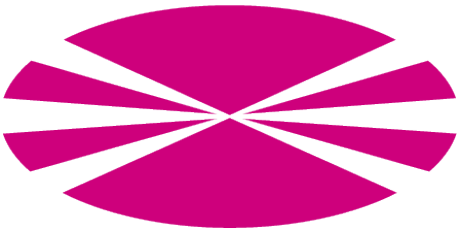


Facultade de Informática
Universidade de A Coruña
Departamento de Computación

PROYECTO DE FIN DE CARRERA
Ingeniería Informática

Desarrollo de un videojuego *roguelike* para invidentes aplicando técnicas de Procesamiento del Lenguaje Natural.

Alumno: Darío Penas Sabín
Director: Carlos Gómez Rodríguez
Director: Jesús Vilares Ferro
Fecha: 5 de junio de 2016

Resumen:

La industria del entretenimiento digital ha crecido inmensamente en los últimos años, llegando a alcanzar números de ventas jamás vistos anteriormente. Parte de la razón de este crecimiento viene dada por una mejora radical en el aspecto visual, necesaria para que el jugador se sienta inmerso en la aventura que se le está planteando. Estas mejoras, sin embargo, dejan de lado a muchos jugadores que, por diferentes motivos, no son capaces de apreciar el contenido visual que se les ofrece o tienen problemas para ello, haciendo imposible que disfruten del contenido ofrecido.

Este proyecto consiste en la creación de "The accessible dungeon", un videojuego de género *roguelike* para invidentes que, desde un principio, parte de la idea de generar contenido específicamente diseñado para que pueda ser jugado por todo el mundo, haciendo énfasis en ofrecer al jugador una diversa cantidad de frases que describan lo que está sucediendo en su alrededor y que serán generadas automáticamente en base a una serie de gramáticas y diccionarios, empleando para ello técnicas de Procesamiento del Lenguaje Natural.

Lista de palabras clave:

- Tiflotecnología
- Procesamiento del Lenguaje Natural
- Accesibilidad
- Entretenimiento Digital
- *Roguelike*
- Java
- *Open Source*

Agradecimientos

Me gustaría aprovechar este pequeño espacio para agredecer a todas las personas que me ayudaron, en diferentes niveles y formas, a elaborar y terminar este proyecto.

Gracias a todos aquellos que dedicaron parte de su tiempo a resolverme dudas sobre tiflotecnología y accesibilidad (sobre todo a los usuarios de *Reddit ais523* y *fastfinge*), así como a todos los profesores que tuve durante los cinco años de carrera y que lograron que llegue al punto en el que me encuentro profesionalmente. En especial me gustaría agredecer a mis directores, Jesús y Carlos, el esfuerzo extra que han hecho durante este tiempo en el que he estado trabajando en el extranjero para mantenerme al tanto y resolverme dudas a distancia cuando estaba perdido.

Tampoco puedo olvidarme de todos los amigos que he hecho durante todos estos años y con los que he pasado tantas horas delante y fuera de la pantalla. Adrián (insaciable compañero de prácticas que me tuvo que aguantar durante tantos años), Daniel, Vanesa, Santi, Chava, Mónica, Brais, María, Xacobe y otros muchos. ¡Gracias por vuestra ayuda y amistad!

Por último, gracias a todas aquellas personas (especialmente a mis padres y a mi novia Diantha) que me ayudaron y presionaron a que terminara el proyecto cuanto antes y no lo dejara pasar.

¡Gracias a todos por todos estos maravillosos años!

Darío Penas Sabín
Amsterdam, 5 de junio de 2016

Índice general

Índice de figuras

Capítulo 1

Introducción

En este capítulo introductorio se explicarán los aspectos necesarios para entender lo más importante del proyecto, la motivación para la realización del mismo y un breve resumen del resto de capítulos que forman parte de la memoria. Cabe destacar que todas las imágenes mostradas en este proyecto son de dominio público.

1.1. Videojuegos y personas invidentes

La mayor parte de los videojuegos comerciales no tienen en cuenta a muchas minorías de la sociedad. Haciendo una pequeña búsqueda online pueden encontrarse miles de usuarios quejándose de *first person shooters. Género de videojuegos en primera persona donde el usuario dispara con diferentes armas a enemigos.* que tienen un *campo de visión* limitado, que termina causándoles mareos al poco rato; usuarios zurdos que tienen que acomodarse a ciertos controles a no existir una opción para cambiarlos; usuarios daltónicos protestando que diferentes juegos (como por ejemplo The Witness[1]), basan buena parte de su mecánica en que el jugador sea capaz de distinguir diferentes colores; usuarios invidentes que no pueden disfrutar de prácticamente ninguno de los títulos que se encuentran en el mercado, etc.

En este proyecto nuestro objetivo ha sido el crear un videojuego desde cero que tenga en cuenta la problemática de las personas con deficiencias visuales, tomando especial relevancia el desarrollo para invidentes y centrándose en los aspectos que sean relevantes para ellos.

[1] Juego de puzzles en primera persona: http://the-witness.net/

1.1.1. Visita a la *ONCE*

El pasado curso, junto a uno de mis co-directores, asistí de tiflotecnología organizado por la Organización Nacional de Ciegos Españoles (ONCE) donde, entre otros temas, uno de sus formadores nos habló sobre la tiflotecnología, nos mostró la forma en la que usaban ordenadores y teléfonos móviles, incluso para leer código fuente y los errores, muchos de ellos fácilmente solventables, que se cometían día a día en temas de accesibilidad por parte de los desarrolladores. Esta visita nos resultó muy interesante y gracias a ella fuimos capaces de detectar posibles mejoras que tener en cuenta en el proyecto y los fallos que no deberíamos cometer en su diseño. También se ofrecieron a probar el proyecto una vez estuviera listo, pero este tema será tratado en las próximas secciones.

1.2. Motivación

A nivel personal, como la mayor parte de las personas de mi generación actual, el entretenimiento digital siempre ha sido parte de mi vida y, en mi caso, una de las razones por las que desde pequeño estuve interesado en la informática y motivo por el que, finalmente, acabé estudiando esta carrera. Del mismo modo, y habiéndome relacionado con bastante gente con toda clase de necesidades especiales durante un gran periodo de tiempo, mi interés por la tiflotecnología y las limitaciones que la tecnología ofrece a millones de personas no hizo más que crecer año a año.

1.3. Estructura de la memoria

La memoria está formada por ocho capítulos en los que se explican los pasos tomados a la hora de crear el proyecto.

Capítulo 1. Introducción. El presente capítulo, donde se explican, de manera general y resumida, en qué consiste el proyecto, la motivación del mismo y la estructura de la memoria.

Capítulo 2. Estado del Arte. En este apartado hablaremos de otros proyectos similares, de la situación actual de la industria sobre el problema que tratamos en este proyecto y del diferente software que es utilizado en relación con la tiflotecnología.

Capítulo 3. Fundamentos Tecnológicos. Trataremos las herramientas y recursos empleados durante la elaboración del proyecto.

Capítulo 4. Metodología. Se detallarán las prácticas y metodologías de desarrollo empleadas y la razón de su uso.

Capítulo 5. Planificación y Seguimento. Detallaremos la planificación y seguimiento de cada una de las etapas del proyecto.

Capítulo 6. Análisis de Requisitos. Se comentará el análisis de requisitos para este proyecto y se explicará en detalle cada uno de los mismos.

Capítulo 7. Diseño e Implementación. En este capítulo explicaremos los detalles del diseño e implementación del sistema, centrándonos especialmente en aquellas partes del programa que consideramos más relevantes.

Capítulo 8. Conclusión y trabajo futuro. En este apartado se comentarán las conclusiones obtenidas y se detallarán los posibles cambios, mejoras y extras que se podrían tener en cuenta en el futuro para su mejora.

Capítulo 2

Estado del Arte

En este capítulo hablaremos, principalmente, de dos importantes aspectos. El primero es la situación de la industria del entretenimiento digital hoy en día, seguido de una introducción a los videojuegos del género *roguelike*. El segundo son los elementos que dificultan y facilitan el uso de diferentes programas y *software* a ciertos sectores de la sociedad como invidentes o daltónicos; cómo algunos programas intentan solventar estos problemas y las razones por las que hemos elegido introducir ciertos elementos en nuestro proyecto para lidiar con los mismos.

2.1. La industria del entretenimiento digital en la actualidad

Desde sus primeros pasos hasta hoy en día, tal y como sucede con muchas de las novedades en el mundo del entretenimiento y la cultura, el sector del ocio digital ha sufrido cierto estigma por parte de una gran parte de la población, siendo censurado y degradado en mayor o menor medida, no tan solo por cierta parte de la sociedad al considerarlo como algo enfocado para niños y sin mucho interés o relevancia cultural, sino también por muchos medios de comunicación e incluso administraciones públicas. A pesar de que hoy en día este problema todavía está activo,[1] la industria se ha expandido tanto (consolas, ordenadores, navegadores, redes sociales, móvil...), que cada vez es más complicado encontrar a alguien que no haya jugado a algún videojuego en algún momento de su vida, ya que algo que forma parte del día a día de buena parte del público y defenderlo como un elemento de gran importancia cultural es cada vez más sencillo.

[1] Es común que cada año en Australia se censuren algunos juegos como Paranautical Activity por razones por las cuales otras formas de entretenimiento y cultura como películas o libros no se ven tan afectados http://goo.gl/H2DXCW

2.1.1. La industria, en números

En todo el mundo, pero especialmente en EEUU, la industria de los videojuegos es uno de los sectores con más crecimiento [The14] llegando a generar, solamente en ventas digitales, alrededor de 61 billones de dólares en el año 2015 [CNB14] y nuevos estudios comentan que no parará de crecer hasta el año 2019 [2]. Ya en 2009 grandes títulos generaban más beneficios que las grandes producciones de *Hollywood* [3] y esta diferencia no ha hecho más que crecer desde entonces. En el año 2013 la industria cinematográfica americana ingresó globalmente 35.9 millones de dólares [4], mientras que la industria del entretenimiento digital en el mismo periodo ingresó 70.9 billones [5] y no ha hecho más que crecer desde entonces. En comparación con la industria musical, solamente el videojuego Grand Theft Auto V prácticamente generó el mismo número de beneficios. [6]

Este gran éxito se debe, en gran parte, a la irrupción de los juegos desarrollados para dispositivos móviles, cuyo beneficio ha ido aumentando enormemente durante los últimos años. [7] Sin embargo, esto no significa que el resto de plataformas no estén triunfando o se hayan quedado estancadas. Solamente Steam,[8] la plataforma de distribución digital para PC por excelencia que ha sido desarrollada por Valve, ha generado alrededor de 3 billones y medio de dólares en el año 2015 [Gam16].

También hemos visto nuevas consolas salir al mercado hace poco más de dos años: PlayStation 4 y Xbox One. La primera de ellas ha logrado vender, a principios del año 2016, alrededor de 36 millones de unidades [Sar16], mientras que la consola de Microsoft llega a los 19.1 millones en el mismo periodo [Goo16], haciendo un total de más de 55 millones de unidades en total, la cual es una gran cifra en comparación con los años anteriores.

Con el mercado del PC resurgiendo, las consolas de sobremesa obteniendo grandes cifras de ventas, las portátiles resistiendo la lucha contra los móviles, el mercado de los videojuegos para móvil en esplendor y los cascos de realidad virtual llegando al mercado este año 2016 y obteniendo grandes números de ventas y preventas, todo parece indicar que la industria de ocio digital no hará más que crecer durante los próximos años.

[2] http://goo.gl/yOlAzc
[3] https://goo.gl/yu3B4O
[4] http://goo.gl/2A44Xk
[5] https://goo.gl/6avRXf
[6] http://goo.gl/E79c68
[7] La venta de videojuegos a nivel global crece año tras año, pero el mayor aumento de beneficio se está centrando en el mercado de los juegos para móvil.
[8] store.steampowered.com

2.1.2. Conclusión

Lo que comenzó hace varias décadas como un modo de entretenimiento minoritario, generalmente enfocado a un público infantil o adolescente y que miraba a otras industrias como la cinematográfica con recelo, tanto en números como en visibilidad, se ha convertido en todo lo que había deseado y más. Gracias a grandes títulos y a su expansión a toda clase de dispositivos, no se puede hablar de la industria del entretenimiento sin hablar de videojuegos y, en muchos casos, algunos de esos títulos han logrado ser nombrados como obras de arte en su género, pasando a la historia y siendo recordados a lo largo de los años. De hecho, varios museos, como el Museum of Modern Art (MoMA), están creando colecciones permanentes de videojuegos [9].

2.2. El género *Roguelike.*

2.2.1. Qué es y orígenes

En 1983, Michael Toy y Glenn Wichman crearon un videojuego llamado *Rogue*,[10] que acabó definiendo el género *roguelike*, el cual sigue vigente y gozando de gran esplendor hoy en día. Principalmente se centra en el control de un personaje que avanza por diferentes tipos de mazmorras derrotando enemigos, consiguiendo mejores estadísticas personales y objetos que permiten al usuario seguir avanzando. Se puede observar en la Figura 2.1.

Las características principales que definieron inicialmente a *Rogue* y, por extensión, al género son:

Permadeath: Se trata de videojuegos con un nivel de dificultad destacado y caracterizados por lo que se denomina*permadeath*. Una vez que el jugador muere, tiene que empezar desde el principio; no hay partidas guardadas (por lo que no puedes volver a atrás), aunque suele permitirse jugarse una misma partida en diferentes sesiones. Esta dificultad obligará al jugador a rejugarlo una y otra vez.

Aleatoriedad: Cada vez que el jugador comienza una nueva partida se encontrará con ciertos elementos que han cambiado con respecto a la vez anterior: el mapa es distinto, los elementos y enemigos se encuentran en sitios diferentes, los objetos obtenidos han cambiado, etc., lo que favorece la rejugabilidad.

[9] https://goo.gl/nioYtu

[10] Desde 2014 este juego se encuentra disponible en su totalidad y completamente gratuito en archive.org

Progresión: Una de las frases más escuchadas en las críticas positivas que *Rogue* recibió tras su lanzamiento era que el jugador sentía la necesidad de intentar llegar más lejos en cada una de las partidas jugadas [Pou84]. Esto viene dado, sobre todo, por la sensación de progresión y de que en cada *run*[11] el usuario vaya mejorando. Dentro de la propia partida también existe una progresión a medida que el usuario derrota enemigos, consiguiendo puntos de experiencia, mejorando su personaje y equipamiento. La intriga por saber los nuevos objetos que se pueden conseguir, los nuevos enemigos con los que nos enfrentaremos y mapas que se generarán y explorarán, hace que el juego sea fácilmente rejugable y atrayente para todo tipo de usuarios.

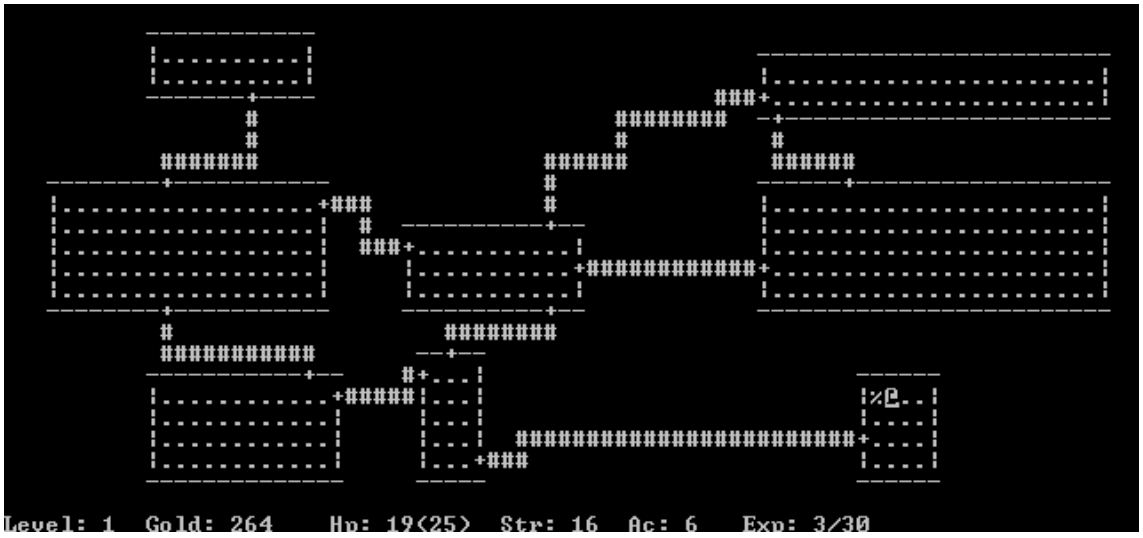


Figura 2.1: Captura de pantalla del videojuego *Rogue* con gráficos ASCII que describen el mapa y los elementos en él, muy usados en muchos *roguelikes*, incluso hoy en día.

A partir de este momento muchos fueron los títulos, algunos de ellos desarrollados por aficionados, que decidieron imitar estas características que hemos mencionado y añadiendo, cambiando o enfatizando diferentes elementos. Por este motivo es por lo que se denominan *roguelikes*, dado que siguen los pasos marcados por *Rogue*. Por ejemplo, *Rogue Legacy* [12] tiene todos los elementos que hemos comentado, pero su combate en vez de ser por turnos es en tiempo real y la progesión viene dada a la hora de elegir el personaje a jugar y los objetos con los que equiparse en cada una de las partidas.

2.2.2. En la actualidad

Tras el éxito de *Rogue*, fueron muchos los juegos que simularon su fórmula de éxito e intentaron mejorarlo, sobre todo gráficamente. Algunos se centran en diferentes aspectos

[11] Palabra comúnmente usada en estos géneros y que se refiere a una partida desde su inicio hasta que el jugador pierde

[12] roguelegacy.com

(combate en vez de exploración, por ejemplo) y llegan a ser completamente diferentes a la hora de jugarlos (por turnos o tiempo real) pero, sin embargo, todos conservan buena parte de las características que hicieron al género famoso hasta hoy en día. Uno de ellos fue Vultures. Figura 2.2.

Figura 2.2: Captura de pantalla del videojuego Vultures

Incluso en la actualidad una gran cantidad de nuevos títulos, muchos de ellos desarrollados por empresas independientes, no dan especial importancia a tener gráficos realistas en tres dimensiones o texturas increíbles (por ello sigue habiendo muchos títulos con pseudo-gráficos en *ASCII*), sino que optan por profundizar en crear una ambientación interesante y ofrecer un sistema de combate y exploración únicos y pulidos. Uno de estos títulos que ha triunfado enormemente desde su lanzamiento en 2012, con más de dos millones de copias vendidas solamente en Steam,[13] es *FTL: Faster Than Light*, del que hemos adjuntado una captura de pantalla en la Figura 2.3, aunque no está considerado un roguelike al 100 %. ADOM,[14] sin embargo, que es un *roguelike* a la vieja usanza, fue un proyecto de *crowdfunding* que consiguió alrededor de cien mil dólares por parte de la comunidad y actualmente se vende en *Steam*.

[13] http://goo.gl/ZqrYbs
[14] http://goo.gl/XHcVQ9

El género *roguelike* también ha influido otros géneros como los *action RPGs* y sagas como *Diablo*[15], que goza de millones de jugadores. Elementos como el mencionado *permadeath* o la generación procedural de mapas y misiones se han incluido en muchos títulos para hacerlos más atractivos, por ejemplo *X-COM 2* [16] o el futuro y muy esperado *No Man's Sky*, donde todo el universo y todos sus elementos son generados proceduralmente.[17]

Figura 2.3: Captura de pantalla del videojuego FTL: Faster Than Light

2.2.2.1. La aparición de subgéneros

Perder todo el progreso y tener que empezar desde el principio sin haber conseguido nada más que la experiencia personal es algo que no atrae al perfil del jugador actual (con una media de edad que no puede permitirse cientos de horas jugando). Por esta razón son numerosos los juegos que han añadido más elementos de progreso general para que el jugador no se sienta frustrado y consiga una recompensa de forma más inmediata. Estos incentivos pueden ser nuevos personajes con los que jugar, puntos de experiencia generales o dinero con lo que poder equipar y mejorar desde un principio a nuestro personaje para poder llegar más lejos que la anterior vez, diferentes modos de juegos que se desbloquean al llegar a una cierta puntuación, etc. También es más común ver juegos de este tipo que se basan en partidas cortas, de como mucho una hora, para que la repetición sea mayor y perder el personaje no se vea como un fracaso, sino algo que forma parte del juego en sí.

[15] us.battle.net/d3/en
[16] https://xcom.com
[17] http://goo.gl/ESoiml

Estos cambios han modificado el género que *Rogue* creó en un inicio y no siempre se han tomado positivamente por parte de la comunidad de aficionados más conservadores, que se suele quejar de que muchos títulos que se definen a sí mismos como *roguelike* no contienen ciertas características, que una vez definieron el género, especialmente en tema de dificultad. Por este motivo se han definido subgéneros como el *roguelite*, que toman muchas de esas ideas iniciales, pero añaden o ignoran otras muchas para crear un título que sea un poco más sencillo y no penalice tanto al jugador. Algunos títulos de este género son *Binding of Isaac*[18] o *Ziggurat*[19].

