Facultade de Informática da Universidade de A Coruña
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: Jesús Vilares Ferro
Director: Carlos Gómez Rodríguez
Fecha: CHANGE: 15 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 su disfrute.

Este proyecto consiste en la creación de un videojuego *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 las gramáticas y diccionario dados.

Para llegar a una mayor cantidad de usuarios y que su ejecución sea sencilla en el mayor número de plantaformas posible, se ha decidido usar el lenguaje de programación Java.

Lista de palabras clave:

- Tiflotecnología
- Lenguajes Naturales
- Accesibilidad
- Entretenimiento Digital
- Roguelike
- Java
- Open Source

Agradecimientos

TODO

Darío Penas Sabín
Amsterdam, PONER FECHA

Índice general

Índice de figuras

Capítulo 1

Introdución

En este capítulo introductorio se explicarán los aspectos necesarios para entender lo más importante del proyecto, la motivación para realización del mismo y un breve resumen del resto de capítulos que forman parte de la memoria.

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 personas quejándose de *first person shooters* que tienen un *FOV*[1] limitado, causándoles mareos al poco rato; daltónicos protestando que diferentes juegos (como por ejemplo The Witness[2]), basan buena parte de su mecánica en que el jugador sea capaz de distinguir diferentes colores; zurdos que tienen que acomodarse a ciertos controles a no existir una opción para cambiarlos; invidentes que no pueden disfrutar de prácticamente ninguno de estos títulos, etc.

En este proyecto nuestro objetivo es crear un videojuego desde cero que tenga en cuenta todo tipo de minorías, tomando especial relevancia los invidentes y centrándose en los aspectos que sean relevantes para ellos; descripciones que sean fácilmente reproducibles que cambien automáticamente y fácil expansión del título, tanto en características generales como en idiomas, gramáticas o palabras empleadas en el mismo.

[1] *Field of view*, campo de visión. extensión de mundo observable en un momento dado
[2] Juego de puzzles en primera persona: http://the-witness.net/

1.1.1. Visita a la *ONCE*

A principios de 2015 Jesús y yo fuimos a un taller de la ONCE donde una persona invidente nos habló sobre la tiflotecnología, nos mostró la forma en la que usaban ordenadores y móbiles, incluso para leer código, y los errores, muchos de ellos fácilmente solventables, que se cometían día a día en temas de accesibilidad. Esta visita nos abrió los ojos y gracias a ella fuimos capaces de detectar posibles mejoras que hacer al proyecto y los fallos que no deberíamos de cometer. También se ofrecieron a probar el proyecto una vez estuviera listo, pero este será un tema que trataremos en las próximas secciones.

1.2. Motivación

El entretenimiento digital siempre ha sido parte de mi vida y una de las razones por las que desde pequeño estuve interesado en la informática, razón por la 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.

A pesar de los grandes avances de la industria de los videojuegos y de la gran cantidad de nuevos estudios y proyectos que se lanzan anualmente, resulta muy complicado poderse dedicar profesionalmente a cualquiera de estas dos cosas (y no digamos ambas a la vez), por lo que en mi futuro profesional no he sido capaz, al menos de momento, de cumplir mi sueño de trabajar en lo que más me apasiona. Por este motivo, cuando Jesús me comentó que él y Carlos llevaban un tiempo con este proyecto disponible, no dudé en un instante en aceptarlo y ponerme manos a la obra.

Este videojuego se ha desarrollado durante el periodo de un par de años en los que se incluyen muchos cambios en mi vida, tales como mi emigración a Holanda hace ya casi dos años y mis primeros pasos en el mundo laboral. Cada día que pasa me alegro más de tener un proyecto como éste, dado que sin la pasión e interés por el mismo estoy seguro de que jamás lo habría terminado.

1.3. Estrutura da memoria

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

Capítulo 1. Introducción. Se explicarán, de manera general y resumida, en qué consiste el proyecto, la motivación del mismo y la estructura que tendrá 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. Citaremos y hablaremos sobre las herramientas y bibliotecas empleadas durante la elaboración del proyecto.

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

Capítulo 5. Planificación y Seguimento. Detallaremos la planificación y seguimiento usados en 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 y de la implementación de ciertas partes del programa.

Capítulo 8. Recepción, conclusión y trabajo futuro. En este apartado mostraremos los comentarios obtenidos por la comunidad durante el desarrollo del proyecto, se relatarán las conclusiones obtenidas y se detallarán los posibles cambios, mejoras y añadidos que se podrán tener en cuenta en el futuro.

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 definición sobre lo que es un *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 diliar 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 una gran parte de la población, siendo censurado y degragado en mayor o menor medida, no tan solo por cierta parte de la sociedad, pero también por muchos medios de comunicación y gobiernos. A pesar de que hoy en día este problema todavía está activo[1], la industria se ha expandido tanto (consolas, ordenadores, navegadores, Facebook, móvil...), que cada vez es más complicado encontrar a alguien que no haya jugado a algún videojuego en las últimas semanas, ya es algo que forma parte del día a día de mucha parte de la población.

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

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[2] llegando a generar, solamente en ventas digitales, alrededor de 61 billones de dólares en el año 2015[3].

Este gran éxito se debe, en gran parte, a la irrupción de los juegos desarrollados para móviles, cuyo beneficio ha ido aumentando enormemente durante los últimos años. [4]. Sin embargo, esto no significa que el resto de plataformas no estén triunfando. Solamente Steam, la plataforma de distribución digital para PC por excelencia desarrollada por Valve, ha generado alrededor de 3 billones y medio de dólares en el año 2015[5].

Con el mercado del PC resurgiendo, las consolas de sobremesa obteniendo grandes números de ventas, las portátiles resistiendo, el mercado de los videojuegos para móvil en esplendor y los cascos de realidad virtual llegando al mercado este año 2016; todo parece indicar que estos números no harán más que crecer durante los próximos años.

2.1.2. Conclusión

Lo que comenzó hace varias décadas como un modo de entretenimiento sin ninguna pretensión, generalmente enfocado a un público infantil o adolescente y que miraba a otras industrias como la cinematográfica con recelo, 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.

2.2. *Roguelikes.*

2.2.1. Qué es y orígenes

En 1983, Michael Toy y Glenn Wichman crearon un videojuego llamado Rogue[6] que acabó definiendo un género.

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

[2] http://www.theesa.com/wp-content/uploads/2014/11/Games_Economy-11-4-14.pdf
[3] http://goo.gl/BgGhXy
[4] La venta de videojuegos en Alemania crece año tras año, pero el mayor aumento de beneficio se está centrando en el mercado de los juegos para móvil
[5] http://goo.gl/Mbjgol
[6] Desde 2014 este juego se encuentra disponible en archive.org

Dificultad : Rogue es un videojuego difícil con *permadeath*[7] que obligará al jugador a rejugarlo una y otra vez, intentando llegar más lejos que la anterior partida gracias a ir aprendiendo los funcionamientos del mismo.

Aleatoriedad : Cada vez que el jugador comienza una partida nueva 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... causando que cada vez que el usuario empiece, tenga un grado de dificultad pseudo-aleatorio dependiendo de la semilla con la que estos elementos han sido generados.

Progresión : Una de las frases más escuchadas en las críticas que Rogue recibió tras su lanzamiento es que el jugador sentía la necesidad de intentar llegar más lejos en cada ocasión[8]. Esto viene dado, sobre todo, por la sensación de progresión y de que en cada *run*[9] el usuario vaya mejorando. Dentro de la propia partida también existe una progresión a medida que el usuario derrota enemigos, consiguiendo puntos de experiencia, subiendo niveles y obteniendo mejores armas y armaduras con las que ser un poco más fuerte.

