso de que no haya colisionado con ningún
82         //objeto pero se haya detectado en el Event System
83         //que detecta si ha pasado por encima de un
84         //objeto de Unity UI)
85         if(EventSystem.current.IsPointerOverGameObject()){
86             CustomCursorController.Instance.
87                 ChangeCursorOverUIElement();
88         }
89
90         //Si no ocurre nada de eso, se restablecen los
91         //valores por defecto
92         else{
93             if(Player.Instance.isUsingItemInventory){
94                 DisplayNameText.text = useText + " "
95                     + lastInventoryItemDisplayName
96                     + " " + withText;
97             }
98             CustomCursorController.Instance.
99                 ChangeCursorToDefault();
100        }
101    }
102
103    //Código que continúa el script....
```

Aunque en el caso específico de **Motor de Aventuras Gráficas 2D de Unity**, más específicamente en `MouseClickHandler.cs`, el método que se tuvo que usar es `RaycastAll`, que devuelve todos los objetos con los que colisiona el rayo. Esto es porque surgió un problema de que cuando dos objetos con `PolygonCollider2D` se solapaban, devolvía uno de los dos aleatoriamente sin importar el orden en el que aparecían en la pantalla. La manera de arreglar esto era usando `RaycastAll`, tal y como hemos dicho previamente, y al obtener la lista con todos los objetos con `PolygonCollider2D` que ha colisionado, los ordenamos por orden de profundidad según el eje `z` del componente `Transform` de cada objeto colisionado.

```

1  private GameObject FirstObjectOnMouseOver () {
2      //Inicializamos la variable que va a contener el objeto a devolver
3      //por el método
4      GameObject _firstGameObject = null;
5      //Coge la posición del cursor y la traduce a posición del escenario
6      Vector2 _mousePosition = Camera.main.ScreenToWorldPoint (Input.
7         .mousePosition);
8
9      //Creamos el array que va a contener los objetos con los que
10     //colisione el rayo
11     RaycastHit2D[] _hits = null;
12
13     //Lanzamos el rayo
14     _hits = Physics2D.RaycastAll (new Vector2 (_mousePosition.x,
15         _mousePosition.y), Vector2.zero, 10f, Physics2D.
16         DefaultRaycastLayers, -2, 4);
17
18     //Comprobamos que el rayo ha colisionado con al menos un objeto
19     if (_hits.Length > 0) {
20         //Como el array devuelve los objetos en un orden aleatorio,
21         //tenemos que comprobar para cada uno su orden en el eje
22         //Z, el que tenga el número más grande, será el que esté m
23         //ás al frente de la pantalla
24         _firstGameObject = _hits[0].collider.gameObject;
25
26         for (int i = 0; i < _hits.Length; i++) {
27             if (_firstGameObject.transform.position.z < _hits[i].
28                 transform.position.z) {
29                 _firstGameObject = _hits[i].collider.
30                     gameObject;
31             }
32         }
33         //....
34     }
35
36     //Finalmente devolvemos el objeto colisionado más al frente
37     return _firstGameObject;
38 }
```

Además de eso, se decidió que el *script* `MouseClickHandler.cs` formarse parte del objeto **Player**. Esto es debido a que dicho *script* solo queremos que se ejecute únicamente cuando el objeto **Player** esté en la escena.

3.6.6. Sistema de inventario

En todos los videojuegos del género de las aventuras gráficas, debe de existir un sistema de inventario donde poder guardar los objetos que va recogiendo el *Jugador* por los escenarios, y que luego dichos objetos dentro del inventario, puedan usarse en otros objetos (o NPCs) del escenario.

En caso de **Motor de Aventuras Gráficas 2D de Unity**, el sistema del inventario se ha implementado como un `PersistentSingleton`, un objeto del que solo puede haber una instancia única suya en todo momento y que además, no se destruye al cambiar de escenario como los objetos normales. Las decisiones de por qué se hizo así fueron:

- Siempre y en todo momento, solo puede haber un único objeto de inventario. Si hubiera más podría haber problemas en el juego debido a que el juego no sabría en cuál inventario añadir los objetos que se cojan. O que cuando se intente seleccionar un objeto del inventario, haya duplicados y no sepa cuál coger tampoco.
- Es menos costoso mantener el objeto del inventario sin destruirse que ir iniciando y destruyéndolo en cada nivel, sobretodo porque entonces tendríamos que instanciar siempre los objetos del inventario que había previamente. Además de saber las posiciones de los objetos del inventario y de si el inventario estaba abierto o cerrado.

```
1  using UnityEngine;
2  using System.Collections;
3
4  //Clase abstracta y genérica, para que pueda ser reutilizado en otros
   scripts al ser heredado
5  public abstract class PersistentSingleton<T> : MonoBehaviour
6  where T : Component{
7
8      protected static T instance;
9
10     public static T Instance{
11         get{
12             return instance;
13         }
14     }
15
16     protected void OnDestroy() {
17         if (instance == this) {
18             instance = null;
19         }
20     }
21
22     protected void OnApplicationQuit(){
23         instance = null;
24     }
25
26     protected void Awake(){
27         if(instance == null){
28             instance = this as T;
29             DontDestroyOnLoad(instance.gameObject);
```

```

30             InitializeOnAwake();
31         }
32     else{
33         Destroy(this.gameObject);
34     }
35 }
36
37 //Pon aquí cualquier código que necesite hacerse en el método Awake
38 protected abstract void InitializeOnAwake();
39 }
```

Sabiendo esto, se creó el objeto **Inventory** con el *script* `Inventory.cs`, de dicho *script* estas son sus funcionalidades más importantes:

- Añadir un objeto al inventario si el inventario no está lleno. En este caso la función de añadido es de coste 1 porque lo añade al final de la lista de objetos de inventario que ya había añadidos previamente.

```

1 public void AddItem(string nameItem) {
2     //Guardamos la posición original por donde se empiezan a
      colocar los objetos del inventario, esto lo hacemos por si
      acaso el inventario está cerrado o abierto (y la posición
      original puede ser distinta)
3     Vector3 _originalItemPosition = this.transform.FindChild(""
      OriginalItemPosition").transform.position;
4
5     //Si el inventario no está lleno
6     if(itemsCount < maxItemsCapacity) {
7         //Creamos el objeto del inventario a partir del nombre
8         GameObject _itemToLoad = Resources.Load<GameObject>(""
      Prefabs/Inventory/Items/" + nameItem);
9
10        //Calculamos su posición en el nivel a partir de la
      posición original y su posición en la lista de
      objetos del inventario
11        _itemToLoad.transform.position = new Vector3(
      _originalItemPosition.x + itemsCount *
      widthBetweenItems, _originalItemPosition.y,
      _originalItemPosition.z);
12
13        //Instanciamos el objeto del inventario
14        GameObject _ newItemAdded = Instantiate(_itemToLoad) as
          GameObject;
15        //Al objeto de inventario recien instanciado le
          ponemos su nombre
16        _ newItemAdded.name = nameItem;
17        //Ponemos que esté anidado dentro del objeto de
          inventario
18        _ newItemAdded.transform.parent = this.transform;
19        //Lo añadimos al array de objetos de inventario
20        items[itemsCount] = _ newItemAdded;
21        itemsCount++;
22    }
```

23 | }

- Eliminar un objeto del inventario si el inventario no está vacío. En este caso la función puede llegar a ser hasta de coste n (donde n es el número de objetos que hay en el inventario), porque al quitar el objeto del inventario reajusta la lista de forma que quite el hueco que se acaba de quedar por eliminar un objeto del inventario.

```
1 public void RemoveItem(string nameItem) {
2     //Comprobamos que el inventario no está vacío
3     if(itemsCount > 0) {
4         //Obtenemos el índice dentro del array del objeto del
5         //inventario a quitar
6         int _indexOfItemToRemove = 0;
7
8         while(items[_indexOfItemToRemove].name != nameItem &&
9             _indexOfItemToRemove < maxItemsCapacity) {
10            _indexOfItemToRemove++;
11        }
12
13        //Si el índice es mayor que la capacidad del
14        //inventario, significará que el objeto de inventario
15        //que le hemos pasado a este método no existe, en
16        //caso contrario si está en el inventario
17        if(_indexOfItemToRemove < maxItemsCapacity){
18            //Guardamos la posición original por donde se
19            //empiezan a colocar los objetos del
20            //inventario, esto lo hacemos por si acaso el
21            //inventario está cerrado o abierto (y la
22            //posición original puede ser distinta)
23            Vector3 _originalItemPosition = this.transform
24                .FindChild("OriginalItemPosition").
25                transform.position;
26
27            //Destruimos el objeto del inventario
28            Destroy(items[_indexOfItemToRemove]);
29            items[_indexOfItemToRemove] = null;
30
31            //Recolocamos el resto del array de objetos
32            //del inventario, se compactarán los objetos
33            //del inventario y no se quedará ningún hueco
34            //vacío a medias
35            for(int i = _indexOfItemToRemove; i < (
36                itemsCount - 1); i++) {
37                items[i] = items[i+1];
38                items[i].transform.position = new
39                    Vector3(_originalItemPosition.x + i
40                        * widthBetweenItems,
41                        _originalItemPosition.y,
42                        _originalItemPosition.z);
43            }
44            for(int i = itemsCount; i < maxItemsCapacity;
45                i++) {
46                items[i] = null;
47            }
48        }
49    }
50}
```

```

28         itemsCount--;
29     }
30 }
31 }
```

- Habilitar y volver visible el menú cuando esté en un escenario donde esté **Player** y no se esté reproduciendo ninguna escena cinemática.

```

1 public void Enable() {
2     isEnabled = true;
3     //Colocamos dentro de la pantalla de juego el inventario
4     this.transform.position = new Vector3(0f, 0f, 0f);
5     //Ponemos el inventario como cerrado
6     Close();
7 }
```

- Inhabilitar y volver invisible el inventario en caso de estar reproduciéndose una escena cinemática o estar en una escena de menú. Así evitamos su uso accidental.

```

1 public void Disable() {
2     isEnabled = false;
3     //Colocamos fuera de la pantalla de juego el inventario
4     this.transform.position = new Vector3(0f, -10f, 0f);
5     Close();
6 }
```

- Obtener una lista de todos los objetos de inventario que hay.

```

1 public List<GameObject> GetItems() {
2     List<GameObject> _currentItemsOnInventory = new List<
3         GameObject>();
4
5     for(int index = 0; index < items.Length; index++) {
6         if(items[index] != null) {
7             _currentItemsOnInventory.Add(items[index]);
8         }
9     }
10
11 }
```

- Buscar si existe en el inventario un objeto con un nombre en concreto.

```

1 public bool HasItem(string nameItem) {
2     int _indexInventory = 0;
3     while(_indexInventory < items.Length) {
4         GameObject _ currentItem = items[_indexInventory];
5         if(_ currentItem != null) {
6             if(_ currentItem.name == nameItem) {
7                 return true;
8             }
9         }
10    }
```

```

9             }
10            _indexInventory++;
11        }
12
13    return false;
14 }
```

- Actualiza los textos de los objetos del inventario al cambiar de idioma.

```

1 public void UpdateAllItems() {
2     for(int i = 0; i < items.Length; i++) {
3         if(items[i] != null) {
4             items[i].GetComponent<ItemInventory>().
5                 UpdateInfo();
6         }
7     }
}
```

Objeto de inventario

Una vez implementado el *script* del inventario, nos hacían falta los objetos del inventario propiamente dichos. Estos objetos debían tener:

- Un componente **SpriteRenderer** con el parámetro **Sorting Layer** en el valor **GUI** dado que se tenía que su gráfico se tenía que dibujar por encima de cualquier otro gráfico de otros objetos cuando fueran a ser usados por el escenario.
- Un componente **PolygonCollider2D** con la forma de la casilla del inventario.
- En el parámetro **Tag** tener el valor de **ItemInventory** para que pudiera ser detectado correctamente por **MouseClickHandler.cs** y se pueda hacer click en él.
- El *script* **ItemInventory.cs**, que hice que fuera una especialización de **InteractiveElement.cs** dado que compartía la mayoría de los métodos de este *script*. Las funcionalidades más importantes a destacar de este *script* son:
 - Al hacer click izquierdo, si el objeto está en el inventario, hace que pase a estar seleccionado por el cursor y lo podamos usar en los distintos elementos del escenario. En el caso de que el objeto del inventario estuviese previamente ya seleccionado, intentará ser usado en el objeto interactivo del escenario al que se le ha hecho click. En este último caso, si el objeto interactivo puede interactuar con el objeto del inventario, hace que **Player** empiece a ejecutar una serie de acciones determinada para esa combinación de objetos.

Fragmento de **MouseClickHandler.cs** que gestiona el uso de **ItemInventory.cs** al hacerle click izquierdo:

```

1 case "ItemInventory":
2   //Si Player estaba usando un objeto del inventario previamente,
3   //intenta usarlo encima de un objeto interactivo del escenario
4   if(Player.Instance.isUsingItemInventory) {
```

```

4     _firstGameObjectHit.GetComponent<InteractiveElement>().
5         ActionOnItemInventoryUsed(lastItemInventoryClicked);
6     }
7     //En caso contrario, selecciona el objeto del inventario
8     else{
9         lastItemInventoryClicked = _firstGameObjectHit;
10        lastItemInventoryName = _firstGameObjectHit.name;
11        lastInventoryItemDisplayName = _firstGameObjectHit.
12            GetComponent<InteractiveElement>().displayName;
13        lastItemInventoryClicked.GetComponent<ItemInventory>().
14            Select();
15    }
16    break;

```

Fragmento de ItemInventory.cs del método Select:

```

1 public void Select() {
2     //Si Player no está haciendo otra acción, usando un objeto
3         //del inventario o se está ejecutando una escena cinemató-
4         //tica, ejecuta la selección del objeto del inventario
5     if(!Player.Instance.isDoingAction && !Player.Instance.
6         isUsingItemInventory && !CutScenesManager.isPlaying()) {
7         if(!isSelectedByMouse) {
8             //Mandamos el evento de que Player está
9                 //usando un objeto
10            BeginUsingItem();
11            //Escondemos el cursor
12            CustomCursorController.Instance.HideCursor
13                ();
14            //Marcamos el objeto del inventario como
15                //seleccionado
16            isSelectedByMouse = true;
17            //Guardamos la posición original del
18                //objeto antes de ser seleccionado
19            originalPosition = this.transform.position
20                ;
21            //Obtenemos la distancia de diferencia que
22                //había entre la posición del objeto del
23                    //inventario y el cursor al ser
24                    //seleccionado
25            mouseOffset = this.transform.position -
26                Camera.main.ScreenToWorldPoint(new
27                    Vector2(Input.mousePosition.x, Input.
28                        mousePosition.y));
29            //Colocamos el orden de dibujado de
30                //SpriteRenderer a 2 para que así se
31                    //dibuje encima de los otros objetos del
32                    //inventario
33            this.GetComponent<Renderer>().sortingOrder
34                = 2;
35            //Colocamos el orden en el eje Z para que
36                //sea detectado correctamente por
37                    //MouseClickHandler
38            this.transform.position = new Vector3(this
39                .transform.position.x, this.transform.

```

```

19             position.y, positionZWhenSelected);
20         }
21     }

```

- Al hacer click derecho, si el objeto está en el inventario, hace que **Player** diga una descripción de este. En caso de que el objeto del inventario estuviese usándose, se deseleccionaría y volvería a su casilla del inventario.

Fragmento de `MouseClickHandler.cs` que gestiona el uso de `ItemInventory.cs` al hacerle click derecho:

```

1 if(Player.Instance.isUsingItemInventory) {
2     lastItemInventoryClicked.GetComponent<ItemInventory>().
3         Deselect();

```

Fragmento de `ItemInventory.cs` del método `Deselect`:

```

1 public void Deselect() {
2     //Si estaba seleccionado previamente, reestablecemos los
3     //valores cambiados
4     if(isSelectedByMouse) {
5         CustomCursorController.Instance.UnhideCursor();
6         isSelectedByMouse = false;
7         this.transform.position = originalPosition;
8         this.GetComponent<Renderer>().sortingOrder = 1;
9
10    }
11 }

```

Para que un objeto interactivo, NPC u otro objeto del inventario pueda interactuar con un objeto del inventario, lo que se ha hecho es crear un método virtual en el *script* `InteractiveElement.cs` de nombre `ActionOnItemInventoryUsed`. En principio este método está vacío, pero puede ser sobrecargado por *scripts* que hereden de `InteractiveElement.cs` e implementarlo de la siguiente manera:

```

1 //Le pasamos al objeto interactivo el objeto del inventario con el que ha
2 //sido clickado
3 public override void ActionOnItemInventoryUsed(GameObject itemInventory) {
4     /* Como usarlo:
5      * if(!Player.Instance.isDoingAction && !CutScenesManager.isPlaying())
6      * {
7      *     //Dependiendo del nombre del objeto del inventario, el
8      *     //objeto interactivo puede interactuar con él o no
9      *     switch(itemInventory.name) {
10        case "name 1":
11            itemInventory.GetComponent<ItemInventory>().
12                Unselect();
13            //Acción 1.....
14            break;
15        }
16    }
17 }

```

```

11     case "name 2":
12         itemInventory.GetComponent<ItemInventory>().
13             Unselect();
14             //Accion 2.....
15             break;
16         }
17     }
18     */
19 }
```

3.6.7. Sistema de audio

El sistema de audio del **Motor de Aventuras Gráficas 2D de Unity** era uno de los que menos importancia han tenido en la etapa de implementación del proyecto. Esto es debido a que al ser un juego 2D del género de aventuras gráficas de corte clásico, el sonido está relegado a un segundo plano.

Al ser un juego 2D, no tenemos que preocuparnos de en qué posición debemos generar los objetos con componente AudioSource (generadores de sonido), pues el componente AudioListener (escuchas de sonido) está incorporado al objeto **MainCamera** y el volumen de los sonidos es el mismo, sin importar la posición de los objetos generadores de sonido.

Por otro lado, al ser la demo técnica de **1812: La aventura** una aventura gráfica que imita a las antiguas, y que carece de diálogos hablados (porque habría supuesto invertir tiempo y dinero en encontrar dobladores y organizarlos, aparte de los otros colaboradores que ha habido) y no se han podido introducir muchas músicas y efectos de sonido debido a los imprevistos comentados en el capítulo de Planificación 2; el audio del proyecto resultaba bastante básico.

Finalmente decidí crear una clase estática (por lo tanto no necesita estar en ningún objeto de *Unity* para poder ser utilizada) llamada `AudioManager.cs`. La funciones de esta clase son las siguientes:

- Cuando se quiera reproducir una música, verifica si un objeto que tenga de nombre `Music_music-name` no existe.

Si no existe, crea un objeto de nombre `Music_music-name`, con Tag con el valor `Music` y de componente un AudioSource con el sonido de la música.

En caso de que previamente existiera, lo que hace es parar y empezar de nuevo la reproducción de la música del componente AudioSource del objeto `Music_music-name`.

En ambos casos, si desde el método para reproducir música se indica que se reproduzca en bucle la música, en el parámetro `loop` del componente AudioSource lo ponemos con valor a `true`. Si por el contrario no deseamos que se reproduzca en bucle, cuando la música termine de reproducirse, se destruirá automáticamente.

```

1  public static void PlayMusic(string name, bool loop = false) {
2      AudioClip _clip = null;
3      GameObject _audioGO = null;
4      AudioSource _source = null;
```

```

6     _audioGO = GameObject.Find("Music_" + name);
7
8     if(_audioGO != null){
9         MonoBehaviour.Destroy(_audioGO);
10        _audioGO = null;
11    }
12
13    try{
14        _clip = Resources.Load<AudioClip>("Sound/Music/" +
15            name);
16    }
17    catch(NullReferenceException){
18        Debug.LogError("Can't find Music audio clip");
19        return;
20    }
21
22    _audioGO = new GameObject("Music_" + _clip.name);
23    _audioGO.tag = "Music";
24    _source = _audioGO.AddComponent< AudioSource >();
25    _source.clip = _clip;
26    _source.volume = GameState.SystemData.AudioVolumeSettings.
27        music / 100f;
28    _source.playOnAwake = false;
29    _source.loop = loop;
30
31    _source.Play();
32    if(!loop){
33        MonoBehaviour.Destroy(_audioGO, _clip.length);
34    }
35}

```

- Cuando se quiera parar de reproducir una música, verifica si un objeto que tenga de nombre Music_music-name existe.

En el caso de que exista, borra el objeto con dicho nombre, en caso contrario no hace nada.

```

1 public static void StopMusic(string name){
2     AudioSource _source = null;
3
4     try{
5         _source = GameObject.Find("Music_" + name).
6             GetComponent< AudioSource >();
7     }
8     catch(NullReferenceException){
9         Debug.LogError("Can't find Music audio clip");
10        return;
11    }
12
13    _source.Stop();
14    MonoBehaviour.Destroy(_source.gameObject);
15}

```

- Cuando se quiera pausar de reproducir una música, verifica si un objeto que tenga de nombre Music_music-name existe.

En el caso de que exista, pausa la reproducción del audio del componente AudioSource si la música se está reproduciendo en bucle, en caso contrario no hace nada.

```
1 public static void PauseMusic(string name) {
2     AudioSource _source = null;
3
4     try{
5         _source = GameObject.Find("Music_" + name) .
6             GetComponent<AudioSource>();
7     }
8     catch(NullReferenceException){
9         Debug.Log("Can't find Music audio clip");
10        return;
11    }
12
13    if(_source.loop){
14        if(_source.isPlaying){
15            _source.Pause();
16        }
17    }
18 }
```

- Cuando se quiera terminar de pausar una música, verifica si un objeto que tenga de nombre Music_music-name existe.

En el caso de que exista, termina de pausar la reproducción del audio del componente AudioSource si la música se está reproduciendo en bucle, en caso contrario no hace nada.

```
1 public static void UnpauseMusic(string name) {
2     AudioSource _source = null;
3
4     try{
5         _source = GameObject.Find("Music_" + name) .
6             GetComponent<AudioSource>();
7     }
8     catch(NullReferenceException){
9         Debug.Log("Can't find Music audio clip");
10        return;
11    }
12
13    if(_source.loop){
14        if(!_source.isPlaying){
15            _source.UnPause();
16        }
17    }
18 }
```

- Cuando se quiera reproducir un efecto de sonido, verifica si un objeto que tenga de nombre SFX_sfx-name no existe.

Si no existe, crea un objeto de nombre SFX_sfx-name, con Tag con el valor SFX y de componente un AudioSource con el sonido del efecto.

En caso de que previamente existiera, lo que hace es parar y empezar de nuevo la reproducción del efecto de sonido del componente AudioSource del objeto SFX_sfx-name.

En ambos casos, si desde el método para reproducir el efecto de sonido se indica que se reproduzca en bucle, en el parámetro `loop` del componente `AudioSource` lo ponemos con valor a `true`. Si por el contrario no deseamos que se reproduzca en bucle, el sonido terminará de reproducirse y se destruirá automáticamente.

```

1  public static void PlaySFX(string name, bool loop = false) {
2      AudioClip _clip = null;
3      GameObject _audioGO = null;
4      AudioSource _source = null;
5
6      _audioGO = GameObject.Find("SFX_" + name);
7      if(_audioGO != null){
8          MonoBehaviour.Destroy(_audioGO);
9          _audioGO = null;
10     }
11
12     try{
13         _clip = Resources.Load<AudioClip>("Sound/SFX/" + name)
14         ;
15     }
16     catch(NullReferenceException){
17         Debug.Log("Can't find SFX audio clip with name " +
18             name);
19         return;
20     }
21
22     _audioGO = new GameObject("SFX_" + _clip.name);
23     _audioGO.tag = "SFX";
24     _source = _audioGO.AddComponent<

```

- Cuando se quiera parar de reproducir un efecto de sonido, verifica si un objeto que tenga de nombre `SFX_sfx-name` existe.

En el caso de que exista, borra el objeto con dicho nombre, en caso contrario no hace nada.

```

1  public static void StopSFX(string name){
2      AudioSource _source = null;
3
4      try{
5          _source = GameObject.Find("SFX_" + name).GetComponent<
6              AudioSource>();
7      }
7      catch(NullReferenceException){
```

```

8             Debug.Log("Can't find SFX audio clip with name " +
9                 name);
10            return;
11        }
12
13        _source.Stop();
14        MonoBehaviour.Destroy(_source.gameObject);
15    }

```

- Si queremos saber si una música se está reproduciendo, verifica que exista un objeto de nombre `Music_music-name`. Si existe mira en su componente `AudioSource` el parámetro `isPlaying` y devuelve su valor, en caso contrario de que no exista, devuelve el valor `false` (no se está reproduciendo).

```

1 public static bool IsPlayingMusic(string name) {
2     AudioSource _source;
3
4     try{
5         _source = GameObject.Find("Music_" + name).
6             GetComponent<catch(NullReferenceException) {
8         return false;
9     }
10
11    bool _isPlaying = _source.isPlaying;
12
13    return _isPlaying;
14 }

```

- Si queremos saber si un efecto de sonido se está reproduciendo, verifica que exista un objeto de nombre `SFX_sfx-name`. Si existe mira en su componente `AudioSource` el parámetro `isPlaying` y devuelve su valor, en caso contrario de que no exista, devuelve el valor `false` (no se está reproduciendo).

```

1 public static bool IsPlayingSFX(string name) {
2     AudioSource _source;
3
4     try{
5         _source = GameObject.Find("SFX_" + name).GetComponent<
6             AudioSource>();
7     }catch(NullReferenceException) {
8         return false;
9     }
10
11    bool _isPlaying = _source.isPlaying;
12
13    return _isPlaying;
14 }

```

- Si queremos parar de reproducir todas las músicas que se están reproduciendo, busca en el nivel todos los objetos que tengan el Tag con valor `Music`. Una vez ha encontrado todos los objetos, los destruye.

```

1 public static void StopAllMusic() {
2     GameObject[] _musics = GameObject.FindGameObjectsWithTag("Music");
3
4     foreach(GameObject music in _musics) {
5         MonoBehaviour.Destroy(music);
6     }
7 }
```

- Si queremos cambiarle el volumen a todas las músicas que se están reproduciendo, busca en el nivel todos los objetos que tengan el Tag con valor Music. Una vez ha encontrado todos los objetos, les cambia el volumen al componente AudioSource.

```

1 public static void ChangeVolumeAllCurrentMusics(int newMusicVolume) {
2     //Guardamos en el archivo de ajustes el volumen para las músicas actual del juego, para que así la próxima vez que abramos el juego siga estando en el mismo volumen
3     GameState.SystemData.AudioVolumeSettings.music =
4         newMusicVolume;
5     SettingsFileManager.Instance.SaveSettingsFile();
6     GameObject[] _musics = GameObject.FindGameObjectsWithTag("Music");
7
8     foreach(GameObject music in _musics) {
9         music.GetComponent<AudioSource>().volume = GameState.
10            SystemData.AudioVolumeSettings.music / 100f;
11    }
12 }
```

- Si queremos cambiarle el volumen a todas las músicas que se están reproduciendo, busca en el nivel todos los objetos que tengan el Tag con valor Music. Una vez ha encontrado todos los objetos, les cambia el volumen al componente AudioSource.

```

1 public static void ChangeVolumeAllCurrentSFX(int newSFXVolume) {
2     //Guardamos en el archivo de ajustes el volumen para los efectos de sonido actual del juego, para que así la próxima vez que abramos el juego siga estando en el mismo volumen
3     GameState.SystemData.AudioVolumeSettings.sfx = newSFXVolume;
4     SettingsFileManager.Instance.SaveSettingsFile();
5     GameObject[] _musics = GameObject.FindGameObjectsWithTag("SFX");
6
7     foreach(GameObject music in _musics) {
8         music.GetComponent<AudioSource>().volume = GameState.
9             SystemData.AudioVolumeSettings.sfx / 100f;
10    }
11 }
```

Gracias a todas estas funcionalidades, tenemos un control básico sobre el audio que pueda necesitar un videojuego 2D del género de las aventuras gráficas más que suficiente.

3.6.8. Interfaces hechas en Unity UI

Para crear parte del menú dentro del juego, el menú principal, y el menú de opciones, tuve que usar el sistema UI [87] de *Unity*. Dicho sistema funciona de forma totalmente diferente a la hora de dibujar *sprites*, detectar eventos y posee sus propios componentes. Además como la mayor parte del tiempo nos dedicaremos a modificar valores en el *Inspector* y estos componentes, y los *scripts* realizados para trabajar con las interfaces gráficas son repetitivos por la naturaleza de tratar a estos componentes (véase los tutoriales de *scripting* de *Unity UI* si quiere saber más del tema [88]), se omitirán la mayoría de los *scripts* referentes a las interfaces gráficas y sólo comentaremos aquí los más importantes.

Inicialización del Canvas

El **Canvas** es el objeto raíz en el que debe ir todo objeto que tenga un componente de *Unity UI*, él se encarga de configurar la resolución base que va a tener la interfaz y de cómo va a escalar según cambie la resolución de pantalla del juego.

Para el caso de la demo técnica de **1812: La aventura**, la resolución base del juego es 800x600 píxeles (por querer mantener una resolución clásica), con lo que solo está permitido escalar en resoluciones de 4:3. En la actualidad la mayoría de los monitores tienen resoluciones 16:9, con lo que deberemos modificar los parámetros de escalado manualmente para que se haga bien.

Así que estos son los pasos para inicializar correctamente un **Canvas** para el proyecto:

1. Una vez creado un **Canvas** en el nivel, lo seleccionamos dentro del panel de *Hierarchy* para que se muestre en el *Inspector* y veremos que tiene los componentes *Rect Transform*, *Canvas*, *Canvas Scaler* y *Canvas Renderer*. En los que debemos fijarnos son los componentes *Canvas* y *Canvas Scaler*.
2. En el componente *Canvas*, deberemos cambiar el parámetro *Render Mode* a *Screen Space - Camera* y donde pondrá *Render Camera* le arrastramos el objeto **MainCamera** desde *Hierarchy*. De esta manera cuando creamos elementos de la interfaz mediante los paneles *Inspector* y *Scene*, estos se verán con el tamaño correspondiente al que tienen los gráficos *SpriteRenderer* del nivel. De la otra manera, los gráficos de la interfaz se verían fuera de sitio en el panel *Scene* y nos costaría más trabajo moldear los elementos de la interfaz respecto al nivel.
3. En el componente *Canvas Scaler*, deberemos cambiar el parámetro *Ui Scale Mode* por *Scale With Screen Size*, y en los parámetros que salgan ponemos lo siguiente:

Reference Resolution Aquí ponemos la resolución base a la que vaya el juego, en nuestro caso la resolución base es 800x600 píxeles.

Screen Match Mode En caso de que la resolución del monitor no pueda tener un múltiplo de la resolución base deseada, intentará escalar la pantalla del juego de diferentes maneras. En este caso concreto, le ponemos de valor *Match Width or Height* y en el *slider* lo ponemos a 1 (que se corresponde a que siempre escala en base a la altura de la pantalla). Así la pantalla del juego intentará siempre ocupar la altura máxima y en vez de estirarse por los lados (estropeando la resolución de los gráficos del juego), dejará dos franjas negras.

De esta manera prepararemos todos los objetos **Canvas** para interfaces gráficas que hay en la demo técnica de **1812: La aventura**.

Botón genérico

Cómo iba a ser necesario programar el comportamiento de muchísimos botones para las interfaces gráficas y la mayoría de ellos tenían un comportamiento bastante similar, creé un *script* con una clase abstracta (de esta manera no se puede añadir a los objetos, solamente los *scripts* que hereden de él) llamado `UIGenericButton.cs` que se encargaba de realizar las siguientes funcionalidades:

- Obtiene el texto del botón mediante el cotejamiento en los ficheros `LocatedTexts.xml` de las IDs que se le pasan al *script* por el Inspector. Y en caso de que ocurra el evento de que cambia de idioma el juego, cambia el texto del botón automáticamente.

```
1 //Método que se ejecuta cuando se inicializa el objeto en una escena
2 //de Unity
3 protected virtual void Start(){
4     //Obtiene el componente botón del objeto
5     thisButton = this.GetComponent<Button>();
6
7     //Intenta obtener el componente de texto del objeto botón (por
8     //defecto es un componente de un objeto hijo) y al campo de
9     //texto le asignamos el texto a partir de las IDs por el
10    //sistema de localización de textos, si ocurre la excepción
11    //significará que el objeto botón no tiene texto.
12    try{
13        buttonText = this.transform.FindChild("Text").
14            GetComponent<Text>();
15        buttonText.text = LocatedTextManager.GetLocatedText(
16            localizedTextGroupID, localizedTextElementID,
17            localizedTextStringID)[0];
18    }catch(NullReferenceException){
19        buttonText = null;
20    }
21}
22
23 //.....
24
25 //Si ocurre el evento de que se ha producido un cambio de idioma en el
26 //juego, automáticamente cambia el campo texto del componente texto
27 //del objeto botón (si lo tiene)
28 protected virtual void ChangeButtonTextLanguage(){
29     if(buttonText != null){
30         buttonText.text = LocatedTextManager.GetLocatedText(
31             localizedTextGroupID, localizedTextElementID,
32             localizedTextStringID)[0];
33     }
34 }
```

- El cursor cambia de gráfico al pasar por encima del botón y al salir de este. Para ello tendremos

que incluir un componente Event Trigger que controle los eventos de entrada y salida del ratón y añadirle dichos métodos.

```
1 //Si el ratón pasa por encima del botón, si el botón no está inactivo  
2 //y/o el cursor no está oculto, cambiará el sprite del cursor por el  
3 //de estar encima de los botones de la interfaz gráfica  
4 public virtual void OnMouseOver(){  
5     if(thisButton.interactable && !CustomCursorController.Instance  
6         .isCursorHidden){  
7         CustomCursorController.Instance.  
8             ChangeCursorOverUIElement();  
9     }  
10 }  
11  
12 //Si el ratón termina de pasar por encima del botón, si el botón no  
13 //está inactivo y/o el cursor no está oculto, cambiará el sprite al  
14 //que tiene por defecto  
15 public virtual void OnMouseExit(){  
16     if(thisButton.interactable && !CustomCursorController.Instance  
17         .isCursorHidden){  
18         CustomCursorController.Instance.ChangeCursorToDefault  
19     }  
20 }
```

- Métodos para inhabilitar y habilitar el botón. Estos métodos se ejecutan automáticamente cuando aparece una ventana modal en el juego, para obligar al *Jugador* a responder y que no pulse por accidente ningún otro botón.