Debido a estos nuevos subgéneros, se han creado definiciones que miden el nivel de *roguelike* que tiene un título. Una de ellas es llamada "Berlin interpretation". [20]

2.2.3. Elementos *roguelike* en nuestro proyecto

En nuestro caso hemos creado un *roguelike* de corte más purista, similar a Rogue, no solamente estéticamente, pero también en diseño y funcionalidad. El jugador explorará un mapa aleatoriamente generado y luchará contra diferentes enemigos que intentarán eliminarlo de diferentes formas. En base al nivel que el usuario tenga, los enemigos serán más o menos poderosos y la recompensa por abatirlos será mayor.

El objetivo del juego es llegar lo más lejos posible dentro de la mazmorra. Cada vez que el jugador entre en un portal se incrementará su puntuación (el número de puntos se mostrará en la pantalla) y, cada vez que esto suceda, un nuevo mapa será generado con diferentes características y contenido, por lo que el jugador tendrá que volver a buscar la salida de este nuevo nivel, batiendo enemigos, descubriendo tesoros, y mejorando sus estadísticas personales durante dicho recorrido. El juego es complicado, aleatorio y con una sensación de progreso, tal y como corresponde al género *roguelike*.

2.3. Problemas de la tecnología para invidentes y daltónicos

Algunos de los factores que hacen de la tecnología un elemento prácticamente imprescindible hoy en día y que facilitan su usabilidad crean, paradójicamente, una barrera para mucha gente que no puede disfrutar de ella, dificultando su día a día enormemente.

En esta sección hablaremos sobre algunos de los problemas que tanto invidentes como daltónicos se encuentran actualmente a la hora de usar diferentes tipos de software (especialmente videojuegos). Asimismo veremos cómo, en bastantes ocasiones, solventarlos no requiere nada más que prestar un poco de atención y tener en cuenta aspectos sim-

[18] bindingofisaac.com
[19] http://goo.gl/wB1kCU
[20] http://goo.gl/29nDSp

ples, lo que pone en manifiesto que muchas de estas limitaciones son dadas más por el desconocimiento que por la dificultad o coste de implementar una solución.

2.3.1. Dificultades a las que se enfrentan los invidentes

A pesar de que han sido muchos los avances que hemos podido observar durante los últimos años en esta área, las dificultades que tienen invidentes o personas con diferentes grados de ceguera en nuestra sociedad son enorme y, tristemente, tanto dispositivos tecnológicos como *software* no se ven excluidos de esta lista. No poder hacer uso de una interfaz gráfica o ver lo que está sucediendo en la pantalla es un problema que automáticamente imposibilita el uso de la mayoría de las aplicaciones y videojuegos. Incluso navegar por Internet, donde la mayoría del contenido es texto que debería poder ser leído fácilmente por un lector de pantalla, puede llegar a ser un quebradero de cabeza.

2.3.2. Cómo abordar parte de estos problemas para los invidentes

Uno de los principales problemas de los que se quejan los usuarios, tal y como nos señalaron los informadores en la charla de la *ONCE*, es que muchos de los desarrolladores de software o adaptaciones específicas para ciegos no tienen en cuenta que, la mayoría de las veces, junto al usuario invidente hay también usuarios videntes que les pueden describir lo que está sucediendo en la pantalla. Por ello es muy importante crear una interfaz que esté actualizada y muestre la misma información que la persona invidente reciba.

En el caso de los videojuegos, algunos de ellos han visto versiones adaptadas para invidentes, la mayoría de ellas creadas por la propia comunidad y no por las empresas desarrolladoras, así como nuevos títulos que se centran en ofrecer una experiencia revolucionaria desde el principio. Un buen ejemplo de ello es Shades of Doom,[21] que solamente usa sonido (principalmente ruidos repetitivos) y muy pocas frases para que el jugador sea capaz de descubrir lo que tiene que hacer, dónde está dentro del mapa, y describir todos los elementos que un usuario vidente recibe a través del uso de una interfaz gráfica.

También existen juegos originales que están completamente adaptados para invidentes, como es el caso de *A Blind Legend*[22] para *Android*.

2.3.3. Tiflotecnología

La tiflotecnología es el conjunto de técnicas, conocimientos y recursos enfocados a ayudar a personas con cualquier tipo de deficiencia visual para que puedan usar dispositivos

[21] http://www.gmagames.com/sod.html
[22] https://goo.gl/nasbIx

electrónicos fácilmente. Ejemplos de tiflotecnología serían lupas magnificadores, tanto a nivel de *software* como *hardware*; *software* de lectura de pantalla como *NVDA*, o barras de braille que permiten al usuario leer lo que está en la pantalla y que suele ser usada para diferentes tareas como programar o realizar cálculos matemáticos.

En nuestro proyecto hemos tenido en cuenta estos elementos para adaptar lo mejor posible nuestro juego a las herramientas que las personas con deficiencia visual usan en su día a día.

2.3.4. Dificultades a los que se enfrentan los daltónicos

El daltonismo es un defecto genético que afecta aproximadamente al 1 % de la población y que tiene diferentes variantes dependiendo de los tipos de colores con los que tienen problema. Lo que en un principio parece como un pequeño inconveniente que no debería afectar demasiado la vida de la persona que lo sufre, lo cierto es que hay muchas ocasiones en las que incluso disfrutar de un simple partido de fútbol puede convertirse en algo casi imposible para ellos, dado que en algunos partidos los jugadores llevan camisetas con colores problemáticos. [23] Incluso ver un mero mapa puede ser una complicación[24].

En el caso de los videojuegos, donde los gráficos y paletas de colores juegan un gran papel en el arte del título, este problema se ve acentuado, especialmente si parte de las mecánicas del juego necesitan que el jugador sea capaz de distinguir colores para obtener cierta información relevante, y esto es algo que sucede en numerosos títulos. En algunos casos puede suponer una pequeña molestia como en *Borderlands*,[25] un juego de acción en primera persona donde hay armas especiales que se diferencian, únicamente, por el color en el que se encuentra un determinado texto. En la Figura 2.4 se puede observar cómo se vería normalmente y en la Figura 2.5 cómo algunas personas daltónicas lo ven. Sin embargo, el mayor problema viene cuando estas limitaciones pueden llegar a arruinar el juego, como ocurre en *The Witness*,[26] un juego de puzzles ya mencionado en apartados anteriores en el que, para poder resolver muchos de ellos, es preciso que el usuario pueda distinguir distintos colores. También nos encontramos con juegos de lucha en dos dimensiones donde la única diferencia entre los escenarios del fondo y los personajes principales es el color de los mismos. No ver la diferencia complica que el jugador detecte si un elemento está al frente o al fondo de dicho escenario.

[23] http://goo.gl/o3GkrP
[24] Imagen que muestra los diferentes tipos de daltonismo y cómo lo perciben personas con diferentes tipos de daltonismo
[25] www.borderlandsthegame.com
[26] http://goo.gl/GYPdhH

Figura 2.4: Captura de pantalla del juego Borderlands para personas sin daltonismo.

Figura 2.5: Captura de pantalla del juego Borderlands para personas con cierto tipo de daltonismo.

2.3.5. Cómo abordar parte de estos problemas para los daltónicos

Para poder dar respuesta a esta cuestión, y debido a su naturaleza, se creyó conveniente informarse respecto a la misma de mano de los propios afectados. En nuetro caso recurrimos al portal *Reddit* para consultar a los usuarios daltónicos sobre cómo desarrollar un juego que sea adecuado para ellos [27]. Fueron muchas las respuestas obtenidas, pero se puede resumir en dos alternativas.

[27] https://goo.gl/d6cTqe

La primera es que, en la medida de lo posible, nunca tengamos que diferenciar dos elementos distintos únicamente por su color. A modo de ejemplo y, tal y como un usuario comentó, si estamos desarrollando un juego tipo "Hundir la Flota" y la diferencia entre un barco intacto y uno averiado es que cambiamos su color de verde a rojo, también deberíamos de añadir llamas u otros elementos distintivos a mayores que faciliten al usuario apreciar visualmente que se ha producido un cambio. Del mismo modo, si tenemos un juego tipo *Tetris* o *Candy Crush* donde las piezas son relevantes, en vez de distinguirlas sólo por su color, podríamos hacerlas de formas diferentes o añadir una imagen a cada una de ellas.

La segunda opción es que, si fuese completamente necesario emplear únicamente diferentes colores para diferenciar ciertos elementos entonces deberíamos, o bien optar por combinaciones de colores que, generalmente, no creen ningún problema (azul, amarillo, verde...) lo cual tampoco nos garantiza que hayamos solventado el problema dado que, como ya hemos comentado, hay muchas formas de daltonismo y en algunas de ellas el usuario todavía puede tener problema diferenciándolos dependiendo del color contreto usado), o bien añadimos una opción mediante la que podamos cambiar la paleta de colores usada. Hay algunos videojuegos, la mayor parte de ellos independientes, donde se da esta última opción.

Capítulo 3

Metodología

En este apartado describiremos la metodología utilizada para el desarrollo del proyecto.

En nuestro caso, al ser un proyecto realizado por una sola persona y con un tiempo diario limitado, hemos optado por adaptar una serie de ideas y valores principales de varias metodologías para optimizar los pocos recursos que disponemos.

3.1. Desarrollo en cascada

El *desarrollo en cascada*, también denominado como modelo en cascada, es una metodología con varias etapas donde, para pasar a la siguiente, es completamente necesario que la fase anterior haya sido completada en su totalidad.

Las ventajas que ofrece el desarrollo en cascada son que es muy sencillo de implementar en comparación con otras metodologías, requiere menos tiempo y capital para hacerlo funcionar de manera óptima y es usado ampliamente, por lo que sus ventajas y desventajas son conocidas de antemano y, gracias a ello, podemos planear las posibles dificultades que puedan surgir. La mayor de estas desventajas es que detectar un error en una de las fases finales puede significar tener que replantear los pasos dados en las etapas anteriores, perdiendo así parte del progreso realizado. Incluso un pequeño cambio puede llegar a afectar otras partes del análisis y diseño, forzándonos a perder y rehacer parte del trabajo ya hecho.

3.1.1. Etapas del desarrollo en cascada

Análisis de requisitos: En esta fase el desarrollador y el cliente definen y determinan los requisitos que el *software* deberá cumplir. Generalmente se generan documentos que formalizan dichas decisiones, dado que un cambio en los requisitos a posteriori puede

significar tener que cambiar gran parte del proyecto, tal y como acabamos de comentar, por lo que es necesario tener que llegar a un consenso.

Diseño: Una vez que la fase del análisis ha finalizado, es hora de pasar a la fase de diseño. La idea principal es descomponer lo detallado durante el análisis de requisitos con el cliente y, en base a ellos, crear diferentes diagramas y diseños que ayuden a los programadores a entender lo que debe ser implementado y cómo debe funcionar, incluyendo pseudocódigo o algoritmos de alto nivel en caso de que fuese necesario.

Implementación: En esta fase es donde se escribe el código fuente en base a lo especificado anteriormente, teniendo presente en todo momento la reutilización del código siempre y cuando sea posible. También es importante tener en cuenta la creación de tests y la realización de las primeras pruebas preliminares por parte de los programadores.

Verificación: Llegados a este momento, el cliente puede finalmente probar la implementación. El sistema deberá estar bien testado previamente por parte de los programadores para evitar la mayoría de los problemas que pueden surgir en esta fase.

Mantenimiento: Una vez el software está terminado y se ha entregado al usuario final, se entra en la denominada fase de mantenimiento. En ella el usuario pedirá que se resuelvan problemas, se añadan nuevas funcionalidades y se cambien otras características ya implementadas en base a los nuevos requisitos que pueden ir apareciendo o a nuevas funcionalidades que puedan hacerse necesarias en un futuro y no hubiesen sido tenidas en cuenta en las anteriores fases del desarrollo. Ésta es una de las fases más críticas, dado que se estima que alrededor de un 80 % de los recursos de un proyecto se emplean en el mantenimiento [Pig96].

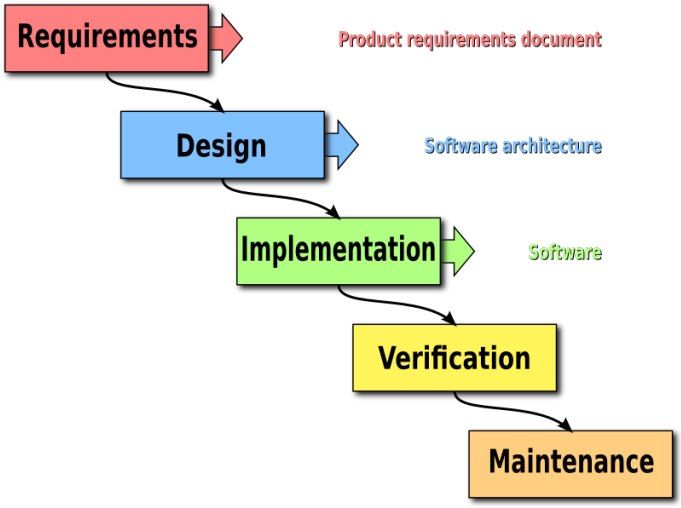


Figura 3.1: Estructura general de una etapa del desarrollo en cascada

3.2. *Scrum*

Scrum pertenece a las denominadas *metodologías ágiles*, siendo na de las más conocidas [Non86]. Su relativa sencillez, así como su flexibilidad y orientación al trabajo en equipo, hace que Scrum sea una de las metodologías más usadas en el desarrollo de software, sobre todo en entornos laborales donde el tamaño del equipo de desarrollo es reducido, aunque su implementación también es posible con equipos mayores, generalmente dividiendo el equipo original en pequeños grupos y manejándolos luego de forma independiente.

3.2.1. Prácticas recomendadas y bases de *Scrum*

Uno de los aspectos esenciales de *scrum* es el *sprint*, un bloque temporal (comúnmente de entre 2 y 4 semanas) donde el equipo de desarrollo trabajará para llegar a un objetivo determinado que, generalmente, consiste en acabar todas las tareas definidas para ese sprint. Al principio de cada *sprint* el *product owner*, persona con altos conocimientos sobre el producto que sirve de unión entre el equipo de desarrollo y el resto de la organización, y *scrum master*, persona encargada de organizar y controlar el *sprint*, junto con el equipo de programadores, definirán lo necesario a realizar en dicho *sprint* en base a las tareas que se encuentran en el *backlog*, que es una lista de tareas que se quieren realizar en el producto y que suele ir creciendo a medida que los usuarios encuentran fallos o necesitan nuevas funcionalidades. Normalmente el *product owner* es el encargado de organizar y decidir aquellas que tienen más prioridad. Todos los integrantes del grupo, así como el propio *product owner* y *scrum master*, decidirán cuánto esfuerzo o tiempo llevará realizar cada una de ellas, dividiéndola en tareas menores en caso de que sea demasiado grande.

También se decidirá quién hará qué, donde los desarrolladores las van cogiendo a medida que finalizan en las que han estado trabajando[1]. De esta manera no todas las tareas tienen un responsable al inicio del *sprint*.

Durante cada día del desarrollo hay una pequeña reunión llamada *sprint stand-up* donde cada uno de los programadores comenta brevemente lo que ha hecho el día anterior, qué tiene planteado realizar ese día y si se ha encontrado con algún problema que le impida continuar.

Al acabar cada *sprint*, el equipo se reúne de nuevo para analizar lo que ha ido bien, qué problemas hubo y las mejoras que deben aplicarse y tenerse en cuenta de cara al futuro, así como elaborar un resumen de estadísticas del equipo. De esta manera, cada nuevo *sprint* será más preciso (dado que sabremos la cantidad de trabajo que el equipo de desarrollo es capaz de hacer en el periodo de tiempo definido) e incluirá el *feedback* de los propios programadores.

Aunque las anteriores son las características esenciales de *Scrum*, al tratarse de una metodología ágil, nada es inamovible y existen bastantes variaciones que se adaptan a todo tipo de equipos tales como la inclusión de otras metodologías. Dependiendo de la empresa, del equipo de trabajo y de las necesidades del mismo, se pueden cambiar o introducir nuevas ideas que mejoren el proceso para ese marco.

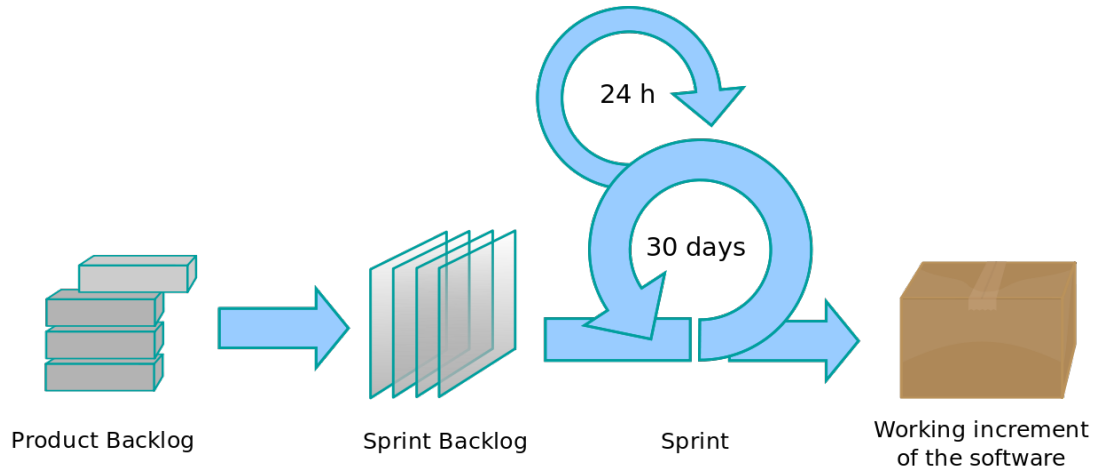


Figura 3.2: Proceso concreto de *Scrum* (en este caso el *sprint* dura cuatro semanas

[1]Algunas compañias llevan Scrum al límite y entrenan a sus empleados para que puedan realizar cualquier tipo de tarea sin limitaciones, por lo que esto es algo que funcionaría muy bien en situaciones similares

3.2.2. Valores

Scrum se basa en flexibilidad y trabajo en equipo. Este es el motivo por el que es muy importante tener un equipo que tenga claro y cumpla con una filosofía y valores básicos. Estos valores son los siguientes:

- Concentración Al tener que finalizar las tareas asignadas al final del *sprint*, la concentración del equipo en general y de cada uno de los miembros en particular, es esencial. Para lograr este objetivo, el *product owner* es el encargado de responder las preguntas del resto de la compañía en nombre de los desarrolladores y solamente molestarlos en caso de que sea realmente necesario.
- Coraje Dado que *Scrum* es una metodología de equipo, la ayuda entre cada uno de los miembros es primordial. De igual modo, cada persona debe ser capaz de enfrentarse a nuevos retos y asumir nuevas responsabilidades para que, en su conjunto, el producto final sea el deseado.
- Compromiso Cada integrante tiene que saber lo que debe hacer y comprometerse a ello. Una vez la reunión inicial haya terminado, es previso que cada desarrollador sepa en lo que tiene que trabajar y pregunte al responsable de la tarea en caso de que no tenga algo claro o crea que no puede llegar a terminar lo acordado durante la reunión. Cada programador debe comprometerse con lo establecido para lograr el éxito del equipo.
- Sinceridad Ser capaz de asumir los errores personales y del propio equipo es la clave para mejorar y evolucionar. Si un miembro del equipo de desarrollo se queda estancado en un problema y no avisa al resto de sus compañeros, el *sprint* en su totalidad puede llegar a fracasar por su culpa. Asumir responsabilidades, errores y pedir ayuda cuando sea necesario debe ser una práctica habitual para que esta metodología funcione sin problemas.
- Respeto En la misma línea que el anterior punto, al trabajar en equipo es imprescindible que tanto los logros como los fracasos se tomen como una forma de aprender y mejorar, por lo que el respeto mutuo y asumir los fallos cometidos durante la iteración es la mejor forma de evolucionar, tanto individualmente como en equipo.

3.3. Metodología elegida

Nuestro enfoque ha sido el de partir de una metodología clásica en cascada integrando elementos de *Scrum* y otras metodologías y prácticas relacionadas, tal y como explicamos a continuación.

Para los elementos principales del sistema, tales como la creación de los mapas, enemigos, motor de generación de frases, etc., nos hemos basado en la metodología cascada para analizar lo que tenemos que realizar, crear el diseño, implementarlo (empezando siempre por los tests, tal y como comentaremos a continuación) y verificarlo. Sin embargo, y dado que los juegos tienden a querer mejorarse continuamente con nuevos elementos de diferente importancia y tamaño (tanto por nuestra parte como de los jugadores-testadores), tiene sentido mantener un *backlog* con todas las nuevas ideas que vayamos recibiendo y el *feedback* recogido, algo para lo que *Scrum* es ideal. También hemos tomado la idea de la creación de tareas cortas y *sprints*, de tal forma que en cada una de las iteraciones intentaremos tener un elemento del juego finalizado, siempre y cuando nuestras estimaciones iniciales no se vean comprometidas. De esta manera seremos capaces de seguir más fácilmente el progreso que hemos realizado y tendremos constancia de los nuevos elementos que queremos implementar en un futuro, así como de los *bugs* a arreglar en próximos *sprints*. Al acabar con el núcleo del juego, la mayor parte de las tareas a realizar a continuación son pequeños cambios y nuevos elementos que no trastocan el diseño inicial del proyecto, por lo que emplear una metodología ágil como *Scrum* es el mejor paso a seguir para terminar las tareas más relevantes lo antes posible. Además, el juego en sí ha sido desarrollado con la idea de hacerlo lo más ampliable y extendible posible. De esta forma, incluso aunque tengamos que cambiar algo en medio del desarrollo, las bases creadas serán sólidas y fácilmente modificables.