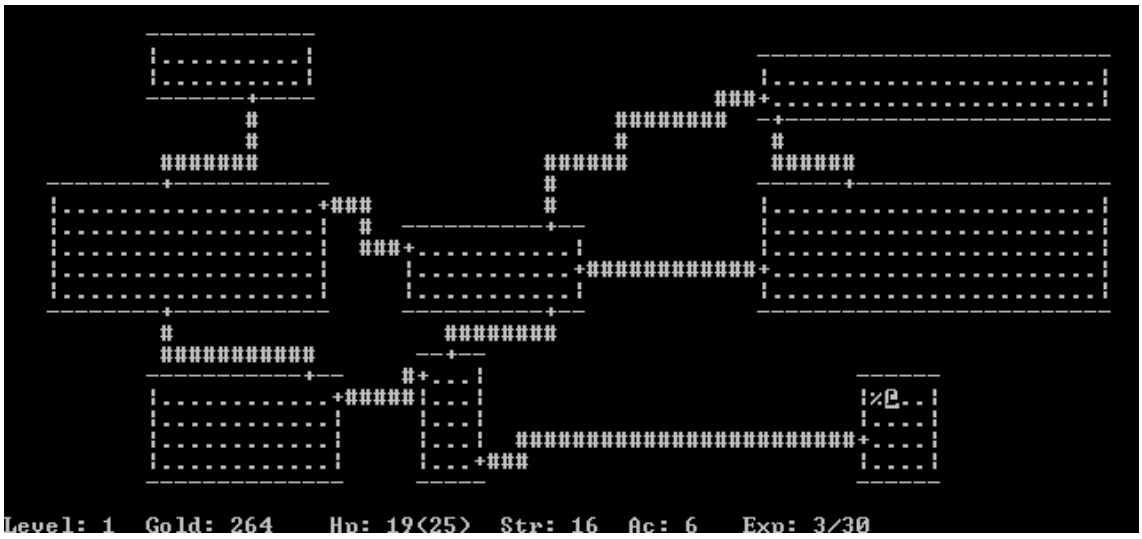


Figura 2.1: Captura de pantalla de dominio público (tal y como todas las imágenes mostradas en este proyecto) del videojuego Rogue

A partir de este momento muchos fueron los juegos que decidieron imitar estas características de Rogue, añadiendo, cambiando o enfatizando diferentes elementos, por eso se han denominado *roguelikes*.

[7] Una vez que el jugador muere, tiene que empezar desde el principio; no hay partidas guardadas

[8] Jerry Pournelle habló de ello en este artículo

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

2.2.2. En la actualidad

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

Figura 2.2: Captura de pantalla del videojuego Vultures

2.2.2.1. La creación de subgéneros

Dado que 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 a mucha gente hoy en día, son numerosos los juegos que han añadido más elementos de progreso general para que el jugador no se sienta frustrado. Estos elementos pueden ser nuevos personajes con los que jugar, puntos de experiencia 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

común ver juegos que se basan en partidas cortas, de como mucho una hora, para que la repetición sea mayor y perder el personaje no sea un gran "castigo" que tener que afrontar.

Estos cambios que se han realizado durante los últimos años y que, de cierta manera, han modificado el género que Rogue creó en un inicio, no siempre se han tomado positivamente por parte de la comunidad, que se suele quejar de que muchos títulos que se definen a sí mismos como *roguelike* no contienen ciertas características, como la dificultad, que una vez definieron el género. 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.

2.2.3. Elementos *roguelike* en nuestro proyecto

En nuestro caso hemos creado un *roguelike* similar a Rogue, no solamente estéticamente, pero también en diseño y funcionalidad. El usuario se moverá por 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 complicados de batir y la recompensa por hacerlo será mayor.

El objectivo del juego es llegar lo más lejos posible dentro de la mazmorra. Cada vez que el jugador entra en un portal se añadirá un punto (el número de puntos se mostrará en la pantalla) y, cada vez que esto suceda, un nuevo mapa, con diferentes características y contenido, será generado. El juego es complicado, aleatorio y con una sensación de progreso, tal y como el género *roguelike* especifica.

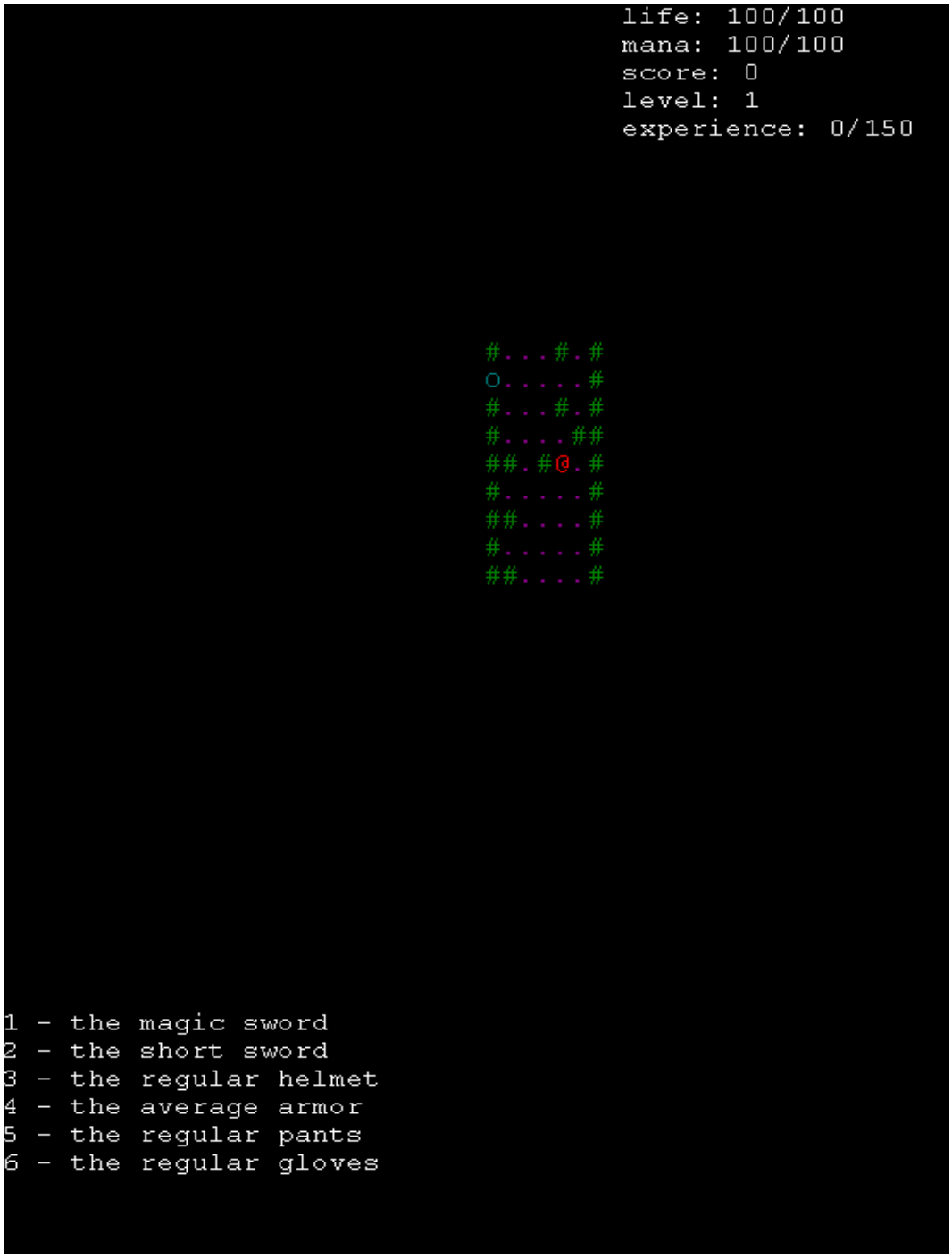


Figura 2.3: Captura de pantalla de la interfaz de usuario de nuestro juego

2.3. Problemas de la tecnología para ciertos sectores de la sociedad

Algunas de las razones que hacen de la tecnología un elemento prácticamente necesario hoy en día y que facilita su usabilidad crea, paradójicamente, una barrera para mucha gente que no puede disfrutar de ella.

En esta sección hablaremos sobre algunos de los problemas que tantos invidentes como daltónicos se encuentran actualmente en diferentes programas y cómo, en bastantes ocasiones, solventarlos no es muy complicado, lo que clarifica que muchas de estas limitaciones son dadas más por el desconocimiento que por la dificultad de la implentación de una solución.

2.3.1. Dificultades a los que se enfrentan los invidentes

Las dificultades que tienen invidentes o personas con diferentes grados de ceguera en nuestra sociedad es enorme y, tristemente, los dispositivos tecnológicos y 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 imposibiliza el uso de la mayoría de las aplicaciones y videojuegos en su totalidad. Incluso navegar por internet, donde la mayoría del contenido es texto que puede ser leído fácilmente por un lector de pantalla, puede llegar a ser un quebradero de cabeza si cierta parte de ese contenido está en flash (el lector no podrá hacer nada en este caso), los enlaces a redes sociales están en posiciones poco recomendables (dado que su contenido se puede llegar a reproducir y generalmente son un suplicio saltarlos), algunos de los nombres usados en programas software son poco descriptivos y dificultan su correcta identificación, etc.