```
1 //Cuando el objeto pasa a estar activo en la escena, le asignamos los  
2 //siguientes métodos para que sean ejecutados en diversos eventos  
3 protected virtual void OnEnable(){  
4     //Si empieza el evento de que ha comenzado una ventana modal, automá-  
5     //ticamente inhabilita el botón  
6     ModalWindowHandler.beginQuestion += DisableButton;  
7     //Si la ventana modal termina, automáticamente habilita el objeto botó-  
8     //n  
9     ModalWindowHandler.endedQuestion += EnableButton;  
10    //Si se cambia de idioma, automáticamente lo cambia (como hemos visto  
11    //antes)  
12    LocatedTextManager.currentLanguageChanged += ChangeButtonTextLanguage;  
13 }  
14  
15 //Cuando el objeto pasa a estar inactivo, quitamos los listeners de  
16 //eventos  
17 protected virtual void OnDisable(){  
18     ModalWindowHandler.beginQuestion -= DisableButton;  
19     ModalWindowHandler.endedQuestion -= EnableButton;  
20     LocatedTextManager.currentLanguageChanged -= ChangeButtonTextLanguage;  
21 }  
22  
23 //Método que habilita el objeto botón  
24 public virtual void EnableButton(){  
25     thisButton.interactable = true;
```

```

21    }
22
23    //Método que inhabilita el objeto botón
24    public virtual void DisableButton() {
25        thisButton.interactable = false;
26    }

```

- Cuando se le hace click encima (siempre y cuando el cursor sea visible, si no, no hará nada), ejecuta el sonido de botón pulsado y después el método ActionOnClick que deberá ser declarado obligatoriamente en los *scripts* hijos de UIGenericButton.cs con las funciones que tenga que hacer el botón una vez clickado.

```

1    //Método que se ejecuta cuando pulsamos encima del objeto botón
2    public virtual void OnClick() {
3        //Si el botón no está inhabilitado y el cursor no está oculto,
4        //ejecuta el sonido de pulsado del botón y ejecuta la
5        //corrutina
6        if(thisButton.interactable && !CustomCursorController.Instance
7            .isCursorHidden) {
8            AudioManager.PlaySFX(sfxOnClick);
9            StartCoroutine(WaitForSFXAndExecuteClickAction());
10       }
11
12   //Corrutina que espera que termine de reproducirse el sonido de
13   //pulsado del botón y ejecuta el método ActionOnClick
14   protected IEnumerator WaitForSFXAndExecuteClickAction() {
15       do{
16           yield return null;
17       }while(AudioManager.isPlayingSFX(sfxOnClick));
18       ActionOnClick();
19   }
20
21   //Método abstracto que deberá ser definido en los scripts que hereden
22   //de UIGenericButton para que su objeto botón haga algo al ser
23   //clickado
24   protected abstract void ActionOnClick();

```

Menú genérico

Aparte del botón genérico, muchos menús del juego al ser llamados por un botón tenían un comportamiento similar, e incluso idéntico en el caso de los menús del menú de opciones. Por lo que también fue necesario crear un *script* llamado MenuButton.cs, que lo que hace es encargarse del comportamiento de traer hacia el frente de las pantallas los menús al pulsar sus respectivos botones. Y no sólo eso:

- Cuando se pulsa un botón de Menú, verifica mediante el parámetro booleano estático isAnyMenuActive si hay algún menú activo previamente. Si el valor es false significa que no hay ningún menú activo, y aparte de traer su menú correspondiente al frente, envía el evento a los demás botones de menú de la escena que se inhabiliten (y al botón de regreso al juego) y cambia

el valor de `isAnyMenuActive` a `true`. En caso de que estuviera ya a `true` el botón no hace nada.

```
1 //Declaración del evento para cuando se active el menú
2 public delegate void SetMenuActive();
3 public static event SetMenuActive menuSetActive;
4
5 //Heredamos el método ActionOnClick de UIGenericButton y lo
6 //sobrecargamos
7 protected override void ActionOnClick () {
8     //Comprobamos que el botón de menú tiene efectivamente un
9     //objeto de menú asignado
10    if(menuPanel != null) {
11        //Comprobamos si no hay un menú previamente activo
12        if(!ReturnToButton.isAnyMenuActive){
13            //Ponemos el gráfico del cursor a su gráfico
14            //por defecto, para que no siga con el grá
15            //fico de estar encima de un botón cuando
16            //salga el menú
17            CustomCursorController.Instance.
18                ChangeCursorToDefault();
19            //Habilitamos el menú
20            EnableMenu();
21        }
22    }
23
24 //Método para habilitar el menú
25 public virtual void EnableMenu() {
26     //Inhabilitamos el botón de menú, dado que el menú esta activo
27     DisableButton();
28     //Comprobamos si hay un objeto de menú asignado
29     if(menuPanel != null) {
30         //Lo traemos al frente de la pantalla
31         menuPanel.SetAsLastSibling();
32     }
33     //Ponemos el booleano estático a true, pues acabamos de
34     //activar un menú
35     isThisMenuActive = true;
36     //Se ejecuta el evento de que hay un menú activo
37     menuSetActive();
38 }
```

- Si se pulsa el botón Volver al Menú con un menú activo, vuelve a colocar el menú en su posición y condiciones iniciales.

```
1 //Cuando el objeto pasa a estar activo en la escena, le asignamos los
2 //siguientes métodos para que sean ejecutados en diversos eventos
3 protected override void OnEnable() {
4     //...
5     //Si un menú se vuelve activo, inhabilitamos el resto de
6     //botones de menú
7     MenuButton.menuSetActive += DisableButton;
8     //Si se pulsa el botón de Volver al Menú, inhabilita el menú
```

```

    activo y lo resetea
7     ReturnToButton.resetUIPositions += DisableMenu;
8     //...
9 }
10 //Cuando el objeto pasa a estar inactivo, quitamos los listeners de
11 //eventos
12 protected override void OnDisable() {
13     //...
14     MenuButton.menuSetActive == DisableButton;
15     ReturnToButton.resetUIPositions == DisableMenu;
16     //...
17 }
18
19 //Método que inhabilita el menú
20 public virtual void DisableMenu() {
21     //Comprueba que hay un objeto de menú asignado al botón
22     if(menuPanel != null) {
23         //Vuelve a llevar el menú al fondo de la escena
24         menuPanel.SetAsFirstSibling();
25     }
26     //Ponemos el booleano estático a false, pues se acaba de
27     //desactivar el menú activo
28     isThisMenuActive = false;
29     //Reseteamos la ventana modal, en el caso de que estuviera
30     //activa
31     ModalWindowHandler.Instance.Disable();
32     //Volvemos a habilitar el botón del menú
33     EnableButton();
34 }

```

Por último, para que funcione un botón de menú, lo que hay que hacer es añadirle el *script* `MenuItem.cs` y desde el `Inspector` asignarle al parámetro `menuPanel` el objeto con el menú desde `Hierarchy`.

Ventana Modal

En algunos momentos específicos en los que estemos usando los menús del juego, por seguridad es necesario que salte una ventana modal para asegurarnos que queremos ejecutar ciertos cambios importantes de verdad y no por accidente. Sobretodo es relevante en el momento de guardar, cargar y borrar partidas, dado que si se les hace click por accidente pueden realizar cambios en el estado del juego que sean irreversibles. Por estos motivos decidí implementar un sistema de ventana modal para los menús.

El sistema de ventana modal es un panel con un texto preguntándonos si queremos realizar la acción de verdad mediante los dos botones que tiene incorporados y controlado por el *script* `ModalWindowHandler.cs`. También por precaución, se deshabilitan todos los demás botones del juego (menos el de Volver al Menú) para obligar al jugador a elegir una de las dos respuestas disponibles. Una vez se termina de elegir la opción, se vuelven a habilitar los botones y si se ha dado el caso de que se ha pulsado el botón Sí en la ventana modal, ejecuta la acción por la que se le preguntaba.

Aquí tenemos un ejemplo del funcionamiento de la ventana modal en el caso del botón para salir del juego:

```
1 //Acción que se ejecuta al pulsar el botón para salir del juego
2 protected override void ActionOnonClick (){
3     CustomCursorController.Instance.ChangeCursorToDefault ();
4     EnableMenu ();
5     StartCoroutine(WaitForExitGameOnClick ());
6 }
7
8
9 //Corrutina que se ejecuta al pulsar el botón para salir del juego
10 private IEnumerator WaitForExitGameOnClick (){
11     //Inicializamos la ventana modal y le pasamos como parámetro el ID
12     //de la pregunta que tiene que mostrar
13     ModalWindowHandler.Instance.Initialize("CLOSE_GAME");
14     do{
15         yield return null;
16         //El bucle se ejecuta hasta que el Jugador eliga una
17         //respuesta en la ventana modal
18     }while(!ModalWindowHandler.Instance.isAnswerSelected);
19
20
21     //Si se ha pulsado el botón de Si
22     if(ModalWindowHandler.Instance.isYesClicked){
23         //Desactiva la ventana modal
24         ModalWindowHandler.Instance.Disable();
25         //Se sale del juego
26         Application.Quit();
27     }
28     //Si se ha pulsado el botón de No
29     else{
30         Deshabilita el 'menú' del botón para salir (actualiza el
31         //botón de Volver al Menú y desactiva la ventana modal)
32         DisableMenu();
33         //Ejecuta el evento de que se ha impedido salir del juego
34         abortedExitGame();
35     }
36 }
```

3.6.9. Integración de base de datos hecha en SQLite y Unity

Si 1812: La aventura fuera como inicialmente se planteó (un juego completo) estaría lleno de detalles e información sobre bastantes y diversos elementos, hechos y anécdotas que ocurrieron en esta época. De hecho, la totalidad de los puzzles usarían algún elemento concreto de la historia que obligue al jugador, ademas de haber usado uno o varios objetos del inventario, a buscar información sobre estos hechos ya sea en el juego o fuera del juego. Podríamos considerar que así instamos al jugador a aprender hechos históricos para avanzar, pero también se podrían dar los casos en que el jugador no recuerde lo leído para cuando necesitara usar la información o que dejase de investigar lo suficiente por su cuenta lo necesario para avanzar. Estos dos últimos casos podrían hacer que un jugador desistiese de seguir jugando al juego.

Por lo tanto, para minimizar estos casos negativos, se ha implementado un pequeño buscador de in-

formación sobre estos hechos para que el jugador pueda usarlo directamente, ahorrándole tiempo y ofreciéndole comodidades. No es un elemento de obligado uso para avanzar, simplemente de refuerzo y totalmente a elección del jugador dependiendo de su modo de juego.

A tal buscador lo he titulado **Gadipedia** en el contexto de este juego, como se puede obviar por su nombre, viene de la combinación de *Wikipedia* y *Gades*, nombre con el que antiguamente se llamaba Cádiz y que aún palabras actuales tienen de raíz esta palabra como “gaditanos” que significa “gente de la ciudad y/o provincia de Cádiz”.

El buscador está montado de manera que escribiéndole una o varias palabras clave, nos cotejará en una base de datos dichos términos y nos devolverá las ID de las posibles entradas relacionadas que dentro del juego se traducirán en botones de acceso directo a dichas entradas. Si clickamos en una de esas entradas, usando la ID que contenga, obtendrá del fichero .xml de los textos localizados el nombre del artículo y su descripción, mostrándolo en una nueva ventana para que el jugador pueda leerlo.

Hablando más en profundidad sobre la base de datos, la elección elegida para montarla es usando *SQLite* [20] porque:

- Nos permite crear una base de datos sin necesidad de usar un servidor.
- No necesita de ninguna instalación previa para poder usarse.
- La base de datos se contiene en un único fichero de extensión .db, con lo cual es fácil de distribuir al incluirlo dentro de los ficheros del juego (aunque en aplicaciones web no podría ser, ya que en ese estado no permite el acceso de ficheros externos dentro de aplicaciones *Unity* en ese caso).

Además, como estamos trabajando con *Unity* usando C# y .NET, disponemos de librerías como System.Data y Mono.Data.SqliteClient que podemos copiarlas al proyecto para que nos permitan trabajar con la base de datos sin necesidad de usar librerías no oficiales.

Para instalar *SQLite* para Windows nos bajamos desde la sección de descargas de la página oficial [21] los siguientes ficheros:

- sqlite-shell-win32-* para poder usar la consola de comandos para crear la base de datos.
- sqlite-dll-win32-* para tener las librerías internas de *SQLite*.

Creamos una variable de entorno PATH en el sistema Windows con la ruta hacia los ficheros que saldrán al descomprimir los archivos .zip para así poder acceder a los comandos de *SQLite* desde la consola de Windows. Para asegurarnos de que *SQLite* se puede acceder correctamente desde la consola, escribimos sqlite3 y debería saltarnos el prompt de *SQLite*, si es así es que todo va bien.

Para más detalles sobre cómo seguir la instalación, por ahora seguir las instrucciones incluidas en la documentación oficial de *SQLite*: http://www.tutorialspoint.com/sqlite/sqlite_installation.htm

En cuanto al diseño de la base de datos, teniendo en cuenta del poco tiempo disponible para realizarla y que solo va a ser un ejemplo para una demo técnica para probar el Motor de Aventuras Gráficas 2D para

Unity, limitaremos a unas pocas ID de entradas relacionadas por palabra clave de búsqueda. Costará más dar con un resultado lo cual por ahora será algo negativo, pero lo positivo es que tal y como montemos la base de datos y el *script* de acceso a esta es que aunque ampliemos el número de campos para añadir más ID de entradas a la tabla de palabras clave, y se añadiesen aún más entradas para añadir más palabras clave con sus respectivas ID de entradas relacionadas, el *script* funcionará independientemente de estos campos. Con lo que solo sería cuestión de disponer tiempo para hacer una base de datos más completa.

La idea para el *script* es que al meter las palabras clave en el buscador y después clickar en el botón *Buscar* ocurra lo siguiente:

1. Recibimos la cadena de texto escrita en el formulario de texto del buscador de la Gadipedia.
2. Separamos las palabras clave según los espacios en blanco, las convertimos a minúsculas y les quitamos las tildes para evitar problemas de palabras parecidas.
3. Hacemos una búsqueda en la base de datos que nos devuelva las entradas relacionadas con las palabras clave introducidas.
4. Se muestran las entradas relacionadas que aparecían con dicha palabra y el jugador clickara en la entrada que él considere oportuno para ayudarle con el juego. En caso de que no haya ninguna entrada relacionada con las palabras clave introducidas, sale el mensaje “*No ha habido ningún resultado*”.

Si en el buscador se han introducido más de una palabra clave en el buscador, se hará una búsqueda en la base de datos por cada una, y en el caso de repetirse alguna de las ID relacionadas, crearemos un `Dictionary<string, int>` en la que el campo `string` será el ID de la entrada y el campo `int` será un contador que aumentará en uno cada vez que se repita la aparición de una ID de entrada al hacer una búsqueda de una palabra clave.

Una vez se acaben las palabras claves a cotejar en la base de datos, cogeremos el `Dictionary` y crearemos una `List<string>` que esté ordenada de mayor a menor según el valor de `int` en el par de valores del `Dictionary`. De esa `List<string>` finalmente se sacarán los ID de los artículos a mostrar de la búsqueda y estarán ordenadas de mayor relevancia a menor relevancia.

Diseño de la base de datos

La base de datos estará formada por una única tabla llamada `keywords_search`. El primer campo se llamará `keyword` y será de tipo `varchar(50) not null`, luego el otro campo será `article_id` de tipo `varchar(100) not null` y como `primary key` en este caso usaremos una clave compuesta de `keyword` y `article_id`.

Normalización del diseño de la base de datos

Tal como se presenta el diseño de esta base de datos, supera la **Primera Forma Normal (1FN)** porque todos sus campos tienen valores atómicos, también la **Segunda Forma Normal (2FN)** porque no existen

dependencias parciales dado que haciendo `article_id->keyword` no nos devuelve los mismos valores que a partir de `keyword->article_id`.

Por lo que en última instancia, también está en **Tercera Forma Normal (3FN)** dado que no existen dependencias transitivas.

Una vez diseñada la base de datos, sólo nos queda crearla mediante la terminal de *SQLite*. En este caso hemos optado crearla mediante los comandos de la consola. En la página oficial disponemos de un tutorial donde nos enseñan los comandos básicos para ello [22].

Creación de la base de datos

Abrimos la terminal de *SQLite* y escribiremos esta serie de comandos:

```
sqlite3> .open gadipedia.db
sqlite3> attach 'gadipedia.db' as 'gadipedia';
sqlite3> CREATE TABLE gadipedia.keywords_search(
sqlite3> keyword VARCHAR(50) NOT NULL,
sqlite3> article_id VARCHAR(100) NOT NULL,
sqlite3> PRIMARY KEY(keyword, article_id)
sqlite3> );
```

O si preferimos escribirlo todo en una misma línea:

```
sqlite3> CREATE TABLE gadipedia.keywords_search(keyword VARCHAR(50) NOT NULL, article_id
VARCHAR(100) NOT NULL, PRIMARY KEY(keyword, article_id));
```

Para asegurarnos que todo está correctamente, introducimos

```
sqlite3> .tables
```

Y nos tendría que salir la tabla `gadipedia.keywords_search` en la consola.

Como nota a tener en cuenta, la base de datos se crea en un fichero `.db` en el mismo directorio en el que esté situado el programa de la terminal de *SQLite* por defecto. Necesitaremos mover a mano este fichero al directorio del proyecto de *Unity* para poder hacer eso de él, pero si no queremos tener que moverlo a mano, lo único que tendríamos que hacer es que cuando usemos el comando `.open` para crear la base de datos, le agregamos la ruta completa al fichero `.db`. Por ejemplo:

```
sqlite3> .open C:/path/to/Unity/project/gadipedia.db
```

Y exactamente lo mismo si queremos abrir una base de datos creada previamente por *SQLite* pero que no esté ubicada en el mismo directorio que la consola.

Ahora para llenar la tabla `gadipedia.keywords_search`, solo tendremos que ir usando desde la consola el comando:

```
sqlite3> INSERT INTO "gadipedia.keywords_search" VALUES('keyword_id','article_id');
```

Es o, como hice yo en mi caso, creé una tabla en *LibreOffice Calc* (también valdría otro software tipo *Microsoft Office Excel* que pudiese importar a .csv separado por comas) con los dos campos, la rellene poco a poco, y después la importé a formato .csv separado por comas. Esto es realmente útil para poder crear una tabla visualmente y poder asegurarse de que no haya fallos antes de meterla en la base de datos. Así nos facilita la detección de errores temprana y no que de otra manera sería más difícil arreglarla una vez dentro de la base de datos en formato .db.

Los comandos usados para importar son:

```
sqlite3> .mode csv
sqlite3> .separator ;
sqlite3> .import C:/path/to/some/csv/file.csv gadipedia.keywords_search
```

Para asegurarnos de que se han importado correctamente, escribimos en la consola:

```
sqlite3> SELECT * FROM gadipedia.keywords_search
```

Una vez hecho todo, los datos se guardan automáticamente en la base de datos contenida en el fichero gadipedia.db y salimos de la consola usando el comando:

```
sqlite3> .exit
```

Una vez terminado con todo el proceso de diseño e implementación de la base de datos, procedemos a trabajar con *Unity* para poder integrar las librerías de .NET que trabajan con *SQLite*.

Este paso fue algo problemático. Primero hacen falta las librerías oficiales proporcionadas por *SQLite* en formato .dll, específicamente el fichero sqlite3.dll. Pero *SQLite* solo provee compilados de la versión de 32 bits de su librería sqlite3.dll, mientras que la de 64 bits tienes que compilarla por ti mismo. Por suerte hay gente que ha liberado para todo el mundo sus propios compilados para 64 bits, como en este plugin de *SQLite* para *Unity*:

https://github.com/codecoding/SQLite4Unity3d/tree/master/Example_unity5/Assets/Plugins

Una vez con las dos librerías sqlite3.dll según para el tipo de sistema, dentro del proyecto en *Unity* los organizamos dentro de los *Assets* en la carpeta *Plugins*. Dentro de este directorio, si ya no lo habíamos creado previamente, creamos las carpetas *x86* y *x64* e introducimos los sqlite3.dll en el directorio correspondiente a su sistema.

Una vez hecho esto, aún no hemos acabado el proceso de integración de librerías, tenemos que especificarle a *Unity* cuando usar estas librerías. Así que dentro del visor del proyecto de *Unity*, hacemos click en un fichero sqlite3.dll y en el Inspector le indicamos cuando deben ejecutarse. En el caso del fichero para 32 bits le indicamos que solo se ejecute en sistemas de 32 bits, y para el otro solo en sistemas de 64 bits.

Si no hiciéramos este proceso, cada vez que intentábamos usar la librería contenida en nos saldría un error en sistemas de 64 bits porque intentaría siempre ejecutar la librería de 32 bits al intentar acceder a una base de datos y saldría un error.

La solución para este error me resultó bastante costoso encontrarlo en mi caso (uso el editor de 64 bits de *Unity*), pero gracias al seguimiento de una entrada en el foro oficial de la comunidad de *Unity* [80] logré encontrar la solución especificada arriba.

Y esto no es todo, aún nos hacen falta librerías para poder usar Clases y métodos que trabajen con las bases de datos en formato .db en los *scripts* dentro de *Unity*. En mi caso opté por usar las librerías Mono.Data integradas en *Monodevelop* (el IDE de desarrollo que he usado para este proyecto) en vez de System.Data, para hacer uso de ellas primero tenemos que ir al directorio de librerías donde tengamos instalado *Monodevelop*. En este caso la dirección resultará más o menos así:

```
C:\Program Files\Unity\Editor\Data\Mono\lib\mono\2.0\Mono.Data.dll  
C:\Program Files\Unity\Editor\Data\Mono\lib\mono\2.0\Mono.Data.Sqlite.dll  
C:\Program Files\Unity\Editor\Data\Mono\lib\mono\2.0\Mono.Data.SqliteClient.dll
```

Y los copiamos dentro de la carpeta *Assets/Plugins* de nuestro proyecto en *Unity*.

Implementación del menú de Gadipedia

Ya por fin, podremos empezar a programar el *script* que haga uso de la base de datos, en mi caso al estar usando C# y librerías .NET conjuntamente con Mono.Data (por usar el IDE *Monodevelop*), luego busqué un tutorial [63] para aprender a realizar mi *script*. Dicho *script* se llama **GadipediaSearchManager.cs**, que se encarga tanto la gestión de la UI gráfica del objeto **GadipediaMenuPanel/SearchPanel** como de la conexión a la base de datos *gadipedia.db*.

```
1 //Corrutina que se ejecuta al pulsar el botón de Buscar en el buscador de  
2 //la gadipedia  
3 private IEnumerator WaitForResults(){  
4     //Obtenemos la cadena de texto introducida en el campo de texto del  
5     //buscador  
6     string _searchText = searchInputField.text;  
7     //Eliminamos los espacios en blanco, saltos de línea y tabuladores  
8     //que hay en los extremos de la cadena  
9     _searchText = _searchText.Trim();  
10    //Si la cadena introducida no está vacía  
11    if(_searchText != ""){  
12        //Hacemos que los caracteres de la cadena pasen a minúscula  
13        string _lowercase = _searchText.ToLower();  
14        //Removemos los acentos de los caracteres  
15        string _noAccents = RemoveDiacritics(_lowercase);  
16        //Separamos la cadena en palabras separadas por espacio,  
17        //coma, punto, dos puntos, tabulador y salto de línea  
18        string[] _keywords = _noAccents.Split(new char[] {' ', ',',  
19            '.', ':', '\t', '\n'});
```

```

17
18     //Comprobamos si hay al menos una palabra despues de
19     //separar la cadena
20     if(_keywords.Length != 0){
21         //Para cada palabra introducida, buscamos si está
22         //en la base de datos
23         foreach(string keyword in _keywords){
24             StartCoroutine(FindArticleIDs(keyword));
25             do{
26                 yield return null;
27                 //Esperamos a que termine la bú
28                 //squeda antes de que continúe
29                 //esta corrutina
30             }while(isSearchinInDB);
31         }
32
33         //Para cada ID de artículo encontrado
34         foreach(string ID in foundedIDs){
35             //Comprobamos si ya existía en el
36             //Dictionary de los IDs de artículos
37             if(articleIDs.ContainsKey(ID)){
38                 //Si ya existía, hacemos que el
39                 //campo Value (número de veces que
40                 //ha aparecido este ID en la bú
41                 //squeda) aumente en uno
42                 int _counter = articleIDs[ID];
43                 _counter++;
44                 articleIDs[ID] = _counter;
45             }
46             else{
47                 //Si no existía, lo añadimos en el
48                 //Dictionary de IDs
49                 articleIDs.Add(ID, 1);
50             }
51
52             //Para cada campo en el Dictionary, mandamos que se
53             //cree un prefab a partir del ID de artículo
54             //correspondiente, todo ordenado mediante el orden
55             //de aparición de más veces a menos veces
56             foreach(KeyValuePair<string, int> article in
57                 articleIDs.OrderByDescending(i => i.Value)){
58                 CreateSearchResultElement(article.Key);
59             }
60
61             searchingBlock.SetAsFirstSibling();
62         }
63
64         //Reseteamos los contenidos de IDs de las palabras clave y
65         //los artículos
66         articleIDs.Clear();
67         foundedIDs.Clear();
68         //Mostramos el panel de resultados
69         ShowResultsPanel();
70     }
71 }

```

```

59 //Corrutina que busca los artículos de la gadipedia a partir de las
60     //palabras clave
61 private IEnumarator FindArticleIDs(string keyword) {
62     //Indicamos que se está realizando una búsqueda (para evitar que el
63         //jugador pueda volver a hacer click mientras se esté realizando
64         //la búsqueda)
65     isSearchingInDB = true;
66
67     //Obtenemos la ID de la palabra clave a partir de los archivos
68         //LocatedTexts.xml
69     string _keywordID = LocatedTextManager.GetGadipediaKeywordID(
70         keyword);
71     //Inicializamos una conexión con la base de datos para poder
72         //introducirle comandos
73     dbCommand = dbConnection.CreateCommand();
74     //Escribimos el comando en formato MySQL de que nos busque todos
75         //los ID de los artículos relacionados con la palabra clave
76     dbCommand.CommandText = "SELECT article_id FROM keywords_search
77         WHERE keyword=' " + _keywordID + " '";
78     //Inicializamos el lector de datos de resultados de la base de
79         //datos y hacemos que ejecute el comando
80     dbReader = dbCommand.ExecuteReader();
81     //Hacemos que espere dos décimas de segundo para asegurarnos que
82         //obtiene todos los resultados antes de seguir la corrutina
83     yield return new WaitForSeconds(0.2f);
84
85     //Bucle que se ejecutará por cada resultado obtenido
86     while(dbReader.Read()){
87         //Añade el resultado actual a la lista de IDs de artículos
88             //encontrados
89         foundedIDs.Add(dbReader.GetString(0));
90         //Esperamos una décima de segundo a que se obtenga el
91             //resultado para que no haya problemas de lentitud a la
92                 //hora de leer de la base de datos
93         yield return new WaitForSeconds(0.1f);
94     }
95
96     //Cerramos el lector de la base de datos
97     CloseDBReader();
98     //Esperamos dos décimas de segundo a que se cierre correctamente el
99         //lector de la base de datos
100     yield return new WaitForSeconds(0.2f);
101
102     //Indicamos que ya se ha terminado de realizar la búsqueda
103     isSearchingInDB = false;
104 }
```

Luego una vez que el *script* GadipediaSearchManager.cs obtiene los IDs de los artículos a partir de las palabras introducidas en el buscador Gadipedia, el *script* GadipediaResultsManager.cs que se encarga de inicializar un *prefab* **GadipediaSearchResultElementButton** por cada ID de artículo resultante dentro del panel de resultados de búsqueda de la Gadipedia, y finalmente muestra el submenú de resultados de la Gadipedia. Dependiendo de si ha habido al menos un resultado positivo en la búsqueda de IDs de artículos, se mostrará en este submenú un panel de resultados, en caso contrario

no se mostrará y saldrá un texto indicando que no se han encontrado resultados.

```
1 //Método para crear nuevos objetos de resultados a partir de las IDs de los
2 //artículos
3 private void CreateSearchResultElement(string ID) {
4     //Instanciamos un objeto a partir del prefab
5     GameObject _newArticle = Instantiate(Resources.Load<GameObject>(
6         pathPrefab + "GadipediaSearchResultElementButton"));
7     //Lo inicializamos
8     _newArticle.name = "Article_" + ID;
9     _newArticle.transform.SetParent(contentPanel, false);
10    //Y finalmente le decimos que inicialice sus propios datos a partir
11    //del ID recibido
12    _newArticle.GetComponent<GadipediaSearchResultElementButton>().
13        Initialize(ID);
14 }
15
16 //Método para mostrar el submenú de resultados
17 public void ShowPanel() {
18     //Obtiene el número de resultados que hay que mostrar
19     int _countSearchResults = contentPanel.transform.childCount;
20
21     //Resetea los objetos del submenú de resultados a sus valores
22     //iniciales
23     ResetResultsPanel();
24
25     //Si el resultado es menor que la cantidad de resultados necesaria
26     //para mostrar la scrollbar, se inhabilita el scrollbar
27     if (_countSearchResults <= maxArticlesInPanelWithoutScrollbar) {
28         resultsPanelScrollbar.gameObject.SetActive(false);
29     }
30
31     //Si los resultados obtenidos son cero, desactiva el panel de
32     //resultados
33     if (_countSearchResults == 0) {
34         resultsEntriesPanel.SetActive(false);
35     }
36
37     //Muestra el submenú de resultados
38     resultsPanel.transform.SetAsLastSibling();
39 }
```

Fragmento de GadipediaSearchResultElementButton.cs con el método Initialize:

```
1 //Método que inicializa los datos del objeto al que está unido este script
2 //con los datos sacados a partir del ID del artículo pasado como parámetro
3 public void Initialize(string ID) {
4     //Ejecuta el método Start heredado de UIGenericButton
5     base.Start();
6
7     //Le asignamos el ID del artículo
8     articleID = ID;
```

```

9     //Le asigna tanto al texto del botón como al componente texto para
10    //el resumen del artículo, el texto localizado del ID a partir de
11    //los archivos LocateTexts.xml. En caso de que no los encuentre,
12    //les pone un texto por defecto.
13    try{
14        buttonText.text = LocatedTextManager.GetLocatedText (
15            "GADIPEDIA_ARTICLES", ID, "NAME") [0];
16        briefArticleText.text = LocatedTextManager.GetLocatedText (
17            "GADIPEDIA_ARTICLES", ID, "BRIEF") [0];
18    } catch(NullReferenceException){
19        buttonText.text = "Article name";
20        briefArticleText.text = "Brief text";
21    }
22}

```

Para el caso de que se haya encontrado al menos un único resultado, se mostrará un panel de resultados con uno o más *prefabs* **GadipediaSearchResultElementButton**, en el que cada uno tendrá un *script* GadipediaSearchResultElementButton.cs que el *script* GadipediaResultsManager.cs habrá inicializado a partir de los textos obtenidos del ID de los artículos al pasarlo por los LocateTexts.xml.

Una vez hayamos decidido en el panel de resultados que artículo queremos leer, al hacer click en uno de los **GadipediaSearchResultElementButton** llamará al *script* GadipediaEntry.cs el ID de artículo que tiene guardado, y así GadipediaEntry.cs mostrará el submenú de artículo completo de la Gadipedia con los textos del artículo de la ID que se le ha pasado.

Fragmento con el método que ejecutamos en GadipediaSearchResultElementButton.cs al clickar un artículo

```

1 protected override void ActionOnClick (){
2     //Llama al evento para que se muestre el submenú de artículo con el
3     //ID de artículo que se le ha pasado
4     showArticleContent(articleID);
5 }

```

Fragmento de GadipediaEntry.cs en el que mostramos cómo se inicializa el submenú del artículo completo

```

1 //Al iniciarse el objeto al que está adjunto este script, se inicializa el
2 //listener de eventos
3 private void OnEnable(){
4     GadipediaSearchResultElementButton.showArticleContent += ShowEntry;
5     //....
6 }
7 //Cuando se desactiva el objeto al que está adjunto este script, se
8 //desactivan los listeners de eventos
9 private void OnDisable(){
10    GadipediaSearchResultElementButton.showArticleContent -= ShowEntry;
11    //....
12 }

```