Por último, también hemos seguido la práctica de *test driven development* (desarrollo guiado por pruebas) cuya base consiste en que por cada nueva tarea a realizar, deberemos implementar primero los tests unitarios (que, lógicamente, fallarán en un principio) y luego programar la solución en sí para que el test sea válido, refactorizando la solución en caso de que no sea la ideal. Esto nos dará desde el primer momento una idea clara de lo que queremos hacer antes de proceder con la implementación y nos asegurará de que siempre tendremos el test hecho.

Capítulo 4

Planificación y Seguimiento

En este capítulo detallaremos la planificación de cada una de las iteraciones que hemos hecho durante todos los meses de trabajo, así como el seguimiento que hemos realizado de la misma. La elaboración de este proyecto se llevó a cabo durante alrededor de dos años, comenzando en junio del año 2014 y sufriendo un par de interrupciones durante varios meses (desde septiembre de 2014 hasta enero de 2015 y desde mayo de 2015 hasta septiembre del mismo año) a causa de tener que realizar otras actividades que no permitían trabajar en el proyecto al mismo tiempo. También el tiempo dedicado en cada una de las secciones difiere dependiendo del tiempo disponible (durante la mayoría de estos dos años fue necesario compaginer la realización del proyecto proyecto con una jornada laboral a tiempo completo), pero esto es algo que se discutirá en cada una de las secciones individualmente.

Para resumir y clarificar lo que acabamos de comentar, mostramos los periodos de actividad real:

- Desde junio del año 2014 hasta septiembre del 2014 (3 meses).
- Desde enero de 2015 hasta mayo del mismo año (4 meses).
- Desde septiembre del 2015 hasta abril de 2016 (8 meses).

En las próximas secciones hablaremos de lo que hemos desarrollado durante estos tres periodos de tiempo y las iteraciones seguidas en los mismos.

Cabe destacar que todos los datos y tareas mostradas a continuación están realizadas de forma lineal, dado que solamente disponemos de un recurso.

4.1. Junio 2014 - Septiembre 2014

Este periodo comienza el 4 de junio, que es cuando se nos comenta la posibilidad de realizar este proyecto, y termina el 24 de septiembre, por lo que consta de un total de 15 semanas. Hay que tener en cuenta que durante parte de este verano (desde agosto) el proyectando recibe una *internship* en Holanda, por lo que las jornadas de trabajo dedicadas al proyecto solamente constaban de una o dos horas y alrededor de 7/8 horas durante los fines de semana.

Sumando todo el tiempo empleado durante estas semanas se calcula que se han empleado 190 horas en total para la realización de las tareas asignadas a esta iteración en su totalidad.

Durante todas estas semanas el principal objetivo fue el de recoger toda la información general posible para poder empezar el proyecto con buen pie, así como comenzar con la implementación de los mapas y habitaciones que se utilizarán en el juego.

4.1.1. Primera iteración: Análisis de requisitos generales, diseño genérico y preparación y configuración de los elementos necesarios para el comienzo de la implementación

Esta primera iteración empieza el día 4 de junio y termina el 29 del mismo mes. Las tareas realizadas se muestran a continuación:

Análisis de requisitos generales: Al desarrollar un proyecto enfocado a un sector de la población con necesidades especiales del que no formas parte, es muy importante documentarse sobre todos los aspectos que hay que tener en cuenta e intentar ponerse en su piel (por ejemplo usando las herramientas que ellos utilizan diariamente y así recabar ideas), además de preguntar a diferentes miembros de dichos colectivos para coger distintas perspectivas que luego se puedan incluir en nuestro proyecto. Asimismo, desarrollar un videojuego puede llegar a ser una tarea sin fin, dado que es común que tanto a desarrolladores como a usuarios se les ocurran continuamente nuevas características o ideas que añadir, y es por eso que debemos sentar las bases que definan lo que es realmente necesario y lo que no para, así, poder priorizar. Del mismo modo, aprender sobre lo básico del género y poner límites es fundamental para centrar los reducidos recursos que tenemos en crear lo necesario y luego, si se dispusiese de tiempo, empezar con la implementación de otras características no prioritarias adicionales.

Diseño genérico del juego a implementar: Crearemos el primer diseño genérico que nos dará una idea sobre lo que tendremos que realizar y nos guiará sobre el proceso de creación del proyecto. Este primer boceto evolucionará a medida que queramos añadir

nuevos elementos y ser más específicos en ellos, pero contendrá las partes generales y más básicas de lo que pretendemos realizar.

Búsqueda de bibliotecas que se adapten a nuestros requisitos: Hay varias bibliotecas con las que se puede crear una interfaz gráfica sencilla como la del videojuego *Rogue* que mencionamos al principio de esta memoria, pero todas ellas tienen sus ventajas e inconvenientes. Debemos averiguar cuáles de ellas son las más adecuadas para nuestro caso y tomar una decisión sobre cuál usar.

Creación y configuración del entorno de desarrollo para poder empezar la implementación: Al empezar un nuevo proyecto debemos crear un repositorio en *git*, instalar todo el software necesario y preparar el entorno de desarrollo para que podamos empezar a programar sin encontrarnos con ninguna dificultad a posteriori.

4.1.1.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

- **WBS 1.1** Análisis de requisitos generales
  - **WBS 1.1.1** Estudio de herramientas para invidentes.
  - **WBS 1.1.2** Estudio de los elementos del género *roguelike*.
  - **WBS 1.1.3** Analizar los elementos encontrados.
- **WBS 1.2** Diseño genérico del juego a implementar.
- **WBS 1.3** Búsqueda de bibliotecas que se adapten a nuestros requisitos.
- **WBS 1.4** Creación y configuración del entorno de desarrollo para poder empezar la implementación.

Como se puede aprenciar en la Figura 4.1, para la realización de todas estas tareas se planificaron 65 horas en total. Esta estimación se cumplió sin ningún imprevisto, por lo que el día 30 de junio todas las tareas habían sido realizadas.

4.1.1.2. Qué se ha conseguido en esta iteración

Comenzar un nuevo proyecto e investigar sobre un área que nunca hemos experimentado anteriormente puede ser bastante complicado. En algunas ocasiones la información que nos encontramos puede ser contradictoria o insuficiente, por lo que reconocer lo relevante

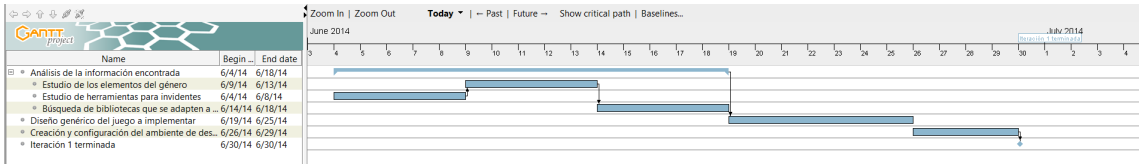


Figura 4.1: Diagrama de Gantt de la primera iteración de la primera etapa

para nuestro caso es muy importante. Asentar las bases sobre las que se desarrollarán las próximas iteraciones siempre es una tarea ardua y complicada pero necesaria. En esta primera iteración hemos sentado estas bases, definiendo lo que debemos realizar y creando un primer boceto de lo que tendremos que cumplir, concretar, diseñar y programar durante el posterior desarrollo.

4.1.1.3. Qué se quiere conseguir en la próxima iteración

Con todos los aspectos básicos definidos, ahora debemos materializarlos. Durante las siguientes iteraciones deberemos empezar a crear el juego en sí, comenzando por la creación de mapas y habitaciones de manera aleatoria.

4.1.2. Segunda iteración: Generación de mapas y habitaciones

Desarrollada entre el 15 de julio y el 24 de septiembre. Se emplearon 7 semanas para terminar esta iteración, trabajando entre 2 y 3 horas durante los días de semana y alrededor de 7 horas al día durante los fines de semana.

Análisis de requisitos de los mapas y habitaciones: Hay mucho material escrito y trabajo realizado sobre la creación de mapas y habitaciones de forma aleatoria y pseudo-aleatoria para el género de los *roguelikes*[NH01]. Con toda esta información recogida y recopilada debemos decidir cómo será nuestra solución en base a diferentes factores como, por ejemplo, si el tamaño de dicho mapa siempre será el mismo o no, cómo y cuántas habitaciones queremos tener en cada mapa y el tipo de aleatoriedad a usar (completamente aleatorio o pseudo-aleatorio). Estas decisiones deben concordar con los objetivos que nos hemos marcado anteriormente.

Diseño de los mapas y las habitaciones: Una vez hayamos creado el análisis de requisitos y tengamos toda la información necesaria sobre la mesa, será hora de crear el diseño. Dicho diseño debe ser fácilmente extensible para que la realización de pequeños cambios no suponga un gran problema.

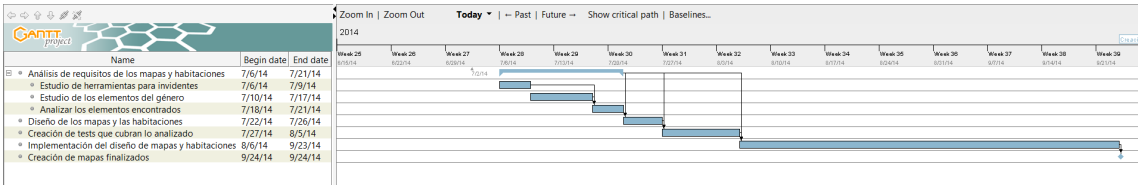


Figura 4.2: Diagrama de Gantt de la segunda iteración de la primera etapa

Creación de tests que cubran lo analizado: Tal y como comentamos anteriormente a la hora de elegir la metodología en el Apartado 3.3, antes de ponernos con la programación crearemos los tests que especifiquen y cumplan lo diseñado para, posteriormente, proceder con la parte de implementación.

Implementación del diseño de mapas y habitaciones: Con el diseño y los tests creados, es hora de dar paso a la implementación del código correspondiente. También deberemos de mostrarlo en la interfaz gráfica.

4.1.2.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

- **WBS 2.1** Análisis de requisitos correspondiente a la generación de mapas y habitaciones.
  - **WBS 2.1.1** Estudio de diferentes algoritmos de creación aleatoria de mapas y habitaciones.
  - **WBS 2.1.2** Decisión sobre la estructura y tamaño a elegir en base al tipo del juego.
- **WBS 2.2** Diseño de los mapas y las habitaciones.
- **WBS 2.3** Creación de tests que cubran lo analizado.
- **WBS 2.4** Implementación del diseño de mapas y habitaciones.

Para la realización de esta segunda iteración se planificaron 125 horas en total, como se puede ver en la Figura 4.2, las cuales fueron insuficientes para terminar todas las tareas asignadas inicialmente. Dado que la estimación inicial no fue correcta y no fuimos capaces de finalizar todo lo necesario, tuvimos que dejar la tarea de representar el mapa en la interfaz gráfica para la siguiente iteración.

4.1.2.2. Qué se ha conseguido en esta iteración

Hemos conseguido generar mapas y habitaciones de manera aleatoria pero todavía no podemos mostrarlos en la interfaz de usuario. Uno de los puntos esenciales del género de los *roguelike* es su aleatoriedad por lo que conseguir que los mapas puedan ser generados de esta forma es un primer gran hito.

4.1.2.3. Qué se quiere conseguir en la próxima iteración

En la siguiente iteración debemos terminar lo que no hemos conseguido hacer en ésta, por lo que representar estos mapas en la interfaz gráfica toma prioridad. A continuación comenzaremos con la creación de objetos. Dichos objetos son otro de los aspectos esenciales del género y poder generarlos de forma sencilla (al igual que mostrarlos en el mapa) es muy importante, por lo que debemos tenerlo disponible lo antes posible.

4.2. Enero 2015 - Mayo 2015

Este periodo comienza el 2 de enero, tras los festivos navideños, y termina el 31 de mayo, en el que da comienzo el periodo de exámenes. Por lo tanto, esta etapa consta de un total de 21 semanas durante las cuales el proyectando estuvo trabajando a tiempo completo y preparando los exámenes, por lo que el tiempo total semanal empleado en el proyecto se redujo a unas 12 horas de media distribuidas irregularmente. La cantidad de horas dedicadas se analizan con más detalle en cada una de las iteraciones mencionadas a continuación.

En total se dedicaron 170 horas a cumplir todas las tareas que teníamos preparadas para este *sprint*.

En este caso nos hemos centrado en continuar lo realizado anteriormente y, en general, seguir con el desarrollo del juego en sí, centrándonos en los objetos, usuarios, enemigos e interfaz gráfica, dejando para los siguientes *sprints* el tema de las gramáticas para la generación de descripciones.

4.2.1. Primera iteración: mapas en IU, análisis, diseño, tests e implementación de los objetos

Esta primera iteración comienza el 2 de enero y termina el 4 de marzo, por lo que consta de unas 9 semanas en total.

Mostrar los mapas y habitaciones en el interfaz de usuario: En la anterior fase implementamos los mapas, pero no se mostraban en la interfaz de usuario. En esta ocasión debemos acabar con esta tarea para terminar con todo lo relacionado con la creación de mapas y poder así continuar con el resto de los cometidos que teníamos planeados desde un principio para este *sprint*.

Análisis de requisitos sobre los objetos: En un *roguelike* los objetos y la interacción con los mismos son primordiales. Debemos estudiar qué objetos vamos a usar y cómo los representaremos en el mapa diseñado en las iteraciones previas.

Crear un diseño simple sobre cómo los objetos interactuarán con el mapa: Crearemos un diseño genérico que nos permita añadir toda clase de objetos de manera sencilla.

Creación de tests que cubran lo analizado: Al igual que antes, es necesario crear primero los tests en vez de empezar con la implementación del diseño directamente. En este caso también añadiremos tests en los que interactúen mapas y objetos para asegurarnos que no vamos a tener ningún error cuando combinemos ambos.

Implementación de los objetos en el juego: Una vez realizado el análisis, el diseño y los tests, podremos implementar la solución encontrada.

4.2.1.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

- **WBS 1.1**Mostrar los mapas y habitaciones en el interfaz de usuario.
- **WBS 1.2** Análisis de requisitos sobre los objetos.
  - **WBS 1.2.1** Buscar información sobre los diferentes tipos de objetos necesarios en el juego.
  - **WBS 1.2.2** Estudiar cómo estos objetos deben interactuar con el mapa.
  - **WBS 1.2.3** Decidir y resumir lo encontrado.
- **WBS 1.3** Crear un diseño simple sobre cómo los objetos interactuarán con el mapa.
- **WBS 1.4** Creación de tests que cubran lo analizado.

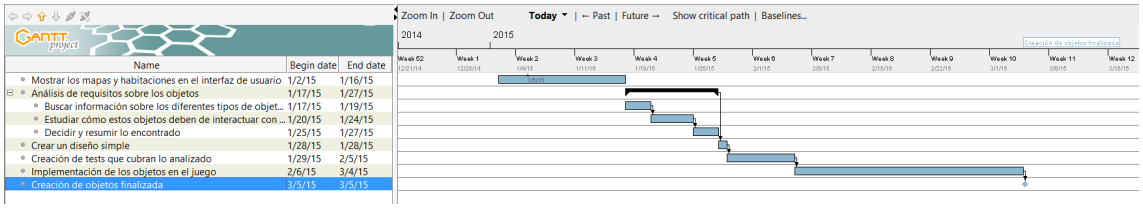


Figura 4.3: Diagrama de Gantt de la primera iteración de la segunda etapa

- **WBS 1.5** Implementación de los objetos en el juego y su asociación con el propio mapa y habitaciones.

Para la realización de la primera iteración del segundo bloque de trabajo se planificaron 55 horas en total. La estimación fue correcta y fue posible cumplirla, por lo que a principios de marzo tuvimos todas estas tareas terminadas. La Figura 4.3 muestra información más detallada sobre la distribución de las mismas.

4.2.1.2. Qué se ha conseguido en esta iteración

Terminar la tarea restante que se había dejado inacabada en el *sprint* anterior y permitir la creación y uso de diferentes objetos (pociones, armas y armaduras) con los que los personajes podrán interactuar posteriormente. También ya podemos representar dichos objetos en el mapa y, además, la creación de nuevos elementos resulta muy sencilla gracias al diseño genérico planteado.

4.2.1.3. Qué se quiere conseguir en la próxima iteración

La próxima iteración tendrá que ver con la creación de los personajes. Éste es uno de los pilares esenciales del género y la última pieza que nos falta para tener los elementos primordiales acabados. Por estos motivos es por los que se decidió abordarlo en la siguiente iteración.

4.2.2. Segunda iteración: Análisis, diseño, tests e implementación de los personajes

La segunda iteración comienza el 5 de marzo y acaba el 23 de mayo, contando con un total de 7 semanas de duración. Durante éstas la cantidad de horas trabajadas fue de 11 a la semana.

Análisis de requisitos sobre personajes: En el juego tendremos un personaje principal, que será controlado por el jugador, y una serie de enemigos con diferentes carac-

terísticas que intentarán acabar con él. El jugador deberá enfrentarse a ellos para lograr su objetivo y conseguir mejores objetos que le ayuden a avanzar en el juego. En esta tarea nos encargaremos de buscar información sobre el tipo de enemigos a crear y diferentes métodos para hacerlo de la forma más genérica posible, dado que ser capaz de crear estos enemigos fácilmente es una característica muy importante de nuestro sistema.

Creación del diseño de los personajes: Una vez hayamos realizado el análisis para hacernos una idea clara de los enemigos y personajes a usar, será hora de crear el diseño. Tal y como hemos comentado en el apartado de análisis, es necesario contar con un diseño fácilmente extendible en el cual aumentar el número de clases generales y tipos concretos de enemigos sea lo más sencillo posible.

Implementación de los tests: Como hasta ahora, antes de empezar con la implementación directamente, debemos de crear tests sobre ello, que nos indicará las características que deben de cumplir y nos ayudará a detectar fallos en la implementación desde el primer momento.

Programación de lo diseñado y analizado previamente: Con el análisis, diseño y tests preparados, podremos realizar la tarea de implementación de la creación de personajes y enemigos. Inicialmente solamente contaremos con tres tipos de enemigos a enfretarse, cada uno con características diferentes. Hemos elegido tres tipos de enemigos diferentes porque es suficiente para mostrar la generalidad y aleatoriedad del sistema y nuestro proyecto está más enfocado al procesamiento de lenguages naturales.

4.2.2.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

- **WBS 2.1** Análisis de requisitos sobre personajes (jugadores y enemigos)
  - **WBS 2.1.1** Buscar información sobre los diferentes tipos de enemigos a los que nos enfrentaremos.
  - **WBS 2.1.2** Estudiar cómo estos enemigos interectuarán con el mapa y los objetos creados anteriormente.
  - **WBS 2.1.3** Decidir y resumir lo encontrado.
- **WBS 2.2** Crear un diseño para crear, fácilmente, nuevos enemigos que aparezcan aleatoriamente en el mapa.

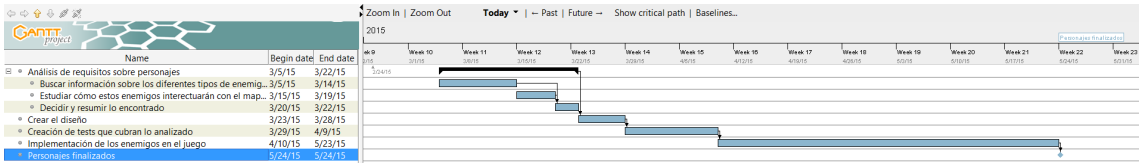


Figura 4.4: Diagrama de Gantt de la segunda iteración de la segunda etapa

- **WBS 2.3** Creación de tests que cubran lo analizado.
- **WBS 2.4** Implementación de los enemigos en el juego y su asociación con el propio mapa, habitaciones y objetos.

Para la realización de todas las tareas de este *sprint* se han asignado 80 horas en total, las cuales fueron suficientes. La Figura 4.4 muestra información más detallada sobre cómo se han dividido el número de horas por cada una de las tareas.

4.2.2.2. Qué se ha conseguido en esta iteración

Al igual que con los objetos, hemos conseguido facilitar la creación de diferentes tipos de personajes, así como la representación del mapa de los mismos en la interfaz gráfica. También hemos creado un personaje principal, que será controlado por el usuario, y diferentes tipos de enemigos a los que enfrentarse: goblins, ratas y dragones. Ambos tipos (el personaje principal y los enemigos) se representan en el mapa de forma diferente para que sea sencilla su identificación.