2.3.2. Cómo solventar parte de estos problemas para los invidentes

Uno de las primeros problemas que nos comentaron en la charla de la ONCE es que muchos de los desarrolladores que programan algo específicamente para ciegos no tienen en cuenta que, la mayoría de las veces, hay gente que puede ver a su alrededor que les puede describir lo que está sucediendo en la pantalla, por lo que es muy importante crear una interfaz que esté actualizada y muestre la misma información que la persona invidente reciba. También es necesario dar la posibilidad al usuario de repetir el contenido generado, dado que algunos lectores tienen problemas para leer ciertos caracteres o lo que leen no es del todo claro y, por lo tanto, debe de ser fácilmente repetido.

Algunos videojuegos han visto versiones adaptadas para invidentes, la mayoría de ellas desarrolladas por la comunidad, o nuevos títulos que se centran en ofrecer una experiencia

revolucionaria desde el principio. Un buen ejemplo de ello es Shades of Doom[10], que solamente usa sonido (principalmente ruidos repetitivos) y muy pocas frases para que el jugador sea capaz de descubrir lo que tiene que hacer y dónde está.

2.3.3. Cómo hemos solventado el problema en nuestro proyecto

En el proyecto, dado que se basa en la generación pseudo-aleatoria de frases, 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, etc. Hemos creado una ventana que se encuentra al lado del juego y donde se van guardando todas las frases generadas, siendo la primera frase la última generada y leyéndose automáticamente por el reproductor de pantalla que esté siendo usado en ese momento. De esta forma, una persona con visión siempre puede leer dicha frase y el jugador repetirla las veces que quiera.

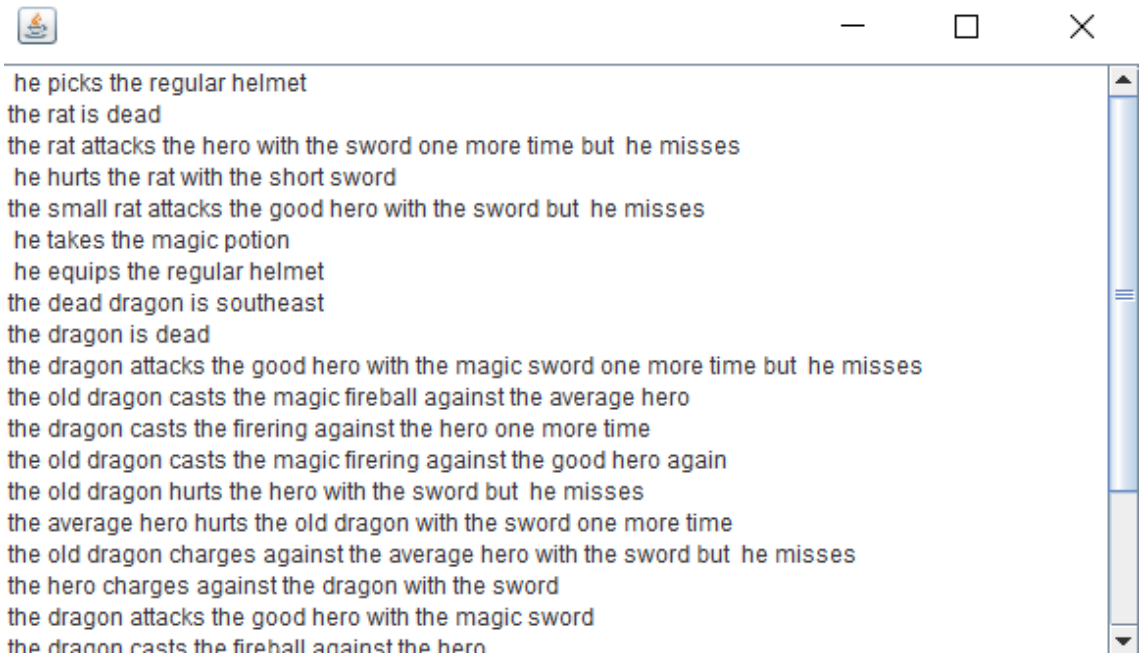


Figura 2.4: Captura de pantalla del area donde mostramos las frases generadas de nuestro juego para los invidentes

2.3.4. Dificultades a los que se enfrentan los daltónicos

Daltonismo es un defecto genérico que afecta, aproximadamente, al 1 % de la población. Lo que en un principio parece como un pequeño inconveniente que no cambia mucho la vida de la persona que lo sufre, lo cierto es que hay muchas ocasiones en las que incluso

[10] http://www.gmagames.com/sod.html

ver un simple partido de fútbol americano correctamente puede convertirse en algo casi imposible para ellos [11]. Incluso ver un mero mapa puede ser una complicación[12].

En el mercado del ocio digital, donde los gráficos y paletas de colores juegan un gran papel en el arte del título, este problema se ve acentuado. La peor parte viene 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 es una pequeña molestia como en *Borderlands*[13] donde hay armas especiales que se diferencian, únicamente, por el color en el que se encuentra un determinado texto. Sin embargo, el mayor problema viene cuando estas limitaciones pueden llegar a arruinar el juego, como ocurre en *The Witness*, un juego de puzzles mencionado en apartados anteriores en el que, para poder solventar muchos de dichos puzzles, requiere que el usuario distinga (o al menos tenga la información), 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 a que el jugador detecte si un elemento está al frente o al fondo de dicho escenario.

2.3.5. Cómo solventar parte de estos problemas para los daltónicos

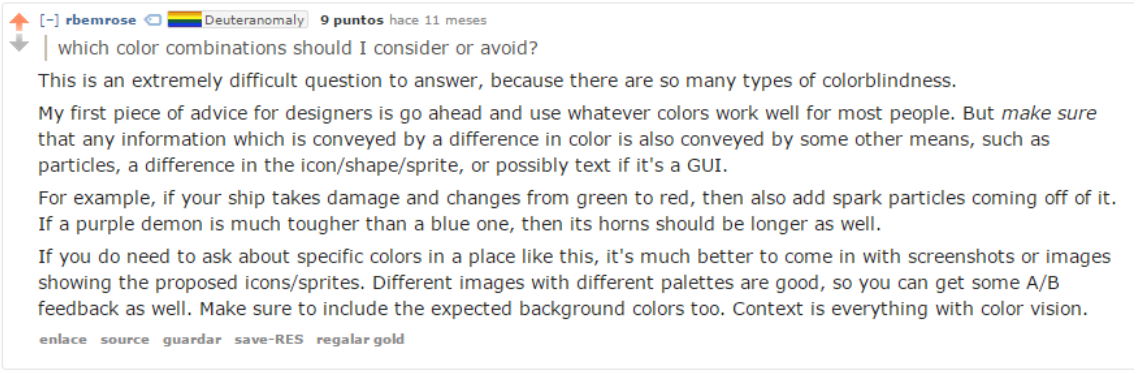
Para poder dar respuesta a esta cuestión, decidí preguntar en *Reddit* a los usuarios daltónicos sobre cómo desarrollar un juego que sea adecuado para ellos [14]. Fueron muchas las respuestas obtenidas, pero todo se puede resumir en un par de ideas.

La primera es que, en la medida de lo posible, nunca tengamos que diferenciar dos cosas distintas simplemente por el color. Tal y como un usuario comentó, si estamos desarrollando un juego naval y la diferencia entre un barco que está "sano" y uno que está "roto" es que cambiamos el color o la silueta del barco de verde a rojo, también podemos añadir chispas u otros elementos a mayores que faciliten la idea de que algo ha cambiado y ayude al usuario a apreciarlo visualmente con elementos a mayores. Del mismo modo, si tenemos un juego como *Tetris* o *Candy Crush* donde las piezas son relevantes, en vez de distinguirlas por color, podríamos hacerlas de diferentes formas o añadir una imagen a cada una de ellas.

La segunda idea es que, si es completamente necesario introducir diferentes colores para diferenciar ciertos elementos, o bien usamos colores que, generalmente, no crean ningún problema como el azul, amarillo, verde... (lo cual no nos garantiza que hayamos solventado el problema, dado que hay muchas formas de daltonismo y en algunas de ellas el usuario todavía puede tener problema diferenciándolos dependiendo del color contreto usado) o

[11] En algunos partidos los jugadores llevan camisetas con colores problemáticos.
[12] Resumen de algunos problemas que tienen los daltónicos a ver mapas
[13] Juego de acción en primera persona lanzado inicialmente en 2009 y que cuenta con varias secuelas
[14] https://goo.gl/d6cTqe

creamos una opción en el que podamos cambiar la paleta de colores usada. Hay algunos videojuegos (la mayor parte de ellos independientes), donde se da esta opción.



[–] **rbemrose** Deuteranomaly **9 puntos** hace 11 meses

> which color combinations should I consider or avoid?

This is an extremely difficult question to answer, because there are so many types of colorblindness.