```

12
13 //Método que se encarga de inicializar el submenú del artículo con el ID
14 //que se le ha pasado por el evento, y lo muestra
15 public void ShowEntry(string ID){
16     List<string> _articleContent = new List<string>();
17     string _finalArticleContentText = "";
18     //Asignamos al submenú el ID que se le ha pasado
19     currentID = ID;
20     //Obtenemos el nombre y el contenido de los textos localizados del
21     //artículo
22     articleNameLabel.text = LocatedTextManager.GetLocatedText("GADIPEDIA_ARTICLES", ID, "NAME")[0];
23     _articleContent = LocatedTextManager.GetLocatedText("GADIPEDIA_ARTICLES", ID, "CONTENT");
24
25     //Para todas las líneas de texto que haya en el artículo, las
26     //separa con un salto de línea
27     foreach(string line in _articleContent){
28         _finalArticleContentText += line + "\n";
29     }
30
31     //Lo asigna finalmente al texto del objeto del submenú
32     articleContentText.text = _finalArticleContentText;
33
34     //Obtiene la longitud del texto del artículo
35     int _countCharactersArticleContent = _finalArticleContentText.
36         Length;
37
38     //Si supera los 182 caracteres, muestra la scrollbar para poder
39     //leer el texto, en caso contrario, la desactiva
40     if(_countCharactersArticleContent < 182){
41         entryScrollbar.gameObject.SetActive(false);
42     }
43     else{
44         entryScrollbar.ResetPosition();
45     }
46
47     //Lleva al frente y muestra en pantalla el submenú del artículo
48     articleEntryPanel.SetAsLastSibling();
49 }

```

Hay que decir que dentro de los submenús que contiene Gadipedia, en todo momento se puede volver al submenú previo con un botón de Volver atrás, o salirse del menú completamente mediante el botón Volver a Menú.

3.6.10. Búsqueda de caminos

Para que el jugador pudiera mover el personaje principal por los niveles jugables, era necesario delimitar las zonas transitables de cada nivel, de esta forma al clickar en el suelo el personaje se movería pero no si se hiciera click en la pared.

En un principio para niveles con zonas transitables de formas simples valía con crear un objeto **Navigation** con un componente PolygonCollider2D (con el polígono en forma de la zona transitable) y poniéndole al objeto desde el panel del Inspector en sus parámetros Tag el valor Navigation-Polygon y en Layer el valor Navigation. De esta manera la zona transitable era detectada por el script MouseClickHandler.cs y al hacer click izquierdo sobre ella, llevaba al personaje principal siguiendo un camino en línea recta hasta la posición clickada.

El problema vino cuando la forma de las zonas transitables se volvió compleja. Por ejemplo, en el nivel del Despacho del Profesor de la demo técnica de **1812: La aventura**, la zona transitable era una especie de U invertida. Al hacer click de un extremo de la U invertida a otra, como el personaje principal solo se podía mover en línea recta, atravesaba obstáculos donde no debía.

Para poder arreglar este problema, tuve que crear un sistema de búsqueda de caminos para el personaje principal. *Unity* ofrecía su propio sistema de navegación llamado NavMesh, pero como solo funcionaba bien en 3D (en 2D tenías que sortear una serie de problemas para hacerlo funcionar correctamente simulando que el escenario es 3D) y al ser la demo técnica de **1812: La aventura** una aventura gráfica 2D de corte clásico, quise crear un sistema de búsqueda de caminos similar al que usaban las antiguas aventuras gráficas 2D pero mediante los componentes que proporcionaba *Unity*.

Lo primero fue buscar documentación sobre cómo se realizaban estos sistemas y lo encontré aquí [23]. Explica que lo que hacían las antiguas aventuras gráficas era crear varios polígonos (a ser posible, polígonos simples de cuatro lados) que designaban las zonas transitables del nivel, luego los unían mediante puntos en el escenario.

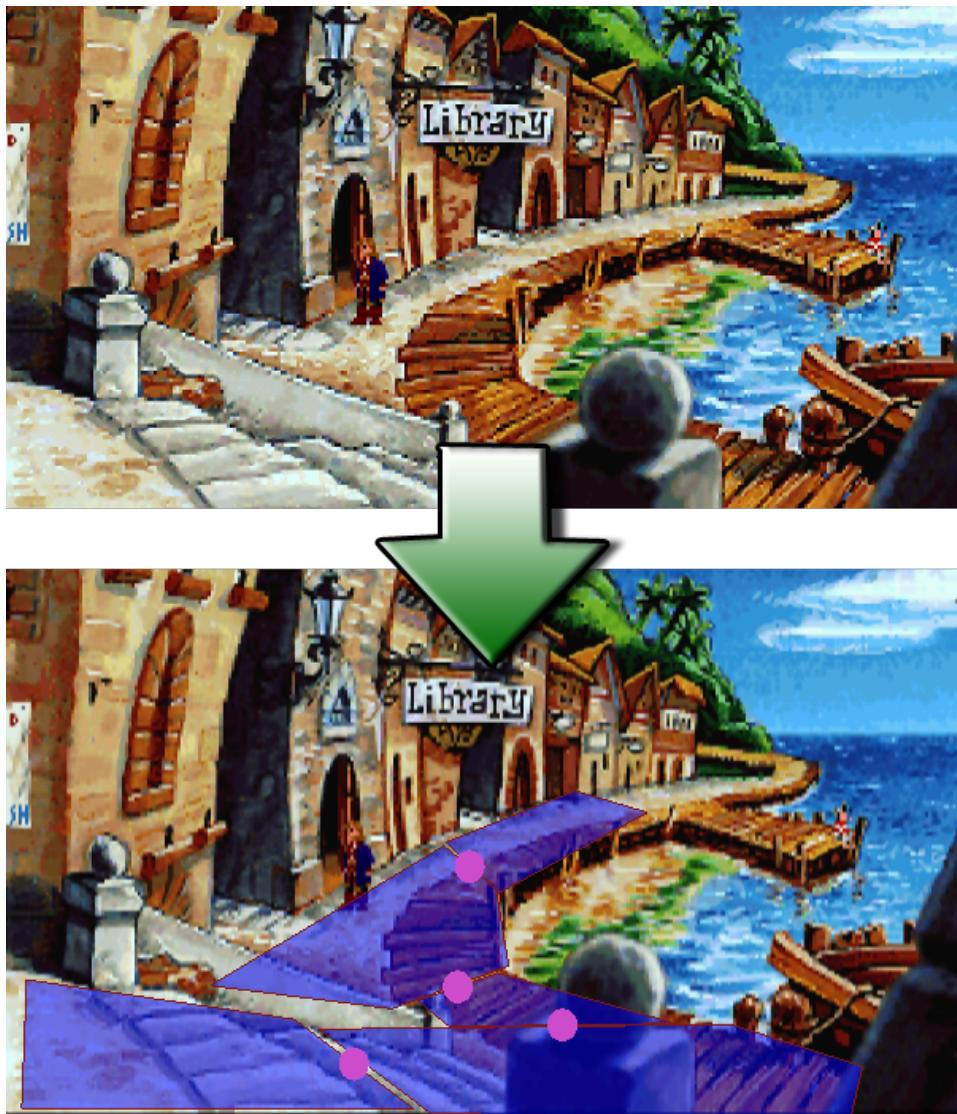


Figura 3.19: Zonas transitables y sus enlaces en un nivel de The Secret of the Monkey Island

A partir de ahí, de cada polígono de zona transitble sacaban su centroide [145] para crear más puntos que luego el sistema de encargaba de juntarlos con los puntos de unión para crear el grafo predefinido con las rutas posibles del nivel. De esta manera, cada vez que se haga click el sistema realiza lo siguiente:

1. Obtiene la posición del personaje principal y en qué zona transitble está.
2. Se añade al grafo predefinido un punto con la posición del personaje principal, unido mediante un arco al centroide de la zona transitble en la que está.
3. Obtiene la posición del punto al que se le ha hecho click y en qué zona transitble está.
4. Se añade al grafo predefinido un punto con la posición del sitio que se le ha hecho click, unido mediante un arco al centroide de la zona transitble en la que está.
5. Mediante un algoritmo de búsqueda de caminos, se calcula en el grafo la ruta más corta entre las dos zonas transitables.

6. El algoritmo de búsqueda de caminos devuelve una lista de las posiciones por las que tiene que pasar el personaje principal para llegar a su objetivo.

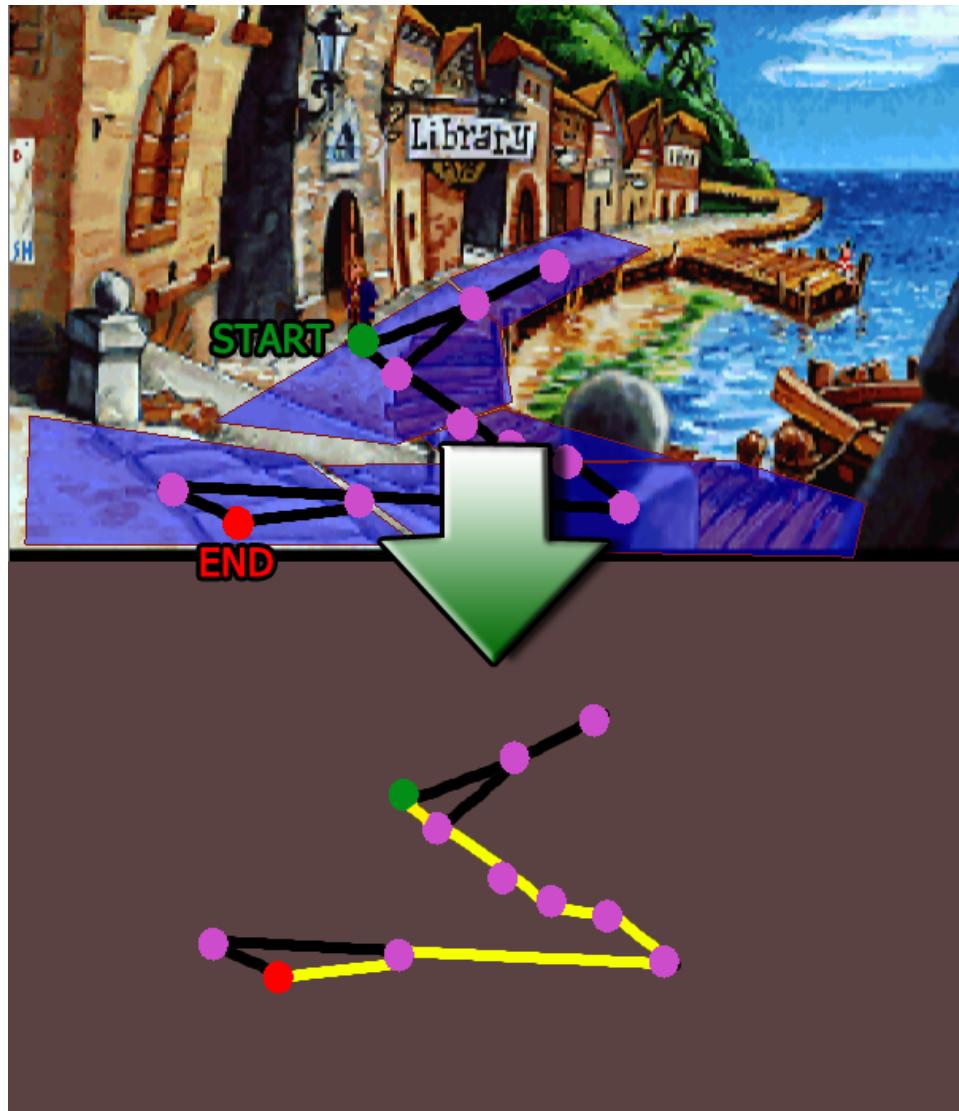


Figura 3.20: Grafo calculado en la búsqueda de caminos en un nivel de The Secret of the Monkey Island

Este fue el sistema elegido para realizar la búsqueda de caminos para el personaje principal. Me permitió realizar algunos cambios dado que *Unity* con sus componentes *PolygonCollider2D* y la capacidad de anidar objetos dentro de otros en una escena, facilitaba el trabajo en algunos puntos.

Para la implementación del sistema de búsqueda de caminos, primero tenía que crear en el escenario un **Navigation**, dicho objeto a su vez tenía los siguientes objetos hijo:

Colliders Este es un objeto vacío que servirá para contener los objetos **Collider_n** (donde n es un valor entre 0 y el número total de zonas transitables menos uno).

Los objetos **Collider_n** contienen un componente *PolygonCollider2D* (con el polígono en forma

de la zona transitable) y en sus parámetros Tag el valor NavigationPolygon y en Layer el valor Navigation. Además de tener el *script* NavigationCollider.cs, este *script* se encarga de calcular el centroide del PolygonCollider2D en tiempo de ejecución al inicial el nivel. Por último, con la posición del centroide crea un objeto **Point_n** que lo añade de hijo al objeto **Points**.

Points Este es un objeto vacío que servirá para contener los objetos **Point_n** (donde n es un valor entre 0 y el número total de puntos menos uno).

Los objetos **Point_n** están situados en las posiciones que formarán los puntos dentro del grafo para la búsqueda de caminos, en sus parámetros tienen Tag con el valor NavigationPoint y en Layer el valor Navigation, además tienen de componente el *script* NavigationPoint.cs (si es un punto que conecta dos zonas transitables) o NavigationPointCentroid.cs (si es un punto generado en tiempo de ejecución que indica el centroide de una zona transitable).

Con todos los objetos que intervienen nombrados, el sistema de búsqueda de caminos funciona de la siguiente manera:

1. El *Jugador* hace click en un **Collider_n** del escenario.
2. El *script* MouseClickHandler.cs del objeto **Player** detecta que se ha hecho click en un objeto con el Tag con valor NavigationPolygon.
3. El *script* MouseClickHandler.cs le manda al objeto **Player** que vaya a la dirección a la que se ha hecho click llamando el método GoTo del *script* Player.cs.

```

1 //...
2 case "NavigationCollider":
3     //Si se ha clickado un objeto de zona transitable, no hace nada si
        //estaba usando un objeto del inventario, pero en caso contrario, le
        //manda al objeto Player que vaya a la posición a la que se ha hecho
        //click.
4     if(Player.Instance.isUsingItemInventory){}
5     else{
6         Vector2 _mousePosition = Camera.main.ScreenToWorldPoint(Input.
           .mousePosition);
7         Player.Instance.GoTo(_mousePosition);
8     }
9     break;
10    //...

```

4. Dentro del método GoTo, este llama al método FindPath del *script* NavigationManager.cs del objeto **Navigation** pasándole como parámetros la posición de **Player** como punto de origen, y la posición del punto al que se le ha hecho click.

```

1 //Método GoTo de Player para ir a una dirección
2 public override void GoTo (Vector2 newPosition){
3     StartCoroutine(WaitForGoToInPathCompleted(newPosition));
4 }
5

```

```

6 //Corrutina que usa Player para ir a una dirección mediante el calculo
7 //de una ruta
private IEnumerator WaitForGoToInPathCompleted(Vector2 newPosition) {
8     //Si la posición a la que ir no es la misma que se tenía de
9     //objetivo anteriormente
10    if(originalTargetedPosition != newPosition){
11        do{
12            yield return null;
13            //Esperamos a que Player termine de ejecutar
14            //otras acciones
15            while(isSpeaking && isInteracting && isInConversation
16                );
17
18            //Si estaba caminando previamente, abortamos la ruta
19            //que estaba realizando antes
20            if(isWalking){
21                StopCoroutine(walkingCoroutine);
22                isPositionInPathReached = true;
23            }
24
25            //Reseteamos la dirección de caminar en caso de que
26            //estuviera ya caminando
27            isWalking = true;
28            moveDirection = Vector2.zero;
29            originalTargetedPosition = newPosition;
30            //Obtiene la ruta de posiciones mediante el sistema de
31            //Navigation
32            List<Vector2> _path = NavigationManager.Instance.
33                FindPath(currentPosition, newPosition);
34
35            //Asigna la corrutina que se encarga de hacer caminar
36            //a Player pasando por todos los puntos de la ruta
37            walkingCoroutine = StartCoroutine(WaitForPathCompleted
38                (_path));
39        }
40    }
41 }
```

5. Dentro del método `FindPath`, tanto para las posiciones de origen y destino calcula el ID de la zona interactiva al que pertenecen cada uno.
6. Agrega las posiciones de origen y destino al grafo de caminos predefinido del nivel, enlazándolos al punto del grafo con el mismo ID de la zona accesible más cercana a estas posiciones.
7. Calcula mediante el algoritmo de *Floyd-Warshall*, la ruta más corta en el grafo desde el punto con la posición de origen, hasta el punto con la posición de destino. Como resultado, devuelve una lista con las posiciones contenidas en los puntos del grafo de la ruta obtenida.

```

1 //Método de NavigationManager.cs que calcula la ruta entre dos posiciones
2 public List<Vector2> FindPath(Vector2 originPos, Vector2 destinyPos){
3     //Calcula el ID de la zona transitable de la posición de origen
4     int _originID = GetNearestNavPointID(originPos);
5     //Calcula el ID de la zona transitable de la posición de destino
6     int _destinyID = GetNearestNavPointID(destinyPos);
7 }
```

```

8     //Declaramos una lista de vectores para guardar posteriormente las
9     //posiciones de la ruta
10    List<Vector2> _path = new List<Vector2>();
11
12    //Si las dos posiciones no estan en la misma zona transitable, se
13    //ejecuta el algoritmo que devuelve la ruta
14    if(_originID != _destinyID){
15        _path = FindMinPathPositions(_originID, _destinyID);
16    }
17
18    //Comprueba que si la ruta devuelta solo tiene una posición, eso
19    //significa que la posición destino está en la misma zona
20    //transitable, así que calcula si es más corto ir directamente al
21    //punto de destino pasando por el centroide de la zona transitable
22    //o ir directamente
23    if(_path.Count == 1){
24        float _distanceToPath = Mathf.Abs(Vector2.Distance(
25            originPos, _path[0]));
26        float _distanceToDestinyPos = Mathf.Abs(Vector2.Distance(
27            originPos, destinyPos));
28        if(_distanceToPath > _distanceToDestinyPos) {
29            _path.Clear();
30        }
31    }
32
33    //Añadimos la posicion destino como última posición de la ruta
34    _path.Add(destinyPos);
35
36    //Devolvemos la ruta
37    return _path;
38}

```

Y así es como funciona el sistema de búsqueda de caminos en el **Motor de Aventuras Gráficas 2D de Unity**. Ahora comentaremos brevemente como el script *Player.cs* gestiona la ruta a seguir mediante el método *GoTo*:

1. Por cada posición que haya en la ruta, hace que el objeto **Player** vaya en línea recta llamando hasta la posición actual mediante el método *GoToInStraightDirection*.
2. Espera a que **Player** llegue a la posición.
3. Una vez llegado a la posición, verifica si esa posición es la del destino final (la que se había hecho click). Si no es el destino final, coge el siguiente punto de la ruta, si se ha llegado al final, el objeto **Player** deja de caminar.

```

1 //Corrutina en Player.cs que se encarga de que el objeto Player vaya a
2 //la posición destino pasando por la ruta dada
3 private IEnumerator WaitForPathCompleted(List<Vector2> path){
4     //Si la ruta solo tiene una posición, calcula el vector de
5     //dirección de movimiento del objeto Player y se espera que
6     //llegue al punto y deje de caminar
7     if(path.Count == 1){
8         isPositionInPathReached = false;
9     }
10    Vector2 targetPosition = path[0];
11    Vector2 movementDirection = targetPosition - transform.position;
12    float distanceToTarget = Vector2.Distance(transform.position, targetPosition);
13    float speed = 5f;
14    float elapsedTime = 0f;
15
16    while(elapsedTime < distanceToTarget / speed)
17    {
18        elapsedTime += Time.deltaTime;
19        transform.Translate(movementDirection * speed * Time.deltaTime);
20        yield return null;
21    }
22
23    isPositionInPathReached = true;
24}

```