4.2.2.3. Qué se quiere conseguir en la próxima iteración

Llegados a este punto, el estado del proyecto está bastante avanzado. Al final de este *sprint* tenemos el mapa generado aleatoriamente y en él podemos mostrar objetos y personajes con facilidad, además de que crear nuevos objetos o personajes es ahora algo trivial. El siguiente paso consiste en que los personajes puedan interactuar con los objetos para que así podamos realizar acciones como recoger objetos, equiparlos, tirarlos, etc. También pretendemos que los personajes interactúen entre sí para que, por ejemplo, sea posible que se ataquen entre ellos de diferentes maneras.

4.2.3. Tercera iteración: interacción entre personajes, tests e implementación

Esta última iteración de la sección, y la más corta, comienza el 24 de mayo y acaba el 31 del mismo mes, por lo que su duración es solamente de una semana, aunque al tratase

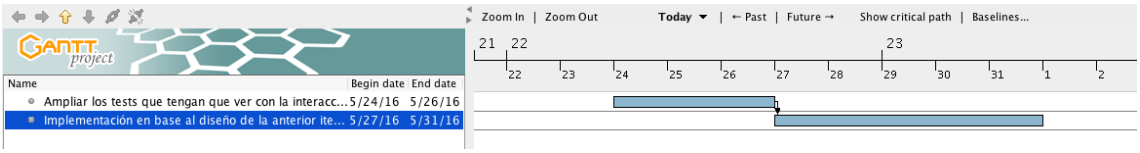


Figura 4.5: Diagrama de Gantt de la tercera iteración de la segunda etapa

de una semana libre, las horas al día que se pudieran dedicar al proyecto fueron alrededor de seis en lugar de una o dos, que es lo que solíamos trabajar en las iteraciones anteriores.

En la anterior iteración de esta sección teníamos el objetivo de crear los personajes y enemigos y que éstos pudiesen ser representados en la interfaz gráfica. En este caso iremos un paso más allá y añadiremos funcionalidades para que dichos personajes puedan interactuar, comunicarse y usar los objetos presentes en e juego. Para ello crearemos un *inventario* para que el personaje pueda almacenarlos y, de este modo, tanto el jugador como los enemigos puedan tener la habilidad de coger objetos (del mapa al inventario), tirarlos (del inventario al mapa), equiparlos (del inventario al personaje) y desequiparlos (del personaje al inventario). Cuando un personaje muera, los objetos se devolverán al mapa para que puedan volver a recogerse.

Ampliar los tests sobre la interacción entre los objetos y personajes: En la iteración anterior fuimos capaces de crear los personajes de forma genérica y en este caso deberemos dotarlos con nuevas funcionalidades que ayuden a la interacción entre los objetos y los personajes, tal y como acabamos de describir.

Implementación en base a los tests, análisis y diseño de la anterior iteración: El diseño y el análisis ya están listos y, una vez tengamos los tests creados, podremos realizar la implementación.

4.2.3.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

- **WBS 3.1** Ampliar los tests que tengan que ver con la interacción entre los objetos y personajes.
- **WBS 3.2** Implementación en base al diseño de la anterior iteración y los nuevos tests.

Al tratase de un pequeño *sprint* corto y disponer de más tiempo del normal, hemos sido capaces de terminarlo sin mayor problema. El diagrama de Gantt puede verse en la Figura

4.5 y en él se indica cómo hemos dividido estas dos tareas y cuánto tiempo dedicamos a cada una de ellas.

4.2.3.2. Qué se ha conseguido en esta iteración

En esta iteración hemos conseguido que personajes y objetos interaccionen de una forma básica, lo que constituye un paso previo que nos permitirá, en un futuro, que los personajes utilicen estos objetos para interaccionar entre ellos mismos.

4.2.3.3. Qué queremos conseguir en la próxima iteración

Aumentar esta interacción entre personajes. Por ejemplo, que un usuario pueda ser capaz de atacar a un personaje y que, dependiendo de los objetos equipados, el daño realizado sea mayor. También dedicaremos algo de tiempo en dar control al usuario, por lo que tendremos que diseñar la manera de que el usuario pueda decidir moverse, atacar, coger un objeto del mapa, etc. y pensar la manera de distribuir todas estas teclas.

4.3. Septiembre 2015 - Abril 2016

Este periodo empieza el 1 de septiembre, tras un paréntesis en verano, y termina el 31 de abril. Durante esta etapa hemos podido dedicar más tiempo al proyecto que en iteraciones anteriores, alrededor de 20 horas a la semana, con lo que los avances en el proyecto también han sido mucho mayores. Asimismo hemos buscado *feedback* sobre lo que hemos realizado por parte de un grupo de jugadores en potencia y, una vez recibido, las modificaciones y mejoras sugeridas fueron aplicadas durante las últimas iteraciones de este *sprint* asegurarnos para, de este modo, asegurarnos de que es bien aceptado por la comunidad de jugadores invidentes.

Tal y como se puede calcular, este periodo consta de un total de 34 semanas, siendo el periodo más largo de todos los que hemos tenido hasta ahora.

En total, 680 horas fueron dedicadas para este sprint.

4.3.1. Primera iteración: Aumentar interacción entre objetos, mapas y personajes, añadir movimiento para el jugador

Esta primera iteración da comienzo el 1 de septiembre y acaba el 5 de octubre. Su duración total es de 5 semanas.

Aumentar la interacción entre objetos, personajes y mapa: Hasta el momento tenemos el mapa, objetos y personajes, pero debemos incrementar su capacidad de interactuar (sobre todo entre los personajes, dado que su interacción con los objetos ya se abordó en iteraciones anteriores) así como sus características. Por ejemplo: añadiendo puertas entre habitaciones que también se muestren en el mapa; que el usuario y enemigos puedan atacarse entre sí (teniendo en cuenta las armaduras y armas que llevan); y la integración del concepto "campo de visión" para que el personaje del jugador solamente pueda ver lo que hay a su alrededor y no la mazmorra completa.

Agregar *listeners* para que el jugador pueda enviar órdenes al juego En este momento el jugador aún no puede hacer nada, dado que no hemos creado un sistema para que el usuario pueda moverse o atacar, por tanto, ahora es el momento de introducir los elementos necesarios para que podamos mover nuestro personaje por el mapa y realizar todas las acciones implementadas anteriormente.

4.3.1.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

- **WBS 1.1** Aumentar la interacción entre objetos, personajes y mapa.
  - **WBS 1.1.1** Crear tests para los elementos posteriores
  - **WBS 1.1.2** Añadir puertas que unan las habitaciones para que el jugador pueda desplazarse de habitación en habitación.
  - **WBS 1.1.3** Añadir la posibilidad de ataque entre personajes.
  - **WBS 1.1.4** Añadir un campo de visión al personaje del jugador para que no sea capaz de ver todo el mapa, sino el área a su alrededor.
- **WBS 1.2** Agregar *listeners* para que el jugador pueda enviar órdenes al juego.

Hemos estimado 100 horas para este *sprint*, las cuales cumplimos sin problemas. Véase la Figura 4.6 para comprobar el diagrama de Gantt y obtener información más detallada sobre la división de tareas.

4.3.1.2. Qué se ha conseguido en esta iteración

Al acabar esta iteración contamos con los elementos y funcionalidad básicas de un juego: el usuario puede moverse por el mapa usando el teclado, coger objetos y combatir enemigos (aunque estos enemigos todavía no tienen inteligencia, por lo que no se mueven).

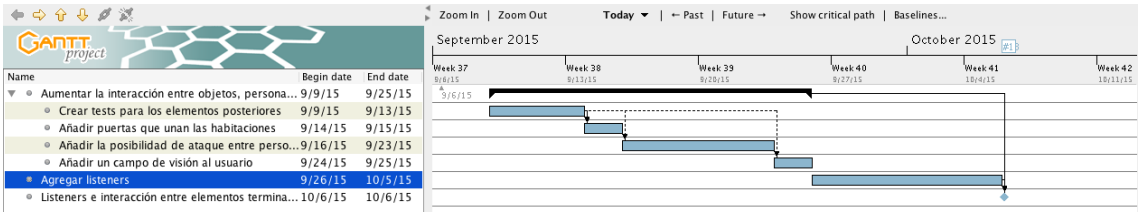


Figura 4.6: Diagrama de Gantt de la primera iteración de la tercera etapa

4.3.1.3. Qué se quiere conseguir en la próxima iteración

En la próxima iteración deberemos completar los elementos básicos del videojuego en sí para poder empezar a diseñar la parte de las gramáticas y la generación automática de descripciones. Para ello tendremos que implementar el sistema de portales (que nos permitirán movernos entre mapas consiguiendo, de esta manera, puntos), dotar de una inteligencia artifial (IA) básica a los enemigos e incorporar la opción de cambiar el color de la interfaz de usuario para que se adapte a los jugadores daltónicos, tal y como ya explicamos en la Sección 2.3.5

4.3.2. Segunda iteración: Diseño e implementación de los portales, IA de los enemigos y accesibilidad para daltónicos

Esta segunda iteración comienza el 6 de octubre y termina el 8 de noviembre, con 4 semanas y media para su completa realización.

Diseño e implementación de los portales: El sistema de portales fue la idea principal que tuvimos para crear un sistema de puntuación y progreso para el usuario. El objetivo es encontrar el portal dentro del mapa y, una vez localizado, un nuevo mapa será generado, con otro portal en una posición diferente a la anterior. De esta manera, la meta del juego se centra en derrotar enemigos (consiguiendo mejores armas y armaduras en el proceso) para desplazarse de mapa en mapa empleando los portales, y así llegar cada vez más lejos, consiguiendo mejor puntuación. Deberemos diseñar e implementar estos portales en esta iteración, dado que dotará de un objetivo al título.

Dotar de inteligencia artificial: Dependiendo del tipo de enemigo al que nos enfrentemos, éste se comportará de forma distinta. Habrá enemigos que serán activos y se lanzan contra el usuario, mientras que otros serán pasivos e inicialmente no querrán atacarlo. Puede ser que en el futuro queramos incrementar la variedad de este tipo de comportamientos. En la Sección [**?**] explicamos cómo se ha desarrollado.

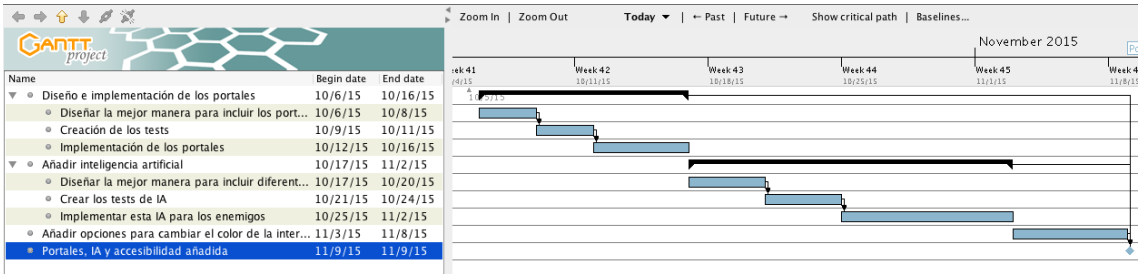


Figura 4.7: Diagrama de Gantt de la segunda iteración de la tercera etapa

Añadir opciones para cambiar el color de la interfaz gráfica: El punto central de este proyecto es la accesibilidad, e incluir una opción para para mejorar la accesibilidad de los jugadores daltónicos a nuestro juego es muy importante.

4.3.2.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

- **2.1** Diseño e implementación de los portales.
  - **2.1.1** Diseñar la mejor manera para incluir los portales en nuestro diseño (en el análisis es algo que se había considerado hacer).
  - **2.1.2** Creación de los tests.
  - **2.1.3** Implementación de los portales.
- **2.2** Dotar de IA a los enemigos.
  - **2.2.1** Diseñar la mejor manera para incluir diferentes tipos de IA.
  - **2.2.2** Crear los tests de IA.
  - **2.2.3** Implementar esta IA para los enemigos.
- **2.3** Añadir opciones para cambiar el color de la interfaz gráfica para usuarios daltónicos

48 horas fueron las que estimamos suficientes para este *sprint*, resultando suficientes para la conclusión del mismo con todas las tareas terminadas. La Figura 4.7 proporciona más información sobre la distribución de las tareas mencionadas.

4.3.2.2. Qué se ha conseguido en esta iteración

Con esta última iteración tenemos la parte básica del juego completada. Hay un objetivo, enemigos con IA que deberemos batir, diferentes objetos que podemos coger, puertas para

conectar distintas habitaciones y varias acciones que nos permitirán interactuar con los elementos que acabamos de mencionar.

4.3.2.3. Qué se quiere conseguir en la próxima iteración

Con el juego en sí completado (o por lo menos la parte primordial), es hora de volver nuestra atención a la parte de generación automática del lenguaje y estudiar su integración con lo ya terminado.

4.3.3. Tercera iteración: Análisis, diseño básico y comienzo de la implementación de las gramáticas

Esta segunda iteración comienza el 9 de noviembre y termina el 31 de diciembre, con 6 semanas y media para su finalización.

Análisis de las gramáticas: Un aspecto clave de nuestro sistema es la generación automática de descripciones de lo que ocurre en el juego para su posterior lectura mediante un lector de pantalla. Buscaremos expresividad y variedad en las descripciones, que se generarán en base a una serie de gramáticas y diccionarios. Esto permitirá, además, traducir nuestro juego a diferentes idiomas de forma relativamente sencilla, pues bastaría con adaptar la gramática y traducir el diccionario empleado. Para construir este motor descriptivo, primero tendremos que investigar cómo podemos hacerlo en nuestro caso de la manera más genérica posible y cómo han solventado este problema otros sistemas similares.

Diseño general sobre la generación del lenguaje: Una vez hemos analizado y decidido lo que debemos realizar, crearemos un diseño lo más simple y adaptable posible para la generación de dichas frases en base a unas gramáticas del lenguaje dadas. El objetivo principal es, en primer lugar, que la adición de nuevas palabras sea lo más fácil posible para que añadir expresividad y variedad sea sencillo y, en segundo lugar, facilitar al máximo la traducción del juego a otros idiomas.

Creación de gramáticas y diccionarios base: Para empezar el desarrollo necesitamos tener una gramática y un diccionario base sobre los cuales se comienza a trabajar, analizando los resultados obtenidos. Empezaremos con gramáticas y diccionarios para inglés, dado que sus restricciones iniciales son más sencillas de tratar que en otros idiomas y el número de usuarios potenciales es mayor, aunque mantendremos en mente la futura extensibilidad a otros idiomas con caracterísctcias morfológicas que compliquen la implementación (como varios géneros para los sustantivos, artículos, etc.).

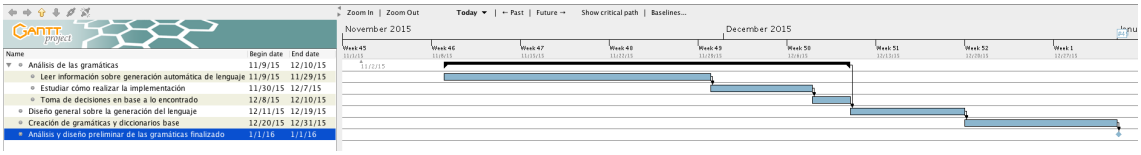


Figura 4.8: Diagrama de Gantt de la tercera iteración de la tercera etapa

4.3.3.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

- **3.1** Análisis de las gramáticas.
  - **3.1.1** Familiarizarse con los principios básicos de generación automática de lenguaje.
  - **3.1.2** Estudiar cómo construir e integrar un sistema de este tipo en nuestro proyecto en base a lo ya existente.
  - **3.1.3** Tomar la decisión en base a lo encontrado y sentar las bases sobre ello.
- **3.2** Diseño general del proceso de la generación del lenguaje y cómo las gramáticas interactuarán con nuestro programa.
- **3.3** Creación de las gramáticas y diccionarios iniciales para tener una base con la que testar nuestra implementación.

Hemos reservado 130 horas para esta iteración y hemos conseguido terminar todas las tareas asignadas a tiempo. En la Figura 4.8 se puede ver el diagrama de Gantt asociado y obtener más información sobre esta iteración.

4.3.3.2. Qué se ha conseguido en esta iteración

Hemos sentado las bases de lo que queremos implementar con las gramáticas en un futuro y hemos creado unas gramáticas y diccionarios iniciales para inglés que iremos ampliando y que servirán de base para el resto de idiomas.

4.3.3.3. Qué se quiere conseguir en la próxima iteración

En la siguiente iteración empezaremos creando los tests para definir cómo las frases automáticas deben generarse e iniciaremos el desarrollo de lo que hemos investigado en esta iteración.

4.3.4. Cuarta iteración: Análisis, diseño y comienzo de la implementación de las gramáticas

Esta segunda iteración comienza el 1 de enero y termina el 14 de enero. Es decir, solamente durará 2 semanas.

Análisis sobre cómo crear las gramáticas NP: Los sintagmas nominales o frases nominales (NP por *noun phrase*) son elementos básicos de cualquier lengua constituidos, en general, por sustantivos acompañados por sus determinantes (ej. artículos) y modificadores (ej. adjetivos). Sin embargo, su estructura, dada por su gramática, no es igual en todos los idiomas, el orden de los elementos cambia entre inglés y castellano o gallego, por ejemplo. Asimismo debemos estudiar qué planteamientos existen que faciliten su implementación de la forma más genérica posible.

Diseño de las gramáticas NP: Las frases de sintagma nominal serán las más usadas dentro de nuestro juego. No solamente serán utilizadas al describir lo que sucede en el juego, sino también a la hora de representar los elementos en la interfaz de usuario. Por ello debemos crearlas de la forma más extensible y accesible que podamos.

Creación de los tests para las gramáticas NP: Al igual que en otras iteraciones, realizamos los tests antes de empezar con ningún tipo de implementación para tener una base sobre cómo las funciones y clases deben de comportase.

Implementación de las gramáticas NP: Con los tests realizados, debemos empezar con la implementación, teniendo en cuenta el formalismo escogido y las restricciones aplicables en base al idioma. En el caso inicial del inglés, se necesitará abordar el caso de la congruencia en númerp, por ejemplo pero, en su momento, con español también tendremos que tener en cuenta el género.

4.3.4.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

WBS 4.1 Análisis sobre el formalismo a emplear para representar las gramáticas NP y cómo construir un motor de generación de textos descriptivos a partir de ellas para su integración en nuestro sistema, todo ello primando la generidad y la extensibilidad.

WBS 4.2 Diseño de las gramáticas NP y del motor descriptivo.

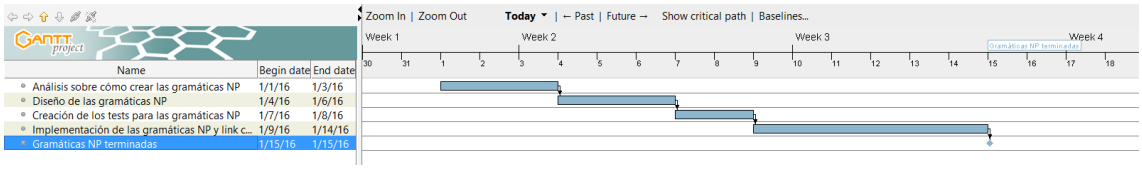


Figura 4.9: Diagrama de Gantt de la cuarta iteración de la tercera etapa

WBS 4.3 Creación de los tests para las gramáticas NP y el motor.

WBS 4.4 Desarrollo de las gramáticas NP, su enlace con el diccionario e implementación del motor.

Hemos reservado 40 horas para esta iteración y terminamos todas las tareas a tiempo. En la Figura 4.9 se puede encontrar más información y para ver el diagrama de Gantt que muestra la distribución de tiempo para estas tareas.

4.3.4.2. Qué se ha conseguido en esta iteración

Al finalizar esta iteración hemos conseguido generar sintagmas nominales destinados a describir la acción del juego en base a las gramáticas y diccionarios dados. De momento solamente está disponible para inglés, pero su ampliación a otros idiomas debería resultar sencillo gracias a nuestro diseño y será algo que se realizará en iteraciones posteriores.

4.3.4.3. Qué queremos conseguir en la próxima iteración

Aumentar el rango de las construcciones sintácticas generables a otro tipo de sinftagmas. De esta forma podremos generar diversos tipos de construcciones que permitirán describir todo lo que ocurre dentro de nuestro juego sin problema.

4.3.5. Quinta iteración: Análisis, diseño e implementación de las gramáticas y frases más complejas

Esta quinta iteración comienza el 15 de enero y termina el 29 de enero. Es decir, tiene 2 semanas y media de duración, un poco más que la anterior.

Análisis sobre cómo crear las gramáticas complejas de forma genérica: Una vez somos capaces de generar sintagmas nominales, pasaremos a combinarlos con otros elementos del lenguaje, tales como verbos para, así, generar estructuras sintácticas más complejas que sean capaces de describir todo lo que realmente deseamos. Deberemos analizar las diferentes formas de hacer esto.

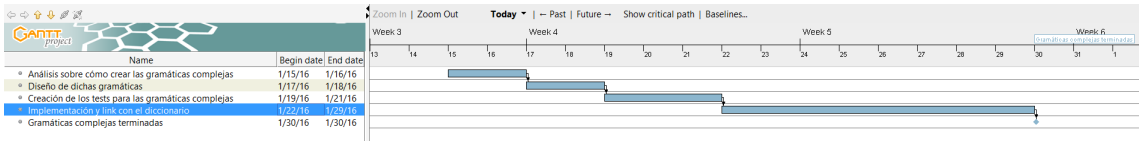


Figura 4.10: Diagrama de Gantt de la quinta iteración de la tercera etapa

Diseño de dichas gramáticas: Una vez hayamos analizado y decidido los pasos que vamos a seguir, nos quedará realizar el diseño del mismo.