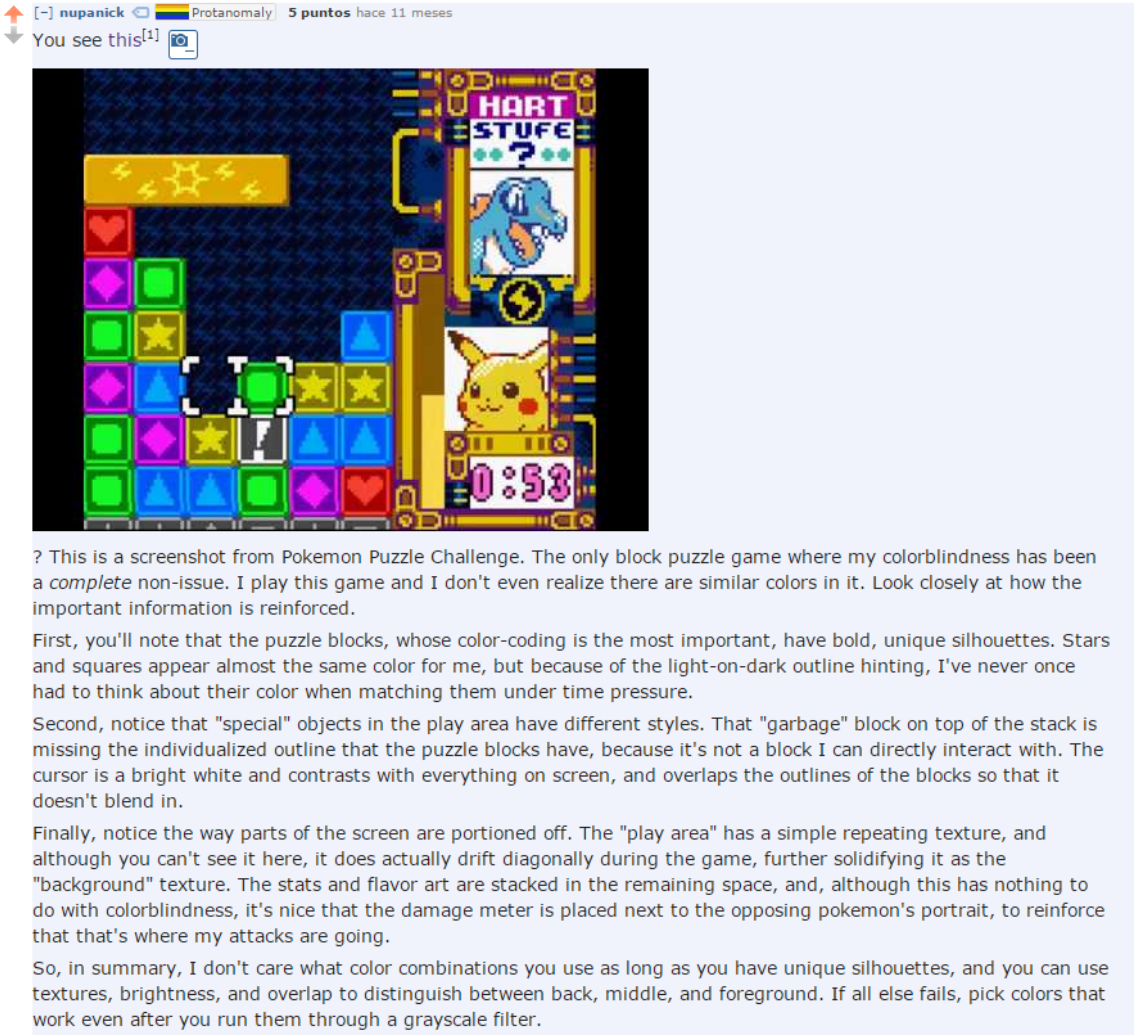
My first piece of advice for designers is go ahead and use whatever colors work well for most people. But *make sure* that any information which is conveyed by a difference in color is also conveyed by some other means, such as particles, a difference in the icon/shape/sprite, or possibly text if it's a GUI.

For example, if your ship takes damage and changes from green to red, then also add spark particles coming off of it. If a purple demon is much tougher than a blue one, then its horns should be longer as well.

If you do need to ask about specific colors in a place like this, it's much better to come in with screenshots or images showing the proposed icons/sprites. Different images with different palettes are good, so you can get some A/B feedback as well. Make sure to include the expected background colors too. Context is everything with color vision.

enlace source guardar save-RES regalar gold

Figura 2.5: Consejo para ayudar a los daltónicos a la hora de desarrollar un juego



[–] **nupanick** Protanomaly **5 puntos** hace 11 meses

You see this[1]

? This is a screenshot from Pokemon Puzzle Challenge. The only block puzzle game where my colorblindness has been a *complete* non-issue. I play this game and I don't even realize there are similar colors in it. Look closely at how the important information is reinforced.

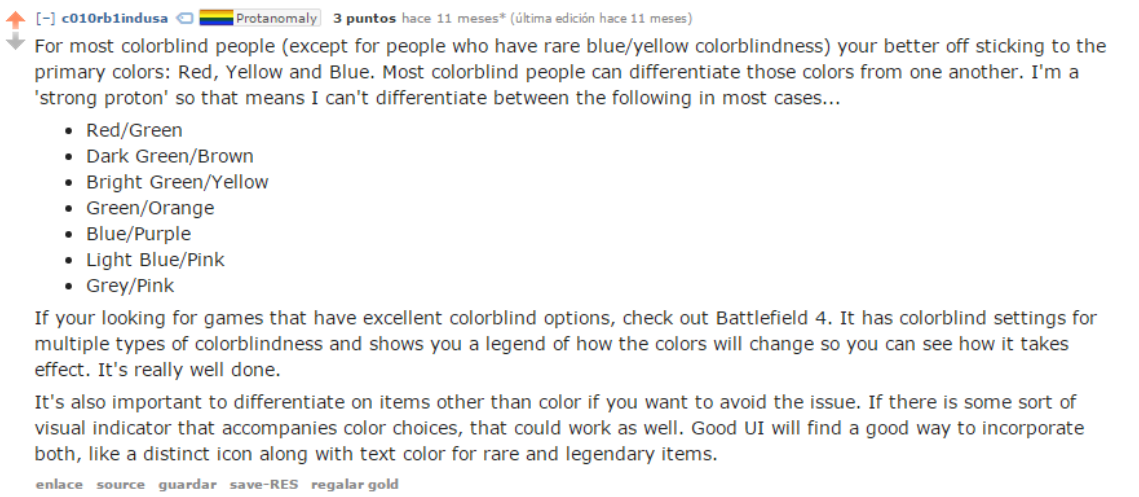
First, you'll note that the puzzle blocks, whose color-coding is the most important, have bold, unique silhouettes. Stars and squares appear almost the same color for me, but because of the light-on-dark outline hinting, I've never once had to think about their color when matching them under time pressure.

Second, notice that "special" objects in the play area have different styles. That "garbage" block on top of the stack is missing the individualized outline that the puzzle blocks have, because it's not a block I can directly interact with. The cursor is a bright white and contrasts with everything on screen, and overlaps the outlines of the blocks so that it doesn't blend in.

Finally, notice the way parts of the screen are portioned off. The "play area" has a simple repeating texture, and although you can't see it here, it does actually drift diagonally during the game, further solidifying it as the "background" texture. The stats and flavor art are stacked in the remaining space, and, although this has nothing to do with colorblindness, it's nice that the damage meter is placed next to the opposing pokemon's portrait, to reinforce that that's where my attacks are going.

So, in summary, I don't care what color combinations you use as long as you have unique silhouettes, and you can use textures, brightness, and overlap to distinguish between back, middle, and foreground. If all else fails, pick colors that work even after you run them through a grayscale filter.

Figura 2.6: Consejos y ejemplos para evitar que las personas daltónicas tengan problemas


[–] c010rb1indusa Protanomaly 3 puntos hace 11 meses* (última edición hace 11 meses)

For most colorblind people (except for people who have rare blue/yellow colorblindness) your better off sticking to the primary colors: Red, Yellow and Blue. Most colorblind people can differentiate those colors from one another. I'm a 'strong proton' so that means I can't differentiate between the following in most cases...

- Red/Green
- Dark Green/Brown
- Bright Green/Yellow
- Green/Orange
- Blue/Purple
- Light Blue/Pink
- Grey/Pink

If your looking for games that have excellent colorblind options, check out Battlefield 4. It has colorblind settings for multiple types of colorblindness and shows you a legend of how the colors will change so you can see how it takes effect. It's really well done.