```

6         currentPathStepPosition = path[0];
7         moveDirection = CalculateMoveDirection(
8             currentPathStepPosition);
9         do{
10             yield return null;
11         }while(!isPositionInPathReached && isWalking);
12     }
13     //En caso de que la ruta posea más de una posición
14     else if(path.Count > 1){
15         for(int i = 0; i < path.Count; i++){
16             //Reseteamos el booleano que indica si ha
17             //llegado a un punto de la ruta
18             isPositionInPathReached = false;
19             //Obtenemos la siguiente posición de la ruta a
20             //la que ir
21             currentPathStepPosition = path[i];
22             //Calculamos el vector de dirección de
23             //movimiento del objeto Player
24             moveDirection = CalculateMoveDirection(
25                 currentPathStepPosition);
26             do{
27                 yield return null;
28                 //Esperamos a que llegue al siguiente
29                 //punto de la ruta
30             }while(!isPositionInPathReached);
31         }
32     }

```

3.7. Pruebas

3.7.1. Pruebas unitarias

Las pruebas unitarias se realizaron junto a la fase de implementación, a medida que se finalizaban los GameObjects con sus correspondientes *scripts* y Componentes. Se optó por un enfoque estructural con pruebas de caja blanca. Se buscaba tomar todas las bifurcaciones posibles, o al menos las más propensas a fallos, en cada GameObject testado. De esta forma todas las sentencias se ejecutarían al menos una vez y los posibles fallos saldrían a relucir con mayor facilidad.

Así mismo, también se procuró que el sistema no accediese a memoria no inicializada a través de GameObjects o Componentes no inicializados o textos no incluidos en los XML.

El mayor número de errores encontrados estaban relacionados con los accesos inválidos a GameObjects y/o Componentes no inicializados y el acceso erróneo por de los textos de los XML. En este caso, el depurador de *Monodevelop* y la clase *Debug* con los métodos *Log* y *.LogError* (mediante la inclusión de instrucciones *try catch*) fueron de gran ayuda para arreglar estos fallos. Así mismo, el sistema de detección de colisiones presentó varios defectos en algunos de los tests de colisión por la complejidad que entrañaban al tener que trabajar con distintas profundidades en un entorno 2D para evitar que se solapasen los PolygonCollider2D de los GameObjects interactivos y tratar luego por otra parte, los elementos de la interfaz.

3.7.2. Pruebas de integración

A medida que el desarrollo de varios módulos de un mismo subsistema finalizaba, se procedían a realizar pruebas de integración entre dichos módulos empleando una aproximación de caja negra. Interesaba que el sistema realizara la tarea para la que había sido diseñado de forma correcta. Cuando todos los módulos del sistema estuvieron listos se realizaron pruebas adicionales de integración a mayor escala. No sólo entre los módulos de un mismo subsistema, sino el funcionamiento conjunto de varios subsistemas.

De nuevo, el mayor número de problemas fue localizado en los accesos inválidos a GameObjects con *scripts* y/o Componentes no inicializados correctamente.

3.7.3. Pruebas de jugabilidad

Una vez se compilaron las primeras versiones usables de la demo técnica de **1812: La aventura** se pidió mediante el uso de las redes sociales, foros de desarrollo, y la página y blog del proyecto a colaboradores externos y jugadores anónimos que probaran el videojuego. Tras estas sesiones se les preguntó (o ellos me comentaron directamente a través de las redes) por aspectos como la capacidad de respuesta del control con el ratón al clickar muy rápido, la comodidad de los controles, la fluidez de las animaciones al caminar con el personaje principal, si el sistema de diálogos les resultaba agradable y otros parámetros.

Posteriormente, se analizaron las respuestas de todos los colaboradores y se llevó a cabo un proceso de balanceo. En el caso de la respuesta del control mediante el ratón, se limitó que solo se pudiera clickar

sobre los elementos cada medio segundo, para evitar que se pudiera clickar demasiado rápido sobre los objetos de forma que no les daba tiempo a actualizar sus estados debidamente y ocurrían errores como solapamiento de diálogos y/o acciones, o que un objeto del escenario que sólo se podía coger una veces para meterlo en el inventario, se incluía varias veces.

También en el caso del Sistema de diálogos, se volvió a rehacer la forma de formatear las líneas de texto en *LocalizedTextManager* de los ficheros XML con los diálogos, para que el tamaño de las cajas de diálogos fuera personalizable dependiendo del número de caracteres máximo por línea, y el número de saltos de línea máximo por caja. Con esta nueva versión del formateo de líneas de texto para diálogos, hubo que crear un script llamado *RelocateTextPosition* en caso de que las cajas de diálogo se saliesen por los bordes de la pantalla y se reajustasen para que se pudieran ver bien dentro de la pantalla.

Por último, se mejoró el método con el que se escogía la dirección a la que miraba el personaje principal (aunque aplicado también a los actores en general, porque era un método de la clase *Actor*) al caminar, de forma que cuando caminaba distancias más pequeñas que el ancho de su *SpriteRenderer*, dependiendo de en qué mitad del ancho estuviera la dirección a caminar, giraba de un lado a otro.

3.7.4. Pruebas de interfaz

Tras completar las pruebas de jugabilidad, se procedió a realizar pruebas sobre la interfaz del juego. Tanto en los menús como en la propia pantalla de juego. Estas pruebas consistieron principalmente en comprobar si se reseteaban las posiciones, textos y parámetros de los menús cada vez que se entraba en un submenú y se salía de este, comprobar que la interfaz fuera intuitiva y que se reajustaba automáticamente ante los cambios en los ajustes de pantalla, audio e idioma del juego. Se probó que estuviese bloqueado el acceso al menú de opciones mientras el jugador ejecutaba una acción en la partida, tratar de cargar y borrar partidas de un bloque de guardado vacío, bloquear los botones cuando saltase una ventana modal hasta que eligiésemos una respuesta, etc.

Se demostró que la interfaz era bastante estable ante los cambios de los ajustes, que se reseteaban bien los submenús a cerrarlos y volver a abrirlos, y que la interfaz resultaba bastante intuitiva.

Lo único que hizo falta cambiar fue a petición de los colaboradores de que se dejase el inventario de objetos siempre abierto cuando no se estuviese ejecutando ninguna *cutscene* dentro del juego, ya que resultaba incómodo e innecesario estar abriendo y cerrando el inventario continuamente.

Capítulo 4

Conclusiones

Tras haber explicado todo el proceso de desarrollo de la demo de **1812: La aventura** y su motor de aventuras gráficas en *Unity* a lo largo del presente documento, en esta sección haremos una valoración acerca del proyecto. En primer lugar hablaré de como me ha afectado en el plano personal y académico para después tratar los aspectos técnicos. Finalmente finalizaremos con las posibles ampliaciones que podría sufrir en el futuro.

4.1. Conclusiones personales

Con la demo técnica de **1812: La aventura** y su motor de aventuras gráficas he adquirido una gran cantidad de conceptos y me he enriquecido muchísimo como persona y desarrolladora. Si bien ya había trabajado un par de videojuegos sencillos por la asignatura *Diseño de Videojuegos* en la carrera o por talleres independientes que realicé, aparte de haber colaborado en el desarrollo de otros juegos en la realización de gráficos, este sin duda era el reto de mayor envergadura al que me había enfrentado jamás dado que nunca había diseñado ni implementado todas las características técnicas de un juego por mi misma. De ahí que la fase de aprendizaje fuese tan larga al comienzo del proyecto aunque después se extendiera durante todo este tiempo, principalmente a la hora de aprender a usar *Unity UI* y *SQLite* para implementarlos dentro del proyecto.

En los siguientes puntos repasaré de forma superficial lo que he aprendido:

1. Aprender a buscar documentación de otros campos (histórica en este caso):

Si bien al final no se le ha podido dar el uso debido a los conocimientos históricos adquiridos al cambiar de orientación de proyecto, aprendí a buscar en bibliografías y bancos de imágenes históricas para posteriormente cribar su información y obtener los datos que me fueran relevantes para diseñar el puzzle implementado posteriormente en la demo técnica de **1812: La aventura**. Algo que me será tremadamente útil en el futuro cuando tenga que documentarme de material ajeno a mi profesión para poder escribir documentos de diseño en posibles futuros proyectos.

2. Metodologías ágiles para la planificación del proyecto:

La aparición de problemas no tenidos en cuenta por la planificación y versión original de este proyecto, hicieron que tuviera que buscar nuevas metodologías de organización y planificación para

proyectos. Aprendí de las metodologías ágiles que se aplican en la actualidad en los desarrollos de proyectos y las adapté a mis necesidades mediante el programa *Trello*. Adquirir la capacidad de flexibilizar las planificación y/o un proyecto ante los problemas que van surgiendo, es una habilidad que será de utilidad en cualquier campo de desarrollo que me depare en el futuro.

3. Programar un videojuego:

Si bien antes había participado en la creación de videojuegos, nunca había llegado a programar un videojuego completo a excepción de un juego pequeño en *GameMaker* y algún que otro ejemplo básico en *Unity*. Que si bien la demo de **1812: La aventura** no pueda considerarse como un juego completo, se han programado para esta demo todos los requerimientos técnicos que serían necesarios para una aventura gráfica completa. Consideré que el Proyecto Final de Carrera me brindaba una gran oportunidad para por fin afrontar la programación de un videojuego de forma serie y me decidí a intentarlo. Aprendí conceptos y técnicas nuevas de programación para cada parte técnica que constituía un videojuego, algo que seguramente, me será de gran provecho para reutilizarlos en próximos proyectos, sean de videojuegos o no.

4. Lenguaje de programación C# y librerías .NET:

Nunca había programado anteriormente en C#, por suerte gracias a las similitudes que poseía respecto a los lenguajes de C++ (aprendido a lo largo de los cursos de la carrera) y, sobretodo, de Java (enseñado en la asignatura de *Programación Concurrente y Distribuida*) el aprendizaje no fue complicado. Las mayores dificultades vinieron de aprender a usar las librerías .NET de las que hacía uso C# para el manejo de lectura y escritura de datos en ficheros en general y el funcionamiento de los XmlNode para poder trabajar con los nodos de un fichero .xml. De todo esto saqué que C# y las librerías .NET es un lenguaje potente pero más sencillo de usar comparado con C++, y que actualmente es muy usado para el desarrollo ya no solo de videojuegos, sino de aplicaciones en general.

5. Las APIs de Unity 2D, UI y métodos avanzados:

Antes de embarcarme con este proyecto, ya había tenido alguna experiencia previa con *Unity* y sabía moverme por su editor, crear GameObjects junto con algunos componentes básicos como algunos tipos de collider, escribir *scripts* con los métodos genéricos del entorno MonoBehaviour (necesaria para poder adjuntar los *scripts* a los GameObjects), la clase Input para obtener información de los periféricos de entrada, y algun que método extra como el de Ray-Cast. Sin embargo, a finales de 2014, *Unity* lanzó su nueva API para crear proyectos 2D, lo que trajo multitud de componentes para GameObjects totalmente nuevos, nuevas herramientas como el Animator para crear máquinas de estado de animaciones 2D, nuevos métodos y clases como SpriteRenderer, etc. Ocurrió lo mismo posteriormente varios meses después con la llegada de *Unity UI*, el nuevo sistema para crear interfaces gráficas, con el que se añadieron funcionalidades nuevas al editor de *Unity* para crear interfaces y una nueva librería con nuevas clases, métodos, y componentes. No siendo bastante aprender todo esto nuevo, me fue necesario buscar y aprender nuevos métodos avanzados de *Unity* como las CoRoutines para que así las acciones de los actores, en el motor de aventuras gráficas, pudieran sincronizarse y esperar las unas a las otras debidamente.

El aprendizaje de *Unity* fue ardua por tener que adaptarme a los cambios que se fueron introduciendo tanto en el motor como en el motor. Sin embargo, si en el futuro contemplo acabar trabajando desarrollando videojuegos, saber programar y usar *Unity* es uno de los requisitos más demandados entre estos perfiles.

6. Sistema de gestión de bases de datos con SQLite:

Para poder implementar el buscador de información básico detallado en los objetivos a conseguir de este proyecto, necesitaba aprender a usar *SQLite* para poder crear una base de datos relacional

que se pudiera albergar dentro de los *Assets* y no tuviese que recurrir a la creación de un servidor. Si bien fue bastante fácil aprender a instalar el software para mi sistema operativo (Windows 7 de 64 bits) y usar la consola, lo más difícil fue aprender como integrar las librerías de *SQLite* y de *.NET* (para poder hacer uno de las librerías de manejo de base de datos en C# y *Monodevelop*) en ficheros .dll con *Unity*, y no solo eso, había que instalar dos versiones de las librerías para que dependiendo del sistema operativo que ejecutase el juego, escogiera entre la versión para sistemas operativos de 32 o 64 bits. Posteriormente aprendí a usar las librerías *System.Data* y *Mono.Data.SqliteClient* para poder conectarme mediante un *script* dentro de *Unity* a la base de datos, mandarle comandos escritos en MySQL y recoger la información en respuesta al comando. Si bien *SQLite* no es un software muy usado dado que ahora se pretende que todo funcione desde la nube, puede ser útil para el desarrollo de aplicaciones y videojuegos que requieran del uso de bases de datos relacionales pero que puedan hacer uso de servidor remoto para alojarlas.

7. Matemáticas para videojuegos:

Para poder implementar la búsqueda de caminos tuve que repasar conceptos matemáticos de la geometría del espacio y de álgebra, además de incorporar otros nuevos como la fórmula de obtener el centroide de un polígono de n -lados. Todo esto para poder crear los puntos necesarios del grafo para la búsqueda de caminos.

8. Técnicas de IA para la búsqueda de caminos y la máquina de estados de animaciones:

Si bien conocía la existencia de algunas técnicas de IA gracias a la asignatura de *Introducción a la Inteligencia Artificial* y cómo funcionaban las máquinas de estado discretas por la asignatura *Teoría de Autómatas y Lenguajes Formales*, nunca había llegado a poner en práctica estos conocimientos. En el caso de las máquinas de estado, su uso fue para crear una *FSM* (*Finite State Machine*) en el editor *Animator* de *Unity* para crear animaciones y transiciones entre ellas mediante los cambios de estado que le indicáramos. Por el otro lado, la búsqueda de caminos lo empleé para que el personaje jugable siguiera rutas preestablecidas por el escenario y no caminase únicamente en línea recta, pudiendo así evitar que salieran fallos de atravesar obstáculos sin querer y poder realizar escenarios con zonas libres para caminar con formas complejas.

9. Diseño de un videojuego:

Anteriormente para la asignatura de *Diseño de Videojuegos* y para un taller universitario online de *Introducción al desarrollo de videojuegos* de la Universidad Internacional de Andalucía ya había creado documentos que detallaban los aspectos básicos de un videojuego: forma de jugar, controles, gráficos y sonidos, etc. Nunca había hecho hasta ahora un documento de diseño de juego (*GDD* o *Game Design Document*) tan completo y elaborado como hasta ahora. Resultó ser de enorme utilidad ya que ha ayudado al proceso de análisis y diseño. Además, los colaboradores iniciales pudieron saber las necesidades del proyecto en cuanto a recursos artísticos se refiere. Los integrantes iniciales del equipo conocían en todo momento el estilo de juego y la apariencia que debía tener **1812: La aventura**.

10. Trabajo en equipo:

En **1812: La aventura** el trabajo en equipo fue inicialmente muy importante, se hizo mucho más evidente ya que colaborábamos al principio seis personas especializadas en disciplinas muy distintas. Hubo que realizar labores de coordinación, comunicación y resolución de conflictos (sobre todo relacionados con el diseño del puzzle y el apartado gráfico para que con las limitaciones de tiempo de la artista, se llegara a unos mínimos estándares de calidad y que a su vez, se pudieran al máximo los gráficos). Lamentablemente solo duró el principio del proyecto antes de la reorientación del proyecto a motor de aventuras gráficas en 2D para *Unity*. Pero igualmente, fue una experiencia muy enriquecedora y que, sin duda, me ayudará en mi futuro profesional.

11. Aplicación de conocimientos:

El desarrollo de este proyecto me ha sido especialmente útil para poner en práctica aquellos conocimientos adquiridos durante la Ingeniería Técnica. Sobre todo me refiero a aquellos relacionados con la Ingeniería del Software, Bases de Datos y Estructuras de Datos II.

4.2. Conclusiones técnicas

En la demo técnica de **1812: La aventura** para crear el **Motor de Aventuras Gráficas 2D en Unity** se han cumplido con los objetivos propuestos en el capítulo introductorio de esta memoria de Proyecto Fin de Carrera. En el proyecto hemos conseguido:

- Diversos scripts de programación, que hacen de clases maestras para los distintos elementos interactivos del escenario. De esta manera, los elementos interactivos básicos solo necesitarán cambiar los valores de un par de parámetros para ser funcionales. Y en el caso de ser elementos que requieran hacer funciones más complejas, se puede crear un *script* que herede de estos y extender las funcionalidades de los métodos iniciales, permitiendo que se reutilice gran parte del código y sólo implementar lo estrictamente necesario para el nuevo elemento.
- Un sistema mediante ficheros .xml que se encargue de procesar los textos descriptivos y conversacionales, en el que pasando como parámetros el nivel del juego, el objeto o personaje, y como parámetro opcional si este ha sido manipulado previamente, y así mostrarlos por pantalla. Creando un sistema independiente del juego, para que así se pueda modificar los textos, como para posibles traducciones futuras.
- Un sistema de elección de respuesta, de forma que dependiendo de la respuesta elegida, podemos hacer que ocurra una acción u otra.
- Un sistema de inventario, en el que los objetos que se pueden coger del escenario, se muestren en el inventario y al ser clickados puedan ser usados sobre otros elementos del escenario.
- Implementación de menús y submenús y elementos de la interfaz durante el juego empleando la tecnología UI de *Unity*.
- Un sub-menú que implemente un sistema de guardado de partidas en cualquier momento. Dichas partidas serán guardadas en ficheros en formato .xml.
- Un sub-menú que incluya un pequeño buscador básico de información para facilitar al jugador la resolución de puzzles usando una pequeña base de datos mediante SQLite, y luego que aparezca otro sub-menú con los resultados ordenados de mayor a menos relevancia. Y que al clickar en un resultado, nos salga un artículo con información sobre Cádiz.
- Un sub-menú que implemente un mapa que permita al jugador trasladarse a otros escenarios del juego en cualquier momento, siempre que lo permita la historia.
- Un sistema de configuración de ajustes que permite mediante una interfaz, modificar ajustes de audio, gráficos y lenguaje y que estos se guarden en un fichero settings.xml. De forma que para la siguiente vez que abramos el juego, mantengan los mismos ajustes.

- Un algoritmo de búsqueda de caminos para que el personaje controlable pueda moverse durante los distintos escenarios e interactuar con ellos mediante la pulsación con ratón del sitio o elemento objetivo al que queremos dirigir al personaje.
- Un sistema de *cutscenes* (o escenas narrativas), que haga que se ejecuten acciones y eventos automáticamente y que cuando terminen, permitan que el jugador siga jugando.
- Implementar un sistema de audio básico para poder escuchar, parar de escuchar, o saber si se está escuchando una canción y/o sonido concreto.

Al estar el repositorio del proyecto alojado en *GitHub*, este incluye un generador de estadísticas básicas que incluyen un calendario de contribuciones dividido por los miembros participantes del repositorio, tráfico de visitas del proyecto, calendario básico de *commits* realizados, otro calendario con la frecuencia de código añadido y borrado al proyecto, los días y horas de la semana donde más *commits* se han realizado, historial de *branches* y *commits* del proyecto y finalmente miembros que han colaborado en el repositorio. Para ver este tipo de estadísticas para el proyecto de **1812: La aventura**, la dirección es <https://github.com/Firenz/1812/graphs>

Como comentario final decir que se realizaron pocos commits, lo cual puede hacer el seguimiento un poco difícil, pero *GitHub* incluye una herramienta `diff` para saber las diferencias de código que ha habido, además la intención de realizar tan pocos commits vino de querer subir las *builds* más estables del proyecto para que pudieran ser probados por jugadores potenciales de **1812: La aventura** y pudieran comentar al momento los fallos y las sugerencias para la próxima *build* del proyecto.

4.3. Trabajos futuros

Es cierto que los objetivos que nos marcamos al comienzo del desarrollo del proyecto (después de haber sido reorientado a demo técnica de motor de aventuras gráficas 2D para *Unity*) han sido cumplidos satisfactoriamente. Sin embargo, en cuanto a la parte de diseño del juego, la propuesta inicial de hacer un juego completo tuvo que ser acortada a una demo técnica y a un único puzzle, además de haber aún algunos puntos respecto a la programación en los que el proyecto podría mejorar. A continuación hacemos una lista de las posibles mejoras de **1812: La aventura**:

- Incluir nuevos escenarios, personajes y puzzles. En definitiva realizar el diseño completo del juego.
- Mejor integración de los datos históricos dentro de la historia y de los puzzles de **1812: La aventura**.
- Mejora del script de recolocación de las cajas de diálogos dentro del juego, para que no se vea un parpadeo de cómo se mueven.
- Mejora del buscador de artículos históricos, en el que aparte de buscar por palabras clave, se incluya que filtre y busque a partir de frases contenidas en los artículos existentes.
- Mejora del sistema de búsqueda de caminos, de forma que coja la posición del personaje jugable como un vértice y genere un grafo de caminos nuevo con ello, en vez de coger la posición del vértice pre-inicializado más cercano a la posición del personaje jugable y que este devuelva un camino pre-inicializado al que se le ha añadido como punto de partida esta nueva posición.

- Conseguir convertir las clase `ActorAnimStateMachine` y `Actor` a plantillas genéricas, de forma que al crear nuevos *scripts* heredados para la creación de nuevos personajes a partir de estas clases, sólo hubiera que expandir los métodos existentes en vez de reescribirlos de cero o acabar en desuso (lo cuál es una mala práctica).
- Mejorar la clase `AudioManager` para que pudiera incluir opciones de pausa en sonidos y música.

Capítulo 5

Software utilizado

En esta sección hablaremos de las herramientas utilizadas durante el desarrollo de **1812: La aventura**. Para cada herramienta ofreceremos una pequeña descripción de sus funcionalidades y adjuntaremos las razones por las cuales han sido elegidos para el desarrollo frente a otros de similares características.

Unity

Unity, también conocido como *Unity3D* (o desde que sacó su motor 2D también puede llamarse *Unity2D*) es un motor de videojuegos multiplataforma creado por *Unity Technologies* con editor de niveles integrado, que puede ser instalado tanto en Windows como en OS X. Permite crear juegos para una gran variedad de plataformas: Windows, OS X, Linux, Xbox 360, PlayStation 3, Playstation Vita, Wii, Wii U, iPad, iPhone, Android y Windows Phone. Y sin la necesidad de crear un proyecto específico para cada plataforma.

Otra ventaja, es la existencia de una versión gratuita del motor, que ahora es prácticamente idéntica a la de pago desde el lanzamiento de la versión 5 de *Unity*. Permitiendo que pequeños, medianos, e incluso grandes, desarrolladores puedan acceder a este motor, ahorrándoles el tiempo y coste que supone crear un motor propio o el pagar uno ajeno.

Sin la necesidad de obtener beneficios, al usar la versión gratuita, ayudó a que proliferasen gran cantidad de juegos de estudios o desarrolladores independientes. Y ya no solo juegos para vender, sino desarrollar juegos sin ánimo de lucro o por amor al arte, aumentando la participación en eventos de creación de videojuegos temáticos en tiempo reducido, generalmente conocidos como **Game Jams** [147], siendo la Game Jam más reconocida de todas la **Ludum Dare** [61].

Entrando en detalles sobre cómo funciona *Unity*, el desarrollo de videojuegos en este entorno está basado en *componentes*, que son insertados en **objetos** y estos colocados a su vez en una **escena o nivel**. Existe una enorme y gran variedad de componentes, cada una para un aspecto específico: gráficos, interfaz, audio, físicas, etc. Con lo que logramos conformar el aspecto y comportamiento básico del objeto al que unamos dichos componentes. Pero sin duda, los componentes más importantes que podemos añadir a cualquier objeto, son los **scripts** de código, permiten funcionalidades tales como:

- Definir y controlar el comportamiento del objeto al que está unido.
- Interaccionar con otros objetos de la misma escena.
- Realizar acciones o ejecutar eventos según los cambios que se produzcan en la escena o en otros objetos (que pueden ser causados por el usuario o no).
- Generar dinámicamente otros componentes y/o objetos dentro de una escena, e incluso generar ficheros.

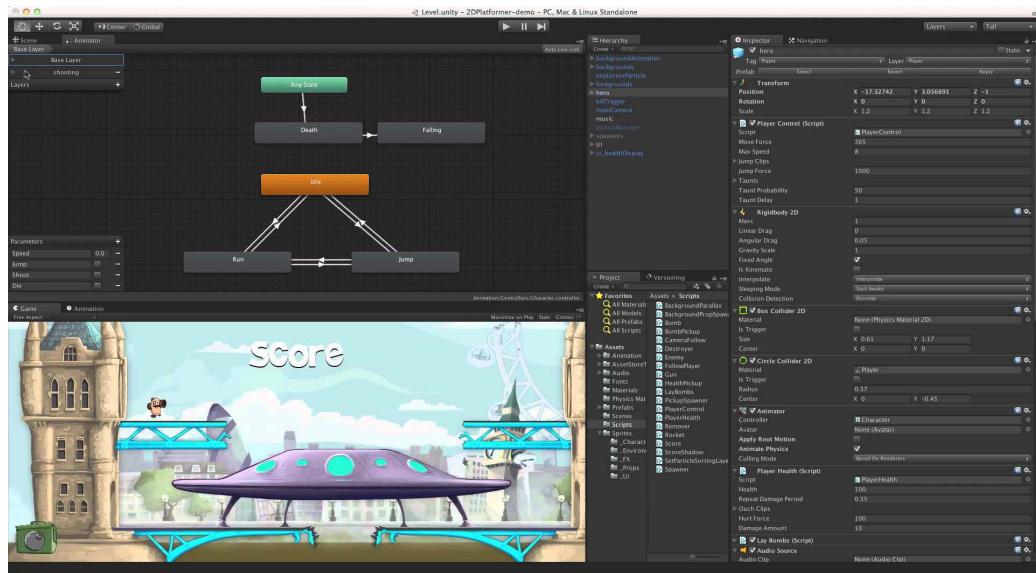


Figura 5.1: Interfaz de *Unity* para proyectos 2D

Monodevelop

Estos scripts pueden ser programados en C# o Javascript, que si bien se pueden programar en cualquier IDE que soporte estos lenguajes. Pero he escogido *Monodevelop* como IDE para programar los scripts del proyecto.

Monodevelop [64] es un IDE libre bajo licencia GPLv2 en su kernel, aunque la mayoría del código y extensiones del IDE están bajo otro tipo de licencia libre, la licencia MIT/X11. Disponible para Windows, OS X y Linux; y enfocado al desarrollo de aplicaciones bajo los lenguajes de C# y .NET, que son los lenguajes que escogí para la realización de la demostración de **1812: La aventura**.

Otros motivos por los que elegí *Monodevelop*, es porque una versión personalizada de este IDE viene incorporado al instalar *Unity*. Esta versión ya viene integrada con la API de *Unity*, con lo que la herramienta de autocompletado de *Monodevelop* puede referenciar las librerías, clases y métodos propios de *Unity* sin tener que realizar ninguna instalación extra.

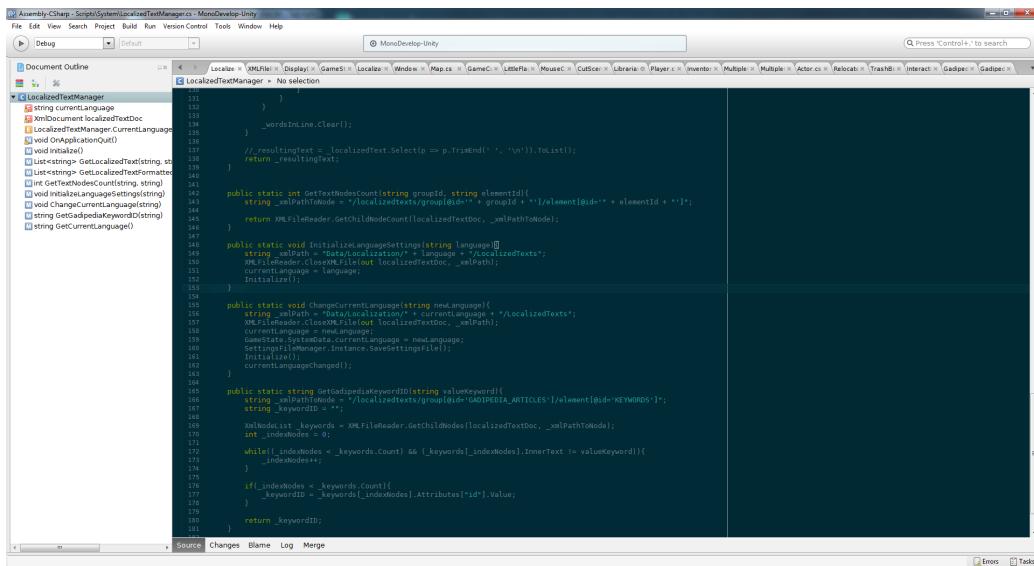


Figura 5.2: Interfaz de *Monodevelop* en uso durante la realización del proyecto

Geany

Geany [29] es un editor de texto ligero, libre y multiplataforma desarrollado con el *GTK2 toolkit*. Sus creadores lo desarrollaron con la intención de crear ser un editor para desarrollar pequeño y rápido con las características básicas que vienen dadas en IDEs de desarrollo más potentes.

Se usó principalmente para trabajar con los ficheros .xml del proyecto justamente por esas características y sus facilidades básicas para editar ficheros de este tipo.

LaTeX

LATEX[55] es un lenguaje de marcado y un sistema de creación de documentos especialmente orientado al mundo científico y técnico. El lenguaje *Tex* fue concebido por Donald Knuth durante los 80 y en 1984 Leslie Lamport creó *LATEX* como un framework para trabajar con cartas, libros y otro tipo de textos.

Los resultados que produce *LATEX* son coherentes, ordenados y muy limpios. Si bien aprender su uso puede ser complejo, los resultados son de una enorme calidad si los comparamos con los documentos que producen los editores convencionales como *Microsoft Word* o *LibreOffice Writer*. Por esta calidad y posibilidad de automatizar formatos ha sido elegido *LATEX* para la redacción de la documentación del proyecto.

TeXstudio

En un principio se eligió ShareLaTeX [38] como el editor de la documentación el L^AT_EXde **1812: La aventura**. Pues es un editor online con herramientas colaborativas y que permite compilar en cualquier equipo al guardarse como si fuera una copia de seguridad. Pero ante el nulo uso colaborativo a la hora de la verdad, los diversos problemas de sincronizar los cambios y de compilar cuando la página se satura de usuarios, finalmente hicieron que optara por otro editor.

En este caso el ganador resultó ser TeXstudio [9], un editor *standalone* multiplataformas libre, cuya interfaz es cómoda a la hora de visualizar la estructura de los ficheros que componen la documentación, además de incluir corrector ortográfico y sintáctico, autocompletado, herramientas avanzadas de resaltado y editado de texto, un visor de PDF integrado y otras muchas más características.

Doxxygen

Doxxygen [26] es una herramienta de documentación automática de código. Mediante la inclusión de comentarios especiales dentro de los ficheros de nuestro proyecto, es capaz de generar documentación con una apariencia atractiva tanto en formato HTML, como en L^AT_EXo muchos otros. Es compatible con los lenguajes C, C++, C#, Fortran, Java, Objective-C, PHP, Python, IDL y algunos más.

Con *Doxxygen* se produce una documentación legible por usuarios (con los conocimientos necesarios) u otros miembros del equipo. Es posible incluir diagramas de colaboración y herencia gracias a su uso de *GraphViz* [6] (paquete de utilidades para generar documentación gráfica). Ha sido elegida como herramienta de documentación de código por su sencillez de uso, limpieza y por su amplia aceptación.

Git

Git [56] es un sistema de control de versiones libre enfocado en flujos de trabajo no lineales, integridad de datos y rapidez. Inicialmente fue diseñado y desarrollado por Linus Torvalds para el kernel de Linux en 2005, y desde entonces se ha convertido en el sistema de control de versiones más usado para el desarrollo de software. *Git* nos permite contar con una copia de seguridad del código de **1812: La aventura** en todo momento. Gracias a esta herramienta podemos guardar un historial de todas las versiones de los ficheros, así como deshacer cambios en caso de que fuera necesario. También permite crear ramas con distintas versiones del proyecto y luego fusionarlas en otra. Con *Git* conseguimos acceso al código del proyecto desde cualquier equipo. Además, pone a disposición de cualquier interesado el código fuente de forma sencilla.

Existen otros sistemas de control de versiones como *Subversion* o *Mercurial*. Se ha elegido *Git* para aprender a desarrollar en este sistema de control de versiones, al ser el más extendido y requerido a la hora de trabajar en equipos con varios programadores en un proyecto de software.

GitHub

Además de *Git*, se ha elegido la forja *GitHub* [35] que usa dicho sistema de control de versiones. Los motivos son que al ser la forja más popular y reconocida a fecha de hoy, e incluir elementos sociales, permite una mayor difusión del proyecto. *GitHub* también proporciona su propia herramienta para Windows, llamada GitHub Windows [37], con la que gestionar el control de versiones de manera sencilla y visual para los más novatos. Para los más experimentados y familiarizados de UNIX, incluye la herramienta Git Shell que es una terminal para Windows que permite la utilización de los comandos bash de *Git*.

Git Shell

Existe otra herramienta llamada también Git Shell [34], siendo esta la oficial usada en el proyecto *Git*, pero por comodidad y sencillez (y la posibilidad de usar comandos *Unix*) se ha optado por la terminal Git Shell de *GitHub*.

GraphicsGale FreeEdition

GraphicsGale [41] es un editor de gráficos orientados al pixel-art para Windows. Permite la utilización de capas, un sistema de animación (con capas de cebolla para facilitar la creación de animaciones), otras muchas características específicas para el uso del pixel-art tales como el control de la paleta de colores.

Ha sido utilizado para la creación y edición de los gráficos de **1812: La aventura** con estética pixel-art. Siendo elegido por ser un editor muy completo y potente para este tipo de gráficos.

GIMP

GIMP [71] es el editor de imágenes libre del proyecto GNU, de hecho su nombre es un acrónimo de *GNU Image Manipulation Program*. Es multiplataforma y está disponible para Windows, GNU/Linux y Mac OS X. No es comparable a soluciones privativas como *Adobe Photoshop* pero es capaz de realizar operaciones bastante avanzadas de forma sencilla. Además de que sus funcionalidades pueden ser expandidas gracias a plugins, extensiones y scripts.

Ha sido utilizado en **1812: La aventura** para crear las transparencias necesarias en las hojas de sprites realizados con *GraphicsGale* [5]. Hemos elegido esta herramienta por contar con una licencia libre, ser lo suficientemente potente para nuestras necesidades y estar disponible en varias plataformas.

Trello

Trello es una aplicación web que sirve para la organización de proyectos, creado por Fog Creek Software en 2011. Su modelo de negocio se basa en el freemium, en el que se puede hacer uso de sus características básicas de manera gratuita, y en caso de necesitar de servicios complementarios (herramientas avanzadas de organización, seguridad, mantenimiento 24/7...) se puede contratar un plan para negocios.

Su funcionamiento se basa en el paradigma Kanban para organizar proyectos. Estos proyectos son representados por tablones, los cuales contienen listas (correspondiéndose con listas de tareas). Las listas contienen tarjetas (correspondiéndose con tareas). Estas tarjetas se tienen que ir cambiando de una lista a la siguiente (vía *drag & drop*) según va avanzando el proyecto, para reflejar el flujo de creación de un producto y/o característica desde su concepción como idea hasta su implementación. Los usuarios pueden ser asignados a las tarjetas. Además, las tarjetas pueden ser personalizadas por un ilimitado número de etiquetas, en la forma de colores que luego incluso pueden ser renombrados y crear otros nuevos. También se pueden crear sub-listas dentro de una misma tarjeta, adjuntar archivos y añadir comentarios. Y por último, los usuarios y tablones pueden ser agrupados en organizaciones.

La aplicación es ampliamente accesible desde la mayoría de los navegadores, incluidos navegadores móvil, o si no, existen aplicaciones específicas para iOS, Android y Windows 8 mobile. Con lo que es muy fácil siempre estar al tanto de la organización del proyecto y poder participar en los cambios organizativos en cualquier momento.

Si bien *Trello* se usa principalmente para la organización de productos y software, al ser una herramienta de organización tan sencilla y multiusos, se puede adaptar a cualquier tipo de organización, desde un diario personal hasta boletines de colegios, incluyendo los ya citados anteriormente.

Gracias a todas las facilidades comentadas, *Trello* fue usado para llevar la organización del proyecto durante toda su implementación.

Cacoo

Cacoo [67] es una aplicación web gratuita para crear diagramas de propósito general con la particularidad de poder elaborarlos colaborativamente con otros usuarios. *Cacoo* tiene una interfaz clara y sencilla que facilita mucho su uso. Además de los iconos correspondientes a las diversas opciones de edición y poseer un historial de estas ediciones, *Cacoo* cuenta con un chat para poder hablar y comunicarse con los usuarios durante los trabajos colaborativos. En los diagramas elaborados con esta herramienta se puede dibujar, incluir texto, imágenes (prediseñadas o subirlas desde nuestro ordenador) y todo tipo de figuras geométricas, bocadillos para texto, iconos y dibujos decorativos.

También podremos exportar los trabajos a formato .png, compartirlos a través de enlaces y en las redes sociales.

Siendo una herramienta altamente intuitiva y fácil de usar, en la que rápidamente podemos acceder a un historial de ediciones en caso de cambiar de parecer, se usó esta herramienta para generar la mayoría de los diagramas presentes en esta memoria.

Gantter

Gantter es una aplicación web gratuita con el que podremos realizar la planificación de nuestros proyectos y generar un diagrama de Gantt con las tareas, fases e hitos del mismo. Si bien estas funcionalidades, más o menos podemos encontrarlas en otras herramientas del mercado, Gantter es interesante porque permite la colaboración de varias personas a la hora de diseñar y generar un diagrama de Gantt. Además, al ser una aplicación web es accesible desde cualquier dispositivo que disponga de internet.

La aplicación nos permite guardar nuestras planificaciones en el propio servicio ofertado desde su página web oficial [48] o en nuestro almacén de Google Drive, exportar las planificaciones a .pdf, .html o .png e, incluso, poder exportar los hitos a un calendario en formato Google Calendar. La integración de Google Drive y dota a la aplicación de mucha flexibilidad y, además, el registro en la herramienta se hace muy simple porque tendremos la posibilidad de vincular nuestra cuenta de Gmail.

Con respecto al acceso, además de utilizar la web del servicio, Gantter dispone de una aplicación para Chrome que nos dotará de un acceso directo desde el navegador.

Todas estas comodidades para poder generar diagramas de Gantt y que sean accesibles en cualquier momento, junto con la capacidad de poder colaborar con gente y su integración con otros servicios, fueron los que hicieron que finalmente me decantase a usar esta herramienta para generar mis propias diagramas de Gantt.

Pencil

Pencil [30] es un editor gratuito con licencia libre, disponible para Linux, OS X y Windows, de prototipado de interfaces para aplicaciones, así podemos crear bocetos para las distintas pantallas que queramos que tenga nuestro software antes de implementarlas.

Incluye varias colecciones de objetos para poder montar los bocetos, y se pueden añadir aún más colecciones con las extensiones y plugins que hay disponibles. Entre dichas colecciones se incluye la de objetos simples con las que también poder elaborar diagramas.

Dichos bocetos y/o diagramas podemos exportarlos a diversos formatos (.png, .svg, .html, .pdf...) según lo que necesitemos.

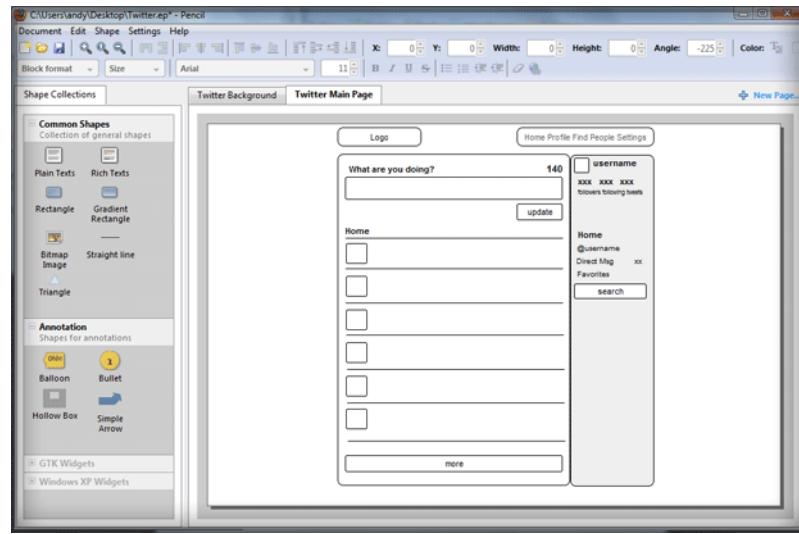


Figura 5.3: Ejemplo de *Pencil* con un boceto de la aplicación web de Twitter.

Con *Pencil* diseñé todos los bocetos de las interfaces para la demo de **1812: La aventura**. Siendo una de las pocas opciones gratuitas y libres que dispusiese de una interfaz fácil de usar y con gran cantidad de imágenes para poder crear bocetos de interfaces.

Bfxr

Bfxr [73] es una aplicación web gratuita (que también puede bajarse como aplicación para Windows) basada en *Sfxr* [77] para crear efectos de sonido retro (solo tiene cinco canales de ondas de formas básicas) de manera sencilla. Además incluye ajustes preestablecidos por si queremos crear cierto tipo de sonidos genéricos que suelen repetirse en los videojuegos.

La facilidad de uso y simplicidad, además de que con solo retocar un par de ajustes podíamos crear sonidos que se ajustaban a nuestras necesidades, fueron los motivos por los que elegí *Bfxr* para elaborar los sonidos restantes que me faltaban para la demo de **1812: La aventura**.

Audacity

Audacity [75] es un editor de audio libre y multiplataforma. Nos permite grabar y modificar audio con un gran número de opciones y variantes. Fue lanzado por primera vez en mayo del 2010 y actualmente cuenta con más de 72 millones de descargas. Si bien es cierto que carece de herramientas avanzadas de edición de sonido, para nuestras necesidades era la herramienta ideal. Fue utilizado en la demo de **1812: La aventura** para convertir y retocar los efectos de sonido y música que me mandaron los músicos del grupo *boredBit* y los creados posteriormente por mí misma mediante *Bfxr*.

Open Broadcaster Software (OBS)

OBS [50] es un programa libre, gratuito y multiplataforma para capturar audio y vídeo, ya sea para grabarlos en archivos de vídeo y/o hacer *streaming* [151]. Escrito en C y C++, *OBS* nos proporciona captura de audio y video en tiempo real a partir de distintas fuentes de entrada de información (la salida de imagen de un ordenador que se usa normalmente para los monitores, aplicaciones, archivos de imagen, vídeo, etc...), codificación, grabación y emisión. La transmisión de los datos es hecha mediante el Real Time Messaging Protocol y puede ser enviado a cualquier destino que soporte esta tecnología (Youtube, Twitch.tv, por poner unos ejemplos) e incluye varios ajustes preestablecidos para las webs de *streaming* más populares.

En el caso de codificación de archivos de vídeo, *OBS* es capaz de la librería libre x264 e Intel Quick Sync Video para codificar la captura de los *streaming* de vídeos a un formato H.264/MPEG-4 AVC. El audio puede ser codificado usando tanto los códecs de formato MP3 o AAC.

OBS fue utilizado por ser un capturador de vídeo y audio muy completo, gratuito y que me permitía hacer *streaming* de los avances del proyecto del motor de **1812: La aventura** y a su vez que guardase lo grabado en ficheros de vídeo .mp4 o .flv.

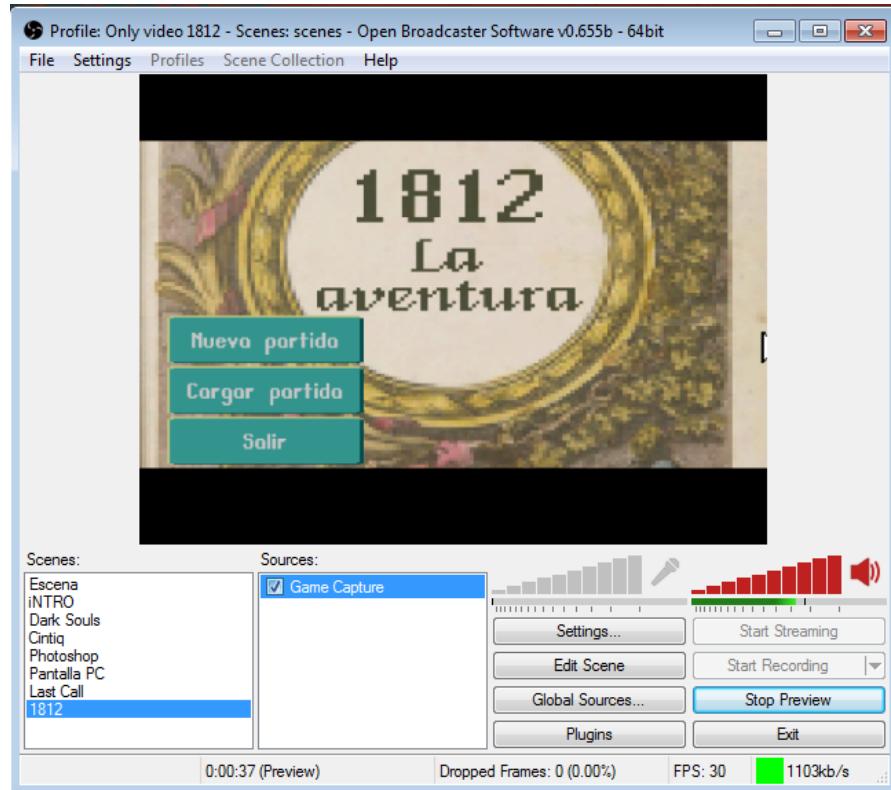


Figura 5.4: *OBS* capturando el vídeo y audio de una *build* de la demo de **1812: La aventura**

Lightworks

Lightworks [27] es un sistema profesional de edición de vídeo no lineal [150] para editar vídeos en una variedad de resoluciones que incluyen las nuevas resoluciones 2K y 4K, así como también dispone de varios formatos de salida tanto para dispositivos HD o SD. Lightworks fue un desarrollo temprano de edición de vídeo no lineal basado en computadora, ha estado en desarrollo desde 1989 y ha ganado varios premios por ser una de las sistemas más potentes para la realización de películas y series de televisión.

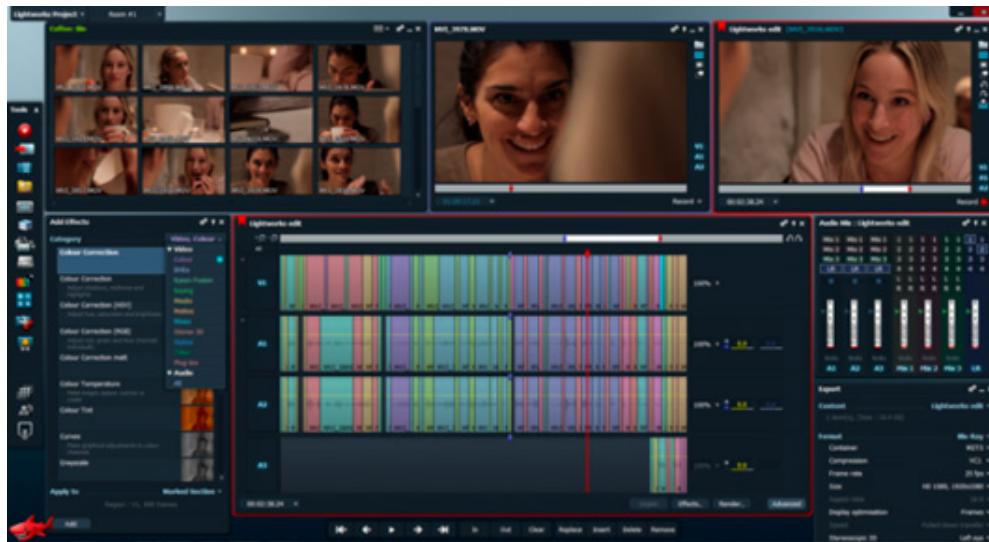


Figura 5.5: Interfaz de *Lightworks* durante un tutorial sobre cómo aprender a usarlo

Actualmente tiene una versión gratuita y multiplataforma que si bien es algo limitada, cumple con creces con las necesidades básicas para edición de vídeos para proyectos no profesionales. Es un sistema muy completo y quizás avanzado, pero gracias a los tutoriales oficiales [28] en un par de días podemos aprender cómo editar, montar vídeos, incluir efectos de transición entre fragmentos de vídeo y mezcla de pistas de audio. Por todos estos motivos, fue elegido como el sistema para el montaje de los vídeos de ejemplo del proyecto, que pueden ser visualizados en Youtube [154].

Capítulo 6

Manual de instalación

Requisitos mínimos

Para poder disfrutar de la demo técnica de **1812: La aventura** sin problemas necesitas disponer de un equipo con unas características iguales o superiores a las que se listan a continuación:

- **Sistema Operativo:** Windows XP+, Mac OS X 10.7+, Ubuntu 12.04+, SteamOS+
- **Procesador:** Pentium 2 GHz o AMD compatible con el conjunto de instrucciones SSE2.
- **Memoria RAM:** 100 MB de memoria RAM disponibles.
- **Tarjeta de vídeo:** 128 MB de memoria y capacidades DX9 (shader modelo 2.0).
- **Espacio en disco:** 100 MB.
- **Control:** Ratón (y teclado de manera opcional).
- **Software opcional:** Si se desea bajar el proyecto y probarlo, es necesario tener instalado el editor de *Unity*. Se puede descargar de forma gratuita en
<https://unity3d.com/es/get-unity/download?ref=personal>

Para saber más sobre los requisitos mínimos genéricos que se necesitan para poder jugar a los juegos realizados en *Unity* o para instalarse el editor de *Unity* visite la siguiente página:

<https://unity3d.com/es/unity/system-requirements>

Descarga e instalación

En esta sección se proporcionan las indicaciones pertinentes para que puedas descargar e instalar la demo técnica de **1812: La aventura** para poder probar el **Motor de Aventuras Gráficas 2D de Unity** en tu sistema. Dependiendo de si quieres probar el proyecto con *Unity* o directamente jugarlo sin *Unity*, existen dos maneras distintas de descargar.

Descarga e instalación para utilizar el proyecto con *Unity* instalado

Si tienes un ordenador con sistema operativo Microsoft Windows o Mac OS X compatible con *Unity* basta con descargarlo e instalarlo. Se puede descargar de forma gratuita en:

<https://unity3d.com/es/get-unity/download?ref=personal>

Una vez con *Unity* instalado (con una versión igual o superior a la 5.0.1) nos podemos descargar el proyecto directamente del repositorio de *GitHub*:

<https://github.com/Firenz/1812/archive/demo.zip>

Tras descargar el proyecto, utilizamos cualquier programa de compresión/descompresión que soporte el formato zip. Si no tenemos ninguno instalado, podemos emplear 7Zip sin problemas ya que es Software Libre. Para descargarlo, dirígete a la siguiente dirección:

<http://www.7-zip.org>

Con el proyecto ya descomprimido, abrimos *Unity* y le damos al botón **Open other**, elegimos el directorio donde hayamos descomprimido el proyecto y le damos a aceptar. Tardará unos segundos en cargar el proyecto en *Unity* y una vez termine ya tendremos a nuestra disposición el proyecto listo para funcionar, solo tendremos que darle al botón de **Play para poder probar la demo técnica y a la vez ver el funcionamiento de esta**.

Descarga únicamente el juego sin *Unity* instalado

Si por algún motivo en concreto no podemos y/o no queremos instalar *Unity* en el ordenador, siempre podemos jugar al juego sin tener que bajarnos el proyecto. Para ello solo hay que descargar desde su página de GameJolt la versión del videojuego para nuestro sistema operativo:

<http://gamejolt.com/games/1812-la-aventura/49809>

Una vez bajado, dependiendo del sistema operativo que utilices, se recomienda lo siguiente para poder instalar y poder jugar a la demo técnica de **1812: La aventura** .

Instalación en Windows

Tras descargar el archivo, utiliza cualquier programa de compresión/descompresión que soporte el formato zip. Si no tienes ninguno instalado, puedes emplear 7Zip sin problemas ya que es Software Libre. Para descargarlo, dirígete a la siguiente dirección:

<http://www.7-zip.org>

Una vez descomprimido, la demo técnica de **1812: La aventura** es completamente portable y autocontenido, así que si lo preferimos podemos guardarla en un lápiz de memoria o en cualquier disco duro

externo. Para iniciar el juego, hacemos doble click sobre el siguiente fichero ejecutable:

```
1812_aventura_win_x86.exe
```

Y listo, ya estamos jugando.

Instalación en Linux

La instalación es prácticamente igual que en Windows, ya que también es portable y autocontenido para este sistema. No obstante, si por algún motivo quieras tener un acceso directo en tanto en el menú de inicio como en el escritorio del juego, accedemos al directorio resultante utilizando la terminal y escribimos lo siguiente:

```
$ sh run_1812.sh
```

Cuando ya nos cansemos de jugar a la demo y decidamos desinstalarla, accedemos al directorio resultante utilizando la terminal y escribimos lo siguiente:

```
$ sh uninstall.sh
```


Capítulo 7

Manual de usuario

Demo técnica de 1812: La aventura

La demo técnica de **1812: La aventura** es un videojuego de aventura gráfica en 2D ambientado en nuestra época actual donde un estudiante que ha suspendido por poco su examen de historia decide reclamar al profesor que le suba un poco la nota, lo cual no será tarea nada fácil: problemas, puzzles, personajes quisquillosos y picarescos se interpondrán entre él y su aprobado.

Historia

Nuestro protagonista en cuestión ha hecho recientemente un odiado examen de historia, el cual ya ha repetido varias veces, y ha suspendido una vez más. Suspender tanto es duro, máxime cuando se ha suspendido por una décima, así que ni corto ni perezoso, nuestro protagonista decide ir a protestar al profesor para reclamarle la nota y demostrar que (esta vez sí) se ha estudiado y aprendido la materia. No obstante, al llegar al despacho del profesor, se da cuenta de que este está ausente, lo cual es un augurio de que no va a ser nada fácil conseguir esa décima extra . . .

Guía de juego

En este bloque explicaremos cómo se juega a la demo técnica de **1812: La aventura**, así como exponer las diferentes pantallas del mismo y explicando la funcionalidad de cada una de ellas.

Menú principal

Nada más abrir el juego, si es la primera vez que lo abrimos, nos saltará un menú donde podemos elegir el idioma con el que queremos jugar a la demo técnica de **1812: La aventura**, si por el contrario ya

no es la primera vez que lo jugamos se saltará este paso y justo después del logo de la **Universidad de Cádiz**, veremos el menú principal.



Figura 7.1: Menú principal

En este menú podremos llevar a cabo las siguientes acciones, pulsando los botones correspondientes:

- **Nueva partida:** Esta opción nos cargará la partida inicial, con la intro del juego.
- **Cargar partida:** Esta opción nos llevará a la pantalla de carga de partidas, donde podremos seleccionar que partida que hayamos guardado anteriormente queremos cargar.
- **Salir:** Cierra la aplicación.
- **Créditos:** Esta opción nos muestra los créditos de todos aquellos que han colaborado en el proyecto.

Cargar partida

Este menú nos permitirá elegir y cargar las partidas que hayamos jugado y guardado previamente. Además, desde este menú, también podremos borrar las partidas que seleccionemos. Simplemente hacemos click izquierdo en la partida que deseemos elegir y después clickamos la opción deseada. Para volver al menú principal, simplemente hacemos clic en “Volver al menú”.



Figura 7.2: Cargar partida

Créditos

En esta pantalla aparecen los créditos de aquellos que han participado en la creación de la demo técnica de **1812: La aventura**. Para volver al menú principal, simplemente hacemos clic en “Volver al menú”.

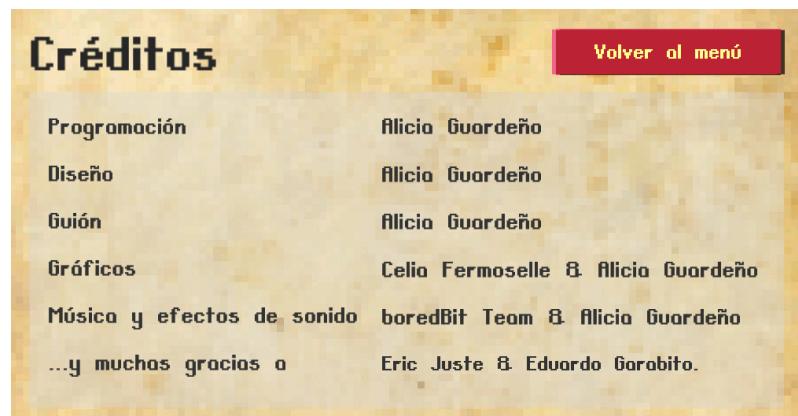


Figura 7.3: Créditos

Jugar

Una vez seleccionada la opción de jugar, se cargará la escena cinematográfica con la introducción de la demo. En ella veremos a nuestro joven y desafortunado protagonista yendo al despacho de su profesor para reclamarle la nota de su examen.



Figura 7.4: Introducción del juego, con el protagonista dirigiéndose al despacho del profesor

Una vez dentro, vemos que el despacho está vacío, y el muchacho decide esperarle dentro para reclamar su nota. Debido al calor, decide abrir la ventana para que se refresque un poco la habitación, con tan mala suerte que el viento echa a volar un montón de las banderitas que estaban pegadas en un mapa situado en el tablón de la pared... ¡Qué mala suerte!



Figura 7.5: Introducción del juego, justo después de que se hayan caído las banderitas del mapa situado en el tablón de la pared

Comprendiendo el lío en el que se acaba de meter, el protagonista decide tratar de solucionar ese embrollo antes de que el profesor vuelva... ¡Aquí empieza nuestra aventura para ayudarle!



Figura 7.6: Después de la introducción, el *Jugador* retoma el control del juego

A partir de aquí asumiremos el control del personaje y le ayudaremos a salir de este embrollo. Podemos controlar todo el juego empleando el ratón para interactuar con los diferentes elementos del juego y con el teclado cuando necesitemos escribir algo.

Utilizando **click izquierdo** en algún lugar del escenario en el que no haya un elemento interactivo podremos mover al personaje hacia donde hicimos click. Si en cambio hacemos **click izquierdo** sobre un elemento que sí sea interactivo, el personaje se acercará a ese objeto y tratará de interactuar con él, llegando incluso a añadirlo a su inventario de ser posible esa opción. Con **click derecho** miraremos (sin interactuar o coger) aquello sobre lo que hayamos hecho click.



Figura 7.7: Controles básicos del juego

Además, en la parte inferior de la pantalla contaremos con nuestro inventario, lugar donde se almacenarán los objetos que nuestro personaje tenga y/o haya conseguido a lo largo de la aventura.

Los objetos del inventario pueden ser observados con **click derecho** al igual que con cualquier otro objeto del escenario, pero la funcionalidad de **click izquierdo** cambia un poco: al hacer **click izquierdo** sobre un objeto de nuestro inventario, “cogeremos” ese objeto con nuestro cursor y, si hacemos **click izquierdo** sobre otro objeto del escenario mientras tenemos “cogido” un objeto de esta forma, usaremos el objeto de nuestro inventario sobre el objeto seleccionado. Si por algún casual no queremos usar el objeto “cogido” sobre ningún otro objeto del escenario y simplemente queremos dejarlo donde estaba, bastará con hacer **click derecho** y automáticamente lo volverá a dejar en el inventario.



Figura 7.8: Usando el inventario dentro del juego

Menú de opciones

Con el apartado anterior hemos cubierto todo lo relevante al juego y como moverse por él. Únicamente queda por comentar el menú de opciones, el cual es accesible solo mientras se juega y el personaje principal no esté realizando una acción en el nivel, haciendo **click izquierdo** sobre el móvil que hay en la esquina superior derecha de la pantalla.



Figura 7.9: Icono del menú de opciones dentro del juego

Una vez dentro, podremos seleccionar entre los diferentes menús de opciones disponibles de acuerdo a nuestros deseos, pudiendo elegir entre:

- Notas
- Gadipedia
- Mapa
- Ajustes
- Partidas
- Salir

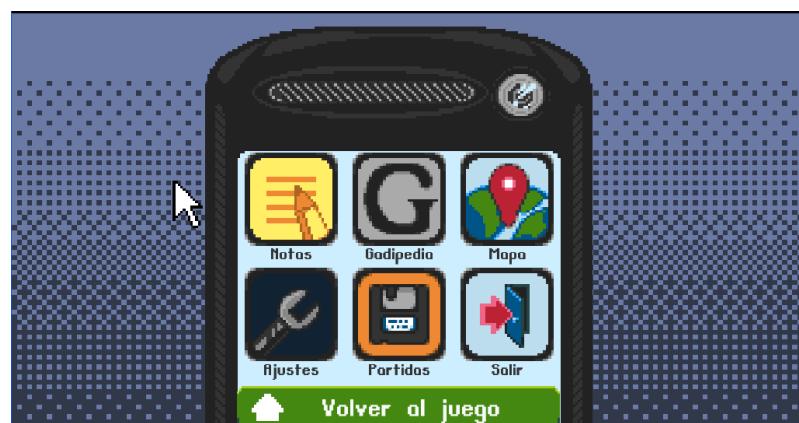


Figura 7.10: Menú de opciones

Notas

Pantalla en la que el sistema carga una lista de notas. Dichas notas serán anotaciones de nuestro protagonista sobre la historia y eventos importantes que ocurrieron, lo cual nos servirá para mantenernos al tanto de lo que ocurre y que no se nos olviden detalles útiles.



Figura 7.11: Menú de notas

Gadipedia

Pantalla en la que podremos buscar información sobre elementos históricos importantes que nos servirán de ayuda en nuestra misión. Para ello, simplemente escribiremos la palabra clave de aquello que deseemos buscar dentro del buscador y hacer click sobre el botón Buscar.



Figura 7.12: Buscador del menú de gadipedia

Acto seguido, nos saldrá un resultado con la búsqueda de aquello que hayamos introducido. Si la búsqueda no coincide con nada, nos devolverá una pantalla que nos avisará de ello.



Figura 7.13: Resultados del menú de gadipedia

Mapa

Pantalla con la que podemos ver el mapa de la universidad en la que se encuentra nuestro protagonista. Sobre el mapa habrán varios puntos, los cuales representarán los distintos lugares y escenarios por los que puede moverse el protagonista. El jugador puede seleccionar cualquiera de ellos haciendo click sobre él y viajar directamente a ese lugar.



Figura 7.14: Menú de mapa

Ajustes

Pantalla en la que se pueden configurar las diferentes opciones que afectan al juego, como la resolución, el modo ventana/pantalla completa, el volumen de la música y efectos sonoros, y el idioma. Al seleccionar el idioma, se nos abrirá una pantalla mostrándonos los diferentes idiomas a elegir. Bastará con elegir cualquiera de ellos para que todos los textos del juego cambien al momento.

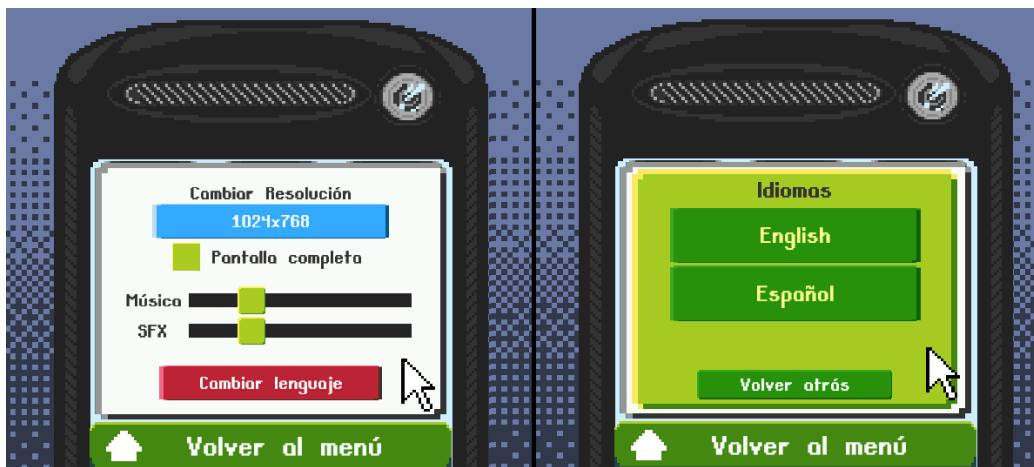


Figura 7.15: Menú de ajustes

Partidas

En esta pantalla podremos ver el listado de partidas guardadas (de forma muy similar al menú de Cargar partida que teníamos en el Menú principal). En él, podremos elegir una ranura de partida, esté vacía o no, y acto seguido guardar nuestra partida haciendo click izquierdo sobre el botón de Guardar (de no estar vacía, sobrescribirá el contenido). También podemos clickar los botones Cargar y Borrar una vez hayamos elegido una ranura de partida no vacía para cargar o borrar esa partida, respectivamente.



Figura 7.16: Menú de partidas guardadas

Salir

Al igual que el botón de salir del menú principal, hacer click sobre esta opción cerrará el juego tras hacer click sobre el botón de “Si” de la pantalla de confirmación que nos aparecerá.



Figura 7.17: Salir del juego

Capítulo 8

Manual básico para editar y traducir los textos de 1812: la aventura

Uno de los objetivos del **Motor de Aventuras Gráficas 2D de Unity** era separar y volver independientes todo texto que aparece en el videojuego de la implementación del motor, de manera que se pudiera modificar el texto en cualquier momento sin afectar el desarrollo ni tocar nada de código. Y no solo eso, podemos crear traducciones del fichero en distintos directorios que el juego los detectará automáticamente y los añadirá como lenguajes a elegir al iniciar el juego por primera vez o desde el menú de ajustes dentro del juego.

Para crear nuestra propia traducción de manera satisfactoria, solo hay que seguir estos pasos:

1. Tenemos que irnos a la carpeta `Data/Localizations/`, si estamos usando el ejecutable este directorio estará dentro de:
`1812_aventura_*)_Data` (donde '*' sería el nombre de la versión correspondiente al sistema operativo y procesador que estaríamos usando)
En caso de que usemos la versión del proyecto para probarlo en *Unity*, el directorio estará dentro de:
`Assets/`
2. Una vez localizado el directorio de `Localizations/`, veremos que hay dos directorios `ES/` con los textos de la demo en español, y `EN/`. Creamos un nuevo directorio con la ID con la que queramos que se identifique el idioma dentro del juego (por ejemplo, si quisieramos hacer una traducción al francés podríamos ponerle de nombre al directorio `FR`).
3. Teniendo ya creado el directorio, copiamos el fichero `LocatedTexts.xml` del directorio `ES/` (si queremos traducir a partir del español) o de `EN/` (si queremos traducir a partir del inglés).
4. Abrimos el fichero con cualquier editor de textos, preferiblemente uno que pueda manejar las etiquetas de un fichero `.xml` automáticamente. En mi caso usé el editor *Geany* que es software libre, gratuito y disponible para Linux, Mac OS X y Microsoft Windows. Este programa se puede descargar para Microsoft Windows y Mac OS X desde aquí:
<http://www.geany.org/Download/Releases> En caso de querer instalarlo para Linux Debian, Ubuntu y Mint, tenemos que abrir la terminal y escribir los siguientes comandos:

```
$ apt-get -y update
$ apt-get -y install geany
```

5. Ya abierto el fichero, antes de ponernos a traducir los textos tenemos que cambiar la información que está dentro del bloque meta.

```
<?xml version = "1.0" encoding = "utf-8"?>
<locatedtexts>
    <meta>
        <GameName>1812: The adventure</GameName>
        <Author>Alicia Guardeño</Author>
        <Traslation>Alicia Guardeño</Traslation>
        <Language>English</Language>
        <LanguageID>EN</LanguageID>
        <Version>version 0.1</Version>
        <Web>1812laaventura.wordpress.com</Web>
        <License>GPL v3</License>
    </meta>
```

Figura 8.1: Bloque de información meta dentro del fichero LocatedTexts.xml

En dicho bloque, tendremos que cambiar alguna que otra información para que pueda el juego detectarla correctamente:

GameName El nombre de la demo técnica de **1812: La aventura**, solo sería necesario traducir “*The adventure*” en este ejemplo.

Traslation El nombre del autor o autores de esa traducción en específico.

Language El nombre del lenguaje de esa traducción en su propio idioma.

LanguageID El nombre del directorio en el que está colocado este fichero LocatedTexts.xml.

Version Este ajuste es opcional. Aquí vendría a ser la versión de la traducción, así se indica si han habido revisiones o que es una versión prematura de los textos.

6. Una vez completado correctamente el bloque meta para que pueda ser detectado por el juego, podemos proceder finalmente con la traducción. Lo único que hay que hacer es traducir el texto dentro de los bloques string.

```
<group id = "LIST_PLAYABLE_SCENES">
    <element id = "PROFESSOR_OFFICE">
        <string id = "NAME">Professor's office</string>
    </element>
    <element id = "CORRIDOR">
        <string id = "NAME">Corridor</string>
    </element>
</group>
<group id = "INVENTORY">
    <element id = "FAILEDTEST">
        <string id = "NAME">Failed test</string>
        <string id = "DESCRIPTION">
            Bloody 4.9-...
        </string>
    </element>
```

Figura 8.2: Fragmento de los bloques string dentro del fichero LocatedTexts.xml

Procura respetar los saltos de línea al traducirlos. Si bien para los textos para la interfaz no son demasiado relevantes (se eliminan), para los diálogos los saltos de línea son importantes porque cada uno equivale a un cuadro de diálogo nuevo dentro del juego.

7. Acaba la traducción, solo tenemos que guardar los cambios y ejecutar la demo técnica de **1812: La aventura** para comprobar que efectivamente se ha traducido de forma correcta. Si por algún motivo sin querer borramos un bloque `string` dentro de un fichero `LocatedTexts.xml`, el juego nos indicará que falta dicha línea de texto sustituyéndola por:

[This string has either not been implemented or needs to be translated.]

Para arreglarlo, tendríamos que cerrar el juego, abrir el archivo `LocatedTexts.xml` que habíamos modificado, buscamos la línea que falta, la añadimos y guardamos los cambios.

Con esto habremos conseguido añadir una nueva traducción completamente nueva al juego. ¡Así nuevos jugadores que no sepan inglés o español podrán disfrutar de esta pequeña aventura gráfica!

Capítulo 9

Manual de uso del Motor de Aventuras Gráficas 2D en Unity

Si lo que queremos es expandir la historia de la demo técnica de **1812: La aventura** o empezar a hacer tu propia aventura gráfica gracias a este motor, el proceso es largo pero no contiene apenas dificultad, sobre todo si seguimos los pasos de este manual.

Instalación de *Unity* y complementos

Si tienes un ordenador con sistema operativo Microsoft Windows o Mac OS X compatible con *Unity* e instalarlo. Se puede descargar de forma gratuita en:

<https://unity3d.com/es/get-unity/download?ref=personal>

Una vez con *Unity* instalado (con una versión igual o superior a la 5.0.1) nos podemos descargar el proyecto directamente del repositorio de *GithHub*:

<https://github.com/Firenz/1812/archive/demo.zip>

Tras descargar el proyecto, utilizamos cualquier programa de compresión/descompresión que soporte el formato zip. Si no tenemos ninguno instalado, podemos emplear 7Zip sin problemas ya que es Software Libre. Para descargarlo, dirígete a la siguiente dirección:

<http://www.7-zip.org>

Con el proyecto ya descomprimido, abrimos *Unity* y le damos al botón **Open other**, elegimos el directorio donde hayamos descomprimido el proyecto y le damos a aceptar. Tardará unos segundos en cargar el proyecto en *Unity* y una vez termine ya tendremos a nuestra disposición el proyecto listo para funcionar, solo tendremos que darle al botón de **Play para poder probar la demo técnica y a la vez ver el funcionamiento de esta**.

Para escribir código, podemos usar el IDE *Monodevelop* que viene por defecto con *Unity*.

Para crear gráficos y animaciones 2D, es necesario que usemos un editor de imágenes. En este caso aconsejo *GIMP*, que es un editor de imágenes genérico gratuito y con licencia libre, se puede usar en Linux, Mac OS X y Microsoft Windows. Está bastante completo y si nos hace falta algo siempre podemos añadirle extensiones y plugins para añadirle más funcionalidades. Además, existen varios tutoriales para preparar *GIMP* a trabajar con estilo *Pixel Art* [78], si deseamos este estilo, o para animar en 2D [74].

Por último, para la edición de los textos de los ficheros .xml podemos usar *Monodevelop* que viene incluido con *Unity* o usar cualquier otro editor que prefiramos.

Requisitos para poder componer un nivel

Para seguir con esta guía es recomendable que tengas unos conocimientos mínimos de sobre arte y animación en 2D, particularmente *Pixel Art* si queremos seguir con la misma estética que la de la demo técnica de **1812: La aventura** .

Además de eso, se requieren unos conocimientos mínimos de C# y .NET junto con nociones básicas de cómo usar el editor y la API de *Unity*.

Creación de un nuevo nivel

Para crear un nuevo nivel en la demo técnica de **1812: La aventura** , tenemos que tener el proyecto abierto dentro de *Unity*. Para ello abrimos *Unity*, le damos al botón **Open other**, elegimos el directorio donde hayamos descomprimido el proyecto y le damos a aceptar. Tardará unos segundos en cargar el proyecto en *Unity* y una vez termine ya tendremos a nuestra disposición el proyecto listo para funcionar.

Ya abierto el proyecto, para crear un nivel nuevo tenemos que abrir el menú **File** de *Unity* y elegimos la opción **New Scene**. Se nos creará una escena totalmente vacía en la que los primeros *prefabs* para lograr que se vea correctamente el nivel dentro del juego.

1. Dentro del panel de **Project**, nos vamos a la carpeta **Assets/Resources/Prefabs/Other/** y arrastramos desde este panel hasta el panel de **Hierarchy** los siguientes *prefabs*:

- **GameController**, que contiene los *scripts* de varios sistemas como el de extracción de textos de los ficheros .xml.
- **CustomCursorController**, que controla el cursor a mostrar en cada momento.
- **MainCamera**, la cámara con los ajustes predefinidos para que se vea la escena con la proporción correcta. No hay que olvidarse que hay que borrar antes la cámara que viene por defecto en la nueva escena después de crearse.
- **BorderScreen**, este objeto es más por estética, simplemente es una imagen con los ajustes ya definidos pero necesario para delimitar los límites del escenario y por dónde comienza la parte de la interfaz gráfica del juego junto con el inventario de objetos.

2. Otra vez dentro del panel de Project, nos dirigimos a la carpeta Assets/Resources/-Prefabs/Inventory/ y arrastramos hasta el panel de Hierarchy el *prefab Inventory*. Con esto ya tendremos la interfaz del inventario lista para usar dentro de la nueva escena.
3. Volvemos al panel de Project y abrimos la carpeta Assets/Resources/Prefabs/UIElements/, cogemos y arrastramos el *prefab InGameCanvas* hasta el panel Hierarchy. Tendremos ya en nuestra escena la interfaz gráfica de la barra superior de la demo técnica de **1812: La aventura**, pero no se verá bien ajustado el tamaño respecto a la cámara de la escena. Para arreglar esto, tenemos que clickar encima de **InGameCanvas** para que salga el Inspector, y arrstrar el *prefab MainCamera* desde Hierarchy hasta el componente Canvas en donde pone Render Camera. Una vez hecho ya se habrá linkeado la cámara con el canvas de la interfaz y se verá con las proporciones correctas.
4. Una vez colocado todos estos *prefabs*, tendremos que decorar el escenario con gráficos de fondo. No importa el tipo de estilo o tamaño de estos, pero intenta que se ajusten bien al tamaño de la escena desde el panel Scene utilizando sus controles. Aparte, dependiendo del tipo de fondo de escenario:

Escenario de fondo: Si es el fondo que se suele ver detrás del jugador, desde el Inspector del gráfico, en el componente SpriteRenderer, en la opción Sorting Layer elegimos BackgroundScenery y en Sorting Order ponemos un número negativo alto (por ejemplo, un valor de -50).

Escenario frontal: Si es un fondo que se ve por delante del jugador, seguimos los mismos pasos que en **Escenario de fondo** pero en la opción Sorting Layer elegimos FrontlineScenery y en Sorting Order ponemos un número positivo alto (por ejemplo, un valor de 100).

5. Por último, nos hace falta colocar al personaje principal en la posición inicial que va a tener al entrar en el escenario. Desde el panel de Project abrimos la carpeta Assets/Resources/Prefabs/Actors y arrastramos hasta Hierarchy el *prefab Player*, una vez dentro de la escena desde el panel Scene y sus controles lo colocamos en el sitio donde queremos que vaya a estar.

Con esto ya tendríamos que tener un escenario vacío listo para ser llenado con objetos, NPCs y algún que otro sistema aparte como el de navegación.

Creación de objetos interactivos

Por ahora nuevo protagonista lo único que puede hacer es estarse quieto sin hacer nada. Lo que vamos a hacer ahora es crear objetos interactivos básicos con los que pueda interactuar una vez hayamos terminado el nivel. Estos son los pasos a seguir para crear uno:

1. Para ello desde el panel de Hierarchy le damos a Create → Create Empty. Se creará un objeto vacío en la escena, el cual lo moveremos mediante los controles del panel Scene hasta el sitio del nivel donde queramos que esté el objeto interactivo y le cambiamos el nombre del objeto a uno que represente (por ejemplo, si es un cartel, renombrarlo a **Cartel**) desde el Inspector.

2. Desde el Inspector, cambiamos el Tag del objeto a InteractivePoint.
3. Desde el Inspector, dentro del componente Transform, buscamos el parámetro Position → z y le cambiamos su valor dependiendo del orden de profundidad en el orden z que queramos que tenga. Esto es así para evitar el solapamiento con otros objetos interactivos o NPCs que hagan que no se detecten correctamente si están en el mismo orden z. El valor por defecto que tenemos que ponerle será 1.
4. Desde el Inspector le añadimos al objeto el componente PolygonCollider2D, le damos a la opción Edit Collider y desde el panel Scene le damos forma al polígono del objeto. La función del PolygonCollider2D es que el objeto interactivo sea detectado por el script MouseClickHandler. De esta forma el cursor cambiará al pasar por encima del objeto, aparecerá su nombre en la interfaz gráfica del nivel, y el objeto Player podrá interactuar con él al clickar encima suya.
5. Desde Hierarchy hacemos click derecho encima del objeto y saldrá un menú, elegimos Create Empty. Nos creará un objeto hijo dentro del objeto interactivo. Nos vamos al Inspector con el objeto hijo seleccionado, le cambiamos el nombre a WalkingPoint y en la opción Layer ponemos Ignore Raycast. Por último desde el panel Scene cambiamos la posición de este objeto hasta la posición dentro del escenario dónde Player irá para interactuar con el objeto interactivo.
6. Volvemos con el objeto interactivo, al que nos falta solamente añadirle el script que definirá su comportamiento dentro del juego. Existen tres tipos básicos de scripts dependiendo del comportamiento que queramos para nuestro objeto:

InteractiveElement

Si solo queremos un objeto interactivo básico que solo sirva para ser inspeccionado, este será el script que tendremos que añadir al objeto interactivo.

a) Desde el Inspector del objeto interactivo, adjuntamos como componente el script InteractiveElement.

b) Una vez añadido, se verá en el Inspector el script dentro del objeto y con algunos campos que hay que llenar:

Group ID Lo rellenamos con el ID del grupo que engloba todos los objetos del nivel que hay dentro de los ficheros LocatedTexts.xml. Por ejemplo, para la escena del pasillo, la ID sería SCENE_CORRIDOR.

Element ID Lo rellenamos con el ID del objeto interactivo que hay dentro de los ficheros LocatedTexts.xml. Por ejemplo, para una planta, el ID del objeto sería OBJECT_PLANT.

Min Distance Es el error de distancia mínimo permitido entre WalkingPoint y Player cuando este último se dirija a interactuar con el objeto interactivo.

PickableElement

Si queremos que un objeto interactivo se pueda coger o manipular en el escenario, este será el script que tendremos que añadir al objeto interactivo.

a) Desde el Inspector del objeto interactivo, adjuntamos un gráfico o sprite mediante el componente SpriteRenderer. Luego en el parámetro Sorting Layer elegimos BackgroundInteractiveObjects y en Sorting Order el número con el orden de dibujado del objeto dentro de la escena (por defecto deberemos poner este valor a 1).

b) Desde el Inspector del objeto interactivo, adjuntamos como componente el script PickableElement.

c) Una vez añadido, se verá en el Inspector el script dentro del objeto y con algunos campos que hay que llenar:

Group ID Lo rellenamos con el ID del grupo que engloba todos los objetos del nivel que hay dentro de los ficheros LocatedTexts.xml. Por ejemplo, para la escena del pasillo, la ID sería SCENE_CORRIDOR.

Element ID Lo rellenamos con el ID del objeto interactivo que hay dentro de los ficheros LocatedTexts.xml. Por ejemplo, para una planta, el ID del objeto sería OBJECT_PLANT.

Min Distance Es el error de distancia mínimo permitido entre WalkingPoint y Player cuando este último se dirija a interactuar con el objeto interactivo.

Current Pickable Position Este valor nos indica si es un objeto que está en el suelo o a la misma altura que el personaje principal, dependiendo de su valor, Player realizará una animación u otra al intentar tocar el objeto para manipularlo y/o cogerlo. Los valores que hay son Down (abajo) y Up (arriba).

Is Destroyed After Being Picked Con esta casilla indicaremos si el objeto después de haber sido manipulado por Player se volverá inactivo o no. El valor por defecto es que no se vuelva inactivo al ser manipulado.

Name List Givable Elements Este parámetro tendrá a su vez otros valores internos. Size sirve para indicar el tamaño de la lista de objetos (el tamaño por defecto es cero) para el inventario que le dará a Player cuando este interactue con el objeto interactivo. Luego según el valor de Size se crearán nuevos valores internos llamados Element n (siendo n un valor entre 0 y el valor de Size menos uno), en cada uno de estos valores tendremos que poner el nombre del prefab del objeto de inventario que deseemos añadir. Para saber de los prefabs de objetos de inventario que hay disponibles dentro del juego, miramos desde el panel de Project en la carpeta Assets/Resources/Prefabs/Inventory/Items/.

WarpElement

a) Desde el Inspector del objeto interactivo, adjuntamos como componente el script WarpElement.

b) Una vez añadido, se verá en el Inspector el script dentro del objeto y con algunos campos que hay que llenar:

Group ID Lo rellenamos con el ID del grupo que engloba todos los objetos del nivel que hay dentro de los ficheros LocatedTexts.xml. Por ejemplo, para la escena del pasillo, la ID sería SCENE_CORRIDOR.

Element ID Lo rellenamos con el ID del objeto interactivo que hay dentro de los ficheros LocatedTexts.xml. Por ejemplo, para una puerta, el ID del objeto sería OBJECT_DOOR.

Min Distance Es el error de distancia mínimo permitido entre WalkingPoint y Player cuando este último se dirija a interactuar con el objeto interactivo.

Destination Scene Nombre de la escena a la que nos transportará el objeto interactivo cuando Player interaccione con él. Por ejemplo, para que nos transporte a la escena del despacho del profesor, el nombre sería ProfessorOffice.

Arrow Direction Dirección hacia la que queremos que apunte el cursor de flecha negra (el que se usa para indicar que este objeto nos puede transportar a otro lugar)

cuando pase por encima de este objeto interactivo. Los posibles valores que hay son Left (izquierda), Right (derecha), Up (arriba) y Bottom (abajo). Por defecto el valor es Left.

Y esto es todo lo necesario para crear un objeto interactivo básico. Si por el contrario queremos crear un nuevo *script* que defina un comportamiento distinto para un objeto interactivo, estos son los pasos a seguir:

1. Nos vamos al panel de Project, nos dirigimos a la carpeta Assets/Scripts/InteractiveElements y creamos una nueva carpeta con el nombre de la nueva escena (por ejemplo NewSceneElements).
2. Una vez dentro del directorio, creamos un nuevo *script* haciendo click derecho y eligiendo la opción Create → C# Script y de nombre le ponemos el tipo de objeto interactivo que va a ser (por ejemplo, si va a ser una ventana, debería llamarse Window).
3. Abrimos el *script* recién creado haciendo doble click y se abrirá Monodevelop con una plantilla de *script* MonoBehaviour.
4. Dependiendo del tipo de comportamiento de objeto interactivo en el que nos queramos basar, cambiaremos la herencia de la clase del *script* de MonoBehaviour a InteractiveElement, PickableElement o WarpElement. Por lo demás, se adjunta un ejemplo de *script* con los métodos que se pueden modificar para extender las funcionalidades del objeto interactivo:

```
1  i»{
2  using UnityEngine;
3  using System.Collections;
4  using System.Collections.Generic;
5
6  public class TemplateElement : PickableElement {
7
8      protected override void InitializeInformation() {
9          //Write here the extra info for your interactive
10         element or warper element
11     }
12
13     protected override void InitializePickableInformation() {
14         //Write here the extra info for your pickable element
15         (don't override InitializeInformation() then)
16     }
17
18     //When this object is left clicked, it executes this coroutine
19     protected override IEnumerator WaitForLeftClickAction(){
20         //Calculate distance between Player and the position
21         of the interactive element in which Player can
22         manipulate this object.
23         float _distanceBetweenActorAndInteractivePosition =
24             Mathf.Abs(Vector2.Distance(Player.Instance.
25                 currentPosition, interactivePosition));
26
27         //If the distance is less than min distance, Player
28         won't move
29     }
30 }
```

```

22         if (_distanceBetweenActorAndInteractivePosition >=
23             minDistance) {
24             Player.Instance.GoTo(interactivePosition); // Tell Player to go to the interactive
25             position of the object.
26             do {
27                 yield return null;
28             }while (Player.Instance.isWalking); //Wait until Player reaches the position
29         }
30         // When Player finish walking, check if it is still
31         // going to this object position or if the Player
32         // target position
33         // has changed
34         if (Player.Instance.originalTargetedPosition ==
35             interactivePosition) {
36             // This tells Player and the other objects
37             // that Player is interacting with this object
38             // , so Player couldn't be interrupted
39             // when doing these actions if clicked another
40             // interactive element or the options menu
41             BeginAction();
42
43             //If we want the Player to speak at the
44             // interactive object
45             Player.Instance.Speak(groupID, nameID, "INTERACTION");
46             do {
47                 yield return null;
48             }while (Player.Instance.isSpeaking); //Wait until Player finishes speaking
49
50             yield return new WaitForSeconds(0.2f); //This
51             // is just for not making the actions be
52             // played so quickly
53             // In case we are extending a PickableElement
54             // class, we can make Player manipulate this
55             // element
56             Player.Instance.Manipulate(this);
57
58             //...More methods can be here to make more
59             // actions
60
61             // This tells Player and the other object that
62             // Player has finished interacting with this
63             // object,
64             // so it can interact with other object or the
65             // options menu now
66             EndAction();
67         }
68     }
69 }
```