Creación de los tests para las gramáticas complejas: Al igual que en las veces anteriores, tendremos que crear los tests antes de empezar con la implementación para estar seguros de que lo que implementamos sea coherente con lo que queremos realizar.

Implementación y enlace con el diccionario: Desarrollar las nuevas gramáticas más complejas y ampliar el motor de generación acorde con ello para que se comuniquen con las gramáticas NP y nos permitan generar frases y oraciones que describan lo que ocurre en el juego.

4.3.5.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

- **WBS 5.1** Análisis sobre cómo crear las gramáticas complejas de forma genérica, extendibles para otros idiomas y que hagan uso de las NP.
- **WBS 5.2** Diseño de dichas gramáticas.
- **WBS 5.3** Creación de los tests para las gramáticas complejas.
- **WBS 5.4** Implementación y enlace con el diccionario.

Hemos reservado 48 horas para esta iteración y la hemos acabado a tiempo. En la Figura 4.10 se muestra el diagrama de Gantt de esta iteración.

4.3.5.2. Qué se ha conseguido en esta iteración

Disponer de gramáticas que generen texto para todas las descripciones que contemplamos actualmente en nuestro proyecto. Estas nuevas gramáticas utilizan lo desarrollado en la iteración anterior para, a su vez, formar frases más complejas. También hemos aumentado el tipo de palabras a usar (ahora incluimos verbos y una mayor cantidad de adjetivos y

sustantivos), por lo que el diccionario del sistema en inglés, va creciendo. En esta iteración todavía no hemos añadido diccionarios o gramáticas para otros idiomas.

4.3.5.3. Qué se quiere conseguir en la próxima iteración

Las frases y oraciones son generadas, pero todavía no las mostramos en ninguna parte de la interfaz gráfica, por lo que el jugador no puede ni verlas ni escucharlas. En el siguiente *sprint* tenemos que añadir esta opción, además de incluir otros idiomas (es decir, crear las gramáticas y diccionarios correspondientes) como el gallego y español.

4.3.6. Sexta iteración: Implementación de las restricciones, gallego y castellano, creación de la interfaz de usuario para con las frases generadas y primer vídeo

La sexta iteración da comienzo el 30 de enero y terminará el 6 de febrero, por lo que solamente tiene 1 semana de desarrollo, una de las más cortas del todo el proyecto, si bien con una jornada de 8 horas al día en la práctica, por lo que en total fueron 56 horas de trabajo en la misma.

Creación de la interfaz de usuario para mostrar las frases generadas al usuario: Hasta ahora somos capaces de generar frases que describan lo que sucede en el juego, pero no mostrarlas. Ahora es el momento de mostrar en una ventana la frase generada (de forma similar a un *popup*).

Implementación de las restricciones para el resto de idiomas: En inglés solamente tenemos que respetar la congruencia en número entre palabras, pero otros idiomas, como el castellano o gallego, tienen otras restricciones adicionales como, por ejemplo, la congruencia en género. Tenemos que tener esto en cuenta e introducirlo en el código.

Añadir gramáticas y diccionarios para gallego y castellano: La idea desde un principio fue la de extender nuestro sistema la mayor cantidad posible de idiomas, por lo que incluir gallego y castellano es importante. Para ello tendremos que "traducir" las gramáticas y diccionarios a ambos idiomas y crear una opción para elegir su uso en un fichero de configuración.

Añadir las teclas necesarias para mostrar las descripciones del inventario y ambiente: El usuario podrá pedir que se genere una frase que describa algo en concreto. Se debería realizar en base a lo que está especificado en el diagrama de casos de uso.

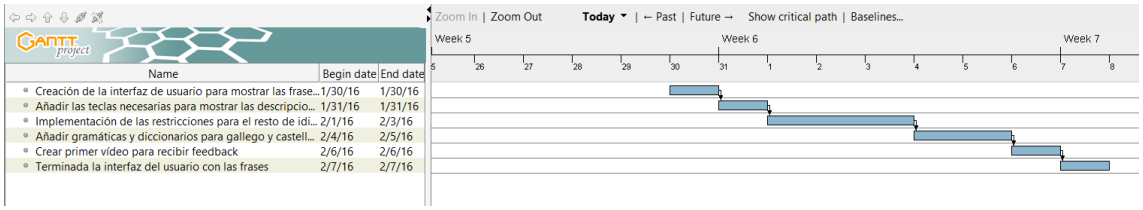


Figura 4.11: Diagrama de Gantt de la sexta iteración de la tercera etapa

Crear primer vídeo para recibir *feedback*: Al terminar esta iteración tendremos el proyecto base completado. Por ello procederemos a grabar un vídeo mostrando el punto en el actual estado de desarollo del sistema con sus funcionalidades y que nos servirá para obtener un primer *feedback* a partir del cual podremos incluir mejoras en futuras iteraciones. A efectos de documentación dicho vídeo aún está disponible en Youtube[1].

4.3.6.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

- **WBS 6.1** Creación de la interfaz de usuario para mostrar las descripciones textuales generadas al usuario.
- **WBS 6.2** Añadir las teclas necesarias para mostrar las descripciones del inventario y ambiente, tal y como está definido en el diagrama de casos de uso.
- **WBS 6.3** Integración en el formalismo de las restricciones para el resto de idiomas.
- **WBS 6.4** Añadir gramáticas y diccionarios para gallego y castellano.
- **WBS 6.4** Crear primer vídeo para recibir *feedback*.

En la Figura 4.11 se muestra el diagrama de Gantt de esta iteración. Hemos reservado, tal y como ya hemos comentado, 56 horas, que fueron suficientes para terminar todas las tareas mencionadas. En dicha figura mostraremos la distribución del tiempo que hemos empleado en cada una de ellas.

4.3.6.2. Qué se ha conseguido en esta iteración

Ampliar la cobertura básica del sistema a dos idiomas adicionales (inglés, español y gallego), permitiendo generar las frases necesarias para que el usuario tenga toda la información necesaria y creando un vídeo que nos ayudará a mostrar el funcionamiento del

[1] https://www.youtube.com/watch?v=RgND1IGZ-68

juego a un primer grupo de jugadores (videntes por ahora) para que podamos mejorarlo en base a sus comentarios.

4.3.6.3. Qué se quiere conseguir en la próxima iteración

Debemos esperar por los comentarios y sugerencias recibidos a raíz del visionado del vídeo y, mientras, mejoraremos diferentes aspectos y características del título realizando alguna de las tareas pendientes que tenemos en el *backlog*.

4.3.7. Séptima iteración: Resolución de bugs detectados y añadidas pequeñas funcionalidades

La séptima iteración empieza el 7 de febrero y acaba el 21 de febrero, con una duración de 2 semanas, volviendo a las 20 horas de trabajo por semana dedicadas al proyecto y 40 horas en total para la iteración. Se ha decidido dar un par de semanas de margen para poder leer y recopilar el *feedback* recibido en el marco de la iteración anterior.

Solucionar *bug* en el que la pantalla no se refresca: En algunas ocasiones y al realizar ciertas acciones, la interfaz gráfica no se actualiza hasta la siguiente acción. Tenemos que solucionar este problema para que lo que se muestre siempre sea lo más nuevo y no haya ningún tipo de retraso.

Cambiar las teclas que usamos por defecto: Las teclas que tenemos por defecto no son del todo intuitivas. Deberemos cambiarlas para que tengan más sentido.

Definir los adjetivos que describen el estado de los personajes: Si, por ejemplo, un enemigo tiene poca vida y el personaje que controla el usuario tiene mucha más vida, el sistema reflejará este hecho a nivel descriptivo calificando al enemigo como "pequeño", "insignificante", etc., mientras que si la situación es al revés, se usará "grande" o "poderoso", de modo que se verá al enemigo de forma distinta dependiendo del contexto.

Añadir opción para que las descripciones sean cualitativas o cuantitativas: Cuando escuchamos una descripción, a veces queremos que los contenidos de dicha descripción sean de un carácter más cuantitativo, es decir, que mencione exactamente la vida o posición en coordenadas de los personajes, por ejemplo. Otras veces deseamos que esto no sea así y que todas las descripciones usen expresiones más cualitativas. Crearemos una opción en la que el usuario será capaz de seleccionar lo que prefiera.

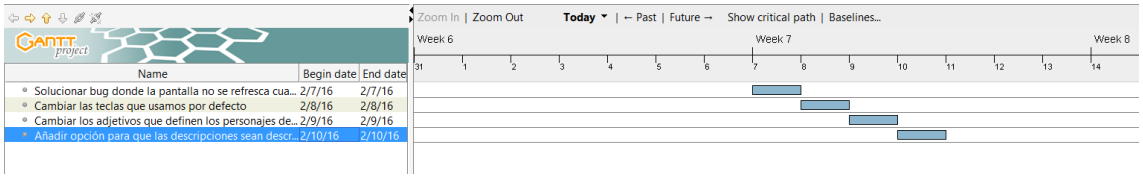


Figura 4.12: Diagrama de Gantt de la séptima iteración de la tercera etapa

4.3.7.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

- **7.1** Solucionar bug en el que la pantalla no se refresca cuando es necesario.
- **7.2** Cambiar las teclas que usamos por defecto.
- **7.3** Cambiar los adjetivos que describen los personajes dependiendo de la vida de dichos personajes.
- **7.4** Añadir opción para que las descripciones sean cualitativas o cuantitativas, dependiendo de lo que el usuario desee.

En la Figura 4.12 se muestra el diagrama de Gantt de esta iteración.

4.3.7.2. Qué se ha conseguido en esta iteración

Mejorar el juego en diferentes aspectos al añadir nuevas funcionalidades que afectarán directamente al jugador y solucionar algunos de los *bugs* que fueron detectados mientras jugábamos.

4.3.7.3. Qué se quiere conseguir en la próxima iteración

El día 19 de febrero es cuando recibimos el *feedback* que presentamos en la anterior iteración, así que crearemos un *sprint* donde podamos analizar e implementar en su caso los cambios y sugerencias que se nos plantean.

4.3.8. Octava iteración: Implementación en base al *feedback* recibido

En esta iteración nos centraremos en realizar los cambios más importantes, fruto del *feedback* recibido a raíz del vídeo. Esta iteración contará solamente con 4 días y 20 horas en total, empezando el 22 y acabando el 26, que es cuando publicaremos un nuevo vídeo con los cambios realizados durante las últimas dos iteraciones.

4.3.8.1. Feedback recibido

Todo este *feedback* es el recibido de mano de mis supervisores que, a su vez, se basan en su propio criterio y experiencia y en lo que otros usuarios les han comentado.

Tener en cuenta la persistencia en el tiempo: Si, por ejemplo, derrotamos a un enemigo, sería una buen adición que las descripciones lo tuvieran en cuenta. De este modo, a la hora de describir de nuevo una sala por la que ya se había pasado y donde se desarrolló un combate, se podría comentar que también se encuentra allí un cadáver.

Cambiar el sistema de salida de las frases generadas: Hasta ahora, cada vez que una frase era generada, tanto a petición del usuario o en base a una acción que ha sucedido en el juego, mostrábamos una nueva ventana con dicho texto, que tendríamos que cerrar en cada ocasión. Esto es un inconveniente para aquellas personas que sí pueden ver y no quieren ser molestadas por este tipo de ventanas. Tampoco es una solución óptima para los usuarios invidentes, dado que no es lo que se suelen encontrar en otros títulos. La solución que se suele plantear en estos casos es emplear una *textarea*, es decir, una ventana aparte donde se vaya almacenando todas las frases generadas, de tal forma que siempre podemos volver a ella cuando queramos y que, del mismo modo, servirá como un *log* de los sucesos acaecidos durante el juego. En el caso de los jugadores invidentes es importante cambiar el "foco" del juego a esta ventana cada vez que una frase sea generada para que el software lector sea capaz de leer lo generado.

Pequeños cambios en los adjetivos usados: En algunas descripciones el sistema usaba adjetivos poco comunes a la hora de calificar ciertos nombres. Tenemos que cambiarlo para que las expresiones resultantes no resulten extrañas.

Adición de niveles y experiencia: Los enemigos, armas y el propio usuario deberían tener niveles para que el juego escale en dificultad y aumente en atractivo. Cada vez que se destruye un enemigo, el jugador ganará una serie de puntos que serán usados para subir niveles y, conscientemente, mejorar sus habilidades a la vez que evoluciona el tipo de enemigos a los que se enfrentarán para que la progresión tenga sentido y la experiencia de juego sea gratificante

Cambio en la aleatoriedad: Hasta ahora, todo lo generado era aleatorio: el tipo de enemigos que nos encontrábamos, los objetos que dejaban atrás éstos al morir, los tesoros etc. Es mejor que esta aleatoriedad venga dada por el nivel del usuario para que el juego escale mejor en dificultad. Esto se explica con mayor profundidad en la Sección: 6.2.2.2.

Mejorar la expresividad: Algunas de las frases que generamos, por ejemplo, tienden a ser bastante repetitivas o cargantes ("el héroe desequipa la espada", "el héroe desequipa la armadura"), en vez de hacer un lenguaje más natural y pulido ("el héroe desequipa la espada y la armadura"). Este tipo de fenómenos han sido tenidos en cuenta.

Darle nombre al héroe: Siempre nos referimos al personaje que controla el usuario como "e héroe" o símplemente "él", lo que resulta impersonal. Podríamos darle un nombre o dejar que el usuario elija para dar una mayor variedad y favorecer la empatía con el personaje.

Mostrar estadísticas al cambiar de mazmorra: Cuando pasamos de una mazmorra a otra usando un portal, podríamos mostrar una serie de estadísticas con los enemigos batidos, la cantidad de experiencia obtenida, el nivel actual del usuario, etc. y, de este modo, dar más sensación de progreso.

Nuevo vídeo Crearemos otro vídeo integrando los cambios que hemos realizado en base a los comentarios recibidos a raíz del anterior. El vídeo sigue disponible en Youtube.[2].

4.3.8.2. Tareas y seguimiento

La descomposición de las tareas que realizaremos este *sprint* es la siguiente:

- **WBS 8.1** Tener en cuenta la persistencia en el tiempo.
- **WBS 8.2** Cambiar el sistema de salida de las frases generadas.
- **WBS 8.3** Pequeños cambios en los adjetivos usados.
- **WBS 8.4** Adición de niveles y experiencia.
- **WBS 8.5** Cambio en la aleatoriedad.
- **WBS 8.6** Mejorar la expresividad.

En la Figura 4.13 se muestra el diagrama de Gantt de esta iteración.

[2] https://www.youtube.com/watch?v=3lS0WFrwOeQ

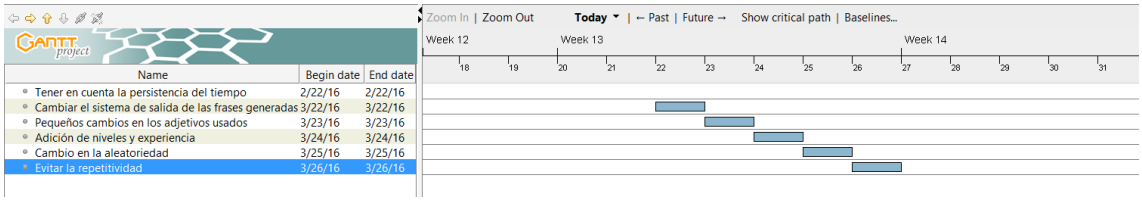


Figura 4.13: Diagrama de Gantt de la octava iteración de la tercera etapa

4.3.8.3. Qué se ha conseguido en esta iteración

Hemos conseguido implementar los cambios más importantes en base a los comentarios recibidos, obteniendo así un producto mucho más sólido y adaptado a lo que la comunidad ha considerado más importante.

4.3.8.4. Qué se quiere conseguir en la próxima iteración

En la próxima iteración, que será la última, tendremos que seguir implementando algunas de las tareas que se encuentran en el *backlog* y debemos de tener en cuenta el *feedback* del resto de usuarios, así como el que recibamos de esta iteración. Las tareas que no vayamos a implementar porque no son completamente necesarias deberán de añadirse al *backlog* para que otros desarrolladores o nosotros mismos podamos tenerlas en cuenta en el futuro.

4.3.9. Novena iteración: Más implementación en base al *feedback* recibido

En esta última iteración daremos los últimos retoques en base al *feedback* recibido de la anterior iteración. Esta última itereación empieza el día 3 de abril, que es cuando recibimos dicho *feedback*, y termina el 13 de abril, fecha que hemos estimado suficiente para terminar las tareas que mencionaremos a continuación. La cantidad de horas usada es la misma que en la anterior iteración, 5 horas al día, por lo que en total trabajaremos 50 horas para finalizar todas las tareas asociadas.

4.3.9.1. Feedback recibido

Añadir el resto de signos de puntuación necesarios: Algunas de las frases son generadas con los signos de puntuación correspondientes, pero todavía existen partes que no tienen estos signos de puntuación, dando lugar a redacciones extrañas.

Incrementar el tamaño de la fuente dentro del área de texto: El texto que usamos en la *textarea* no es muy grande, por lo que podría causar problemas a aquellos

usuarios videntes con dificultad para leer letras pequeñas. Debemos incrementar el tamaño de la fuente.

Generar todas las frases con las gramáticas: En el juego tenemos algunas frases que no usan las gramáticas para ser generadas como, por ejemplo, cuando el personaje del jugador muere. Tenemos que cambiar esto para que se adapte al resto del juego y puedan ser fácilmente traducibles.

Cuando pulsemos una tecla en la *textarea*, el resultado debería de afectar el juego: Cuando pulsamos una tecla para, por ejemplo, desplazar al héroe, mientras el foco está en el área de texto, el foco cambia para el juego, pero la acción correspondiente a la tecla pulsada no es ejecutada. Deberemos solucionar dicho problema.

Cambiar el tamaño y situación de las ventanas: La interfaz del juego consta de dos ventanas: la primera de ellas corresponde a la interfaz gráfica que muestra el juego en sí, mientras que la segunda es donde se encuentran las descripciones generadas por el sistema. Haremos cambios para que desde el inicio ambas se coloquen y adecúen a la pantalla y no se molesten entre sí.

Añadir sonido cuando el jugador se mueve: Uno de los consejos que recibimos de la comunidad es el de agregar algún tipo de sonido cuando movemos al personaje, dado que el usuario no recibe ningún tipo de *feedback* de que la acción ha sido realizada.

Refactorizar el código: Durante el último *sprint* algunos de los módulos contienen código que no es el adecuado. Debemos de refactorizarlo.

4.3.9.2. Tareas y seguimiento

La descomposición de las tareas que realizaremos este *sprint* es la siguiente:

- **WBS 9.1** Añadir el resto de signos de puntuación necesarios.
- **WBS 9.2** Incrementar el tamaño de la fuente dentro del área de texto.
- **WBS 9.3** Generar todas las frases con las gramáticas.
- **WBS 9.4** Cuando pulsemos una tecla en la *textarea*, el resultado debería afectar el juego.
- **WBS 9.5** Cambiar el tamaño y situación de las ventanas.

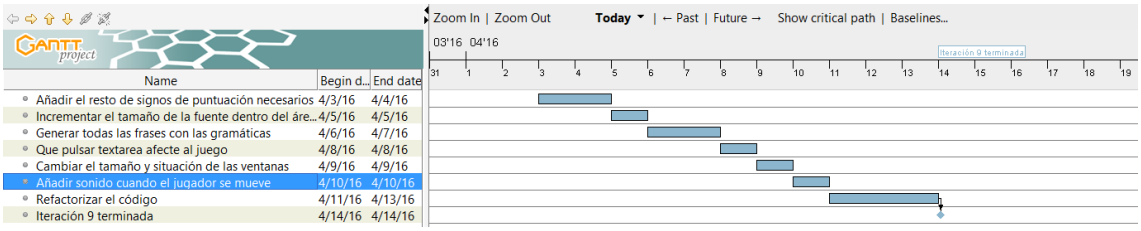


Figura 4.14: Diagrama de Gantt de la novena iteración de la tercera etapa

- **WBS 9.6** Añadir sonido cuando el jugador se mueve.
- **WBS 9.7** Refactorizar el código.

La figura 4.14 se muestra el diagrama de Gantt de esta iteración.

4.3.9.3. Qué se ha conseguido en esta iteración

Terminamos de implementar los puntos más importantes que faltaban o necesitaban revisión, por lo que al final de esta iteración tenemos un juego sólido, extensible y listo para ser jugado y modificado por el resto de la comunidad.

4.3.9.4. Qué queremos conseguir en la próxima iteración

Ésta ha sido la última iteración "oficial" del proyecto. En el futuro el juego seguirá evolucionando con nuevas características y aspectos que vendrán dados tanto por nosotros como por la propia comunidad de usuarios, si bien actualmente ha llegado al punto necesario para considerar esta primera versión como finalizada.