It's also important to differentiate on items other than color if you want to avoid the issue. If there is some sort of visual indicator that accompanies color choices, that could work as well. Good UI will find a good way to incorporate both, like a distinct icon along with text color for rare and legendary items.

enlace source guardar save-RES regalar gold

Figura 2.7: Un usuario nos comenta la combinación de colores que son problemáticas

2.3.6. Cómo hemos solventado el problema en nuestro proyecto

En nuestro caso, al haber preguntado de antemano a los usuarios, siempre tuvimos desde el primer momento la idea de crear la interfaz gráfica con soporte para daltónicos en mente, a pesar de que, al haber tenido en cuenta a las personas invidentes, cualquier persona podría jugarlo sin ninguna dificultad.

Todos los elementos que se encuentran en la pantalla están diferenciados con caracteres completamente diferentes por lo que, incluso aunque todos los colores fueran iguales, sería sencillo identificar cada elemento.

También hemos creado una opción que cambia la paleta de colores a utilizar y facilita su visualización para aquellos usuarios con este inconveniente.

Capítulo 3

Fundamentos Tecnológicos

En este capítulo hablaremos sobre los fundamentos tecnológicos que vamos a usar en este proyecto y, si cabe, la razón por la que fueron elegidas. En primer lugar citaremos las herramientas que hemos usado y, en segundo lugar, las bibliotecas que hemos decidido utilizar en el programa en sí.

3.1. Herramientas empleadas

Java Lenguaje de programación orientado a objetos cuya primera aparición fue en 1995. Es uno de los lenguajes de programación más utilizados en la industria y 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[1]). Esta ventaja es esencial en nuestro caso, dado que la mayoría de las personas que pueden estar interesadas en el proyecto usan una gran variedad de sistemas operativos.

Eclipse Es un IDE[2] usado para escribir código en múltiples idiomas. También incluye una serie de *plugins* que facilitan y automatizan muchas de las labores a realizar como el uso de sistema de controles, ejecución de código y tests, herramientas de debug, autocompletado de código, etc.

Git Sistema de control de versiones distribuido introducido en 2005 y desarrollado principalmente por Linus Torvalds. Es el control de versiones referencia en la mayoría de

[1] *Write once, run anywhere.* Eslogan creado por Sun Microsystems para mostrar los beneficios de la multiplataforma

[2] *Integrated Development Environment.* Entorno de desarrollo integrado

empresas y proyectos de software libre gracias a su rapidez y, al ser distribuida, permite trabajar y realizar *commits* del código sin necesidad de conexión a internet.

GitHub 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 también tiene la opción de almacenar el código de forma privada tras, previamente, realizar un pago.

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 dadas por los mismos. En nuestro caso la hemos usado para comunicarnos con un grupo de usuarios y desarrolladores de videojuegos para invidentes.

Reddit 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`https://www.reddit.com/r/ColorBlind/` y gente ciega`https://www.reddit.com/r/blind/`.

Dia Aplicación informática que permite la creación de todo tipo de diagramas. En nuestro caso lo hemos usado para crear los diagramas UML que se encuentran en esta memoria.

Gantt Project Programa de software libre diseñado para la creación de diagramas de Gantt.

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. Hay numerosas bibliotecas que nos permiten analizar y trabajar con este formato en Java. La que nosotros usamos es *Gson*.

LaTeX Sistema de composición de textos altamente usado por la mayoría de textos científicos dada la facilidad de su composición, simpleza, alta calidad y herramientas que a ayudan a la creación de fórmulas, inserción de imágenes y muchos otros elementos. Muy modificable. Es el sistema que hemos usado para la creación de este documento.

NVDA Lector de pantalla de código libre para Windows. Orca es, en cierta medida, su equivalente en Linux.

3.2. Bibliotecas empregadas

Gson Biblioteca usada para transformar archivos JSON a objetos de Java y viceversa.

JCurses JCurses[3] es una biblioteca para el desarrollo de aplicaciones de terminal para JAVA. Es similar a AWT[4], pero basada en el sistema de ventanas Curses de UNIX.

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

[3] *The Java Curses Library*
[4] *Abstract Window Toolkit.* Kit de herramientas de interfaz de usuario de la plataforma original de Java

Capítulo 4

Metodología

En este apartado describiremos la metodología llevada a cabo en el proyecto. Una metodología es un conjunto de procesos, métodos y prácticas llevadas a cabo para asegurar, en la mayor medida posible, calidad en el producto final y en el tiempo acordado.

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.

4.1. 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 anterior haya acabado y esté completada.

Las ventajas que ofrece en desarrollo en cascada es 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 con bastante frecuencia.

La mayor desventaja es que detectar un error en una de las fases finales puede significar tener que replantear los pasos tomados en las anteriores, perdiendo parte del progreso realizado.

4.1.1. Etapas del desarrollo en cascada

Análisis de requisitos :

En esta fase se definen y determinan los requisitos que el software debe de cumplir entre el desarrollador y el cliente. Generalmente se generan documentos que formalizan

dichas decisiones para no tener que cambiarlas en el futuro, dado que un cambio en los requisitos puede significar cambiar gran parte del proyecto, por lo que llegar a un consenso es necesario.

Diseño :

Una vez del análisis haya finalizado, es hora del diseño. La idea principal es descomponer lo detallado durante el análisis de requisitos con el cliente y crear diferentes diagramas y diseños que ayuden a los programadores a entender lo que debe de ser realizado y cómo debe de funcionar, incluyendo pseudocódigo o algoritmos a alto nivel en caso de que sea necesario.

Implementación :

En esta fase es donde se escribe el código fuente en base a lo especificado anteriormente, intentando dar gran importancia a 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 :

Este es el momento final en el que el cliente es capaz de probar lo desarrollado. El programa debe de estar bien testado por parte de los programadores para evitar la mayoría de los problemas que pueden ocasionarse en esta fase.

Mantenimiento :

Una vez el software ha finalizado y se ha entregado al usuario final, empieza la fase de mantenimiento. En ella el usuario final pedirá que se resuelvan problemas, se añadan nuevas funcionalidades y se cambien otras ya añadidas en base a los nuevos requisitos que vayan apareciendo y no se tuvieron en cuenta en el principio. Esta 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 [1].

[1] http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471170011.html

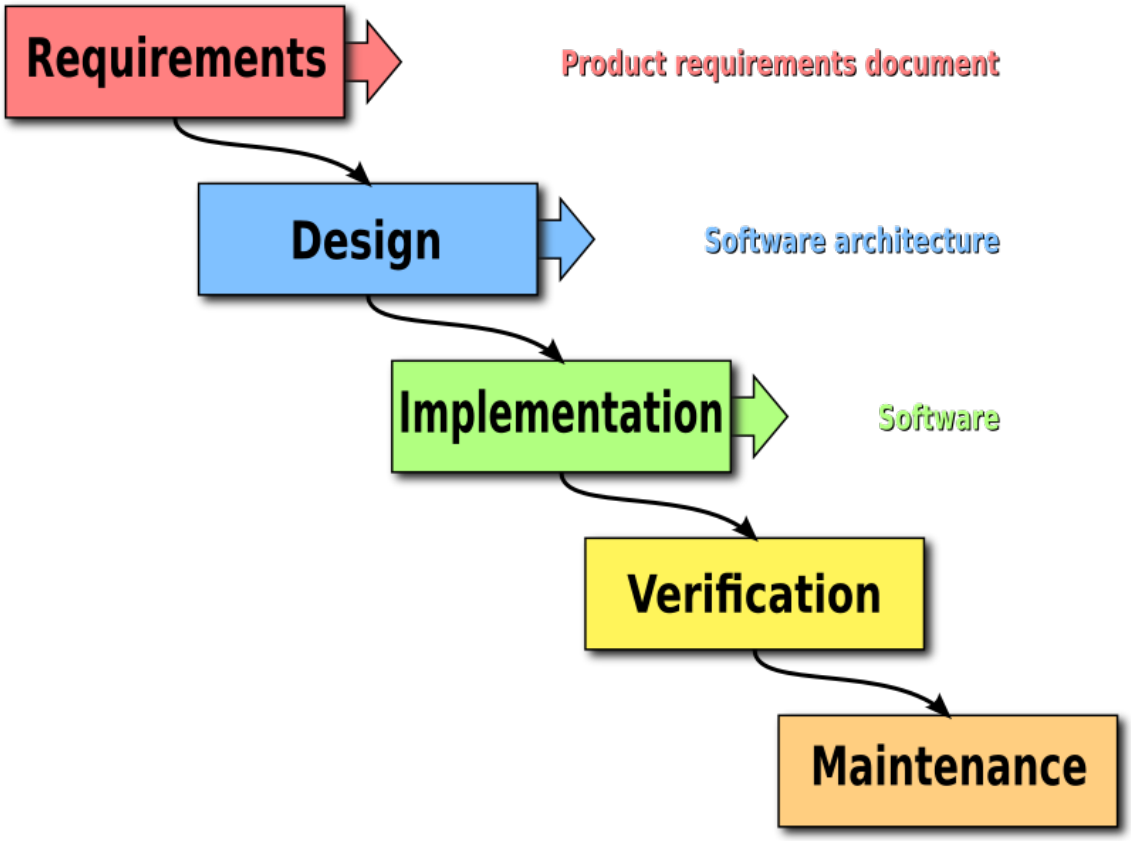


Figura 4.1: Estructura general de una etapa del desarrollo en cascada

4.2. Scrum

Scrum es una metodología ágil definida inicialmente por Hirotaka Takeuchi y Ikujiro Nonaka en 1986 [2]. Su relativa sencillez, así como su flexibilidad y orientación al trabajo en equipo, hace que Scrum sea una de las metologías más usadas en el desarrollo de software, sobre todo en entornos laborales donde el equipo de desarrollo no es gigante (aunque su implementación también es posible en estos casos).