```

57     //If you want that this object can interact with an specific
58     // item of the inventory
59     public override void ActionOnItemInventoryUsed(GameObject
60         itemInventory){
61         /* How to use it:
62          if(!Player.Instance.isDoingAction && !CutScenesManager
63              .IsPlaying())){
64              switch(itemInventory.name) {
65                  case "name 1":
66                      itemInventory.GetComponent<
67                          ItemInventory>().Unselect()
68                      ;
69                      DoSomething1.....
70                      break;
71                  case "name 2":
72                      itemInventory.GetComponent<
73                          ItemInventory>().Unselect()
74                      ;
75                      DoSomething2.....
76                      break;
77                  ...
78              }
79          */
80      }
81
82      // If this object is extending PickableElement class, this
83      // method is needed in order to
84      // make the Manipulate method work when the Player is making
85      // the animation of touching
86      // this object
87      // Also, this method can be override to make Player actions
88      // completely differents that picking
89      // a list of items for the inventory, but here will be an
90      // example of picking items
91      public override void OnPlayerTouchingAction(){
92          // If the inventory is full of objects, the Player
93          // will say it instead of picking
94          // the items of the nameListGivableElements
95          if(Inventory.Instance.IsInventoryFull()){
96              Player.Instance.Speak("GUI", "DEFAULT", "FULL_INVENTORY");
97              do{
98                  yield return null;
99              }while(Player.Instance.isSpeaking); //Wait
100                 until Player finishes speaking
101          }
102          else{
103              // This method adds the items in
104              // nameListGivableElements to the inventory
105              Player.Instance.GrabItem(
106                  nameListGivableElements);
107
108              // If isDestroyedAfterBeingPicked was
109              // initialized with true, it will set
110              // this object inactive

```

```

96         if (isDestroyedAfterBeingPicked) {
97             SetInactive();
98         }
99     }
100 }
101 }
```

Por último, puede darse el caso de que queramos que **Player** pueda dibujarse por delante del objeto interactivo (en el caso de que tenga un componente `SpriteRenderer`) cuando su eje Y sea más bajo que el del objeto, y que cuando el eje Y de **Player** sea más alto, el objeto interactivo se dibuje por encima de él. Para poder simular este cambio de profundidades en el juego, deberemos añadir al objeto interactivo desde el Inspector el script `SortingOrderLayerController` y veremos que tiene los siguientes parámetros:

Change At Player Position Y Le decimos la posición de **Player** en el eje Y donde queremos que se cambie el orden de dibujado del *sprite* de este objeto.

Sorting Layer Name After Change Aquí indicamos a la capa de dibujado del *sprite* que queremos que cambie cuando **Player** alcance un valor en el eje Y mayor que el indicado en **Change At Player Position Y**. Por defecto como el *sprite* del objeto interactivo se dibuja en la `Sorting Layer` de `BackgroundInteractiveObjects` (que se dibuja por detrás de **Player**), el nombre de la `Sorting Layer` que viene por defecto en este parámetro es `FrontlineScenery` (que se dibuja por delante de **Player**). No obstante, estos valores los podemos invertir si se diera el caso de que el objeto interactivo inicialmente se dibujase por delante de **Player**.

Sorting Layer Position After Change En caso de que el *sprite* del objeto interactivo pueda solaparse con otro a la otra de dibujarse en la misma `Sorting Layer`, deberemos cambiar el valor del orden de dibujado del *sprite* dentro de la nueva `Sorting Layer`. Si no es el caso, dejamos el valor por defecto a 1.

Con esto ya podremos realizar todos los objetos interactivos que queramos. Por suerte una vez aprendida la base de cómo hacer los objetos interactivos, el crear los personajes no controlables (o NPCs), no se hará complicado.

Creación de Personajes No Controlables (NPCs)

Con el escenario lleno de objetos interactivos, ya no se siente tan vacío, pero no hay ningún personaje, ser (o cosa) con la que se pueda entablar una conversación. Para crear un personaje, deberemos implementar un NPC dentro del escenario en *Unity*. El proceso es parecido al de crear un objeto interactivo personalizado, pero más largo dado que hay que añadirle más componentes, *scripts* y algunos objetos.

En este caso, cada NPC tiene que tener dos *scripts*: `Actor` y `ActorAnimStateMachine`. Para el caso de `ActorAnimStateMachine` hacen falta conocimientos de cómo crear animaciones 2D en *Unity* e integrarlas su nuevo sistema Mecanim, esta guía no cubre esa información, pero puedes visitar los tutoriales oficiales de *Unity* para aprenderlos <https://unity3d.com/es/learn/tutorials/topics/animation>.

Habiendo puntualizado esta información necesaria para aprender poder crear un NPC, tenemos que seguir los siguientes pasos:

1. Con el nuevo escenario abierto dentro de *Unity*, dentro del panel de *Hierarchy* le damos a *Create* → *Create Empty* y al objeto recién creado le ponemos el nombre del personaje (para facilitar la nomeclatura, lo llamaremos **NewNPC**).
2. Desde el *Inspector* de **NewNPC**, le daños al botón *Add Component* y elegimos *Sprite Renderer*. Este nuevo componente tendrá varios parámetros, de todos ellos tenemos que modificar los siguientes:

Sprite Aquí colocamos el gráfico o *sprite* que vayamos a usar inicialmente para el personaje.

Luego una vez comience la partida, este gráfico será sustituido por el del estado inicial de animación que vaya a tener **NewNPC**. No obstante, es aconsejable colocar el mismo *sprite* que se vaya a utilizar para la animación pasiva del personaje, con el propósito de saber sus dimensiones y saber dónde colocarlo después.

Sorting Layer Si queremos que *sprite* de **NewNPC** se dibuje detrás del de **Player**, le pondremos de valor *BackgroundInteractiveObjects*. En caso contrario, tenemos dos opciones:

- Pondremos el valor de *Actors* si queremos que se dibuje por delante de **Player**, pero detrás de los objetos interactivos con *FrontlineScenery* en su *Sorting Layer*.
- Pondremos el valor de *FrontlineScenery* si queremos que se dibuje por delante de **Player** y los objetos interactivos con *FrontlineScenery* en su *Sorting Layer*.

Sorting Order Aquí indicamos el orden de dibujado del *sprite* con respecto a los objetos que estén en la misma *Sorting Layer*, en caso de que se solape con otro objeto interactivo o NPC, deberemos ir cambiando el número de aquí según el orden de dibujado de *sprites* que queramos. Por el valor por defecto que podemos ponerle será 1.

3. En el panel *Scene* usamos sus controles y lo colocamos en el lugar del escenario que queramos.
4. Habiendo colocado en el sitio que queramos que esté **NewNPC** dentro del escenario, lo siguiente es volver al *Inspector* y añadir el componente *Animator* con el botón *Add Component*. En el *Animator*, deberemos cambiar el valor del siguiente parámetro:

Controller Aquí insertaremos el controlador de la máquina de estado de animaciones para **NewNPC** creada mediante *Mecanim* de *Unity*. Dicho controlador deberíamos encontrarlo en el panel *Project* dentro del directorio *Assets/Resources/Animations/NewNPC/* y arrastrarlo desde ahí hasta este parámetro.

5. Desde el *Inspector* le añadimos al objeto el componente *PolygonCollider2D*, le damos a la opción *Edit Collider* y desde el panel *Scene* le damos forma al polígono de **NewNPC**. La función del *PolygonCollider2D* es que **NewNPC** sea detectado por el *script MouseClickHandler*. De esta forma el cursor cambiará al pasar por encima de **NewNPC**, aparecerá su nombre en la interfaz gráfica del nivel, y el objeto **Player** podrá interaccionar con él al clickar encima suya.
6. Desde el panel *Project*, seleccionamos los objetos hijos del *prefab* de **Player** dentro del directorio *Assets/Resources/Prefabs/Actors/* (los *prefabs* hijos son: **dialogText**, **WalkingPoint** y **Canvas**) y lo arrastramos dentro de **NewNPC** en el panel *Hierarchy*. Con cada uno de estos objetos deberemos hacer lo siguiente:

dialogText Desde el panel Scene cambiamos su posición por dónde sería la cabeza del personaje que simbolice **NewNPC**. Donde coloquemos dialogText será donde aparezcan los diálogos de este personaje cuando hable con **Player**.

WalkingPoint Desde el panel Scene cambiamos su posición por dónde queremos que **Player** vaya cuando se haga click encima de **NewNPC**.

Canvas Desde Hierarchy, seleccionamos **Canvas** y buscamos el objeto hijo **Canvas/Panel/ActorUIText**. Seleccionamos **ActorUIText** para que aparezca en el Inspector. Dentro del Inspector buscamos el componente Text y cambiamos el parámetro Color por el color que queramos que tengan los diálogos de **NewNPC**.

7. Desde el panel Project desde el directorio Resources/Assets/Scripts/GUI/InGameUIElements/ cogemos el script *DisplayDialogueText* y lo arrastramos hasta el objeto **NewNPC** dentro de Hierarchy. De esta manera se unirá como componente al objeto **NewNPC**. El script *DisplayDialogueText* sirve para controlar que los diálogos del personaje se dibujen correctamente en pantalla durante el juego.
8. Desde el panel Project desde el directorio Resources/Assets/Scripts/Actors/ cogemos el script *Actor* y lo arrastramos hasta el objeto **NewNPC** dentro de Hierarchy. Seleccionamos el objeto **NewNPC** dentro de Hierarchy para que aparezca en el Inspector, veremos que en el script que acabamos que añadir aparecen varios parámetros. De todos ellos, tendremos que modificar los siguientes parámetros que aparecen:

Name ID El ID de la etiqueta element dentro de los ficheros LocatedTexts.xml de donde sacará **NewNPC** los diálogos. El valor de este parámetro tendremos que cambiarlo por el ID correspondiente al de este NPC. Por ejemplo, para el NPC bibliotecaria, el ID es LIBRARIAN.

Movement Speed Este parámetro sirve para indicar como de rápido se mueve el NPC al caminar de un sitio a otro del escenario. El valor por defecto de la velocidad es 1.

9. Desde el panel Project desde el directorio Resources/Assets/Scripts/Actors/ cogemos el script *ActorAnimStateMachine* y lo arrastramos hasta el objeto **NewNPC** dentro de Hierarchy.

De manera opcional, al igual que con los scripts de objetos interactivos, los scripts *Actor* y *ActorAnimStateMachine* pueden ser heredados en nuevos scripts para crear NPCs con más funcionalidades aparte de las básicas.

En primer lugar, estas son las acciones básicas que puede hacer un NPC:

Speak Acción de hablar.

Go To Acción de ir a un sitio específico del escenario.

Si queremos que nuestro NPC haga más acciones que las siguientes, deberemos crear un script que herede del script *Actor* y que le añada más acciones. Un ejemplo de NPC extendido sería tal que así:

```
1  using UnityEngine;
2  using System.Collections;
```

```

3   using System.Collections.Generic;
4
5   public sealed class NewNPC : Actor {
6       public bool isWaiting { get; private set; }
7       public bool isDoingAction { get; private set; }
8       public bool isUsingItemInventory { get; private set; }
9       public bool isGrabbingUpperItem { get; private set; }
10      public bool isGrabbingBottomItem { get; private set; }
11      public bool isTouchingItemAnimEventActivated { get; private set; }
12      public bool isDoingAnotherAction { get; private set; }
13      //Mas acciones declaradas...
14
15      //Nos indica si ha alcanzado la posicion objetivo del escenario
16      private bool isPositionInPathReached;
17      //Nos indica la posicion el camino a la que esta apuntando ahora mismo
18      //el NPC
19      //para llegar a la posicion objetivo del escenario mediante el sistema
19      //de navegacion
20      private Vector2 currentPathStepPosition;
21
22      //El tiempo en segundos que va a esperar el NPC sin hacer nada hasta
22      //realizar
23      //la accion de esperar
24      [SerializeField]
25      private const float maxTimeIdleUntilWaitingAnimation = 6f;
26      //Inicializamos el contador de segundos para calcular luego si ha
26      //pasado el
27      //tiempo maximo de espera indicado arriba
28      private float timeCounterUntilWaitingAnimation;
29
30      protected override void InitializeAdditionalActorInformation() {
31          isWaiting = false;
32          isDoingAction = false;
33          isUsingItemInventory = false;
34          isGrabbingUpperItem = false;
35          isGrabbingBottomItem = false;
36          isTouchingItemAnimEventActivated = false;
37          //isDoingAnotherAction = false;
38          //Mas acciones...
39
40          //Podemos hacer que dependiendo del estado de las cinematicas o de
40          //otros estados
41          //guardados en la case GameState se pueda volver inactivo al NPC
42          if(!GameState.CutSceneData.isPlayingCutscene) {
43              SetInactive();
44          }
45
46          timeCounterUntilWaitingAnimation = Time.time;
47      }
48
49      //Metodo a ejecutar los cambios en las acciones que realice el NPC en
49      //cada ciclo de actualizacion
50      //de datos mientras se juega
51      protected override void Update (){
52          //Dependiendo de si el NPC no esta inactivo, ejecuta lo siguiente
52          if(!isInactive){

```

```

53     //Si el NPC se esta moviendo
54     if(moveDirection != Vector2.zero){
55         //Calculamos la distancia absoluta entre la posicion del
56         //NPC con la posicion objetivo
57         float _distance = Mathf.Abs(Vector2.Distance(
58             currentPosition, currentTargetedPosition));
59
60         //Si la distancia es mayor que la distancia minima
61         if(_distance > minDistance){
62             //El NPC se sigue moviendo
63             this.transform.position = new Vector2(currentPosition.x
64                 + moveDirection.x, currentPosition.y +
65                 moveDirection.y);
66             currentPosition = this.transform.position;
67         }
68         //Si la distancia hacia la posicion objetivo es menor que
69         //la distancia minima
70         else{
71             //El NPC deja de moverse
72             moveDirection = Vector2.zero;
73             //Indicamos que ha llegado a la posicion objetivo
74             //dentro del camino
75             isPositionInPathReached = true;
76             //Si la ultima posicion objetivo alcanzada era igual a
77             //la posicion objetivo final a la que queriamos
78             //que fuera el NPC
79             if(currentPathStepPosition == originalTargetedPosition)
80                 {
81                     //El NPC deja de caminar
82                     isWalking = false;
83                 }
84             }
85         }
86
87         //Si el NPC no esta realizando una accion, no se esta
88         //producido ninguna escena cinematica en el juego
89         //y tampoco esta realizando la accion de espera
90         if(IsIdle() && !CutScenesManager.isPlaying() && !isWaiting){
91             //Si se ha superando el tiempo maximo de espera sin
92             //realizar la animacion
93             if((Time.time - timeCounterUntilWaitingAnimation) >
94                 maxTimeIdleUntilWaitingAnimation){
95                 //El NPC empieza a realizar la animacion de esperar
96                 isWaiting = true;
97                 //Reseteamos el contador de espera a cero
98                 timeCounterUntilWaitingAnimation = Time.time;
99             }
100         }
101         //Si el NPC esta realizando una accion, se esta produciendo una
102         //escena cinematica o
103         //se esta realizando la accion de espera
104         else{
105             //Reseteamos el contador de espera a cero
106             timeCounterUntilWaitingAnimation = Time.time;
107         }
108     }

```

```

97         //Mas cosas....  

98     }  

99  

100    //Metodo para ir en linea recta hacia la nueva posicion objetivo  

101   public void GoToInStraightDirection(Vector2 newPosition){  

102       StartCoroutine(WaitForGoToCompleted(newPosition));  

103   }  

104  

105    //Metodo para ir a la nueva posicion objetivo mediante de busqueda de  

106     caminos que hay  

107     //en el escenario  

108   public override void GoTo (Vector2 newPosition){  

109       StartCoroutine(WaitForGoToInPathCompleted(newPosition));  

110   }  

111  

112    //Corrutina para caminar hacia la nueva posicion objetivo mediante la  

113     busqueda de caminos que  

114     //hay en el escenario  

115   private IEnumerator WaitForGoToInPathCompleted(Vector2 newPosition){  

116       //Si la nueva posicion no es la misma que habiamos ido antes  

117       if(originalTargetedPosition != newPosition){  

118           do{  

119               yield return null;  

120           }while(isSpeaking && isInteracting && isInConversation/* &&  

121             isAnotherAction... */); //Esperamos a que termine de  

122             ejecutar otras acciones  

123  

124           if(isWalking){ //Si estaba caminando previamente cuando le  

125             mandamos la nueva posicion objetivo  

126             //Paramos la corutina de caminar que se estaba ejecutando  

127             antes  

128             StopCoroutine(walkingCoroutine);  

129             //Le decimos que ha llegado al final de esa ruta  

130             isPositionInPathReached = true;  

131           }  

132  

133           //Reseteamos los valores para empezar a caminar de nuevo  

134           isWalking = true;  

135           moveDirection = Vector2.zero;  

136           originalTargetedPosition = newPosition;  

137           //Obtiene la lista de las posiciones del camino a las que hay  

138             que ir hasta llegar al objetivo  

139           List<Vector2> _path = NavigationManager.Instance.FindPath(  

140             currentPosition, newPosition);  

141  

142           //Asigna la corutina de caminar que se esta ejecutando  

143             actualmente  

144           walkingCoroutine = StartCoroutine(WaitForPathCompleted(_path));  

145       }  

146   }  

147  

148   //Corrutina que se encarga de calcular por cual posicion y direccion  

149     tiene que ir el NPC  

150     //mientras camina siguiendo una ruta  

151   private IEnumerator WaitForPathCompleted(List<Vector2> path) {  

152

```

```

143     if(path.Count == 1) {
144         isPositionInPathReached = false;
145         currentPathStepPosition = path[0];
146         moveDirection = CalculateMoveDirection(currentPathStepPosition)
147             ;
148         do{
149             yield return null;
150             while(!isPositionInPathReached && isWalking);
151         }
152         else if(path.Count > 1){
153             for(int i = 0; i < path.Count; i++){
154                 isPositionInPathReached = false;
155                 currentPathStepPosition = path[i];
156                 moveDirection = CalculateMoveDirection(
157                     currentPathStepPosition);
158                 do{
159                     yield return null;
160                     while(!isPositionInPathReached);
161                 }
162             do{
163                 yield return null;
164                 while(isWalking);
165             }
166
167             //Para que el NPC pueda interaccionar con un objeto del inventario
168             public override void ActionOnItemInventoryUsed(GameObject itemInventory
169                 ) {
170                 /* Como usarlo:
171                 if(!Player.Instance.isDoingAction && !CutScenesManager.isPlaying())
172                 {
173                     switch(itemInventory.name){
174                         case "name 1":
175                             itemInventory.GetComponent<ItemInventory>().Unselect();
176                             DoSomething1....;
177                             break;
178                         case "name 2":
179                             itemInventory.GetComponent<ItemInventory>().Unselect();
180                             DoSomething2....;
181                             break;
182                         ...
183                     }
184                 */
185
186             //Metodo para que el NPC interactue con un objeto interactivo del
187             //escenario
188             public void Manipulate(PickableElement element){
189                 //Dependiendo de si el objeto interactivo esta a la altura del
190                 //suelo o del NPC
191                 //ejecuta una corrutina u otra
192                 if(element.currentPickablePosition == PickableElement.PickableFrom.
193                     up){
194                     StartCoroutine(WaitForUpperInteractionCompleted(element));

```

```

192     }
193     else{
194         StartCoroutine(WaitForBottomInteractionCompleted(element));
195     }
196 }
197
198 //Metodo para que el NPC interactue con un objeto interactivo a su
199 //altura
200 public void UpperInteraction(PickableElement element){
201     StartCoroutine(WaitForUpperInteractionCompleted(element));
202 }
203
204 //Corutina para que el NPC interactue con un objeto interactivo a su
205 //altura
206 private IEnumerator WaitForUpperInteractionCompleted(PickableElement
207     element){
208     do{
209         yield return null;
210         //Si el NPC esta realizando otra accion, espera a que termine
211         //para poder realizar esta
212         while(isSpeaking && isWalking && isInteracting && isInConversation
213             );
214
215         //Indicamos que el NPC esta interactuando y
216         isInteracting = true;
217         isGrabbingUpperItem = true;
218         isPlayingAnimation = true;
219
220         do{
221             yield return null;
222             //Esperamos que la animacion llegue al punto en que esta
223             tocando
224             //el objeto interactivo
225         }while(!isTouchingItemAnimEventActivated);
226
227         //Ejecuta la accion que hace el objeto interactivo al ser
228         //manipulado
229         element.OnPlayerTouchingAction();
230
231         do{
232             yield return null;
233             //Esperamos que termine de ejecutarse la animacion
234         }while(!isPlayingAnimation);
235
236         //Indicamos que el NPC ya no esta realizando ninguna accion
237         isGrabbingUpperItem = false;
238         isInteracting = false;
239     }
240
241 //Metodo para que el NPC interactue con un objeto interactivo a la
242 //altura del suelo
243 public void BottomInteraction(PickableElement element){
244     StartCoroutine(WaitForBottomInteractionCompleted(element));
245 }
246
247 //Corrutina para que el NPC interactue con un objeto interactivo a la

```

```

altura del suelo
241 private Ienumerator WaitForBottomInteractionCompleted(PickableElement
242     element){
243     do{
244         yield return null;
245         while(isSpeaking && isWalking && isInteracting && isInConversation
246             );
247
248         isInteracting = true;
249         isGrabbingBottomItem = true;
250         isPlayingAnimation = true;
251
252         do{
253             yield return null;
254             while(!isTouchingItemAnimEventActivated);
255
256             element.OnPlayerTouchingAction();
257
258             do{
259                 yield return null;
260                 while(!isPlayingAnimation);
261
262                 isGrabbingBottomItem = false;
263                 isInteracting = false;
264             }
265
266             //Metodo que indica que se ha comenzado a realizar una animacion
267             public void StartOfAnimationEvent(){
268                 isPlayingAnimation = true;
269                 endOfAnimationEvent = false;
270                 isTouchingItemAnimEventActivated = false;
271             }
272
273             //Metodo que indica que la animacion ha llegado al punto en el
274             //que toca a un objeto interactivo
275             public void TouchingItemAnimEvent(){
276                 isTouchingItemAnimEventActivated = true;
277             }
278
279             //Metodo que indica que se ha finalizado la animacion que se estaba
280             //realizando
281             public void EndOfAnimationEvent(){
282                 isPlayingAnimation = false;
283                 endOfAnimationEvent = true;
284                 isTouchingItemAnimEventActivated = false;
285             }
286
287             //Metodo que nos indica si el NPC en este momento esta realizando una
288             //accion o no
289             public override bool IsIdle(){
290                 if(isWalking || isDoingAction || isInteracting || isSpeaking ||
291                     isGrabbingUpperItem || isGrabbingBottomItem || isInConversation
292                     /* || isAnotherAction.... */){
293                     return false;
294                 }
295             else{

```

```

290         return true;
291     }
292 }
293 /*
294 public void Waiting(){
295     //Do Waiting Action...
296 }
297
298 public void AnotherAction(){
299     //Do Another Action.....
300 }
301
302 /*
303 //Mas metodos de nuevas acciones.....
304 */
305
306
307 //Corrutina que se ejecuta cuando el NPC es clickado con el boton
308 //derecho del raton para que interactue con Player
309 protected override IEnumerator WaitForRightClickAction(){
310     //Calcula la distancia que queda entre el NPC y Player
311     float _distanceBetweenActorAndInteractivePosition = Mathf.Abs(
312         Vector2.Distance(Player.Instance.currentPosition,
313         interactivePosition));
314
315     //Si la distancia es mayor que la distancia minima, hacemos que
316     //Player vaya a la direccion donde este el NPC
317     ////(o pueda interactuar con el)
318     if(_distanceBetweenActorAndInteractivePosition >= minDistance){
319         Player.Instance.GoTo(interactivePosition);
320
321         do{
322             yield return null;
323             }while(Player.Instance.isWalking);
324     }
325
326     //Una vez Player termina de caminar, nos aseguramos que no ha
327     //cambiado de ruta
328     //Si no es asi, ejecutamos lo siguiente
329     if(Player.Instance.originalTargetedPosition == interactivePosition)
330     {
331         //Metodo que inicializa el evento para indicar que Player ha
332         //comenzado a realizar una accion
333         BeginAction();
334
335         //Hacemos que Player mire hacia donde este el NPC
336         if(Player.Instance.transform.position.x < (interactivePosition.
337             x + spriteWidth * 0.5f)){
338             Player.Instance.LookToTheRight();
339         }
340         else{
341             Player.Instance.LookToTheLeft();
342         }
343
344         //Player hara una serie de dialogos describiendo el NPC
345         Player.Instance.Speak(groupID, nameID, "DESCRIPTION");
346     }
347 }
```

```

338     do{
339         yield return null;
340         //Esperamos que Player termine de hablar
341     }while(Player.Instance.isSpeaking);
342
343         //Esperamos una decima de segundo para que la accion no termine
344         //tan rapido
345     yield return new WaitForSeconds(0.1f);
346
347         //Metodo que inicializa el evento que indica que Player ha
348         //finalizado de realizar una accion
349     EndAction();
350 }
351
352 //Corrutina que se ejecuta cuando el NPC es clickado con el boton
353 //izquierdo del raton para que interactue con Player
354 protected override IEnumerator WaitForLeftClickAction(){
355     //Calcula la distancia que queda entre el NPC y Player
356     float _distanceBetweenActorAndInteractivePosition = Mathf.Abs(
357         Vector2.Distance(Player.Instance.currentPosition,
358         interactivePosition));
359
360     //Si la distancia es mayor que la distancia minima, hacemos que
361     //Player vaya a la direccion donde este el NPC
362     ////(o pueda interactuar con el)
363     if(_distanceBetweenActorAndInteractivePosition >= minDistance){
364         Player.Instance.GoTo(interactivePosition);
365
366         do{
367             yield return null;
368         }while(Player.Instance.isWalking);
369     }
370
371     //Una vez Player termina de caminar, nos aseguramos que no ha
372     //cambiado de ruta
373     //Si no es asi, ejecutamos lo siguiente
374     if(Player.Instance.originalTargetedPosition == interactivePosition)
375     {
376         //Metodo que inicializa el evento para indicar que Player ha
377         //comenzado a realizar una accion
378         BeginAction();
379
380         //Hacemos que Player mire hacia donde este el NPC
381         if(Player.Instance.transform.position.x < (interactivePosition.
382             x + spriteWidth * 0.5f)){
383             Player.Instance.LookToTheRight();
384         }
385         else{
386             Player.Instance.LookToTheLeft();
387         }
388
389         //Si el NPC esta ejecutando la accion de espera, hacemos que
390         //Player se espere hasta
391         //que acabe de realizarla
392         if(isWaiting) {

```

```

383     Player.Instance.Speak(groupID, nameID, "NPC_IS_BUSY");
384
385     do {
386         yield return null;
387     }while(Player.Instance.isSpeaking);
388
389 }
390 //Tambien podemos hacer que dependiendo del estado de GameState
391 //de la nueva escena
392 //se haga una interaccion entre NPC y Player distinta
393 else if(!GameState.NewSceneData.isEvent1Finished) {
394     //Podemos hacer que haya una conversacion entre Player y
395     //NPC
396     BeginConversation();
397
398     Player.Instance.Speak(groupID, nameID, "
399         PLAYER_CONVERSATION_NEW-NPC_1");
400     do {
401         yield return null;
402     }while(Player.Instance.isSpeaking);
403
404     this.Speak(groupID, nameID, "NEW-NPC_CONVERSATION_PLAYER_1"
405         );
406     do {
407         yield return null;
408     }while(this.isSpeaking);
409
410     //Sigue la conversacion...
411
412     //O que ocurra una eleccion de dialogo que dependiendo del
413     //resultado el NPC dira una cosa u otra
414     MultipleChoiceManager.Instance.CreateMultipleSelection("NEW-
415         -NPC_CONVERSATION_CHOICE");
416     do {
417         yield return null;
418         //Esperamos a que el jugador termine de elegir una
419         //respuesta
420     }while(!MultipleChoiceManager.Instance.isSelectionEnded);
421
422     int _resultChoice = MultipleChoiceManager.Instance.
423         GetSelectionResult();
424     switch(_resultChoice){
425         case 0:
426             this.Speak(groupID, nameID, "NEW-
427                 NPC_CONVERSATION_RESULT_CHOICE_0");
428             yield return null;
429
430             do {
431                 yield return null;
432             }while(this.isSpeaking);
433
434             this.Speak(groupID, nameID, "
435                 PLAYER_CONVERSATION_RESULT_CHOICE_0");
436             yield return null;
437
438             do {

```

```

429             yield return null;
430         }while(this.isSpeaking);
431
432         //...
433
434         //Esperamos a que se terminen de ejecutar las
435         //acciones por elegir la opcion 0.....
436         yield return StartCoroutine(
437             PlayerActionsOnChoiceResult0());
438
439         break;
440     case 1:
441         this.Speak(groupID, nameID, "NEW-
442             NPC_CONVERSATION_RESULT_CHOICE_1");
443         do{
444             yield return null;
445         }while(this.isSpeaking);
446
447         this.Speak(groupID, nameID, "
448             PLAYER_CONVERSATION_RESULT_CHOICE_1");
449         yield return null;
450
451         do{
452             yield return null;
453         }while(this.isSpeaking);
454
455         //...
456
457         //Esperamos a que se terminen de ejecutar las
458         //acciones por elegir la opcion 1.....
459         yield return StartCoroutine(
460             PlayerActionsOnChoiceResult1());
461
462         break;
463     default:
464         //Si elige otra opcion, podemos hacer que se
465         //termine la interaccion o que se continue
466         //sin que haya importado la eleccion elegida
467         break;
468     }
469
470     //Sigue la conversacion...
471
472     //Finalizamos la conversacion
473     EndConversation();
474
475     //Y otras acciones mediante los metodos creados antes....
476 }
```

```
477     }
478 }
```

Como podemos ver en el ejemplo, el NPC personalizado puede llegar a ser tan complejo como queramos, sin límites de acciones a agregar, o de acciones cuando interactúa con **Player**.

En segundo lugar, las animaciones básicas que puede controlar el *script* ActorAnimStateMachine son las siguientes:

Idle Animación pasiva en la que el personaje no está realizando ninguna acción.

Speaking Animación en la que el personaje está hablando.

Walking Animación en la que el personaje está caminando.

Si queremos añadir a un personaje más animaciones de las aquí presentadas (por ejemplo, una animación de coger un objeto o de hacer alguna actividad mientras está esperando que ocurra algo) porque realiza más acciones que un NPC básico, tendríamos que crear un *script* que herede de ActorAnimStateMachine tal que así:

```
1  using UnityEngine;
2  using System.Collections;
3
4  //Dado que NewNPCAnimStateMachine depende en todo momento de las acciones
   //que este realizando
5  //el NPC llamado NewNPC, hacemos que obligue que al objeto al que se le añ
   //ada este script
6  //tambien se le añada el otro
7  [RequireComponent(typeof(NewNPC) )]
8
9  //Hacemos que herede del script ActorAnimStateMachine para poder
10 //expandir las animaciones posibles que puede hacer
11 public sealed class NewNPCAnimStateMachine : ActorAnimStateMachine {
12     //El script que se encarga de controlar las posibles acciones que puede
       //hacer el NPC
13     private NewNPC newNPC;
14     //El estado de animacion actual del NPC
15     private NewNPCStates currentState;
16
17     //Inicializacion de la informacion extra necesaria al cargar el script
18     protected override void InitializeAnimData(){
19         //Inicializamos el parametro con el script del NPC cogiendolo del
           //mismo objeto
20         //al que esta unido este script
21         newNPC = this.GetComponent<NewNPC>();
22     }
23
24     //Para cada ciclo de actualizacion de datos del objeto durante el juego
25     //se comprueba si el NPC ha empezado una nueva accion o no
26     protected override void Update() {
27         //Si el personaje ha empezado a caminar
28         if(newNPC.isWalking){
```

```

29     //Dependiendo del lado hacia el que esta mirando
30     if(newNPC.isFacingLeft){
31         OnAnimationStateChange (NewNPCStates.
32             leftWalking);
33         currentState = NewNPCStates.leftWalking;
34     }
35     else if(newNPC.isFacingRight){
36         OnAnimationStateChange (NewNPCStates.
37             rightWalking);
38         currentState = NewNPCStates.rightWalking;
39     }
40     //Si el personaje ha empezado a hablar
41     else if(newNPC.isSpeaking){
42         //Dependiendo del lado hacia el que esta mirando
43         if(newNPC.isFacingLeft){
44             OnAnimationStateChange (NewNPCStates.
45                 leftSpeaking);
46             currentState = NewNPCStates.leftSpeaking;
47         }
48         else if(newNPC.isFacingRight){
49             OnAnimationStateChange (NewNPCStates.
50                 rightSpeaking);
51             currentState = NewNPCStates.rightSpeaking;
52         }
53     }
54     //Dependiendo de si ha empezado a coger un objeto a la misma altura
55     //del NPC
56     else if(newNPC.isGrabbingUpperItem){
57         //Dependiendo del lado hacia el que esta mirando
58         if(newNPC.isFacingLeft){
59             OnAnimationStateChange (NewNPCStates.
60                 leftUpperGrabbing);
61             currentState = NewNPCStates.
62                 leftUpperGrabbing;
63         }
64         else if(newNPC.isFacingRight){
65             OnAnimationStateChange (NewNPCStates.
66                 rightUpperGrabbing);
67             currentState = NewNPCStates.
68                 rightUpperGrabbing;
69         }
70     }
71     //Dependiendo de si ha empezado a coger un objeto a la altura del
72     //suelo
73     else if(newNPC.isGrabbingBottomItem){
74         //Dependiendo del lado hacia el que esta mirando
75         if(newNPC.isFacingLeft){
76             OnAnimationStateChange (NewNPCStates.
77                 leftBottomGrabbing);
78             currentState = NewNPCStates.
79                 leftBottomGrabbing;
80         }
81         else if(newNPC.isFacingRight){
82             OnAnimationStateChange (NewNPCStates.
83                 rightBottomGrabbing);

```