Durante el resto de semanas nos dedicaremos a integrar nuevos idiomas (el holandés podría estar disponible en un futuro cercano) e intentaremos buscar miembros de la comunidad que nos puedan ayudar en dicha tarea; dar a conocer nuestro juego a distintas personas y comunidades online para que lo prueben y valores, documentando todo el *feedback* recibido; y crear la documentación del propio título.

4.4. Coste del proyecto

Durante un periodo de trabajo de 15 meses (casi dos años si no contamos los meses sin realizar grandes cambios en la práctica), hemos empleado 1040 horas en la realización del mismo. El único recurso necesario es el personal, dado que no necesitamos comprar nada de hardware ni ninguna licencia a mayores. Por lo tanto, el coste total del proyecto, suponiendo que el programador trabajara a tiempo completo y cobrará 1200 euros al mes

(con una media de 5 euros la hora, contando fines de semana y festivos), sería de 5200 euros.

Capítulo 5

Análisis de Requisitos globales

En este capítulo explicaremos el proceso de análisis de requisitos llevado a cabo para la elaboración de la aplicación y toda la información recibida por la comunidad, así como las decisiones tomadas en base a la experiencia de otros proyectos similares al nuestro.

5.1. Consultas con la comunidad

A la hora de realizar las consultas con la comunidad hemos decidido contactar con la sede local de la ONCE para recabar información sobre las dificultades que los invidentes se encuentran a la hora de utilizar diferentes dispositivos y programas informáticos. También hemos preguntado a personas daltónicas para que nos aconsejarán desde su experiencia sobre cómo se deberían realizar juegos sin que ellos se vean afectados por su diseño.

Ambas consultas, así como el resultado obtenido de las mismas, se detallaron en la Sección 2.3

5.1.1. Resumen de las peticiones recibidas

Posibilidad de cambiar el color de la interfaz La interfaz gráfica tendrá diferentes colores que diferencien los distintos tipos de enemigos y elementos del juego de cara a los jugadores videntes. Tal y como se menciona en la Sección 5.1.3, debemos tener en cuenta específicamente el caso de los usuarios daltónicos a pesar de que dichos elementos son fácilmente diferenciables.

Variedad en las descripciones automáticas de los elementos del juego Las personas invidentes necesitan tener un *feedback* auditivo para saber qué es lo que está sucediendo y así poder tomar una decisión razonada en base a la situación en la que se encuentran.

Debemos generar estas descripciones de la forma más variada y correcta posible para que no sean repetitivas.

Diferentes idiomas Al ser un videojuego en el cual el lenguaje es esencial, debemos tener en cuenta la posibilidad de incluir otros idiomas.

Utilizar el idioma en nuestro favor Usar elementos de temporalidad o diferentes adjetivos para definir ciertos elementos para que el lenguaje sea parte de la experiencia.

Multiplataforma Tal y como mencionamos en la introducción, nuestro software debe poder ser ejecutado en varios sistemas operativos (Linux, Mac OS, Windows...).

5.1.2. Cómo hemos abordado el problema de accesibilidad para invidentes en nuestro proyecto

En el proyecto hemos creado descripciones para todos los elementos de la pantalla, por lo que el jugador siempre puede saber dónde se encuentra, qué hay a su alrededor, cuáles son sus características, las del enemigo y las de los elementos equipables, etc. También hemos creado una ventana que se encuentra al lado del juego y donde se van guardando de forma ordenada todas las frases generadas, siendo la primera de ellas la última generada y leyéndose automáticamente por el reproductor de pantalla que esté siendo usado en ese momento (en nuestro caso, *NVDA*). De esta forma, una persona con visión siempre puede leer dicha frase mientras que el jugador invidente puede volver a escucharla las veces que quiera. Además, dicho listado funciona a modo de *log*, por lo que el jugador también puede recordar lo sucedido anteriormente gracias a que estos mensajes permanecen disponibles en la pantalla y pueden ser leidos por el software lector. La Figura 5.1 muestra la captura de pantalla de esta ventana para una sesión de juego.

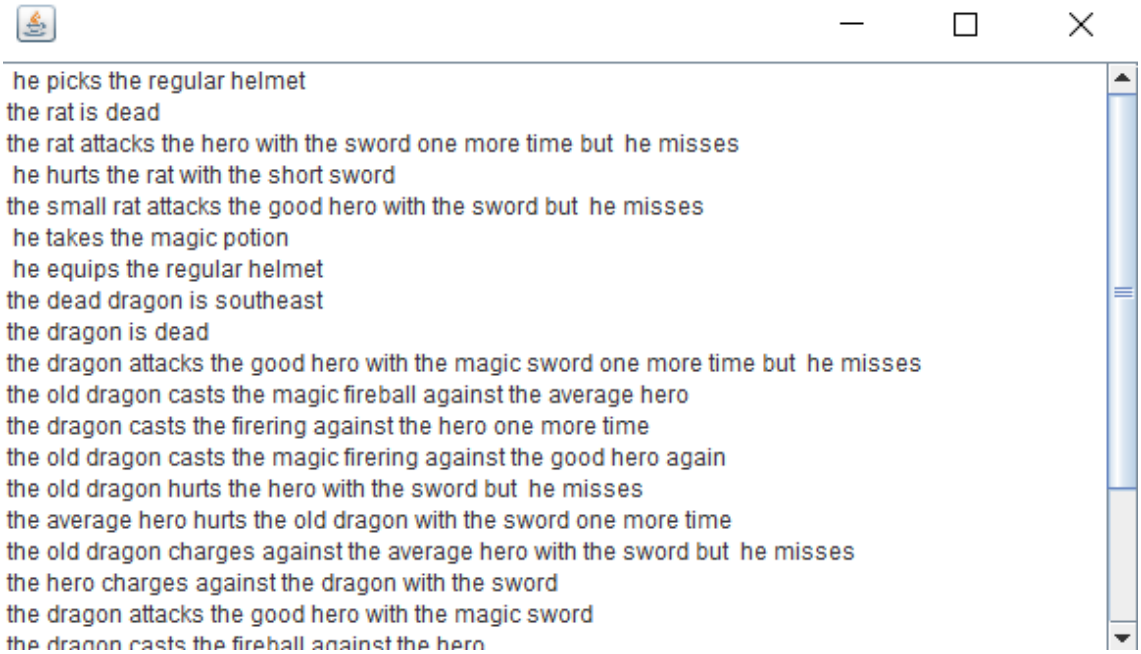


Figura 5.1: Captura de pantalla del área donde mostramos las frases generadas por nuestro juego para invidentes

5.1.3. Cómo hemos abordado el problema de accesibilidad para daltónicos en nuestro proyecto

En nuestro caso, al haber preguntado de antemano a los potenciales usuarios, siempre tuvimos la idea de crear la interfaz gráfica con soporte para daltónicos en mente.

De este modo, todos los elementos del juego que se encuentran en la interfaz gráfica son distinguibles entre sí gracias al uso de caracteres completamente diferentes por lo que, incluso aunque todos los colores fueran iguales, sería sencillo identificar cada elemento sólo por su forma en vez de por su color.

De todas maneras, hemos creado una opción que cambia la paleta de colores a utilizar y facilita su visualización para aquellos usuarios que sufren daltonismo.

5.2. Análisis de los elementos del juego

Al ser un *roguelike*, debemos incluir los elementos característicos del género: aleatoriedad, dificultad, progreso, etc. Los detalles acerca de los mismos, así como la forma en la que han sido introducidos en el proyecto han sido ya comentados en la Sección 2.2.

5.3. Requisitos del aplicativo

Con toda la información obtenida y analizada, creamos una lista con los casos de uso que nuestro proyecto debe cumplir y cuyo diagrama mostramos en la Figura 5.1:

- **Movimiento** Un usuario siempre será capaz de moverse con su personaje a una posición válida dentro de la habitación donde se encuentra.
- **Ataque normal** Cuando el personaje está en la misma posición que un enemigo, éste será capaz de atacar cuerpo a cuerpo.
- **Ataque mágico** Cuando el personaje está dentro de una distancia determinada de un enemigo, aquél será capaz de realizar un ataque mágico, siempre y cuando tenga sufiencience maná para el mismo. Si el enemigo se encuentra demasiado lejos, el ataque mágico se realizará, pero sin afectar a ningún enemigo (por lo que el usuario perderá maná).
- **Coger elemento** En las habitaciones del mapa habrá elementos que se pueden recolectar, así como enemigos que soltarán diferentes objetos.
- **Equipar elemento** Poder equipar a nuestro personaje con un objeto que está en el inventario, siempre y cuando no tengamos un objeto del mismo tipo ya equipado.
- **Desequipar elemento** De la misma manera, podremos desequipar un objeto que tenemos equipado mientras tengamos espacio en el inventario.
- **Tirar elemento** En algunas ocasiones el jugador podría encontrarse con que no dispone de espacio suficiente en el inventario para almacenar nuevos elementos, por lo que podrá arrojar objetos al suelo para hacer hueco a aquellos nuevos que queramos recoger.
- **Descripciones** Durante el transcurso del juego podremos generar diferentes descripciones de lo que ocurre en el juego dependiendo de lo que queramos saber. Por ejemplo: lo que el personaje del jugador tiene en el inventario, las posiciones a las que nos podemos mover, la descripciones de los enemigos a los que nos enfrentamos, las estadísticas del héroe y de los enemigos, etc.
- **Activación de descripciones numéricas** Algunas de estas descripciones corresponden a posiciones o características de los enemigos que el usuario podría querer escuchar como valor numérico (por ejemplo la cantidad de vida de un monstruo) o mediante expresiones del lenguage (“mucha”, “poca”, “bastante”, etc.), dependiendo de lo que prefiera.

- **Cambio de colores** Para usuarios daltónicos hemos incluido una opción para cambiar los colores de la interfaz gráfica.

Asimismo, al realizar algunas de estas acciones (coger, equipar, desequipar y tirar un objeto, moverse y atacar), un turno será consumido. Al consumir el jugador su turno, pasará entonces el turno a los enemigos, que realizarán sus acciones como, or ejemplo, acercarse o atacar al jugador.

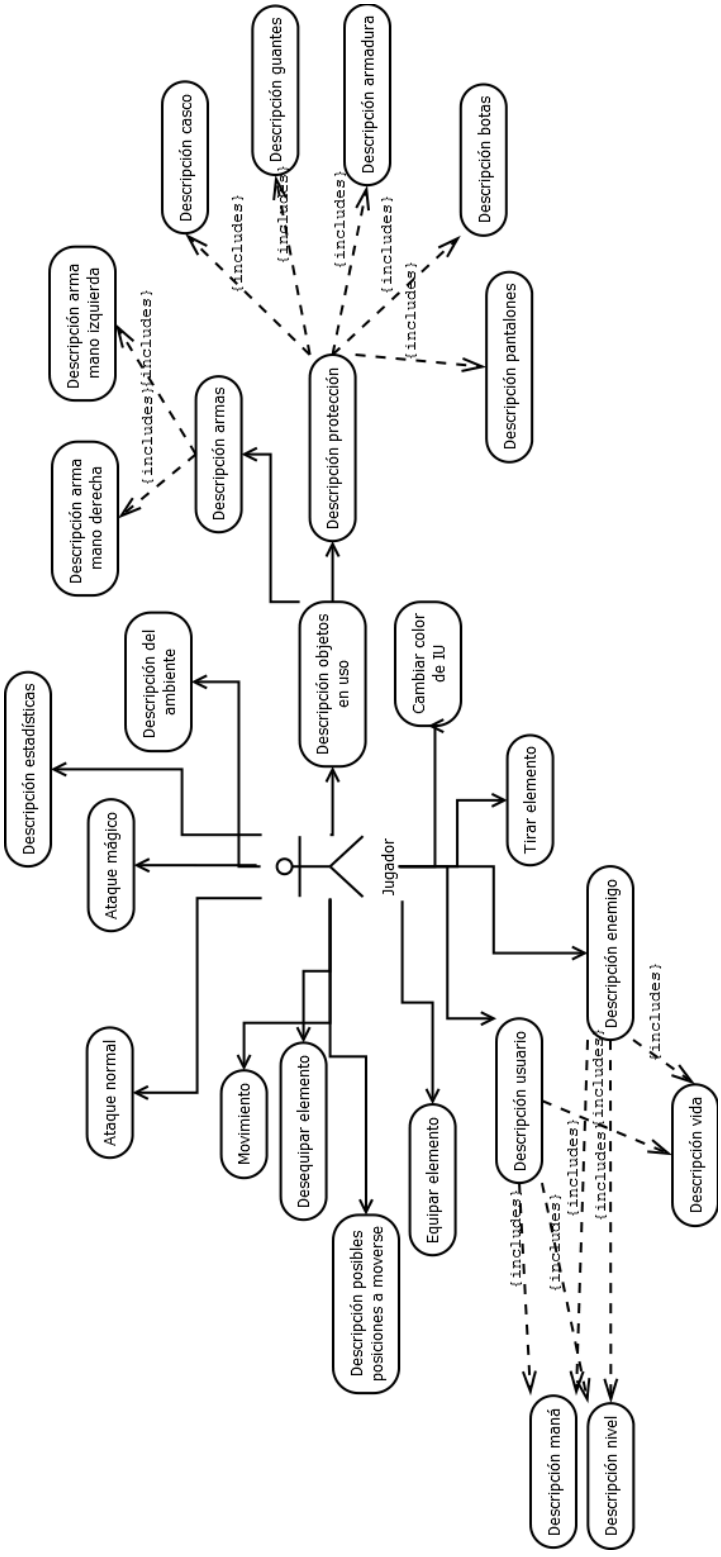


Figura 5.2: Diagrama UML de casos de uso del proyecto

Capítulo 6

Diseño e Implementación

En este capítulo mostraremos los detalles del diseño e implementación de diferentes partes del juego.

6.1. Gramáticas y frases

La parte correspondiente a la definición de gramáticas y generación automática de frases es la más relevante del proyecto y lo que la diferencia del resto de juegos de similares características. Por ello consideramos esencial explicar su funcionamiento.

Tal y como se muestra en la Figura 6.1, la parte esencial de este módulo consta de seis clases, cuyo funcionamiento detallamos a continuación.

6.1.1. Explicación general del funcionamiento

Tenemos dos clases (`GrammarSelectorS` y `GrammarSelectorNP`) que se encargan de generar las frases en base al nombre o nombres sobre los que queremos obtener información. Esta frase se generará en base a gramáticas y diccionarios dados. La clase `GrammarSelectorNP` crea sintagmas nominales mientras que `GrammarSelectorS` crea, a su vez, frases usando dichos sintagmas nominales. Ambas usan diccionarios y gramáticas para el idioma concreto especificado en el archivo de configuración *language.properties* (inglés en este caso):

```
language=EN
```

Parte de la dificultad de generar estas frases está en que los elementos de la misma deben ser congruentes entre sí en base a las restricciones del idioma en cuestión. Por ejemplo,

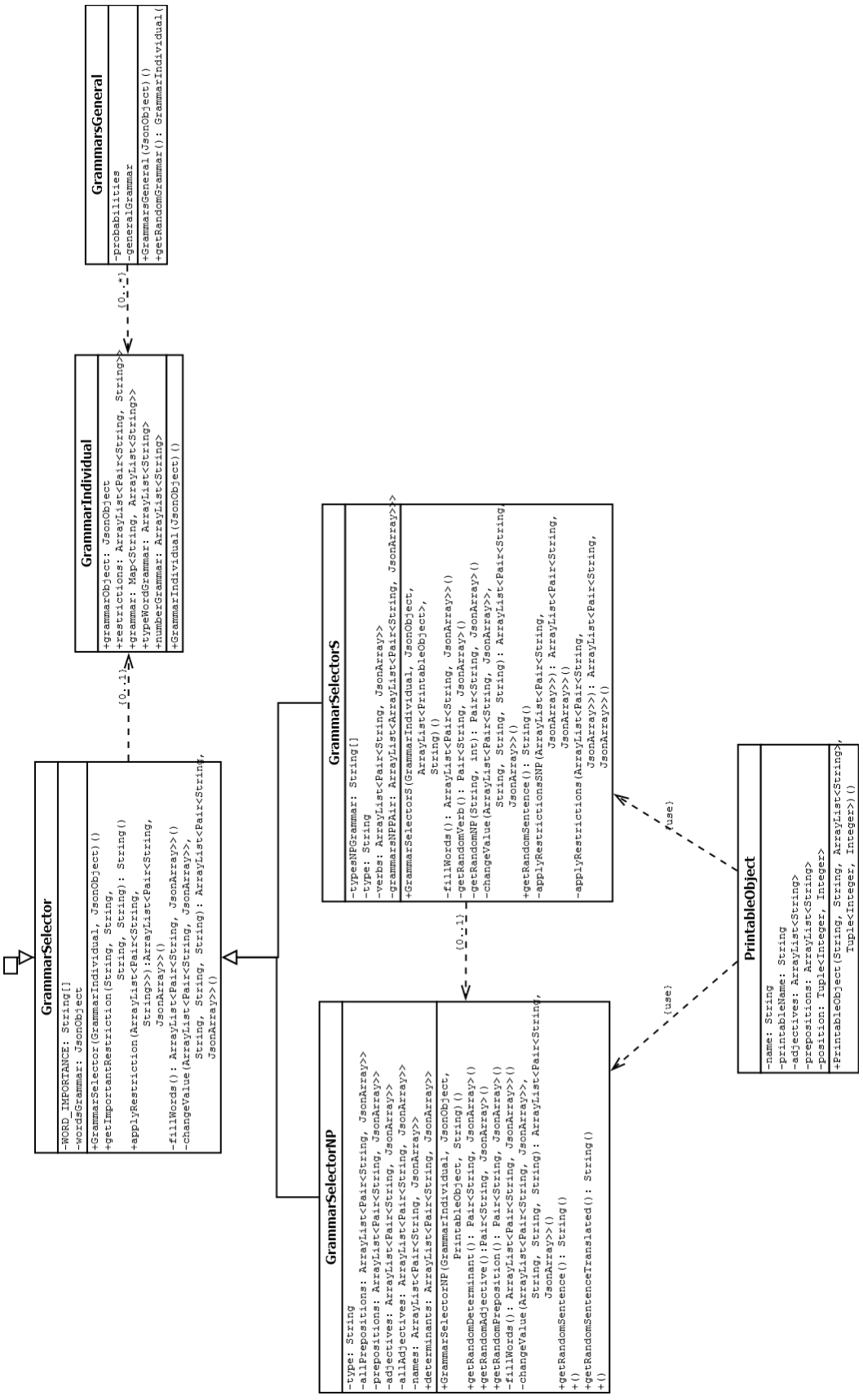


Figura 6.1: Diagrama de clases de las gramáticas

en el caso del español, los nombres, adjetivos y determinantes de una frase nominal deben coincidir en género y número, mientras que en inglés el género no es tenido en cuenta y, en el caso de los adjetivos, tampoco el número. Este tipo de restricciones vienen dadas por las gramáticas, que están especificadas en archivos JSON y que se comentarán más adelante.

Una vez detectemos que dos de las palabras no coinciden, cambiaremos la flexión de la palabra de en base al tipo de la incongruencia (género o número). Por ejemplo, un adjetivo o determinante respecto a un sustantivo.

```
if (toChange.equals(value1)) {
    changeToValue = JSONParsing.getElement(restrictions1, type
        + "opposite");
    typeChangeToValue = typeFirstRestriction;

} else {
    changeToValue = JSONParsing.getElement(restrictions2, type
        + "opposite");
    typeChangeToValue = typeSecondRestriction;
}
this.changeValue(sentenceArray, toChange, changeToValue,
   typeChangeToValue);
```

De esta manera, siempre que haya una discordancia entre algunos de los contribuyentes de una frase, iteraremos entre sus elementos ella buscando una solución hasta que sea coherente, cambiando sucesivamente la flexión de las palabras de menos rango dentro de dicha frase para que se adapten al resto.

La clase `GeneralGrammar`, por su parte, obtiene la información de varias gramáticas, ya que es así como vienen dadas en el fichero, puesto que para describir una misma situación puede haber varias gramáticas aplicables válidas, lo que concuerda con el concepto de *variedad lingüística* del lenguaje humano, es decir, cómo nuestro concepto permite expresar una misma idea o mensaje de diferentes formas. Por ejemplo, "el héroe ataca al dragón" y "él se lanza contra el dragón con la espada mágica". Para ello la clase `GeneralGrammar`, dispone de una función que devuelve una gramática individual (de la clase `GrammarIndividual`) de todas las disponibles para cierto caso y es la que usaremos a la hora de generar la frase en sí. Es decir, cuando queremos generar una frase para describir una situación en concreto (por ejemplo cuando un personaje ataca a otro), seleccionaremos una de las gramáticas de todas las que estén disponibles para este tipo de encuentros en el idioma actual (esto lo realiza la clase `GrammarsGeneral`) y lue-

go usamos esa gramática en particular (clase `GrammarIndividual`). De esta forma no siempre usaremos la misma para describir un mismo evento (o similar), aumentando la expresividad del sistema y evitando así resultar cargantes.

La clase `PrintableObject` es una superclase abstracta de la que heredan todos los objetos que pueden ser representados en el juego. Esta clase se encarga de obligar que dichos objetos tengan ciertas variables como el nombre o la capacidad de traducir su nombre a diferentes idiomas, así como una función que devuelve la posición en la que se encuentra en base al usuario y que emplearemos a la hora de describir lo que hay alrededor del personaje.