4.2.1. Prácticas recomendadas y bases de Scrum

Uno de los aspectos esenciales de scrum es el "sprint", que se trata de un bloque temporal de tiempo (generalmente de entre 2 y 4 semanas) donde el equipo de desarrollo trabajará para llegar a un objetivo determinado, generalmente acabar todas las tareas definidas para ese sprint. Al principio de cada sprint el "product owner" y "scrum master" definirán las

[2] https://cb.hbsp.harvard.edu/cbmp/product/86116-PDF-ENG

tareas que se querrán realizar en dicho sprint y que se encuentran en el backlog[3]. Todos los integrantes del grupo de desarrollo, así como el propio product owner y scrum master, decidirán cuánto esfuerzo o tiempo llevará realizar cada tarea (hay algunas formas de decidir esto, como planning poker, pero no es muy relevante en nuestro caso), dividiéndola en tareas menores en caso de que sea demasiado grande. También se decidirá quién hará qué.

Durante cada día del desarrollo hay una pequeña reunión llamada "sprint stand-up" donde cada desarrollador 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 se tener en cuenta de cara al futuro. De esta manera, cada nuevo sprint será más preciso (dado que sabremos la cantidad de trabajo que cada desarrollador es capaz de hacer en el periodo definido de tiempo) e incluirán el "feedback" de los propios programadores.

Estas son las características eseciales de Scrum, pero al tratarse de una metodología ágil, nada es inamovible. Dependiendo del equipo y de las necesidades del mismo, se pueden cambiar o introducir nuevas ideas que mejoren el proceso para ese grupo de programadores en concreto.

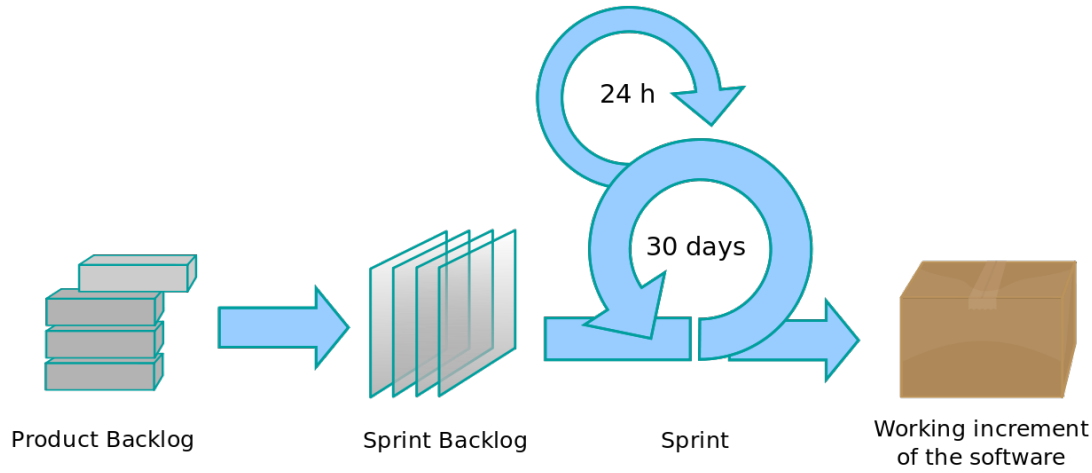


Figura 4.2: Proceso concreto de scrum. En este caso el sprint dura 4 semanas

[3]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. Generalmente el product owner es el encargado de organizar y decidir las tareas que tienen más prioridad

4.2.2. Valores de Scrum

4.2.2.1. 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 es esencial. Para logar 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.

4.2.2.2. Coraje

Dado que Scrum es una metodología de equipo, la ayuda entre cada uno de los miembros es algo esencial. De la misma forma, cada persona debe de ser capaz de enfrentarse a nuevos retos y asumir nuevas responsabilidades para que, en conjunto, el producto final sea el deseado.

4.2.2.3. Compromiso

Cada integrante tiene que saber lo que puede hacer y comprometerse a ello. Una vez la reunión inicial se haya completado, es importante que cada uno sepa lo que tiene que hacer y pregunte en caso de que no tenga algo claro o crea que no puede llegar a terminar lo especificado durante la reunión. Cada programador debe de comprometerse con lo establecido para lograr el éxito del equipo.

4.2.2.4. Sinceridad

Ser capaz de asumir los errores es la clave para mejorar y evolucionar como un equipo. 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. Asumir responsabilidades, errores y pedir ayuda cuando sea necesario debe de ser una práctica habitual en Scrum.

4.2.2.5. Respeto

Similar al anterior punto. Al trabajar en equipo es imprescindible que tanto los logros como los fracasos se tomen como una nueva forma de aprender y mejorar, por lo que el respeto mutuo y asumir los errores es la mejor forma de evolucionar, tanto individualmente como en equipo.

4.3. Metodología elegida

Mayoritariamente hemos usado una metodología en cascada con algunos elementos de Scrum.

Para las características grandes del proyecto como la creación de los mapas, enemigos, sistema creación de frases, etc. nos hemos basado en 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 de nuestra parte como de la gente que lo ha probado), tiene sentido mantener un backlog con todas las nuevas ideas y *feedback* recogido, algo de lo que Scrum es maestro. También hemos seguido la idea de la creación de tareas cortas (de no más de 8 horas, en caso de que haya algo de mayor cantidad deberá de ser dividido en subtareas) y sprints, donde cada mes intentaremos tener un elemento del juego finalizado con varias tareas finalizadas a final de cada semana. De esta manera somos capaces de seguir más fácilmente el progreso que hemos realizado y tenemos constancia de los nuevos elementos que queremos implementar en un futuro, así como los bugs que 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 realizado del proyecto, por lo que seguir una metodología ágil como scrum es el mejor paso a seguir para terminar las tareas más relevantes lo antes posible.

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

Capítulo 5

Planificación y Seguimiento

En este capítulo detallaramos la planificación de cada una de las iteraciones que hemos hecho, así como el seguimiento realizado. La elaboración de este proyecto se llevó a cabo durante dos años, comenzando en junio de 2014 y teniendo un par de parones durante varios meses (desde septiembre de 2014 hasta enero de 2015 y desde mayo de 2015 hasta septiembre del mismo año). Por lo tanto, los periodos de actividad serían los siguientes:

- 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.

5.1. 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) es cuando me mudé a Holanda y empiecé mi *internship*, por lo que las jornadas en días laborables solamente constaban de una o dos horas y alrededor de 8/9 horas durante los fines de semana.

Sumando todo el tiempo empleado durante estas semanas se calcula que se han empleado 190 horas en total.

Durante este tiempo nos centramos en recabar información general para poder empezar el proyecto con buen pie, así como la implementación de los mapas y habitaciones.

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

Desde el 4 de junio hasta el 29 de junio.

Análisis de requisitos generales Al desarrollar un proyecto enfocado a una parte de la población de la 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 usar las herramientas que utilizan diariamente para recabar ideas). Àsí mismo, desarrollar un videojuego puede llegar a ser una tarea infinita, dado que nuevas características o ideas que añadir es algo que sucede casi diaremente. Aprender sobre lo básico del género y poner límites es fundamental para centrar los pocos recursos que tenemos en crear lo completamente necesario.

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 juego. Este primer boceto cambiará a medida que querramos añadir nuevos elementos y querramos especificar más otros.

Búsqueda de bibliotecas que se adapten a nuestros requisitos Hay varias bibliotecas con las que se puede crear una interfaz gráfica sencilla como la de Rogue, pero todas ellas tienen sus ventajas e inconvenientes. Debemos de averiguar cuál de ellas es la más adecuada para nuestro caso.