```

72                     currentState = NewNPCStates.
73                         rightBottomGrabbing;
74                 }
75             }
76             //Dependiendo de si ha empezado la animacion de espera mientras no
77             //estaba realizando ninguna accion
78             else if(newNPC.isWaiting){
79                 //Dependiendo del lado hacia el que esta mirando
80                 if(newNPC.isFacingLeft){
81                     OnAnimationStateChange (NewNPCStates.
82                         leftWaiting);
83                     currentState = NewNPCStates.leftWaiting;
84                 }
85                 else if(newNPC.isFacingRight){
86                     OnAnimationStateChange (NewNPCStates.
87                         rightWaiting);
88                     currentState = NewNPCStates.rightWaiting;
89                 }
90             }
91             /*
92             //Si el personaje esta realizando otra accion
93             else if(newNPC.isDoingAnotherAction){
94                 //Dependiendo del lado hacia el que esta mirando
95                 if(newNPC.isFacingLeft){
96                     OnAnimationStateChange (NewNPCStates.leftDoingAnotherAction)
97                     ;
98                     currentState = NewNPCStates.leftDoingAnotherAction;
99                 }
100                else if(newNPC.isFacingRight){
101                    OnAnimationStateChange (NewNPCStates.rightDoingAnotherAction
102                        );
103                    currentState = NewNPCStates.rightDoingAnotherAction;
104                }
105            }
106            /*
107            //Si el personaje no esta realizando ninguna accion
108            else /*if(newNPC.IsIdle*/{
109                //Dependiendo del lado hacia el que esta mirando
110                if(newNPC.isFacingLeft){
111                    OnAnimationStateChange (NewNPCStates.
112                        leftIdle);
113                    currentState = NewNPCStates.leftIdle;
114                }
115                else if(newNPC.isFacingRight){
116                    OnAnimationStateChange (NewNPCStates.
117                        rightIdle);
118                    currentState = NewNPCStates.rightIdle;
119                }
120            }

```

```

121 //Cada vez que queramos cambiar de estado de animacion del NPC
122 //llamamos este NPC
123 private void OnAnimationStateChange(NewNPCStates newState) {
124     //Si al estado de animacion al que queremos cambiar es el mismo con
125     //el que estabamos
126     //anteriormente, se sale de este metodo sin hacer nada
127     if(newState == currentState){
128         return;
129     }
130
131     //En caso de que al nuevo estado de animacion al que queremos estar
132     //es distinto del estado
133     //de animacion que teniamos actualmente, cambiamos de estado en el
134     //componente
135     //Animator: el controlador de animaciones del NPC
136     switch(newState) {
137         //Si el nuevo estado es la animacion pasiva mirando a la
138         //izquierda
139         case NewNPCStates.leftIdle:
140             animator.SetBool("isWalking", false);
141             animator.SetBool("isSpeaking", false);
142             animator.SetBool("isUpperGrabbing", false);
143             animator.SetBool("isBottomGrabbing", false);
144             animator.SetBool("isFacingLeft", true);
145             animator.SetBool("isFacingRight", false);
146             //animator.SetBool("isWaiting", false);
147             //animator.SetBool("isAnotherAction", false);
148             break;
149         //Si el nuevo estado es la animacion pasiva mirando a la
150         //derecha
151         case NewNPCStates.rightIdle:
152             animator.SetBool("isWalking", false);
153             animator.SetBool("isSpeaking", false);
154             animator.SetBool("isUpperGrabbing", false);
155             animator.SetBool("isBottomGrabbing", false);
156             animator.SetBool("isFacingLeft", false);
157             animator.SetBool("isFacingRight", true);
158             //animator.SetBool("isWaiting", false);
159             //animator.SetBool("isAnotherAction", false);
160             break;
161         //Si el nuevo estado es la animacion de caminar mirando a
162         //la izquierda
163         case NewNPCStates.leftWalking:
164             animator.SetBool("isWalking", true);
165             animator.SetBool("isSpeaking", false);
166             animator.SetBool("isUpperGrabbing", false);
167             animator.SetBool("isBottomGrabbing", false);
168             animator.SetBool("isUsingPhone", false);
169             animator.SetBool("isFacingLeft", true);
170             animator.SetBool("isFacingRight", false);
171             //animator.SetBool("isWaiting", false);
172             //animator.SetBool("isAnotherAction", false);
173             break;
174         //Si el nuevo estado es la animacion de caminar mirando a
175         //la derecha
176         case NewNPCStates.rightWalking:

```

```

170         animator.SetBool("isWalking", true);
171         animator.SetBool("isSpeaking", false);
172         animator.SetBool("isUpperGrabbing", false);
173         animator.SetBool("isBottomGrabbing", false);
174         animator.SetBool("isUsingPhone", false);
175         animator.SetBool("isFacingLeft", false);
176         animator.SetBool("isFacingRight", true);
177         //animator.SetBool("isWaiting", false);
178         //animator.SetBool("isAnotherAction", false);
179         break;
180         //Si el nuevo estado es la animacion de hablar mirando a la
181         izquierda
182     case NewNPCStates.leftSpeaking:
183         animator.SetBool("isWalking", false);
184         animator.SetBool("isSpeaking", true);
185         animator.SetBool("isUpperGrabbing", false);
186         animator.SetBool("isBottomGrabbing", false);
187         animator.SetBool("isUsingPhone", false);
188         animator.SetBool("isFacingLeft", true);
189         animator.SetBool("isFacingRight", false);
190         //animator.SetBool("isWaiting", false);
191         //animator.SetBool("isAnotherAction", false);
192         break;
193         //Si el nuevo estado es la animacion de hablar mirando a la
194         derecha
195     case NewNPCStates.rightSpeaking:
196         animator.SetBool("isWalking", false);
197         animator.SetBool("isSpeaking", true);
198         animator.SetBool("isUpperGrabbing", false);
199         animator.SetBool("isBottomGrabbing", false);
200         animator.SetBool("isUsingPhone", false);
201         animator.SetBool("isFacingLeft", false);
202         animator.SetBool("isFacingRight", true);
203         //animator.SetBool("isWaiting", false);
204         //animator.SetBool("isAnotherAction", false);
205         break;
206         //Si el nuevo estado es la animacion de coger un objeto (a
207         la altura del NPC) mirando a la derecha
208     case NewNPCStates.rightUpperGrabbing:
209         animator.SetBool("isWalking", false);
210         animator.SetBool("isSpeaking", false);
211         animator.SetBool("isUpperGrabbing", true);
212         animator.SetBool("isBottomGrabbing", false);
213         animator.SetBool("isUsingPhone", false);
214         animator.SetBool("isFacingLeft", false);
215         animator.SetBool("isFacingRight", true);
216         //animator.SetBool("isWaiting", false);
217         //animator.SetBool("isAnotherAction", false);
218         break;
219         //Si el nuevo estado es la animacion de coger un objeto (a
220         la altura del NPC) mirando a la izquierda
221     case NewNPCStates.leftUpperGrabbing:
222         animator.SetBool("isWalking", false);
223         animator.SetBool("isSpeaking", false);
224         animator.SetBool("isUpperGrabbing", true);
225         animator.SetBool("isBottomGrabbing", false);

```

```

222         animator.SetBool("isUsingPhone", false);
223         animator.SetBool("isFacingLeft", true);
224         animator.SetBool("isFacingRight", false);
225         //animator.SetBool("isWaiting", false);
226         //animator.SetBool("isAnotherAction", false);
227         break;
228     //Si el nuevo estado es la animacion de coger un objeto (a
229     //la altura del suelo) mirando a la derecha
230     case NewNPCStates.rightBottomGrabbing:
231         animator.SetBool("isWalking", false);
232         animator.SetBool("isSpeaking", false);
233         animator.SetBool("isUpperGrabbing", false);
234         animator.SetBool("isBottomGrabbing", true);
235         animator.SetBool("isUsingPhone", false);
236         animator.SetBool("isFacingLeft", false);
237         animator.SetBool("isFacingRight", true);
238         //animator.SetBool("isWaiting", false);
239         //animator.SetBool("isAnotherAction", false);
240         break;
241     //Si el nuevo estado es la animacion de coger un objeto (a
242     //la altura del suelo) mirando a la izquierda
243     case NewNPCStates.leftBottomGrabbing:
244         animator.SetBool("isWalking", false);
245         animator.SetBool("isSpeaking", false);
246         animator.SetBool("isUpperGrabbing", false);
247         animator.SetBool("isBottomGrabbing", true);
248         animator.SetBool("isUsingPhone", false);
249         animator.SetBool("isFacingLeft", true);
250         animator.SetBool("isFacingRight", false);
251         //animator.SetBool("isWaiting", false);
252         //animator.SetBool("isAnotherAction", false);
253         break;
254     /*
255     //Si el nuevo estado es la animacion de esperar hasta que haga
256     //el NPC una accion mirando a la derecha
257     case NewNPCStates.rightWaiting:
258         animator.SetBool("isWalking", false);
259         animator.SetBool("isSpeaking", false);
260         animator.SetBool("isUpperGrabbing", false);
261         animator.SetBool("isBottomGrabbing", false);
262         animator.SetBool("isFacingLeft", false);
263         animator.SetBool("isFacingRight", true);
264         animator.SetBool("isWaiting", true);
265         animator.SetBool("isAnotherAction", false);
266         break;
267     //Si el nuevo estado es la animacion de esperar
268     //hasta que haga el NPC una accion mirando a la
269     //izquierda
270     case NewNPCStates.leftWaiting:
271         animator.SetBool("isWalking", false);
272         animator.SetBool("isSpeaking", false);

```

```

273         animator.SetBool("isAnotherAction", false);
274             break;
275
276     //Si el nuevo estado es la animacion de esperar hasta que haga
277     //el NPC una accion mirando a la derecha
278     case NewNPCStates.rightAnotherAction:
279         animator.SetBool("isWalking", false);
280         animator.SetBool("isSpeaking", false);
281         animator.SetBool("isUpperGrabbing", false);
282         animator.SetBool("isBottomGrabbing", false);
283         animator.SetBool("isFacingLeft", false);
284         animator.SetBool("isFacingRight", true);
285         animator.SetBool("isWaiting", false);
286         animator.SetBool("isAnotherAction", true);
287         break;
288     //Si el nuevo estado es la animacion es la nueva animacion del
289     //NPC mirando a la izquierda
290     case NewNPCStates.leftAnotherAction:
291         animator.SetBool("isWalking", false);
292         animator.SetBool("isSpeaking", false);
293         animator.SetBool("isUpperGrabbing", false);
294         animator.SetBool("isBottomGrabbing", false);
295         animator.SetBool("isFacingLeft", true);
296         animator.SetBool("isFacingRight", false);
297         animator.SetBool("isWaiting", false);
298         animator.SetBool("isAnotherAction", true);
299         break;
300
301     //Mas estados de animacion posibles....
302     /*
303 }
304
305     //Asignamos como estado de animacion actual el nuevo estado
306     currentState = newState;
307 }
308
309 //Ampliamos tambien la clase ActorStates que se usa para simular un enum
310 //pero que puede ser
311 //ampliado con nuevos estados, no como en el caso de los enum
312 public class NewNPCStates : ActorStates{
313     //Partimos del numero 5, dado que los estados basicos estan
314     //inicializados en la clase ActorStates
315     public const int leftUpperGrabbing = 6;
316     public const int rightUpperGrabbing = 7;
317     public const int leftBottomGrabbing = 8;
318     public const int rightBottomGrabbing = 9;
319     //public const int leftWaiting = 10;
320     //public const int rightWaiting = 11;
321     //public const int leftDoingAnotherAction = 12;
322     //public const int rightDoingAnotherAction = 13;
323     //Mas estados....
324
325     //El constructor de la clase si no se le pasa ningun valor
326     public NewNPCStates() {}

```

```

325
326 //El constructor de la clase si le pasamos un valor, el valor que le
327 //asignaremos
328 //será el estado actual que tendría asignada la clase
329 public NewNPCStates(int value) {
330     Value = value;
331 }
332
333 //Convertimos a int el valor que le asignamos de un NewNPCStates
334 public static implicit operator int(NewNPCStates type) {
335     return type.Value;
336 }
337
338 //Convertimos a NewPCStates el valor que le asignamos de un int
339 public static implicit operator NewNPCStates(int value) {
340     return new NewNPCStates(value);
341 }

```

Creación del sistema de navegación para el nivel

Ya tenemos nuestro nuevo escenario lleno de objetos interactivos y quizás con algún NPC, pero aunque dentro de *Unity* le demos al botón **Play** para probar el nuevo escenario, **Player** no se moverá aunque clickemos (es más, puede que de algún error al intentarlo). Esto es porque aún nos falta por crear los objetos con las zonas por las que **Player** podrá caminar. Los pasos para crear las zonas transitables del nuevo escenario son:

1. Con el nuevo escenario abierto en *Unity*, desde el panel **Project** vamos a **Assets/Resources/Prefabs/Navigation/** y arrastramos el *prefab* **Navigation** hasta el panel de **Hierarchy**.
2. Ahora dependiendo de la forma poligonal que vaya a tener la zona transitable del nuevo escenario (si es un polígono complejo, hay que dividir su forma en distintos polígonos de cuatro lados), añadiremos uno o varios *prefabs* de nombre **Collider** desde el panel **Project** hasta el panel **Hierarchy**, pero dentro de **Navigation/Colliders**.
3. Para cada **Collider** que hayamos introducido dentro de **Navigation/Colliders**, deberemos ir al **Inspector** y hacer lo siguiente:
 - a) Buscamos el componente **PolygonCollider2D** y darle al botón **Edit Collider**. Desde el panel **Scene** le damos la forma deseada, pero hay que intentar que sean polígonos lo más simples (a ser posible de cuatro lados) que se puedan.
 - b) Una vez acabado de moldear los **PolygonCollider2D**, volvemos al **Inspector**. Dentro del *script* **NavigationCollider** que hay en cada **Collider**, deberemos ponerles un número del 0 a $n - 1$ (donde n es el número total de **Collider** que hayamos metido en el nuevo escenario). No hay que saltarse ningún número de esa sucesión o podrían ocurrir fallos durante el juego.

- c) Por último, otra vez desde el Inspector, para cada **Collider** cambiamos el Tag por NavigationCollider, y Layer por Navigation.
4. Una vez agregados todos los **Collider** necesarios para el nuevo escenario, deberemos crear los puntos de unión entre ellos (si hubiera más de un **Collider**). Para ello, volvemos al panel Project a la ruta Assets/Resources/Prefabs/Navigation/, y arrastramos un *prefab* **Point** por cada conexión entre **Collider** que queramos hacer hasta **Navigation/Points** dentro del panel Hierarchy. Una vez colocados, por cada **Point** que hayamos metido, deberemos hacer lo siguiente:
 - a) Los seleccionamos dentro de Hierarchy y desde el panel Scene con sus controles, lo colocamos en la posición concreta que queramos que esté el **Point**. Tiene que estar exactamente en el punto de conexión entre dos **Collider** por el que queramos que pase **Player** al caminar por el escenario.
 - b) Con el **Point** aún seleccionado en Hierarchy, vamos al Inspector. Desde ahí buscamos el *script* NavigationPoint y colocamos dos IDs de los **Collider** que queramos conectar. Dichos IDs están situados en sus *scripts* en NavigationCollider (los podemos ver seleccionándolos y viéndolos desde el Inspector).
 - c) Por último, otra vez desde el Inspector, para cada **Point** cambiamos el Tag por NavigationPoint, y Layer por Navigation.

5. Guardamos los cambios en la nueva escena.

Y estos son los pasos básicos para crear las zonas transitables para **Player** dentro de la nueva escena. De manera opcional, desde el panel Hierarchy al objeto **Navigation/CutsceneNavPoints** podemos añadirle los *prefabs* **CutscenePoint** (desde el panel Project en la ruta Assets/Resources/-Prefabs/Navigation/) de manera similar que los **Point**. La diferencia entre **CutscenePoint** y **Point**, es que el primero puede ser colocado en cualquier sitio del nuevo escenario, dado que su utilidad es la de marcar sitios en el escenario donde los NPCs o **Player** puedan ir sin tener que escribir las coordenadas a mano.

Añadir nuevos textos y diálogos

Si queremos que tanto **Player** como los NPCs puedan hablar nuevos diálogos aparte de los que ya hay contenidos en los `LocatedTexts.xml`, deberemos escribir nuevos diálogos y textos dentro de estos ficheros. El proceso para hacerlo es tal que así:

1. Abrimos un fichero `LocatedTexts.xml` situado dentro de una de las carpetas contenidas en Assets/Data/Localization/ con un editor de textos (en mi caso elegí *Geany*).
2. Dentro de la etiqueta `locatedtexts`, escribimos una nueva etiqueta de grupo con el ID correspondiente a la nueva escena creada. Por ejemplo:

```
1 | <group id = "SCENE_NAME-NEW-SCENE"></group>
```

3. Una vez creada la etiqueta de grupo, dentro de esta tendremos que crear las etiquetas de los elementos con sus propias IDs que vayan a estar dentro del nuevo escenario:

```

1 <group id = "SCENE_NAME-NEW-SCENE">
2     <element id = "OBJECT_NAME-NEW-OBJECT-1"></element>
3     <element id = "OBJECT_NAME-NEW-OBJECT-2"></element>
4 </group>
```

4. Ya añadido todos los objetos interactivos que vayan a estar en la nueva escena, hay que añadir las diferentes cadenas de textos que se van a poder decir. Para ello, creamos las etiquetas de *string* cada una con la ID de con qué tipo de acción se va a decir. Y siempre tendremos que añadir por defecto dos *string*: la de NAME con el nombre del objeto interactivo y la de DESCRIPTION con la descripción de este objeto a decir por **Player** cuando se pulse el click derecho sobre él. Finalmente debería quedar algo similar a esto:

```

1 <group id = "SCENE_NAME-NEW-SCENE">
2     <element id = "OBJECT_NAME-NEW-OBJECT-1">
3         <string id = "NAME">Nombre de objeto interactivo 1</
4             string>
5         <string id = "DESCRIPTION">Descripción del objeto
6             interactivo 1</string>
7         <string id = "INTERACTION">
8             Texto por defecto que dice Player al hacer click
9                 izquierdo sobre este objeto.
10            Recuerda que cada salto de línea que hagas aquí,
11                comenzará un nuevo cuadro de diálogo,
12                así que úsalo a tu favor para cortar los diálogos por
13                    los sitios donde tú quieras.
14            </string>
15        </element>
16        <element id = "OBJECT_NAME-NEW-OBJECT-2">
17            <string id = "NAME">Nombre de objeto interactivo 2</
18                string>
19            <string id = "DESCRIPTION">Descripción del objeto
20                interactivo 2</string>
21            <string id = "INTERACTION_SPECIAL">
22                Este texto no se leerá como texto por defecto cuando
23                    Player haga click izquierdo
24                sobre este objeto, así que tendremos que ponerlo
25                    expresamente en un script que herede de
26                    InteractiveElement y añadírselo al objeto interactivo
27                        en cuestión.
28            </string>
29            <string id = "INTERACTION_SPECIAL_2">
30                Segundo texto de interacción que no viene por defecto.
31            </string>
32        </element>
33    </group>
```

5. Repetimos el proceso para los demás archivos `LocatedTexts.xml` que haya en los directorios contenidos en `Assets/Data/Localization/`, traduciéndolos a sus respectivos idiomas.

En el caso de los textos dentro de los etiquetados de *string* como INTERACTION_SPECIAL o con otros nombres que no sean NAME, DESCRIPTION o INTERACTION, para que puedan ser utilizados dentro de los *scripts* de los objetos interactivos, tendremos que añadir el siguiente fragmento de código:

```

1 Player.Instance.Speak(groupID, nameID, "INTERACTION_SPECIAL");
2 do{
3     yield return null;
4 }while(Player.Instante.isSpeaking);

```

En el otro supuesto de que tuviéramos que añadir textos para un NPC nuevo, los textos tendrían que ir dentro del grupo con ID NPC. Como tal que así:

```

1 <group id = "NPC">
2     <element id = "NPC-NAME">
3         <string id = "NAME">Nombre del NPC</string>
4         <string id = "DESCRIPTION">Descripción del NPC</string>
5         <string id = "PLAYER_CONVERSATION_1">
6             Frase que le dice Player al NPC
7         </string>
8         <string id = "NPC_CONVERSATION_1">
9             Frase que le responde el NPC al Player
10        </string>
11        <string id = "PLAYER_CONVERSATION_2">
12            Siguiente diálogo que le dije Player a NPC
13        </string>
14        <!>Más diálogos...<!>
15    </element>
16    <!>Más NPCs...<!>
17 </group>

```

También está el caso de que queramos añadir un nuevo objeto al inventario, entonces los textos tendrían que ir dentro del grupo con ID INVENTORY:

```

1 <group id = "INVENTORY">
2     <element id = "ITEM-NAME">
3         <string id = "NAME">Nombre del objeto del inventario</
4             string>
5         <string id = "DESCRIPTION">Descripción del objeto del
6             inventario</string>
7     </element>
8     <!>Más objetos del inventario...<!>
9 </group>

```

Y no nos olvidemos que también podemos añadir nuevas elecciones para el sistema de decisión en objetos interactivos y/o NPCs:

```

1 <group id = "CHOICES">
2     <element id = "NPC_OR_OBJECT_CONVERSATION_CHOICE_1">
3         <string id = "CHOICE_0">Opción 1 a elegir</string>
4         <string id = "CHOICE_1">Opción 2 a elegir</string>
5         <string id = "CHOICE_2">Opción 3 a elegir</string>

```

```

6           <!>Más opciones...</!>
7       </element>
8   <!>Otros cuadros de elecciones...</!>
9 </group>

```

Por último no nos olvidemos que también se pueden agregar nuevas etiquetas de `element` en grupos ya creados con ID de otros niveles. Para añadir textos a los distintos menús que contengan notas y artículos, lo veremos en sus respectivas secciones.

Hacer que los cambios en los objetos interactivos se guarden al cambiar de nivel

Si bien en la subsección de **Creación de objetos interactivos** veíamos cómo crear objetos interactivos básicos y una plantilla de cómo crear nuestro propio *script* de objeto interactivo que expandiera o modificase las funcionalidades de estos. Ciertamente sólo desde un *script* personalizado de objeto interactivo podemos hacer que usen los datos del *script* `GameState` para su estado actual.

Así que primero, deberemos abrir el *script* `GameState` en *Monodevelop* (o el IDE que estemos usando para programar los *scripts* de *Unity*). Donde se inicializan los parámetros tendremos que incluir una `static class` para el nuevo nivel:

```

1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4
5  public static class GameState{
6      //Muchas cosas...
7
8      public static class NewSceneData{
9          // Ultima posición de Player dentro del escenario
10         public static Vector2 playerPosition;
11         //Ultima posición clickada como objetivo dentro del
12             escenario
13         public static Vector2 lastTargetedPositionByMouse;
14         // Posición por defecto de Player al usar el teletransporte
15             mediante el menú de Mapa
16         public static Vector2 defaultPosition;
17         //Y aquí ponemos los valores de los objetos interactivos
18         //del escenario que deban guardarse al cambiar de escenario
19         public static bool isObjectAPicked;
20         public static int timesObjectBHasBeenInteracted;
21         public static bool isObjectCHasBeenManipulated;
22             //etc...
23     }
24
25     //Muchas cosas...
26 }

```

Y luego inicializar los datos contenidos en esta clase en el método `InitializeGameState()` del script `GameState`:

```
1 public static void InitializeGameState(){
2     //Muchas cosas inicializadas aparte...
3
4     //Posición inicial de Player al entrar por primera vez en el nuevo
5     //escenario
6     NewSceneData.playerPosition = new Vector2(236.0f, 68.5f);
7     //Posición por defecto de Player al usar el teletransporte mediante
8     //el menú de Mapa
9     NewSceneData.defaultPosition = NewSceneData.playerPosition;
10    //Los objetos interactivos que se pueden coger ponen este valor a
11    //false indicando que no han sido codigos
12    NewSceneData.isObjectAPicked = false;
13    //El número de veces que Player ha interactuado con el objeto
14    //interactivo
15    NewSceneData.timesObjectBHasBeenInteracted = 0;
16    //Estado inicial con el que se indica que aún no ha sido manipulado
17    //por Player
18    NewSceneData.isObjectCHasBeenManipulated = false;
19
20 }
```

Una vez añadido los datos para que se inicialicen, tenemos que actualizar algunos parámetros y métodos que controlan el guardado de datos al cambiar de escenarios dentro del juego.

En el parámetro `lastPlayerPosition` deberemos añadir la última posición de **Player** del nuevo escenario, así podrá cargar la posición correcta en la que inicializar la posición de **Player** cuando vuelva:

```
1 public static Vector2 lastPlayerPosition{
2     get{
3         switch(lastPlayedLevel){
4             //....
5
6             //Nombre de la nueva escena en el fichero .scene de Unity
7             case "NewScene" :
8
9                 //Posición guardada de Player en el nuevo escenario
10                return NewSceneData.playerPosition;
11
12                //....
13
14                default:
15                    return Vector2.zero;
16                }
17            }
18
19        private set {}
20    }
```

En el parámetro `lastTargetedPositionByMouse` añadiremos la posición del último objetivo clickado por **Player** dentro del nuevo escenario:

```

1  public static Vector2 lastTargetedPositionByMouse{
2      get{
3          switch(lastPlayedLevel){
4              //....
5
6              //Nombre de la nueva escena en el fichero .scene de Unity
7              case "NewScene" :
8
9                  //Ultima posición clickada en el nuevo escenario
10                 return NewSceneData.lastTargetedPositionByMouse;
11
12                 //....
13
14             default:
15                 return Vector2.zero;
16             }
17         }
18
19     private set {}
20 }
```

Y en el método SaveGameState guardaremos los datos en la class del nuevo nivel con respecto a **Player** para cuando vuelva a este escenario otra vez:

```

1  public static void SaveGameState(bool isWarpedThroughMapsMenu = false) {
2      //Si se va a teletransportar Player a otro escenario mediante el
3      // el menú de Mapa, se cambia la última posición de Player del
4      // nivel jugable
5      // actual a la que se puso por defecto en vez de guardar la que
6      // tiene
7      if(isWarpedThroughMapsMenu){
8          switch(lastPlayedLevel){//Ultimo nivel jugable
9
10             //....
11
12             case "NewScene" :
13                 NewSceneData.playerPosition = NewSceneData.
14                     defaultPosition;
15                 NewSceneData.lastTargetedPositionByMouse =
16                     NewSceneData.defaultPosition;
17                 break;
18
19             //....
20         }
21     else{
22         switch(Application.loadedLevelName){ //Escenario que está
23             // cargado actualmente (pudiendo ser un menú)
24             //....
25     }
```

```

25
26     case "NewScene" :
27         NewSceneData.playerPosition = Player.Instance.
28             transform.position;
29         NewSceneData.lastTargetedPositionByMouse = Player.
30             Instance.originalTargetedPosition;
31         break;
32
33     default:
34         break;
35     }
36 }
37 }
```

Por último, solo nos quedaría que en el *script* personalizado para el objeto interactivo interactue con alguno de los parámetros de la clase NewSceneData. Por poner un ejemplo:

```

1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4
5  public static class GameObjectA : PickableElement {
6      //.....
7
8      protected override void InitializeInformation() {
9          if(GameState.NewSceneData.isObjectAPicked) {
10              SetInactive();
11          }
12          else{
13              //.....
14          }
15      }
16
17      //.....
18
19      public virtual void OnPlayerTouchingAction(){
20          if(Inventory.Instance.IsInventoryFull()){
21              Player.Instance.Speak("GUI", "DEFAULT", "FULL_INVENTORY");
22              do{
23                  yield return null;
24              }while(Player.Instance.isSpeaking);
25          }
26          else{
27              Player.Instance.GrabItem(nameListGivableElements);
28              if(isDestroyedAfterBeingPicked){
29                  GameState.NewSceneData.isObjectAPicked =
30                      true;
31                  SetInactive();
32              }
33          }
34      }
35 }
```

```
34 }  
35 //.....  
36 }
```

Hacer que los cambios se guarden en los ficheros de guardado

Para que los cambios que se guarden en los parámetros del *script* GameState se puedan guardar y cargar desde un fichero, para que así se pueda salir del juego y jugar en otra ocasión en el punto donde lo dejamos, deberemos ampliar un par de métodos en el *script* GameFileManager.

En primer lugar, dentro del método BuildXMLData, que es el encargado de guardar los datos de GameState en un fichero .xml:

```
1 private string BuildXMLData(){  
2     //Creamos un nuevo documento XML  
3     XmlDocument _ xmlDoc = new XmlDocument();  
4     XmlElement _rootElement = _ xmlDoc.CreateElement("GameFile");  
5  
6     //Creamos el nodo raíz del documento XML  
7     _ xmlDoc.AppendChild(_rootElement);  
8  
9     //Le añadimos la información de la última sesión: último nivel  
10    jugable y fecha de cuando se guardó  
11    //la partida  
12    XmlElement _lastSessionData = _ xmlDoc.CreateElement("LastSessionData");  
13    _lastSessionData.SetAttribute("lastPlayableLevel", GameState.  
14        lastPlayedLevel);  
15  
15    string _date = DateTime.Now.ToString("dd/MM/yyyy HH:mm");  
16    _lastSessionData.SetAttribute("lastTimePlayed", _date);  
17    _rootElement.AppendChild(_lastSessionData);  
18  
18    //Más cosas.....  
19  
20    //Creamos un nodo XML para los datos de NewSceneData  
21    XmlElement _newSceneData = _ xmlDoc.CreateElement("NewSceneData");  
22    //Añadimos como atributos del nodo, los valores de los parámetros  
23    // de la clase NewSceneData  
24    _newSceneData.SetAttribute("isObjectAPicked", GameState.  
25        NewSceneData.isObjectAPicked.ToString());  
26    _newSceneData.SetAttribute("timesObjectBHasBeenInteracted",  
27        GameState.NewSceneData.timesObjectBHasBeenInteracted.ToString());  
28    _newSceneData.SetAttribute("isObjectCBeenManipulated", GameState.  
29        NewSceneData.isObjectCBeenManipulated.ToString());  
30  
30    //Una vez completado los datos del nodo XML de NewSceneData, lo  
31    // incluimos dentro del  
32    //nodo raíz del documento XML  
33    _rootElement.AppendChild(_newSceneData);
```

```

30
31 //Más cosas.....
32
33 //Guardamos los nombres de los objetos que hay en el inventario
34 XmlElement _inventory = _xmlDoc.CreateElement("Inventory");
35 List<string> itemsInInventory = Inventory.Instance.GetItemsName();
36 foreach(string itemName in itemsInInventory){
37     XmlElement _itemInventory = _xmlDoc.CreateElement("item");
38     _itemInventory.SetAttribute("Name", itemName);
39     _inventory.AppendChild(_itemInventory);
40 }
41 _rootElement.AppendChild(_inventory);
42
43 return FileManager.XmlDocToString(_xmlDoc);
44 }
```

Luego, una vez ya se puedan guardar los datos, tendremos que modificar el método ParseXMLFile del script GameFileManager para que puedan ser cargados los datos guardados en GameState al iniciar nuevamente la partida:

```

1 private void ParseXMLFile(string directory, string filename, string
2   filetype, string mode){
3   XmlDocument _xmlDoc = new XmlDocument();
4
5   //Dependiendo de si el fichero de guardado estaba encriptado o no,
6   //lo abrimos de una manera u otra
7   if(mode == "plaintext"){
8       _xmlDoc.Load(texttt + "/" + directory + "/" + filename + "."
9           + filetype);
10  }
11  else if(mode == "encrypt"){
12      string _filedata = FileManager.ReadFile(directory, filename
13          , filetype);
14      _filedata = FileManager.DecryptData(_filedata);
15      FileManager.CreateFile(directory + "/", "/tmp_" + filename,
16          filetype, _filedata);
17      _xmlDoc.Load(texttt + "/" + directory + "/tmp_" + filename
18          + "." + filetype);
19  }
20
21  //Cargamos los datos de la sesión guardada en la partida
22
23  //Creamos un nodo con los datos de la etiqueta LastSessionData y
24  //cargamos en GameState.lastPlayedLevel el valor contenido en su
25  //atributo
26  XmlNode _lastSessionData = _xmlDoc.SelectSingleNode("//"
27      "LastSessionData");
28  GameState.lastPlayedLevel = _lastSessionData.Attributes[""
29      "lastPlayableLevel"].Value;
30
31  //Más cosas.....
32
33  //Creamos un nodo con los datos de la etiqueta de NewSceneData y
34  //cargamos en GameState.newSceneData los valores contenidos en los
```

```

    atributos del nodo
27 //mediante el parseo de string al tipo nativo del parámetro
28 XmlNode _newSceneData = _xmlDoc.SelectSingleNode("//NewSceneData");
29 GameState.NewSceneData.isObjectAPicked = bool.Parse(_newSceneData.
    Attributes["isObjectAPicked"].Value);
30 GameState.NewSceneData.timesObjectBHasBeenInteracted = int.Parse(
    _newSceneData.Attributes["timesObjectBHasBeenInteracted"].Value)
    ;
31 GameState.NewSceneData.isObjectCBeenManipulated = bool.Parse(
    _newSceneData.Attributes["isObjectCBeenManipulated"].Value);

32 //Más cosas.....
33
34
35 //Cargamos los items del inventario que había en la partida
36 XmlNodeList _items = _xmlDoc.GetElementsByTagName("item");
37 //Borramos previamente los objetos que había actualmente en el
    inventario
38 //por si cargamos la partida mientras estamos jugando, así no se
    solapan
39 //los objetos de la partida ya comenzada
40 Inventory.Instance.DeleteAllItems();
41 foreach(XmlNode item in _items){
    //Añadimos un objeto al inventario por cada nodo etiquetado
        como item
    Inventory.Instance.AddItem(item.Attributes["Name"].Value);
42 }
43
44 }
45
46 }

```

Con esto ya los cambios que hagamos a los objetos interactivos o NPCs del nuevo escenario podremos guardarlos, salir del juego y volver a cargarlos posteriormente sin ningún tipo de problema.

Añadir nuevas entradas al Menú de Notas

Para poder añadir más notas al Menú de Notas dentro del Menú de Opciones, primero deberemos crear los textos de las nuevas notas en los archivos `LocatedTexts.xml` incluidos dentro de los directorios que están en `Assets/Data/Localization/`. Abrimos los `LocatedTexts.xml` con un editor de textos y buscamos la etiqueta `group` con el ID `OPTIONS_MENU` y dentro de este la etiqueta `element` con el ID `JOURNAL`. Una vez localizados, para poder añadir una nueva nota deberemos hacer lo siguiente:

1. Añadimos una etiqueta `string` con el ID `ENTRY_n` (donde n es el número de la nota). Rellenamos el contenido de la etiqueta con el título que tendrá esta nota nueva en el juego.
2. Añadimos una etiqueta `string` con el ID `ENTRY_n_DESC` (donde n es el número de la nota). Rellenamos el contenido de la etiqueta con la descripción que tendrá esta nota nueva en el juego.

Una vez hecho esto, debería quedarnos algo tal que así:

```

1  <!>.....</!>
2
3 <element id = "JOURNAL">
4   <string id = "NAME">Notas</string>
5   <!>Otras títulos de notas.....<!>
6   <string id = "ENTRY_4">Título para la nota número 4</string>
7   <string id = "ENTRY_4_DESC">
8     Descripción para la nota número 4
9
10    Aquí puedes incluir todo el texto que tu quieras, pero intenta que
11      el del título de la nota
12      sea escueto o no se podrá leer bien dentro del juego.
13    </string>
14 </element>
15 <!>.....</!>

```

Después de agregar la nueva nota a los ficheros `LocatedTexts.xml`, deberemos modificar el *script* `JournalEntriesManager` de forma que al inicializarse cada vez que el *Jugador* entre dentro del menú de opciones, añada la nueva nota. Los pasos a seguir son:

1. Desde el panel de Project en *Unity* nos vamos a `Assets/Scripts/GUI/OptionsMenu-Buttons/JournalButtons/` y abrimos con *Monodevelop* (o con otro IDE que prefiramos) el *script* `JournalEntriesManager`.
2. Una vez abierto, nos dirigimos al método `CheckGameStateForAddingJournalEntries` y le agregamos lo siguiente:

```

1  private void CheckGameStateForAddingJournalEntries() {
2      //Dependiendo de las escenas cinemáticas que se han
3      //reproducido y otros eventos dentro del juego,
4      // las entradas son agregadas al diario
5
6      //Se añaden otras entradas al menú de Notas....
7      AddJournalEntry(1);
8      AddJournalEntry(2);
9
10     //Si ha ocurrido alguna escena cinemática en concreto, añ
11     //adimos otras entradas
12     if(GameState.CutSceneData.isPlayingIntroProfessorOfficePart) {
13         AddJournalEntry(3);
14         //Más cosas...
15     }
16
17     //Añadimos la nota nueva 4 aquí o donde nos parezca
18     AddJournalEntry(4);
19
20     //Más cosas...
21 }
```

Con esto la nueva nota creada se cargará satisfactoriamente en el Menú de Notas la próxima vez que abramos el juego y usemos el Menú de Opciones.

Añadir nuevos artículos al Buscador de Gadipedia

El proceso para añadir nuevos artículos al Buscador de Gadipedia tiene un esfuerzo añadido, no solo basta con añadir nuevo texto con artículos de información histórica a los archivos `LocatedTexts.xml` sino que tendremos que instalar `SQLite` en nuestro ordenador para poder actualizar el archivo `gadipedia.db` que contiene la base de datos del buscador.

Añadir los artículos a `LocatedTexts.xml`

En primer lugar deberemos añadir los artículos nuevos que vayamos a incluir en el buscador de Gadipedia a los ficheros `LocatedTexts.xml`. Todos estos ficheros están contenidos en directorios siguiendo la ruta `Assets/Data/Localization/`.

Una vez localizados los archivos y abiertos en el editor de textos de nuestra preferencia, buscamos la etiqueta `group` con el ID `GADIPEDIA_ARTICLES`. Para cada artículo que queramos añadir, escribimos una nueva etiqueta `element` de la siguiente manera:

```
1 <element id = "ARTICLE_NAME-OF-THE-ARTICLE">
2   <string id = "NAME">Nombre del artículo</string>
3   <string id = "BRIEF">Resumen breve del contenido del artículo</
4     string>
5   <string id = "CONTENT">
6     Contenido del artículo nuevo para el buscador de Gadipedia.
7
8     Hay que recordar que este buscador es de información histórica, así
      que cualquier información que vayamos a introducir aquí, tiene
      que estar mínimamente contrastada.
9   </string>
</element>
```

Una vez añadido los artículos a los `LocatedTexts.xml`, si consideramos que con las palabras clave que hay ya contenidas en la etiqueta `element` con ID `KEYWORDS` (dentro de la etiqueta `group` con ID `GADIPEDIA_ARTICLES`) no tienen relación con los nuevos artículos, deberemos añadir nuevas palabras clave. Para poder añadir nuevas palabras clave solo tendremos que escribirlas dentro de la etiqueta `element` con ID `KEYWORDS`:

```
1 <group id = "GADIPEDIA_ARTICLES">
2   <element id = "KEYWORDS">
3     <!>Otras palabras clave...<!/!
4     <string id = "PALABRA-CLAVE-DE-NUESTRO-ARTICULO-1">clave1</
          string>
5     <string id = "PALABRA-CLAVE-DE-NUESTRO-ARTICULO-2">clave2</
          string>
```

```
6 |     </element>
7 | </group>
```

Como se puede ver en el ejemplo, deberemos añadir etiquetas `string` en el que sus ID serán la palabra clave que se usará para cotejar en la base de datos, y el contenido de la etiqueta deberá ser la misma palabra clave pero en el idioma propio de cada archivo `LocatedTexts.xml` que haya.

El texto contenido de la etiqueta `string` con el ID de la palabra clave, tiene que estar en minúsculas y sin ningún acento. Esto es porque el *script* encargado de cotejar los textos introducidos en el buscador son formateados de esta manera, de esta forma evitamos que no busque un resultado por un acento o mayúscula mal colocado en una palabra.

Añadir los artículos y las palabras clave a la base de datos

Una vez añadidos los textos de los artículos y las palabras clave a los ficheros `LocatedTexts.xml`, deberemos crear la relación que hay entre estos datos dentro de la base de datos. Para ello tendremos que descargar *SQLite* en el ordenador desde su página web oficial <https://www.sqlite.org/download.html> y descomprimirlo (o crear el binario ejecutable después de compilar el código fuente) en directorio que veamos oportuno.

Una vez descomprimido, solo tendremos que hacer doble click encima de `sqlite3` o `sqlite3.exe` en caso de usar Microsoft Windows, y se abrirá la terminal. Ahora deberemos abrir la base de datos `gadipedia.db` contenida en el proyecto *Unity* e insertarle las nuevas palabras clave con el ID del artículo con el que se relacionan:

```
sqlite3> .open C:/path-to-project/Assets/Data/DB/gadipedia.db
sqlite3> INSERT INTO "gadipedia.keywords_search"
VALUES('ARTICLE_NAME-OF-THE-ARTICLE','PALABRA-CLAVE-DE-NUESTRO-ARTICULO-1');
sqlite3> INSERT INTO "gadipedia.keywords_search"
VALUES('ARTICLE_NAME-OF-THE-ARTICLE','PALABRA-CLAVE-DE-NUESTRO-ARTICULO-2');
sqlite3> .quit
```

Para comprobar que se han insertado correctamente en la base de datos y las IDs tanto del artículo como de las palabras clave son las correctas, dentro de *Unity* le damos al `Play` del proyecto y probamos a introducir las palabras en el buscador de la Gadipedia, si al darle al botón `Buscar` sale el artículo que habíamos creado, es que funciona correctamente.

Añadir nuevas localizaciones al Menú de Mapa

Para poder añadir nuevos sitios a los que poder trasladarse mediante el Menú de Mapa dentro del Menú de Opciones del juego, primero deberemos modificar brevemente los ficheros `LocatedTexts.xml` contenidos en el directorio `Assets/Data/Localization/` para incluir el nombre del nuevo escenario. Buscamos la etiqueta `group` con el ID `OPTIONS_MENU`, luego la etiqueta `element` con el ID `MAPS` y añadimos lo siguiente:

```

1 <element id = "MAPS">
2   <string id = "NAME">Mapa</string>
3   <!>Otros nombres de escenarios...</!>
4   <string id = "NEW_SCENE_LOCATION">Nombre del nuevo escenario</
5     string>
6   <!>Otros nombres de escenarios...</!>
</element>

```

Una vez terminado de modificar los ficheros `LocatedTexts.xml` deberemos seguir los siguientes pasos:

1. Tendremos que abrir la escena que contiene el Menú de Opciones, para ello desde el panel `Project` buscamos y abrimos la escena `OptionsMenu`, que está contenida en `Assets/Scenes/Menus/`.
2. Una vez abierta la escena, nos vamos al panel `Hierarchy`, buscamos dentro de la jerarquía de objetos `OptionsMenuCanvas/MapMenuPanel/Background/MapPanel/ScrollView/MapContent` y nos quedamos con **MapContent**.
3. Volvemos al panel de `Project`, esta vez para coger el *prefab* **LocalizationMark** que está en `Assets/Resources/Prefabs/UIElements/` y arrastrarlo dentro del objeto **MapContent** que habíamos encontrado antes en el panel de `Hierarchy`.
4. Seleccionamos la nueva **LocalizationMark** dentro de `Hierarchy` y modificamos su posición dentro del mapa mediante el panel `Scene` y sus controles hasta que quede en el sitio que prefiramos.
5. Ya colocado en su sitio, seleccionamos desde `Hierarchy` el objeto hijo **Button** del **LocalizationMark** para que aparezca en el `Inspector`.
6. Dentro del `Inspector`, buscamos donde esté el *script* `LocalizationMarkButton` y rellenamos los parámetros que se ven:

Localized Text Group Este campo indica el grupo dentro de los ficheros `LocatedTexts.xml` al que pertenece este objeto. Por defecto su valor es `OPTIONS_MENU` y no tenemos que modificarlo.

Localized Text Element Este campo indica el elemento del grupo dentro de los ficheros `LocatedTexts.xml` al que pertenece este objeto. Por defecto su valor es `MAPS` y no tenemos que modificarlo.

Localized Text String Este campo indica el ID del campo `string` dentro de los ficheros `LocatedTexts.xml` que contiene el nombre del nivel al que transportará este **LocalizedMark** a **Player**. Aquí deberemos cambiar el valor por defecto por el ID del nuevo nivel.

Warp Scene Name Aquí deberemos cambiar el valor por el del nombre del archivo `.scene` del nuevo escenario. Así cuando clickemos sobre este **LocalizedMark**, transportará a **Player** correctamente a este nuevo escenario (si en la ventana modal que sale le da al botón **Sí**).

Con eso ya tendremos añadido la marca en el Menú de Mapa del nuevo escenario y podremos transportar a **Player** mediante esta opción.

Capítulo 10

Comunidad y difusión

Al inicio de embarcarme en este proyecto, era imprescindible llevar a cabo acciones para difundir el proyecto para poder buscar colaboradores para el desarrollo del proyecto. A continuación, se listan los medios empleados para contribuir a la difusión del proyecto.

Blog de desarrollo

Creé un blog en la plataforma *Wordpress* específico para el proyecto con el propósito de informar de los avances del proyecto, escribir sobre conceptos básicos de diseño de videojuegos y de cuestiones técnicas básicas que hay que aprender para desarrollar software general y/o para *Unity*. En total se han redactado unos 15 artículos y se han recibido más de 2.000 visitas. Puede accederse desde la siguiente dirección:

<https://1812laaventura.wordpress.com/>

Repositorio

El proyecto se ha alojado en el repositorio de *GitHub* la cual no ha sido utilizada únicamente por su repositorio Git. Se ha hecho uso de la sección *Issues* para gestionar la lista de sugerencias y fallos que iban apareciendo en el proyecto y así ir agregándolos como trabajo pendiente en el *Trello*. El sitio del proyecto en el repositorio de *GitHub* puede ser accedido desde la siguiente dirección web:

<https://github.com/Firenz/1812>

Web en el repositorio

El repositorio de *GitHub* proporciona a los proyectos un pequeño espacio web [36]. Si bien no otorga libertad absoluta para publicar contenido (solo se permiten webs estáticas en HTML) cuenta con un posicionamiento extremadamente favorable en buscadores. Se ha aprovechado dicho espacio con una

web a modo de índice indicando brevemente en qué consiste el proyecto y enlazando a los medios oficiales. Puede accederse desde la siguiente dirección:

<http://firenz.github.io/1812/>

Hashtag en Twitter

Twitter es una red social de microblogging en la que los usuarios publican mensajes cortos. Es ampliamente utilizada para seguir noticias y estar informado de la actualidad en diversos sectores muy específicos. Existe una comunidad de desarrolladores hispanohablantes muy activa dentro de Twitter por lo que se decidió que el proyecto tuviera presencia en dicha red social. La comunicación en Twitter ha sido muy útil para acercarme a comunidades, lectores del blog, recibir sus sugerencias y conseguir colaboradores para el proyecto. Usé mi cuenta personal para ir mostrando los avances en Twitter y etiquetaba los mensajes referentes al proyecto con el *hashtag #1812laaventura*, así con solo hacer click en el *hashtag* dentro del mensaje o escribiéndolo en el buscador de Twitter, salen todos los mensajes con noticias del proyecto. A continuación la lista de mensajes en Twitter con dicho *hashtag* usado en el seguimiento del proyecto:

<https://twitter.com/hashtag/1812laaventura>

Lista de reproducción en Youtube

Durante todo el desarrollo se han ido subiendo los progresos de la demo técnica de **1812: La aventura** al servicio de vídeo vía streaming por excelencia. Esto ha permitido que el interés por el proyecto creciera parcialmente. En total se han publicado cuatro vídeos los cuales se han reproducido en más de 800 ocasiones. Puede accederse a la lista de vídeos sobre **1812: La aventura** en la siguiente dirección:

<https://www.youtube.com/playlist?list=PLpWTFopWU5f3U7w7qQykKsniv16T0vBYl>

Perfiles en comunidades de desarrollo

Aparte de las redes sociales, también me involucré en un par de comunidades de desarrolladores de videojuegos en los que algunos de sus miembros me ofrecieron consejos y sugerencias, además de que también se logró promocionar un poco más el proyecto. Los perfiles más destacados que hice en las comunidades son:

<http://gamejolt.com/games/1812-la-aventura/49809>

<http://www.zehngames.com/proyectos/unity-1812-la-aventura/>

Difusión

Algunos magazines y comunidades de videojuegos en español mostraron un pequeño interés al principio cuando **1812: La aventura** iba a ser un juego completo, principalmente por ser una aventura gráfica con trasfondo histórico. De forma más secundaria, otros también se interesaron por el **Motor de Aventuras Gráficas 2D de Unity** hecho desde cero sin necesidad de usar *Assets* comprados de la *Unity Store*. A continuación, enlazamos los medios que han publicado artículos sobre el proyecto:

<http://indiefence.blogspot.com.es/2014/08/que-se-cuece-agosto-de-2014.html?m=1>
http://nuevobag.blogspot.com.es/2014/08/noticias-cortas-del-10-de-abril-al-11_20.html?m=1
http://www.abandonsocios.org/wiki/1812:_La_Aventura
<http://comosehaceunvideojuego.com/episodios/celebrando-el-2-cumpleanos-de-como-se-hace-un-videojuego-1a-parte/>

Bibliografía

- [1] Clark C. Abt. *Serious Gamesn*. University Press of America, 1970.
- [2] AI Lowe. AI Lowe's Game Designs. <http://www.allowe.com/games/games-designs.html>.
- [3] Antonio G. Alba. Wikijuegos. <http://wikis.uca.es/wikijuegos/w/index.php?title=Portada>.
- [4] Asi Burak, Emily Treat, Susana Pollack, Victoria Abrash, Meghan Ventura y Hsing Wei. Games for Change. <http://www.gamesforchange.org/>.
- [5] Asociación Española de Empresas Productoras y Desarrolladoras de Videojuegos y Software de Entretenimiento. Libro Blanco del Desarrollo Español de Videojuegos 2015. <http://www.dev.org.es/es/publicaciones/libro-blanco-dev>.
- [6] AT & T Labs Research and Contributors. Pagina oficial de Graphviz. <http://www.graphviz.org>.
- [7] Atlassian. Source Tree. <https://www.sourcetreeapp.com/>.
- [8] Ayuntamiento de Cádiz. Bibliografía del Bicentenario. <http://www.bicentenario2012.org/biblioteca.asp>.
- [9] Benito van der Zander. TeXstudio. <http://texstudio.sourceforge.net/>.
- [10] Biblioteca Nacional Hispánica. Búsqueda de Mapas de Cádiz. <http://bdh.bne.es/bnesearch/Search.do?lengua=&text=&field2Op=AND&field1val=Cadiz&showYearItems=&numfields=3&fechaHdesde=&field3Op=AND&completeText=off&fechaHasta=&field3val=&field3=todos&fechaHsearchtype=0&field2=todos&field1Op=AND&fechaHen=&exact=on&advanced=true&textH=&field1=geo&field2val=&doctype=Material+cartogr%C3%Alfico+impreso&doctype=Material+cartogr%C3%Alfico+manuscrito&pageNumber=5&pageSize=10&language=es>.
- [11] Biblioteca Nacional Hispánica. Map of the Country round Cadiz. <http://bdh.bne.es/bnesearch/detalle/bdh0000020677>.
- [12] Biblioteca Nacional Hispánica. Plan de Cadiz et de ses environs. <http://bdh.bne.es/bnesearch/detalle/bdh0000033472>.
- [13] Biblioteca Nacional Hispánica. Plano de Cádiz en 1812. <http://bdh.bne.es/bnesearch/detalle/bdh0000033478>.
- [14] Biblioteca Nacional Hispánica. Plano de la bahía de Cádiz y sus contornos. <http://bdh.bne.es/bnesearch/detalle/bdh0000178866>.

- [15] Biblioteca Nacional Hispánica. Plano de la Plaza de Cadiz y Fuertes dependientes de ella hasta la cortadura de S. Fernando. <http://bdh.bne.es/bnesearch/detalle/bdh0000033476>.
- [16] Biblioteca Nacional Hispánica. Vista de Cádiz y sus contornos. <http://bdh.bne.es/bnesearch/detalle/bdh0000033399>.
- [17] Bryant Francis. Where in the world did blockbuster educational games go? http://www.gamasutra.com/view/news/243424/Where_in_the_world_did_blockbuster_educational_games_go.php.
- [18] Centro Internacional para Académicos Woodrow Wilson. Serious Game Initiative. <http://www.seriousgames.org/>.
- [19] Comunidad, y desarrolladores de CollabNet, Elego, VisualSVN, WANdisco. Apache Subversion. <http://subversion.apache.org/>.
- [20] D. Richard Hipp. SQLite. <https://www.sqlite.org/>.
- [21] D. Richard Hipp. SQLite download page. <http://www.sqlite.org/download.html>.
- [22] D. Richard Hipp. SQLite tutorials. <https://www.sqlite.org/cli.html>.
- [23] Dan Schuller. How to make an Adventure Game. <http://www.godpatterns.com/2010/08/how-to-make-adventure-game.html>.
- [24] Dave Calabrese. *Unity 2D Game Development*. Packt Publishing, 2014.
- [25] David Saltares Márquez. Primera Versión del GDD de Sion Tower. <http://saltares.com/blog/games/primera-version-del-gdd-de-sion-tower/>.
- [26] Dimitri Van Heesch. Pagina oficial de Doxygen. <http://www.doxygen.org>.
- [27] EditShare LLC. Lightworks. <https://www.lwks.com/>.
- [28] EditShare LLC. Lightworks tutorials. <https://www.lwks.com/tutorials>.
- [29] Enrico Tröger. Geany. <http://www.geany.org/>.
- [30] Evolus. Pencil. <http://pencil.evolus.vn/>.
- [31] Fictiorama Studios. Dead Synchronicity. <http://www.deadsynchronicity.com/es/home-2/>.
- [32] Fictiorama Studios. The Longest Night. <http://www.deadsynchronicity.com/es/the-longest-night-2/>.
- [33] Fundación Descubre. Aplican un juego virtual para mejorar el aprendizaje del alemán en universitarios. <https://fundaciondescubre.es/blog/2013/07/03/investigadores-de-la-universidad-de-cadiz-aplican-un-juego-virtual-para-mejorar-el-aprendizaje-del-aleman-en-universitarios/>.
- [34] GitHub. Git for Windows. <http://msysgit.github.io/>.
- [35] GitHub Inc. GitHub. <https://github.com/>.

- [36] GitHub, Inc. GitHub Pages. <https://pages.github.com/>.
- [37] GitHub Inc. GitHub Windows. <https://windows.github.com/>.
- [38] Henry Oswald. ShareLaTeX. <https://www.sharelatex.com/>.
- [39] Hilda Martín. *Cádiz y los lugares del Doce : de la Puerta de Tierra a Cortadura.* Cádiz : Consorcio para la Conmemoración del II Centenario de la Constitución de 1812, 2012.
- [40] Hilda Martín. *Cádiz y los lugares del Doce : de la Puerta de Tierra a La Caleta.* Cádiz : Consorcio para la Conmemoración del II Centenario de la Constitución de 1812, 2012.
- [41] HUMANBALANCE Ltd. GraphicsGale. <http://www.humanbalance.net/gale/us/>.
- [42] Wikipedia inglesa. EcoQuest. <http://en.wikipedia.org/wiki/EcoQuest>.
- [43] Wikipedia inglesa. Manhunter: New York. http://en.wikipedia.org/wiki/Manhunter:_New_York.
- [44] Wikipedia inglesa. Sam & Max: Save the World. http://en.wikipedia.org/wiki/Sam_&_Max_Save_the_World.
- [45] Wikipedia inglesa. Telltale Games. http://en.wikipedia.org/wiki/Telltale_Games.
- [46] Wikipedia inglesa. The Pink Panther: Passport to Peril. http://en.wikipedia.org/wiki/The_Pink_Panther:_Passport_to_Peril.
- [47] Wikipedia inglesa. Zack & Wiki: Quest for Barbaro's Treasure. http://en.wikipedia.org/wiki/Zack_&_Wiki:_Quest_for_Barbaryos_Treasure.
- [48] InQuest Technologies, Inc. Ganttter. <http://www.ganttter.com/>.
- [49] Jacob Pennock. Unity Automatic Documentation Generation (An Editor Plugin). <http://www.jacobpennock.com/Blog/unity-automatic-documentation-generation-an-editor-plugin/>.
- [50] Jim. Open Broadcaster Software. <https://obsproject.com/>.
- [51] Junta de Andalucía. Bibliografía del Bicentenario. http://www.juntadeandalucia.es/educacion/vscripts/w_bcc1812/w/lista_031.htm.
- [52] Junta de Andalucía. Publicaciones de La Pepa. <http://www.juntadeandalucia.es/educacion/webportal/web/la-pepa/recursos/publicaciones>.
- [53] Kickstarter. A Hole New World. <https://www.kickstarter.com/projects/971128630/a-hole-new-world/description>.
- [54] Sam Lantinga. libSDL. <http://www.libsdl.org/>.
- [55] Leslie Lamport. LATEXproject site. <http://latex-project.org/>.
- [56] Linus Torvalds. Git. <http://git-scm.com/>.
- [57] Linus Torvalds. Git Official Shell. <https://git-scm.com/downloads>.
- [58] Mad Gear Games. Mad Gear Games. <http://madgeargames.com/>.

- [59] Microsoft. Biblioteca de clases de .NET Framework. [https://msdn.microsoft.com/es-es/library/ggl45045\(v=vs.110\).aspx](https://msdn.microsoft.com/es-es/library/ggl45045(v=vs.110).aspx).
- [60] Microsoft. Tutoriales de C#. [https://msdn.microsoft.com/es-es/library/aa288436\(v=vs.71\).aspx](https://msdn.microsoft.com/es-es/library/aa288436(v=vs.71).aspx).
- [61] Mike Kasprzak. Ludum Dare. <http://ludumdare.com/>.
- [62] Ministerio de Educación, Deporte y Cultura. Biblioteca Nacional Hispánica. <http://bdh.bne.es/>.
- [63] Mono-Project. SQLite. <http://www.mono-project.com/docs/database-access/providers/sqlite/>.
- [64] MonoDevelop Project. Monodevelop. <http://monodevelop.com/>.
- [65] Nexus Game Studios. Randal's Monday. <http://www.randsmonday.com/es/>.
- [66] Nintendo. Wii Fit. <http://wiifit.com/es/>.
- [67] Nulab Inc. Cacoo. <https://cacoo.com/>.
- [68] Omnium Lab. U-Startup. <http://omniumlab.com/nuestrosproductos/detalle?id=2>.
- [69] Ramón Solís. *El Cádiz de las Cortes : la vida cotidiana en la ciudad en los años 1810 a 1813.* Sílex, 2012.
- [70] RedIRIS. Forja de RedIRIS. <http://forja.rediris.es>.
- [71] Spencer Kimball, Peter Mattisf. GIMP. <http://www.gimp.org/>.
- [72] Stack Exchange, Inc. StackOverflow. <http://stackoverflow.com/>.
- [73] Stephen Lavelle. Bfxr. <http://www.bfxr.net/>.
- [74] Tanda. Animaciones para la web con GIMP. <http://www.gimp.org.es/tutoriales/animaciones/>.
- [75] The Audacity Team. Audacity. <http://sourceforge.net/projects/audacity/>.
- [76] Tim Schafer, Peter Tsacle, Eric Ingerson, Bret Mogilefsky, Peter Chan. Grim Fandango Puzzle Document. http://www.przygodoskop.pl/021/inne/gf_doc.pdf.
- [77] Tomas Pettersson. Sfxr. http://drpetter.se/project_sfxr.html.
- [78] Toni Martin. Configurar Gimp para Pixel art : Tutorial. <http://www.pixelesmil.com/2011/09/configurar-gimp-para-pixel-art-tutorial.html>.
- [79] Trello, Inc. Trello. <https://trello.com/>.
- [80] Unity Official Forum. Using Plugins for x86 and x64 architectures (issue with sqlite3.dll in beta 9) SOLVED. http://forum.unity3d.com/threads/using-plugins-for-x86-and-x86_64-architectures-issue-with-sqlite3-dll-in-beta-9-solved.276719/.
- [81] Unity Technologies. Input. <http://docs.unity3d.com/es/current/Manual/Input.html>.

- [82] Unity Technologies. Requerimientos mínimos de *Unity*. <http://unity3d.com/es/unity/system-requirements>.
- [83] Unity Technologies. Sprite Renderer. <http://docs.unity3d.com/es/current/Manual/class-SpriteRenderer.html>.
- [84] Unity Technologies. Unity Answers. <http://answers.unity3d.com/>.
- [85] Unity Technologies. Unity comes to Linux: Experimental build now available. <http://blogs.unity3d.com/es/2015/08/26/unity-comes-to-linux-experimental-build-now-available/>.
- [86] Unity Technologies. Unity Learn. <http://unity3d.com/es/learn>.
- [87] Unity Technologies. Unity UI. <http://docs.unity3d.com/es/current/Manual/UISystem.html>.
- [88] Unity Technologies. Unity UI Tutorials. <https://unity3d.com/es/learn/tutorials/topics/user-interface-ui>.
- [89] Unity Technologies. Vista General del Audio. <http://docs.unity3d.com/es/current/Manual/AudioOverview.html>.
- [90] Unity Wiki. Csharp Coding Guidelines. http://wiki.unity3d.com/index.php/Csharp_Coding_Guidelines.
- [91] Universidad Autónoma de Madrid. Virtuam. <http://aida.ii.uam.es/virtuam/proyectos/proyectos-educativos/proyecto-desarrollo-ensenanzas-uam-2011>.
- [92] Universidad de Cádiz. Buscador de la biblioteca de la UCA. http://diana.uca.es/search*spi.
- [93] Universidad de Cádiz. Recursos doceañistas de la biblioteca de la UCA. https://diana.uca.es/search~S7*spi?/dRecursos+Docea{u00F1}istas/drecursos+docean~aistas/-3%2C-1%2C0%2CB/exact&FF=drecursos+docean~aistas&1%2C394%2C.
- [94] Vince Twelve. Resonance's Original Design Document. <http://www.adventuregamestudio.co.uk/forums/index.php?topic=47824.0>.
- [95] WikiCAAD. Aventuras AD. http://wiki.caad.es/Aventuras_AD.
- [96] Wikipedia. Army Battlezone. <http://es.wikipedia.org/wiki/Battlezone>.
- [97] Wikipedia. Back to the Future: The Game. http://es.wikipedia.org/wiki/Back_to_the_Future:_The_Game.
- [98] Wikipedia. Botanicula. <http://es.wikipedia.org/wiki/Botanicula>.
- [99] Wikipedia. Broken Age. http://es.wikipedia.org/wiki/Broken_Age.
- [100] Wikipedia. Broken Sword: La leyenda de los templarios. http://es.wikipedia.org/wiki/Broken_Sword:_La_leyenda_de_los_templarios.
- [101] Wikipedia. Carmen Sandiego. http://es.wikipedia.org/wiki/Carmen_Sandiego.

- [102] Wikipedia. Clock Tower. http://es.wikipedia.org/wiki/Clock_Tower:_The_First_Fear.
- [103] Wikipedia. Day of the Tentacle. http://es.wikipedia.org/wiki/Day_of_the_Tentacle.
- [104] Wikipedia. Dinamic Software. https://es.wikipedia.org/wiki/Dinamic_Softwares.
- [105] Wikipedia. Don Quijote (videojuego de 1987). [https://es.wikipedia.org/wiki/Don_Quijote_\(videojuego_de_1987\)](https://es.wikipedia.org/wiki/Don_Quijote_(videojuego_de_1987)).
- [106] Wikipedia. Double Fine Productions. http://es.wikipedia.org/wiki/Double_Fine_Productions.
- [107] Wikipedia. Dreamfall: The Longest Journey. http://es.wikipedia.org/wiki/Dreamfall:_The_Longest_Journey.
- [108] Wikipedia. Erbe Software. https://es.wikipedia.org/wiki/Erbe_Software.
- [109] Wikipedia. Full Throttle. http://es.wikipedia.org/wiki/Full_Throttle.
- [110] Wikipedia. Grim Fandango. http://es.wikipedia.org/wiki/Grim_Fandango.
- [111] Wikipedia. Heavy Rain. http://es.wikipedia.org/wiki/Heavy_Rain.
- [112] Wikipedia. Hollywood Monsters. http://es.wikipedia.org/wiki/Hollywood_Monsters.
- [113] Wikipedia. Hollywood Monsters 2. http://es.wikipedia.org/wiki/Hollywood_Monsters_2.
- [114] Wikipedia. Hotel Dusk: Room 215. http://es.wikipedia.org/wiki/Hotel_Dusk:_Room_215.
- [115] Wikipedia. Human Entertainment. http://es.wikipedia.org/wiki/Human_Entertainment.
- [116] Wikipedia. Igor: Objetivo Uikokahonia. http://es.wikipedia.org/wiki/Igor:_Objetivo_Uikokahonia.
- [117] Wikipedia. Indiana Jones and the Last Crusade: The Graphic Adventure. http://es.wikipedia.org/wiki/Indiana_Jones_and_the_Last_Crusade:_The_Graphic_Adventure.
- [118] Wikipedia. Kickstarter. <http://es.wikipedia.org/wiki/Kickstarter>.
- [119] Wikipedia. King's Quest. http://es.wikipedia.org/wiki/King'_s_Quest_I:_Quest_for_the_Crown.
- [120] Wikipedia. King's quest. http://es.wikipedia.org/wiki/King_Quest.
- [121] Wikipedia. La Aventura Original. http://es.wikipedia.org/wiki/La_Aventura_Original.
- [122] Wikipedia. Leisure Suit Larry in the Land of the Lounge Lizards. http://es.wikipedia.org/wiki/Leisure_Suit_Larry_in_the_Land_of_the_Lounge_Lizards.

- [123] Wikipedia. Leisure Suit Larry: Reloaded. http://en.wikipedia.org/wiki/Leisure_Suit_Larry:_Reloaded.
- [124] Wikipedia. Lucasarts. <http://es.wikipedia.org/wiki/LucasArts>.
- [125] Wikipedia. Machinarium. <http://es.wikipedia.org/wiki/Machinarium>.
- [126] Wikipedia. Maniac Mansion. http://es.wikipedia.org/wiki/Maniac_Mansion.
- [127] Wikipedia. Mario is Missing. https://es.wikipedia.org/wiki/Mario_is_Missing!
- [128] Wikipedia. Micromecenazgo. <https://es.wikipedia.org/wiki/Micromecenazgo>.
- [129] Wikipedia. Myst. <http://es.wikipedia.org/wiki/Myst>.
- [130] Wikipedia. Mystery House. http://es.wikipedia.org/wiki/Mystery_House.
- [131] Wikipedia. New York Crimes. http://es.wikipedia.org/wiki/New_york_crimes.
- [132] Wikipedia. Pendulo Studios. http://es.wikipedia.org/wiki/Pendulo_Studios,_S.L.
- [133] Wikipedia. Policenauts. <http://es.wikipedia.org/wiki/Policenauts>.
- [134] Wikipedia. Portal. [http://es.wikipedia.org/wiki/Portal_\(videojuego\)](http://es.wikipedia.org/wiki/Portal_(videojuego)).
- [135] Wikipedia. Runaway: A Road Adventure. http://es.wikipedia.org/wiki/Runaway:_A_Road_Adventure.
- [136] Wikipedia. Shadow of the Comet. http://es.wikipedia.org/wiki/Shadow_of_the_Comet.
- [137] Wikipedia. Sierra Online. http://es.wikipedia.org/wiki/Sierra_Online.
- [138] Wikipedia. Strong Bad's Cool Game for Attractive People. http://en.wikipedia.org/wiki/Strong_Bad'_s_Cool_Game_for_Attractive_People.
- [139] Wikipedia. Tales of Monkey Island. http://es.wikipedia.org/wiki/Tales_of_Monkey_Island.
- [140] Wikipedia. The Secret of Monkey Island. http://es.wikipedia.org/wiki/The_Secret_of_Monkey_Island.
- [141] Wikipedia. The Walking Dead. [http://es.wikipedia.org/wiki/The_Walking_Dead_\(videojuego\)](http://es.wikipedia.org/wiki/The_Walking_Dead_(videojuego)).
- [142] Wikipedia. Tim Schafer. http://es.wikipedia.org/wiki/Tim_Schafer.
- [143] Wikipedia. Wizard and the Princess. http://es.wikipedia.org/wiki/Wizard_and_the_Princess.
- [144] Wikipedia española. Brain Training del Dr. Kawashima. https://es.wikipedia.org/wiki/Brain_Training_del_Dr._Kawashima_%C2%BFCu%C3%A1ntos_a%C3%B3los_tiene_su_cerebro%3F.

- [145] Wikipedia española. Centroide. <https://es.wikipedia.org/wiki/Centroide>.
- [146] Wikipedia española. Desarrollo en Cascada. https://es.wikipedia.org/wiki/Desarrollo_en_cascada.
- [147] Wikipedia española. Game Jams. https://es.wikipedia.org/wiki/Game_jam.
- [148] Wikipedia española. Pixel Art. https://es.wikipedia.org/wiki/Pixel_art.
- [149] Wikipedia española. Scrum. <https://es.wikipedia.org/wiki/Scrum>.
- [150] Wikipedia española. Sistema de edición no lineal. https://es.wikipedia.org/wiki/Sistema_de_edici%C3%B3n_no_lineal.
- [151] Wikipedia española. Streaming. <https://es.wikipedia.org/wiki/Streaming>.
- [152] Wikipedia inglesa. Space Quest. http://en.wikipedia.org/wiki/Space_Quest.
- [153] Julian Raschke y Jan Lücker. Gosu. <http://www.libgosu.org/>.
- [154] Youtube. Lista de reproducción de vídeos de 1812: La Aventura. <https://www.youtube.com/watch?v=Ei3XxmpahSA&list=PLpWTFopWU5f3U7w7qQykKsniv16T0vBYl>.
- [155] Myke Zynda. From visual simulation to virtual reality to games. *Computer*, (38):25–32, September 2005.

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with … Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.