6.1.2. *Input* de gramáticas y diccionario

Tanto las gramáticas como los diccionarios empleados por el sistema son almacenados como ficheros JSON que el usuario puede modificar o intercambiar, lo que afectará al resultado de las frases generadas.

El sistema requiere tres archivos de entrada para un idioma dado, dos correspondientes a las gramáticas y el tercero correspondiente al diccionario. En el caso de las gramáticas, uno es empleado para generar los sintagmas nominales, mientras que con el otro generamos expresiones más complejas, en su mayor parte a partir de estos sintagmas nominales.

6.1.2.1. Gramáticas: Sintagmas nominales

A continuación mostramos un ejemplo de gramática generadora de sintagma nominal en inglés correspondiente a una estructura DETERMINANTE + ADJETIVO + NOMBRE:

```
"DETADJN": {
    "GM_1": {
        "S":
            [
            {"DET_1": ""},
            {"ADJ_1": ""},
            {"N_1": ""}
        ],
        "restrictions": [
            {"DET_1.num": "N_1.num"},
            {"N_1.num": "ADJ_1.num"}
        ]
    }
}
```

En estas gramáticas especificamos, primero, el nombre del grupo de gramáticas (en este caso, `DETADJN`). Este identificador es usado por el programa y ayudará al traductor del sistema a entender el tipo que son.

El siguiente identificador es el nombre de la gramática en particular. Éste no es empleado en el programa, pero su uso es necesario para poder definir varias de ellas sobre el mismo árbol. Por ejemplo, si queremos más gramáticas de tipo `DETADJN`, necesitaremos que se especifique un nombre para cada una de ellas.

A continuación viene la definición de la gramática en sí. Primero nos encontramos con el apartado `S`, que es donde se almacenará su definición; y luego el apartado `restrictions` que, como su nombre indica, es donde especificaremos las restricciones aplicables al anterior.

En el caso de nuestro ejemplo, en la parte de la gramática tenemos, en este caso, `DET_1`, que es el primer determinante (y único) de la gramática; `ADJ_1`, el primer y único adjetivo que tiene dicha gramática y `N_1`, que es el nombre o sustantivo. En las restricciones detallamos las partes que tienen que coincidir. En este caso solamente tenemos que el determinante, nombre y adjetivo tienen que ser iguales (congruentes) en número. Es decir, que "the red swords" no sería generada, sino que generaría "the red sword".

Este primer era para inglés, cuyas restricciones son más sencillas que en otros idiomas como el español o el gallego, donde los nombres, determinantes y adjetivos no solamente tienen que coincidir en número, sino también en género. De este modo, la gramática equivalente para el español (y también gallego) sería la siguiente:

```
"DETADJN": {
    "GM_1": {
        "S":
            [
            {"DET_1": ""},
            {"N_1": ""},
            {"ADJ_1": ""}
        ],
        "restrictions": [
            {"DET_1.num": "N_1.num"},
            {"N_1.num": "ADJ_1.num"},
            {"DET_1.gen": "N_1.gen"},
            {"N_1.gen": "ADJ_1.gen"}
        ]
    }
```

```
}
```

Nótese que no solamente hemos añadido el género a las restricciones (para que no podamos generar frases como “la espada rojo” o “el espada roja”), sino que también hemos cambiado el orden del nombre y del adjetivo para que correspondan a la estructura típica en español (y gallego).

A la hora de cambiar la flexión de las palabras que componen la frase dispondremos de un array donde ordenar las palabras por rango de importancia. Un sustantivo es más importante, o tiene mayor rango que un determinante en cuanto a que el sustantivo, como núcleo de la frase nominal, es el que determina el género y número del sintagma con el que deben de coincidir determinantes y adjetivos. De este modo, si alguna de las restricciones no se cumple entre de estos dos elementos, el determinante será el que cambiará para adaptarse al género y número del nombre.

De esta manera podremos adaptar las gramáticas a diferentes idiomas de una forma sencilla y sin necesidad de tocar nada de código.

6.1.2.2. Gramáticas: Estructuras complejas

Las gramáticas para la generación de este tipo de estructuras también pueden hacer a su vez uso de aquellas gramáticas de sintagmas nominales. En este caso, a modo de ejemplo mostramos una gramática correspondiente a la descripción (sencilla) de un ataque:

```
"ATTACK": {
  "S1": {
      "S": [
          {"DETADJN_1": ""},
          {"V_1": ""},
          {"DETADJN_2": ""},
          {"SIMPLEPREP_1": ""}
      ],
      "restrictions": [
          {"DETADJN_1.num": "V_1.num"}
      ]
  },
  "S2": {
      "S": [
          {"GENERAL_1": ""},
          {"V_1": ""},
          {"SIMPLE_2": ""},
```

```
            {"SIMPLEPREP_1": ""}
        ],
        "restrictions": [
            {"GENERAL_1.num": "V_1.num"}
        ]
    }
}
```

La estructura de esta gramática es la ya mencionada anteriormente. En este caso vemos que dentro de `ATTACK` disponemos de dos gramáticas distintas (éste es solamente un ejemplo, en el juego tenemos muchas más disponibles) y éstas llaman a sintagmas nominales como el anterior. Con la primera gramática podríamos generar una frase del estilo "The mighty dragon attacks the brave hero with the sword". En este ejemplo cabe destacar que las restricciones se pueden definir no solamente entre palabras, pero también entre sintagmas nominales (siempre y cuando sea en comparación con una palabra que pueda ser cambiante, en este caso el verbo) y que los sintagmas nominales que son creados dentro de esta gramática tienen sus propias restricciones, por lo que siempre serán generadas de manera correcta.

6.1.2.3. Diccionarios

Al igual que las gramáticas, los diccionarios también están divididos por idiomas y son archivos JSON. Éste es un ejemplo de parte de un diccionario para inglés:

```
"DET": {
    "the": [
        {"num": ""},
        {"translation": "the"},
        {"numopposite": "the"},
        {"genopposite": ""}
        {"gender": ""}
    ]
},
```

El primer elemento, `DET`, especifica el tipo de palabra que es (determinante). El siguiente elemento especifica la palabra en sí (`the`). El resto de elementos estarán en un array de pares (CLAVE, VALOR), aunque en este caso no importa el orden. La clave del primer elemento en esta lista es `num` correspondiendo al número del término en cuestión, es decir, si el elemento es singular o plural (para esta palabra esta información es innecesaria,

por eso es un string vacío) mientras que en `numopposite` guardaremos la flexión de la palabra en el número contrario (si la palabra es singular, entonces guardaremos la palabra en plural). El segundo es `translation`. Todas las palabras entre los diccionarios serán las mismas o, por lo menos, las que están definidas en inglés también deben estarlo en otros idiomas. En este valor almacenaremos la traducción de esta palabra al idioma del diccionario que estaremos definiendo. Los elementos `gender` y `genopposite`, al igual que con `num` y `numopposite`, almacenan el género de la palabra y la palabra de género contrario.

Retomando nuestro ejemplo anterior, con los determinantes en inglés no hay mucho cambio, pero en español sí que es bastante diferente:

```
"DET": {
        "el": [
            {"num": "sing"},
            {"translation": "el"},
            {"numopposite": "los"},
            {"genopposite": "la"},
            {"gen": "mas"}
        ],
        "la": [
            {"num": "sing"},
            {"translation": "la"},
            {"numopposite": "las"},
            {"genopposite": "the"},
            {"gen": "fem"}
        ],
        "los": [
            {"num": "plural"},
            {"translation": "los"},
            {"numopposite": "the"},
            {"genopposite": "las"},
            {"gen": "mas"}
        ],
        "las": [
            {"num": "plural"},
            {"translation": "las"},
            {"numopposite": "la"},
            {"genopposite": "los"},
            {"gen": "fem"}
```

```
            ]
        },
```

En este caso vemos que `numopposite` y `genopposite` apuntan ahora a `la` y `los`, que corresponden a la flexión en número y género, respectivamente. De esta forma obtendremos el término que queramos siempre y cuando el programa precise obtener un tipo de palabra concreta en base a las restricciones que contenga la gramática.

6.2. Otras partes del sistema

Las gramáticas juegan un papel muy importante en el proyecto, pero hay otras partes relevantes del sistema que merecen una atención especial. Aquí mencionamos las más importantes:

6.2.1. Interfaz de usuario para el texto generado

Para que nuestro sistema pueda aprovecharse del software de lectura de pantalla empleado por usuarios invidentes, no basta simplemente con que saque el texto por pantalla. Debe de hacerse de forma que el software de lectura pueda leerlo, lo que no es del todo trivial dado que hay que tener en cuenta diferentes aspectos para que el lector lea lo que queramos y no otras partes que no nos interesan. Más información sobre este tema se encuentra disponible en el Apéndice B de esta memoria.

6.2.1.1. Primer planteamiento: *pop-ups*

La forma más sencilla de mostrar este texto es mostrar un *pop-up* cada vez que una frase es generada. De esta forma el usuario se enterará inmediatamente de lo que está sucediendo y, en caso de que haya una persona vidente a su lado, ésta también podrá leer la frase que se ha generado. Además, la frase no desaparecerá hasta que se presione la tecla de *intro* o se haga click en *Aceptar*, por lo que es muy sencillo reproducirla la cantidad de veces que se desee.

La Figura 6.3 muestra un ejemplo de uso sencillo para esta implementación.

6.2.1.2. Segundo planteamiento: *TextArea*

En base al *feedback* recibido decidimos abandonar esta prima solución por tres razones. La primera es que resulta desconcertante para el jugador vidente, dado que, en su caso, no es necesario. Esto se podría solventar añadiendo una opción que permita activar o

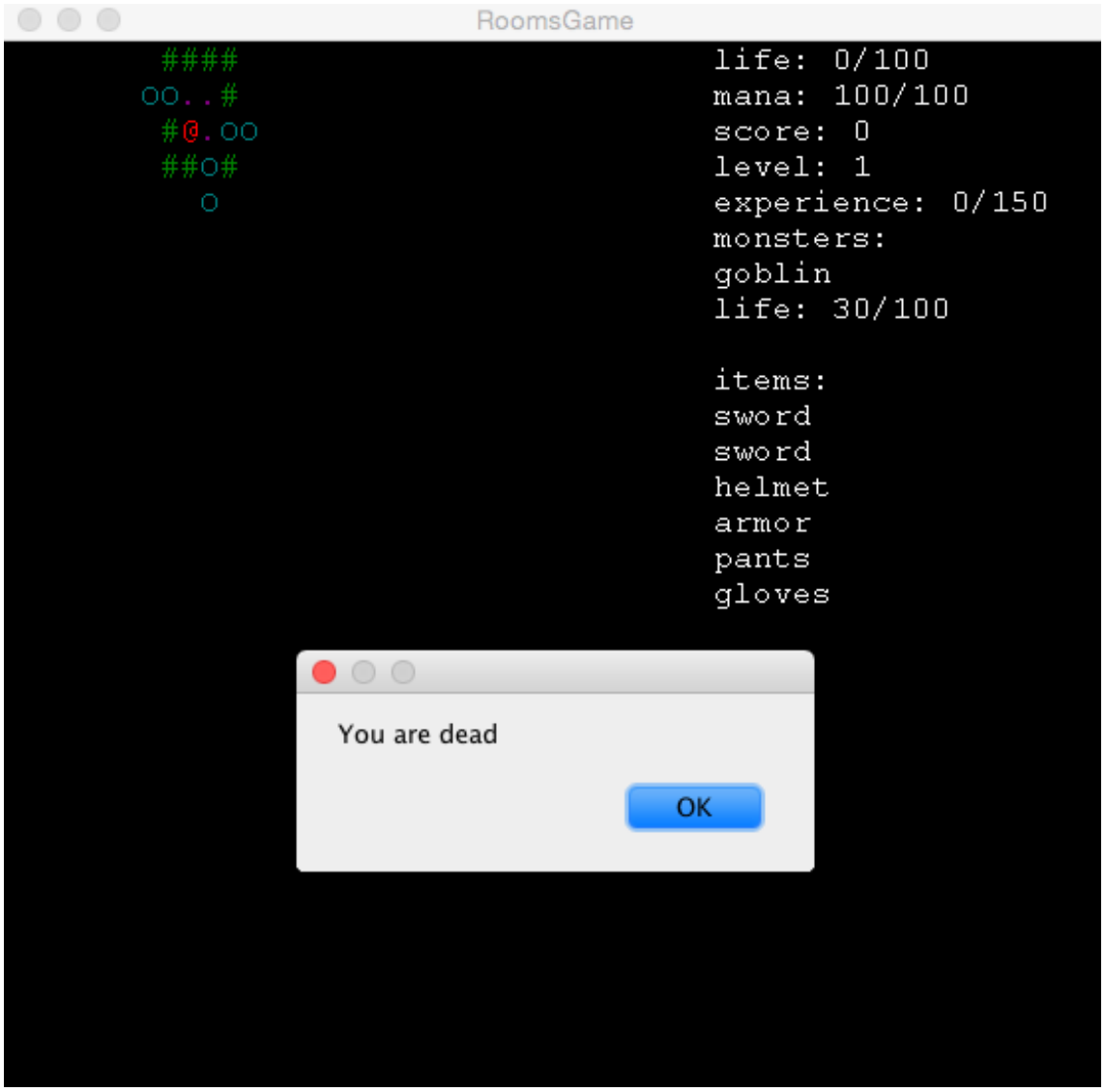


Figura 6.2: Versión preliminar de la interfaz de usuario para el texto generado basada en el uso de *pop-ups*

desactivar esta función, aunque no es la mejor solución para este problema. La segunda razón es que estos *pop-ups* no son un mecanismo óptimo para mostrar salidas rutinarias, como en nuestro caso, pues rompe el "flujo de trabajo". La tercera y última razón es que los usuarios invidentes no suelen hacer uso de elementos de este tipo, sino que están más acostumbrados a emplear áreas de texto donde se almacenan las últimas frases generadas por el programa en cuestión.

En base a estas ideas decidimos crear un área de texto que permanecerá abierta siempre y cuando el jugador no decida cerrarla, (permitiendo a los usuarios videntes prescindir de ella). En el momento en que sucede algo en el juego que requiere generar una descripción, dicha descripción será enviada al área de texto, el "foco" de las aplicaciones cambiará para que se sitúe en esa ventana y el lector de pantalla que estamos usando leerá dicha descripción. Mientras generemos frases el "foco" seguirá en ese área de texto, si bien las teclas que pulsemos seguirán actuando sobre el juego para evitar romper el flujo de trabajo). El foco solamente volverá al juego cuando pulsemos una tecla que no genere una nueva frase. De esta forma solucionamos todos los inconvenientes que los *pop-ups* causaban a los usuarios, mejorando la interfaz gráfica y la accesibilidad del proyecto.

En la Figura 6.3 se puede encontrar una captura de pantalla donde se muestra un ejemplo de uso para esta nueva implementación.

6.2.2. Generación aleatoria de mapas y elementos

Como ya se indicó en la Sección 2.2.1, la generación aleatoria de los mapas y de los elementos dentro de los mismos es una de las características que un juego de género *roguelike*. A continuación explicaremos las decisiones tomadas a este respecto.

A la hora de generar el mapa en sí tenemos en cuenta tres elementos fundamentales: mapas, habitaciones y puertas. Cada mapa generado consta de un conjunto de habitaciones (que vienen limitadas por sus propias paredes), cuyo tamaño y forma cambia de manera aleatoria. Cada puerta une dos habitaciones, asegurándonos que ninguna de las estancias queda sin la posibilidad de ser alcanzable por el jugador. También se han introducido columnas que se encuentran dentro de cada habitación para que tanto el jugador que controla el usuario como los enemigos tengan que esquivarlas y puedan usarlas en su favor en los combates. Hay que tener en cuenta que ningún personaje puede moverse a una posición donde se encuentra una columna por lo que, a la hora de generarlas, se comprueba que no bloquean el acceso a la puerta o, si lo hacen, que haya otra entrada por la que se pueda acceder a dicha estancia.

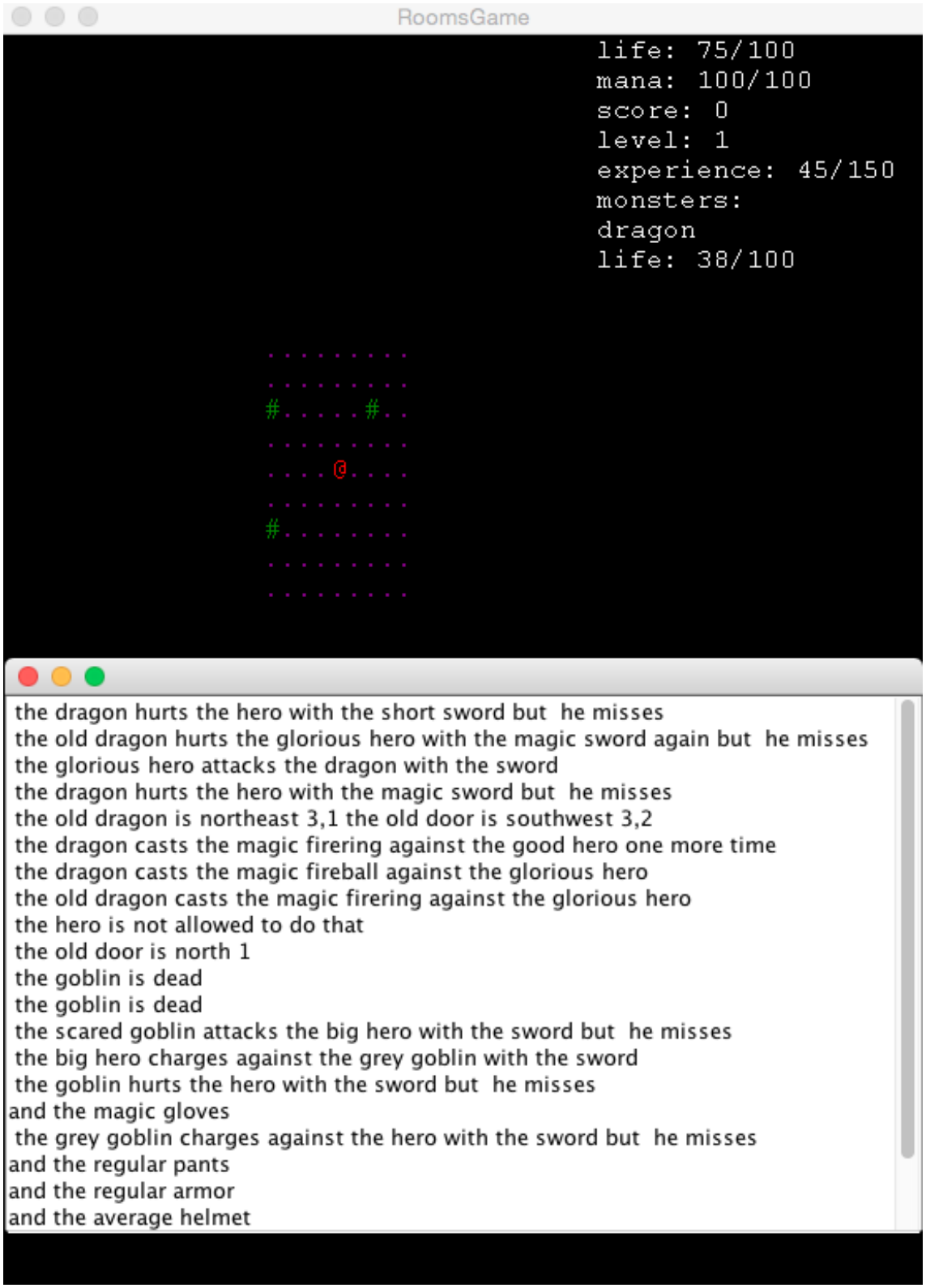


Figura 6.3: Versión definitiva de la interfaz de usuario para el texto generado basada en el uso de una *TextArea*

6.2.2.1. Primer planteamiento: Generación completamente aleatoria

En un primer momento nuestra decisión fue la de generar todo de manera aleatoria, sin considerar ningún aspecto externo, pero manteniendo la generalidad en el código para poder cambiar su comportamiento en cualquier momento. Esto causaba que algunas mazmorras fueran muy complicadas cuando el jugador todavía no tenía el equipamiento necesario para enfrentarse a dichos enemigos o demasiado sencilla en otros momentos.

Este problema fue mencionado la primera vez que recibimos *feedback* y por ello decidimos cambiarlo por algo un poco más complejo.

6.2.2.2. Segunda idea: Generador de encuentros

En el mundo del rol, tener en cuenta ciertas características del usuario para generar elementos externos se denomina *generación de encuentros*. De este modo, en vez de generar mapas, enemigos y elementos de manera completamente aleatoria, podemos generarlos en base al nivel que tienen ciertos personajes. Si el personaje que controla el usuario tiene nivel 10, entonces los adversarios que se encuentren deberían ser de un nivel similar, por ejemplo entre 8 y 12 y así mantener un nivel de dificultad equilibrado, no demasiado fácil ni demasiado difícil, envitando que el jugador caiga en el tedio o en la frustración. De forma similar, los elementos que dichos oponentes suelten al morir, tales como espadas y armaduras, también deberían tener un nivel similar para que la progresión tenga sentido y eliminar enemigos implique un cierto nivel de recompensa.