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

5.1.1.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

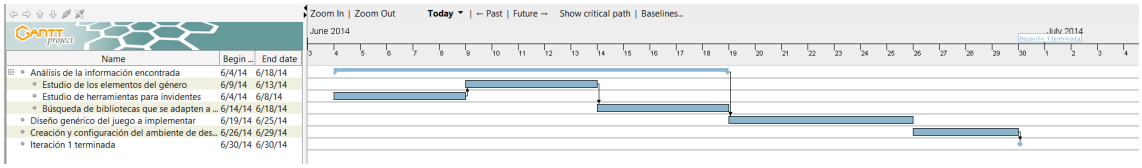


Figura 5.1: Diagrama de Gantt de la primera iteración de la primera etapa

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 y especificación de lo que queremos en nuestro caso.

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 ambiente de desarrollo para poder empezar la implementación.

Para la realización de todas estas tareas se planificaron 65 horas en total. Esta estimación se cumplió, por lo que al final de la iteración todas las tareas habían sido realizadas. Ver la figura **??**

5.1.1.2. Qué hemos conseguido en esta iteración

Comenzar un nuevo proyecto y recabar toda la información necesaria para tener una idea donde asentar las bases en las que se sembrarán las próximas iteraciones siempre es una tarea ardua y complicada. En esta iteración hemos sentado estas bases, definiendo lo que debemos de realizar y creando un primer boceto de lo que tendremos que cumplir durante los próximos meses.

5.1.1.3. Qué queremos conseguir en la próxima iteración

Con todo lo básico definido, ahora debemos de materalizarlo. Durante las siguientes iteraciones deberemos de empezar a crear el juego en sí, empezando por los mapas y habitaciones.

5.1.2. Segunda iteración: Generación de mapas y habitaciones

Desde el 15 de julio hasta el 24 de septiembre.

Análisis de requisitos de los mapas y habitaciones Hay mucho escrito y realizado sobre la creación de mapas y habitaciones de forma aleatoria. En base toda esta información, debemos de elegir lo que queremos realizar en base a, 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 aletoriedad que queremos (completamente aleatorio o pseudo-aleatorio).

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

Creación de tests que cubran lo analizado Tal y como comentamos anteriormente, antes de ponernos con la programación, crearemos los tests que especifiquen y cumplan lo diseñado para, posteriormente, empezar con la codificación.

Implementación del diseño de mapas y habitaciones Con el diseño y los tests creados, es hora de la implementación del código analizado y diseñado.

5.1.2.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

WBS 2.1 Análisis de requisitos de los 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. La estimación no fue correcta y no fuimos capaces de finalizar todo lo necesario, por lo que tuvimos que dejar la tarea de representar este mapa para la siguiente iteración. Ver la figura **??**

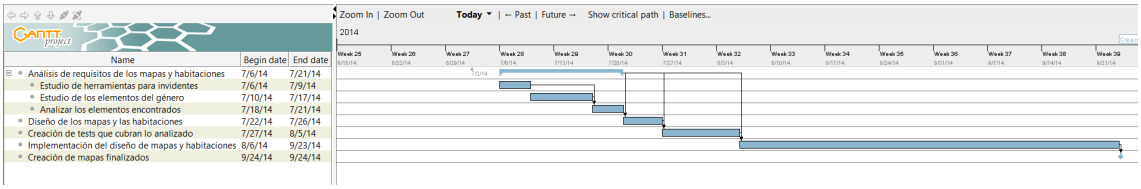


Figura 5.2: Diagrama de Gantt de la segunda iteración de la primera etapa

5.1.2.2. Qué hemos conseguido en esta iteración

Hemos conseguido generar mapas y habitaciones de manera aleatoria, pero todavía no mostrarlos en la interfaz de usuario. Uno de los puntos esenciales del género de los *roguelike* es su aleatoriedad y conseguir que los mapas es el primer gran punto.

5.1.2.3. Qué queremos conseguir en la próxima iteración

En la siguiente iteración debemos de 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 es otro de los puntos esenciales del género.

5.2. Enero 2015 - Mayo 2015

Este periodo comienza el 2 de enero, tras los días libres de navidad y nuevo año, y termina el 31 de mayo, en el que pararemos para preparar los exámenes. Por lo tanto, este periodo consta de un total de 21 semanas. Durante estas semanas he estado trabajando a tiempo completo y algunos días se los dediqué a estudiar para los exámenes, por lo que el tiempo semanal empleado fue de alrededor de 12 horas de media, aunque algunas de estas semanas fueron de más trabajo que otras.

En total, 170 horas fueron dedicadas para este sprint inicialmente.

Durante este sprint nos hemos centrado en continuar lo realizado anteriormente y, en general, seguir con el desarrollo del juego en sí.

5.2.1. 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.

Mostrar los mapas y habitaciones en el interfaz de usuario En la anterior fase implementamos los mapas, pero no los enseñamos en la interfaz de usuario. En esta ocasión debemos de acabar con esta tarea para terminar con todo lo relacionado con la creación de mapas.

Análisis de requisitos para sobre los objetos En un *roguelike* los objetos son primordiales. Debemos de investigar qué objetos vamos a usar y cómo los representaremos en el mapa diseñado anteriormente.

Crear un diseño simple sobre cómo los objetos interactuarán con el mapa Debemos de crear un diseño genérico que nos permita añadir 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 que interactuen 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.

5.2.1.1. 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 1.1.1** Buscar información sobre los diferentes tipos de objetos necesarios en el juego.
- **WBS 1.1.2** Estudiar cómo estos objetos deben de interactuar con el mapa.
- **WBS 1.1.3** 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.

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. Ver la figura **??**

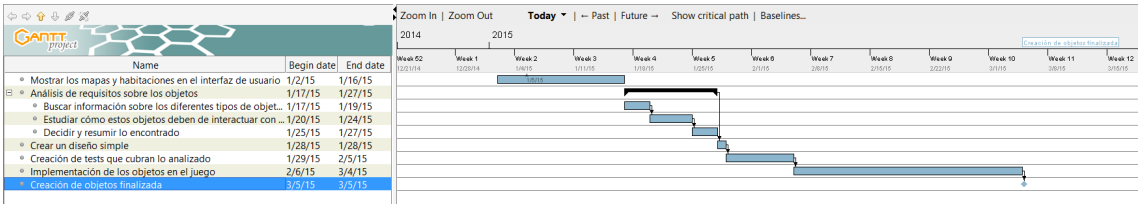


Figura 5.3: Diagrama de Gantt de la primera iteración de la segunda etapa

5.2.1.2. Qué hemos conseguido en esta iteración

Terminar la tarea restante del sprint anterior y la creación de diferentes objetos (pociones, armas y armaduras) que los personajes podrán utilizar. También podemos representar dichos objetos en el mapa, además de que la creación de nuevos elementos sea muy sencilla.

5.2.1.3. Qué queremos conseguir en la próxima iteración

Crear los personajes. Ésta es uno de los últimos pilares esenciales del género, por lo que tomará prioridad y será realizado a continuación.

5.2.2. Segunda iteración: análisis, diseño, tests e implementación de los personajes

La segunda iteración comienza el 5 de marzo y acaba el 23 de mayo.

Análisis de requisitos para sobre personajes En el juego tendremos un personaje principal (controlado por el usuario) y una serie de enemigos con diferentes características que intentarán destruirle y que el usuario deberá de batir. En esta tarea nos encargaremos de buscar información sobre el tipo de enemigos a crear y diferentes métodos para hacerlo de forma más genérica posible, dado que ser capaz de crear estos enemigos fácilmente es una característica fundamental.

Creación del diseño de los personajes Una vez hayamos realizado el análisis para obtener la idea de los enemigos y personajes a usar, es hora de crear el diseño. Tal y como hemos comentado anteriormente, es necesario hacer un diseño extendible donde crear diferentes clases 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.

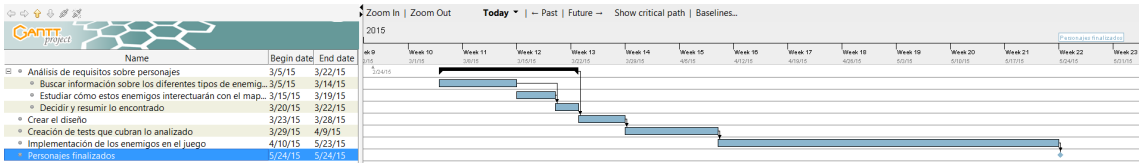


Figura 5.4: Diagrama de Gantt de la segunda iteración de la segunda etapa

Programación de lo diseñado y analizado previamente Con el análisis, diseño y tests listos, podremos realizar la tarea de implementación.