De esta forma, en base al nivel dado, podremos generar contrincantes con diferentes características que se adapten a lo que necesitamos. Mostramos un ejemplo cómo generar un enemigo en una habitación en concreto:

```
Rat rat = new Rat(this.getMap(), this, position, new ArrayList
    <String>(), level);
```

6.2.3. Comportamiento de los enemigos

Como cabe suponer, los adversarios que nos encontramos en el juego se moverán de diferentes formas, por lo que tendremos que diseñarlo de una manera genérica que nos permita realizar esto de la forma más simple posible. Para ello nos hemos basado en el patrón de diseño *estrategia* porque se adapta perfectamente a nuestros requisitos.

Los tipos de movimiento que tenemos disponibles en el juego actualmente de cara a los enemigos son los siguientes:

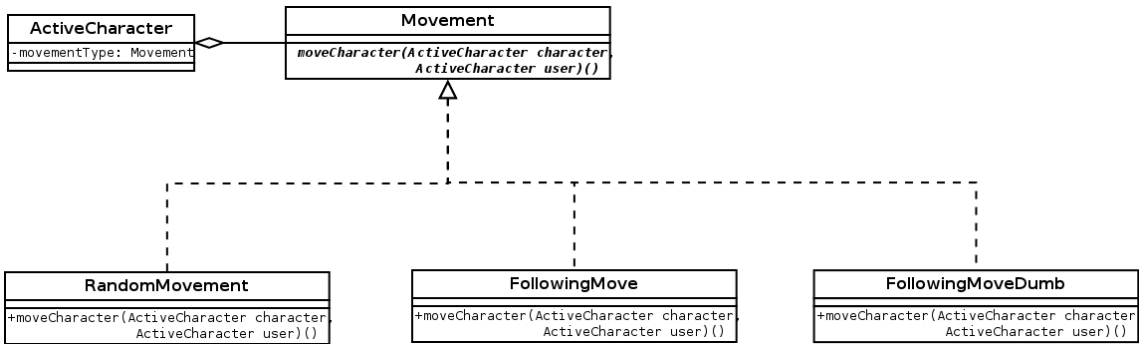


Figura 6.4: Diagrama de clases para el comportamiento de los enemigos

RandomMovement: Los enemigos son "pasivos", es decir, no atacarán a ningún personaje ni tampoco irán hacia él en ningún momento.

FollowingMovementDumb: Los contrincantes que tengan este tipo de comportamiento son más "agresivos". Seguirán al jugador e intentarán atacarle, pero no se mueven de una forma óptima para lograr su meta, por lo que esquivarlos no suele ser muy complicado.

FollowingMovement: Este tipo de comportamiento es el más complejo y "agresivo". El enemigo se moverá de manera precisa para llegar lo antes posible al usuario y usará todo tipo de tácticas para derrotarlo, tales como el uso de magia o atacándole cuerpo a cuerpo.

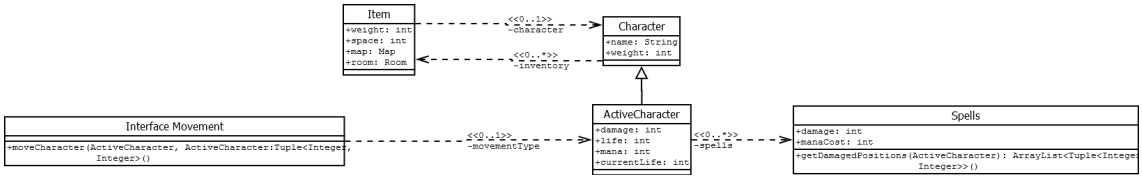
Gracias a nuestro diseño, crear nuevos tipos de comportamientos es una tarea trivial, ya que solamente tendríamos que implementar la interfaz *Movement* con la única función que posee e insertar en él el código del nuevo tipo de movimiento.

La Figura 6.4 mostramos el diagrama de clases que hemos implementado en nuestro sistema y que contiene el patrón de diseño *estrategia*, tal y como hemos comentado.

6.2.4. Interacción entre objetos, personajes y hechizos

La Figura **??** muestra las relaciones entre las clases abstractas e interfaces de los objetos, personajes, movimientos y hechizos de nuestro juego.

La clase `Character` contiene la información básica de los personajes, mientras que en `ActiveCharacter` se encuentra información adicional que posibilita que el personaje se mueva en base a una clase que implemente la interfaz `Movement` e interactúe con el resto de elementos del juego, por lo que es la clase de la que todos los enemigos heredan. Esto ayuda a que en un futuro puedan crearse personajes como vendedores que no necesiten moverse de manera sencilla. Todos los personajes pueden tener `Items`, que son consumibles (pociones)



o elementos que el jugador puede equipar (espadas, cascos, botas, etc.). Sin embargo, éstos no tienen por qué pertenecer a ningún personaje, se pueden encontrar en el mapa para que el personaje que el jugador controla pueda cogerlos. Por último, un personaje en concreto también puede tener diferente número de hechizos, representados en el diagrama por la clase abstracta `Spell`, que harán daño en cierta área y que también costarán maná. De la misma manera que con las clases `Movement` o `ActiveCharacter`, extenderla para crear nuevos hechizos es muy sencillo, lo que permite que los elementos primordiales del juego sean fácilmente extendibles.

6.3. Herramientas empleadas

En esta sección hablaremos sobre las tecnologías y herramientas empleadas en este proyecto y, si cabe, las razones por la que fueron elegidas. En primer lugar describiremos las herramientas y bibliotecas que hemos usado y, en segundo lugar, las herramientas de comunicación con usuarios y *testers*.

6.3.1. Herramientas software empleadas

Java: Este lenguaje de programación orientado a objetos es uno de los lenguajes de programación más utilizados en la industria. Una de sus principales características es que es multiplataforma, es decir, puede ser ejecutado en cualquier sistema operativo que tenga la *Java Virtual Machine* instalada sin necesidad de realizar cambios en el código (WORA, *"Write once, run anywhere"*). Esta característica es primordial en nuestro caso, dado que nuestros usuarios potenciales usan gran variedad de sistemas operativos, mayor incluso que respecto al usuario estándard .

Eclipse: [1] Es un IDE (entorno de desarrollo integrado) usado para escribir código en múltiples lenguajes. También incluye una serie de *plugins* que facilitan y automatizan muchas de las labores a realizar, tales como el uso de sistema de controles, ejecución de código y tests, herramientas de *debug*, autocompletado de código, etc.

[1] https://eclipse.org

Git: [2] Sistema de control de versiones distribuido. Es el control de versiones referencia en la mayoría de empresas y proyectos de software libre gracias a su rapidez y, al ser distribuido, permite trabajar y realizar *commits* del código sin necesidad de conexión a Internet.

GitHub: [3] Plataforma de desarrollo colaborativo usada para alojar proyectos usando el sistema de control de versiones Git. La mayoría de proyectos de código abierto lo usan, dado que es gratuito, aunque permite la opción de almacenar el código de forma privada previo pago.

JSON: *JavaScript Object Notation*. Es un formato muy usado en APIs para intercambio de datos, similar a XML. En nuestro caso lo usamos para definir las gramáticas y diccionarios de nuestro proyecto, dado que es muy sencillo de leer y especificar.

***NVDA*:** [4] Lector de pantalla de código libre para Windows que lee el tipo de ventanas y texto que se encuentran en la pantalla, facilitando el uso del ordenador y otros dispositivos a los usuarios invidentes. Orca[5] es, en cierta medida, su equivalente en Linux. Otras alternativas son BrowseAloud[6] o Microsoft Narrator[7]. Hemos elegido *NVDA* por ser la alternativa libre más usada por este tipo de usuarios.

***Gson*:** [8] Hay numerosas bibliotecas que nos permiten analizar y trabajar con este formato en Java. En nuestro caso hemos usado *Gson* para transformar archivos JSON a objetos de Java y viceversa.

JCurses: [9] *The Java Curses Library* es una biblioteca para el desarrollo de aplicaciones de terminal para JAVA. Es similar a AWT,[10] pero basada en el sistema de ventanas *Curses* de *UNIX*.

Libjcsi: [11] Biblioteca de representación gráfica que trabaja sobre JCurses y simplifica la tarea de representar y refrescar elementos del terminal.

[2] https://goo.gl/IKsKt5
[3] github.com
[4] www.nvaccess.org
[5] https://goo.gl/nW9UaZ
[6] https://goo.gl/GoghW7
[7] http://goo.gl/OAB7VC
[8] https://goo.gl/zPCXen
[9] https://github.com/sunhong/jcurses
[10] *Abstract Window Toolkit*. Kit de herramientas de interfaz de usuario de la plataforma original de Java
[11] www.slashie.net/libjcsi

6.3.2. Herramientas de comunicación con usuarios y *testers*

Listas de Correo: Las listas de correo son un método de comunicación muy usado por diferentes comunidades, especialmente en el desarrollo de software, que ayudan a los usuarios que participan en ellas a enviar correos a múltiples personas que lo deseen de forma anónima y, al mismo tiempo, tener un historial de las respuestas devueltas por los mismos. En nuestro caso las hemos usado para comunicarnos con un grupo de usuarios y desarrolladores de videojuegos para invidentes[12] y recibir *feedback* por parte de la comunidad.

Reddit: [13] Web creada en 2005 y que actualmente se encuentra en el top 50 de las más visitadas del mundo. Cuenta con una comunidad gigante que está dividida en muchísimos subgrupos dependiendo del tema a tratar. La hemos usado como una herramienta de *feedback*. Especialmente los *subreddits* de daltónicos, de personas que sufren de ceguera, desarrolladores de videojuegos y roguelikes.

[12] tiflo-juegos@googlegroups.com
[13] www.reddit.com

Capítulo 7

Conclusiones y trabajo futuro

En este último capítulo detallaremos las conclusiones sacadas de la elaboración del proyecto y el posible desarrollo futuro del mismo.

7.1. Conclusiones

Realizar un proyecto de estas características no es sencillo. Dejando a un lado la parte técnica y de diseño, que nunca es trivial, en este caso se requiere un gran grado de investigación, análisis, asimilación del *feedback* y determinación para poder sacarlo adelante. Sin embargo, también es cierto que durante su desarrollo siempre me he encontrado a una comunidad muy ilusionada por la existencia del proyecto y encantada de colaborar y resolver mis dudas. Es una pena que algo tan esencial como son la mayoría de elementos tecnológicos de hoy en día (y su *software* en particular) todavía no estén adaptados para que toda la parte de la población pueda usarlos sin grandes restricciones. Espero que con este proyecto haya gente que se anime a intentar hacer algo similar (o mejorar lo ya creado) y que entre todos podamos formar una sociedad donde no haya tanta discriminación tecnológica.

7.2. Trabajo Futuro

Cuando se crea cualquier tipo de *software*, siempre existen mejoras y nuevas funcionalidades que puede ser añadidas para mejorar lo creado, sobre todo si se trata de un juego de *software libre*, dado que las comunidades de desarrolladores y jugadores suelen ser bastante apasionadas. Ésta es la razón por la que, desde un principio, hemos establecido los elementos básicos a desarrollar para luego incluir otros elementos que no son tan esenciales.

Durante el desarrollo del proyecto se han tenido muchas ideas que hemos ido recopilando e implementando en caso de que las considerásemos necesarias. Actualmente todas estas ideas están recogidas en la página del proyecto en Github,[1] donde cualquier persona puede añadir su comentario o incluso realizar el cambio necesario y crear una *pull request* para que sea incluido en el juego final.

De entre éstas, alguna de las ideas más interesantes con las que podemos mejorar el juego en el futuro son las siguientes:

Creación de un menú de configuración: En la actualidad no disponemos de un menú porque podemos realizar todo lo necesario dentro del propio juego. Sin embargo, y a medida que la cantidad de opciones de configuración crece, sería conveniente su creación para que nos mostrase lo que podemos cambiar sin tener que entrar en el juego en sí.

Añadir más variedad de enemigos y elementos: La cantidad de enemigos diferentes que tenemos actualmente no es muy grande ya que desde el punto de vista académico aumentar dicha variedad no nos aportaba nada. Sin embargo, la experiencia de juego se beneficiaría teniendo más enemigos y objetos con los que interactuar, aunque sin olvidar que deben escalar de manera apropiada.

Integrar un modo historia: El juego actual es abierto en el sentido de que las partidas son independientes entre sí y no hay un principio ni final definidos, sino que el usuario es el que crea, en cierta manera, su historia. No estaría de más integrar un nuevo modo de juego en el que se desarrolle una historia con argumento. Para ello se añadirían ciertos niveles y pantallas predefinidos mediante los cuales se hilvanaría la historia para aquellos usuarios que prefieran este tipo de aventuras.

Crear un tutorial: Las instrucciones del juego se encuentran disponibles en la página de Github del proyecto, pero no estaría de más la creación de un tutorial o pantalla de inicio que explicase al usuario qué hacer y cómo desenvolverse por los entornos del juego.

Integrar un "modo resumen": Con todas las frases que generamos durante la partida podríamos crear una especie de historia que relatase, en la medida de lo posible, lo que el jugador ha hecho durante dicha partida, dándole cierta importancia a los "hitos" conseguidos durante la misma. Otros *roguelikes* como *Dwarf Fortress* disponen de este elemento.[2]

[1] https://github.com/dpenas/roomsgame/issues
[2] http://www.bay12games.com/dwarves/

Incluir elementos sonoros: Algunos juegos para invidentes solamente tienen elementos auditivos para informar al usuario sobre lo que tiene a su alrededor. En nuestro caso, al poder crear frases en base a unas gramáticas dadas, no resulta ser algo necesario, pero sí que podríamos crear una opción para que (en vez de) usar siempre descripciones, podamos reproducir sonidos a diferentes grados de volumen e incluso empleando técnicas de sonido tridimensional para informar al jugador sobre distintos elementos y enriquecer la atmósfera del juego. Por ejemplo, acompañar las descripciones de los combates con entrechocar de espadas.

Añadir más complejidad a las gramáticas: Los textos generados se basan en las gramáticas que se hayan definido para el sistema, pero hay estructuras del lenguaje (como frases reflexivas), que son más complejas de crear que otras. Podríamos introducir estos nuevos elementos para que la cantidad de frases generadas y su variedad sea todavía mayor.

Integrar recursos lógicos de terceros: Tener un diccionario propio del sistema para el idioma en el que vayamos a jugar es necesario, dado que no todos los idiomas disponen del mismo tipo de recursos lingüísticos y su uso limitaría la cantidad de idiomas con los que podríamos jugar. Sin embargo, si el juego está, por ejemplo, en inglés, sí que podríamos usar recrsos como la base de datos léxica *WordNet* para que, en vez de usar la palabra que se encuentra en nuestro diccionario, busque sinónimos en dichas bibliotecas, lo que aumentaría de manera significante la riqueza y expresividad del sistema.

Añadir opción de jugar en base a una semilla: Algunos *roguelikes* o *roguelites* como *The Binding of Isaac: Rebirth*,[3] permiten que el jugador introduzca un código que equivale a una semilla de aleatoriedad para que los elementos generados sean en base a esa semilla. Esto permitiría que los usuarios compartan partidas donde todo lo generado sea igual para ambos.

[3] http://store.steampowered.com/app/250900

Apéndice A

Escoger la licencia

Cuando tenemos que escoger una licencia para un nuevo programa también debemos tener en cuenta las licencias de las bibliotecas y recursos que hemos usado en el mismo. Las bibliotecas que hemos usado en nuestro proyecto y sus respectivas licencias son éstas:

- **Gson** Apache 2.0 license
- **JCurses** GNU Lesser General Public License v3
- **libjcsi** GNU Lesser General Public License v2.1

Como podemos apreciar, Gson usa una licencia Apache 2.0[web] mientras que JCurses y libjcsi tienen GNU Lesser GPL [web07]. La licencia de Apache 2.0 no es compatible con las versiones de GNU Lesser GPL anteriores a la 3, pero las versiones de GNU Lesser GPL son compatibles entre sí. Por este motivo decidimos usar GNU General Public Licence v3, dado que es compatible con todas las licencias de las librerías que hemos usado.

Apéndice B

Instalación e Instrucciones del Juego

Nuestro *roguelike* es software libre y el código fuente está libremente disponible en Github,[1] al igual que varias versiones ejecutables del mismo. Dichas versiones ejecutables contienen un archivo `.jar` que solamente requiere tener una versión de JRE[2][3] superior a la 6 instalada en el sistema a usar.

Para su ejecución basta con hacer doble clic en el archivo `.jar` o ir al directorio donde se encuentre dicho archivo e introducir:

Fragmento de Código B.1: Comando para la ejecución del videojuego

```
java -jar game.jar
```

Para cambiar el idioma del proyecto se debe de localizar el archivo `languages.properties` y cambiar el idioma a ES, GL, EN o NL para jugar en español, gallego, inglés u holandés, respectivamente. También es posible cambiar las teclas del propio juego. Las que vienen por defecto se puede encontrar en la wiki del proyecto.[4].

En algunas ocasiones las frases generadas no se leen directamente. Esto viene causado por no tener activado el *Java Access Bridge* o si no está instalada la versión correcta del mismo (se recomienda tener instalada la versión de 32 y 64 bits para evitar problemas). En sistemas Windows *Java Access Bridge* se puede activar fácilmente de la forma siguiente:

- Ir a **Start →Control Panel →Ease of Access →Ease of Access Center**

[1] https://github.com/dpenas/roomsgame
[2] http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html
[3] Java Runtime Environment
[4] https://github.com/dpenas/roomsgame/wiki

- Seleccionar **Use the computer without a display**
- En la sección **Other programs installed**, seleccionar: **Enable Java Access Bridge**

Estas instrucciones deberían servir para Windows Vista y posteriores. En caso contrario, también se podrá hacer:

```
%JRE_HOME%\bin\jabswitch -enable
```

donde *%JRE_HOME%* es el directorio donde Java está instalado.

Para otros sistemas operativos será necesario la descarga y activación de manera manual.[5]

[5] http://www.oracle.com/technetwork/articles/javase/index-jsp-136191.html

Apéndice C

Bibliografía

[Bec02] Kent Beck. *Test Driven Development: By Example*. 2002.

[CDM99] Hinrich Schütze Christopher D. Manning. *Foundations of Statistical Natural Language Processing*. 1999.

[CNB14] CNBC. Digital gaming sales hit record $61 billion in 2015: Report. http://www.cnbc.com/2016/01/26/digital-gaming-sales-hit-record-61-billion-in-2015-report.html, 2014.

[DJ08] James H. Martin Daniel Jurafsky. *Speech and Language Processing*. 2008.

[EG+95] Richard Helm Erich Gamma et al. *Elements of Reusable Object-Oriented Software*. 1995.

[fou12] The Ablegamers foundation. A Practical Guide to Game Accessibility. http://www.includification.com/AbleGamers_Includification.pdf, 2012.

[Gam16] PC Gamer. Steam games market worth $3.5 billion in 2015. http://www.pcgamer.com/steam-games-market-worth-35-billion-in-2015/, 5 de Enero, 2016.

[Git] Git. Git Documentation. https://git-scm.com/documentation.

[Goo16] Owen S. Good. EA exec's remarks shine a light on Xbox One sales. http://www.polygon.com/2016/1/30/10876042/ps4-xbox-one-sales-lifetime-million, 30 de Enero, 2016.

[JS] Ken Schwaber Jeff Sutherland. Scrum Guides. http://www.scrumguides.org/.

[NH01] Jana Halford Neal Halford. *Swords & circuitry: a designer's guide to computer role playing games*. 2001.

[Non86] Hirotaka Takeuchi; Ikujiro Nonaka. NEW NEW PRODUCT DEVELOPMENT GAME. https://cb.hbsp.harvard.edu/cbmp/product/86116-PDF-ENG, 1 de enero, 1986.

[Ora] Oracle. Java Documentation. https://docs.oracle.com/javase/7/docs/api/.

[Pig96] Thomas M. Pigoski. Practical Software Maintenance: Best Practices for Managing Your Software Investment. http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471170011.html, Noviembre, 1996.

[Pou84] Jerry Pournelle. New Machines, Networks and Sundry Software. http://archive.org/stream/byte-magazine-1984-03/1984_03_BYTE_09-03_Simulation#page/n47/mode/2up, 1984.

[Rad12] Roguelike Radio. Designing for the Visually Impaired. http://www.roguelikeradio.com/2012/10/episode-48-designing-for-visually.html, 2012.

[Sar16] Samit Sarkar. PS4 sales surpass 35.9M units worldwide. http://www.polygon.com/2016/1/5/10717142/ps4-lifetime-sales-35-9-million-holiday-2015, 5 de Enero, 2016.

[The14] Theesa. Games: Improving the Economy. http://www.theesa.com/wp-content/uploads/2014/11/Games_Economy-11-4-14.pdf, 2014.

[web] Apache License. http://www.apache.org/foundation/license-faq.html.

[web07] GNU Licenses. http://www.gnu.org/licenses/lgpl-3.0.en.html, 2007.