5.2.2.1. 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.

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 este *sprint* se han asignado 80 horas, las cuales fueron suficientes para cubrir todo lo que se quiso realizar. Ver la figura **??**

5.2.2.2. Qué hemos conseguido en esta iteración

Al igual que con los objetos, hemos conseguido facilitar su creación, así como la representación del mapa de los mismos. Hemos creado un personaje principal (controlado por el usuario) y diferentes tipos de enemigos (doblins, ratas y dragones). Ambos tipos se muestran en el mapa.

5.2.2.3. Qué queremos conseguir en la próxima iteración

Tal y como tenemos el proyecto actualmente, tenemos el mapa creado aleatoriamente y podemos mostrar objetos y personajes con facilidad. El siguiente paso consiste en que los personajes puedan interactuar con los objetos.

5.2.3. 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.

En la anterior iteración teníamos el objetivo de crear los personajes y enemigos y que éstos fueran capaces de ser representados en la interfaz gráfica. En este caso daremos un paso más allá y deberemos de añadir funciones que dichos personajes puedan realizar con los objetos: creación de un inventario para que puedan almacernarlos y, de tal modo, el usuario y 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).

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 crear nuevas funciones que ayuden a la interacción entre los objetos y los personajes, tal y como hemos descrito anteriormente.

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

5.2.3.1. 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 creados.

Al ser un pequeño sprint, hemos sido capaces de terminarlo sin problema. El diagrama de Gantt puede verse en la figura **??**

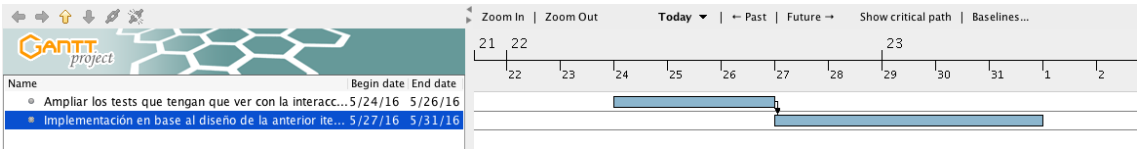


Figura 5.5: Diagrama de Gantt de la tercera iteración de la segunda etapa

5.2.3.2. Qué hemos conseguido en esta iteración

Interacción básica entre personajes y objetos.

5.2.3.3. Qué queremos conseguir en la próxima iteración

Aumentar esta interacción (que podamos atacar a los personaje, por ejemplo) y 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.

5.3. Septiembre 2015 - Abril 2016

Este periodo empieza el 1 de septiembre, tras un breve descanso en verano, y termina el 31 de abril. Durante este tiempo hemos empleado más tiempo que anteriormente en el proyecto y desarrollado mucho más (20 horas a la semana). También hemos buscado *feedback* una vez tuvimos tiempo disponible para poder tener tiempo para implementarlo y asegurarnos de que lo que hemos realizado es aceptado por la comunidad. Este periodo consta de un total de 34 semanas.

En total, 680 horas fueron dedicadas para este sprint inicialmente.

5.3.1. 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, es decir, damos 5 semanas para su realización.

Aumentar la interacción entre objetos, personajes y mapa Hasta el momento tenemos el mapa, objetos y personajes, pero debemos de ser capaces de interactuar completamente y aumentar en características. Por ejemplo añadiendo puertas entre habitaciones, que también se muestren en el mapa, que los personajes puedan atacarse entre sí (teniendo en cuenta las armaduras y armas que llevan) y la creación de un campo de visión para el usuario para que solamente pueda ver lo que hay a su alrededor y no todo el mapa.

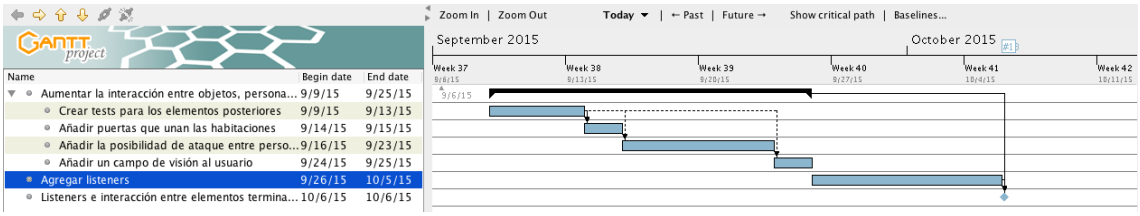


Figura 5.6: Diagrama de Gantt de la primera iteración de la tercera etapa

Agregar *listeners* para que el jugador pueda decidir lo que hacer En este momento el jugador no puede hacer nada. Ahora es el momento de introducir los elementos necesarios para que nos podamos mover por el mapa y realizar todas las acciones implementadas anteriormente con nuestro personaje.

5.3.1.1. 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 pasar de habitación a habitación.
- **WBS 1.1.3** Añadir la posibilidad de ataque entre personajes.
- **WBS 1.1.4** Añadir un campo de visión al usuario para que no sea capaz de ver todo el mapa, pero solamente cierta área a su alrededor.

WBS 1.2 Agregar *listeners* para que el jugador pueda decidir lo que hacer.

Hemos asignado 100 horas para la estimación de este *sprint*, las cuales cumplimos sin problema. Ver la figura **??**

5.3.1.2. Qué hemos conseguido en esta iteración

Al acabar esta iteración tenemos lo básico de un juego: nos podemos mover por un mapa, coger objetos y combatir enemigos (aunque estos enemigos todavía no tienen inteligencia artificial, por lo que no se mueven).

5.3.1.3. Qué queremos conseguir en la próxima iteración

En la próxima iteración deberemos terminar lo básico del juego. Para ello tendremos que implementar el sistema de portales (que nos transportarán a un mapa completamente

diferente, consiguiendo puntos de esta manera), añadir básica inteligencia artifial 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 hemos explicado en la sección

5.3.2. 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 finalización.

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 está en encontrar el portal dentro del mapa y, una vez encontrado, un nuevo mapa será generado, igual que otro portal en ese mapa. De esta manera la meta del juego se centra en derrotar enemigos (consiguiendo mejores armas y armaduras en el proceso) para encontrar el mayor número de portales posibles. Deberemos de diseñar e implementar estos portales en esta iteración.

Añadir inteligencia artificial Dependiendo del enemigo al que nos enfrentemos, éste se comportará de forma distinta. Habrá enemigos que sean activos y vayan contra el usuario, mientras que otros serán pasivos y no quieran atacar al usuario. Puede ser que en el futuro queramos incrementar el número de este tipo de comportamientos, por lo que deberemos de diseñarlo de la mejor forma.

Añadir opciones para cambiar el color de la interfaz gráfica El punto central de este proyecto es la accesibilidad e incluir una opción para que daltónicos puedan disfrutar de nuestro juego es decisivo.

5.3.2.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

WBS 2.1 Diseño e implementación de los portales.

- **WBS 2.1.1** Diseñar la mejor manera para incluir los portales en nuestro diseño (en el análisis es algo que se había considerado hacer).
- **WBS 2.1.2** Creación de los tests.
- **WBS 2.1.3** Implementación de los portales.

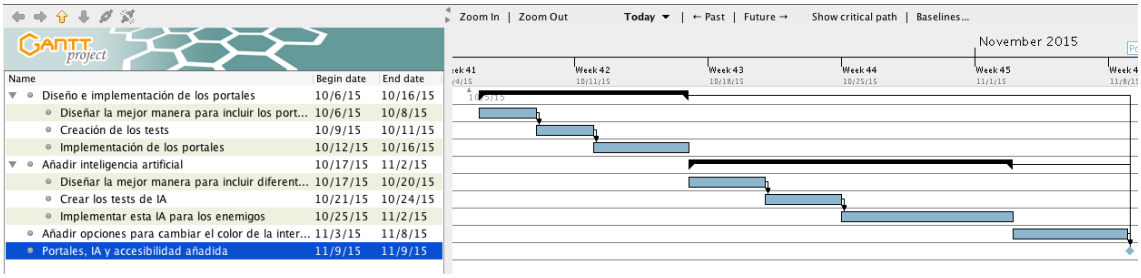


Figura 5.7: Diagrama de Gantt de la segunda iteración de la tercera etapa

WBS 2.2 Añadir inteligencia artificial.

WBS 2.2.1 Diseñar la mejor manera para incluir diferentes tipos de IA.

WBS 2.2.2 Crear los tests de IA.

WBS 2.2.3 Implementar esta IA para los enemigos.

WBS 2.3 Añadir opciones para cambiar el color de la interfaz gráfica para usuarios daltónicos

48 horas fueron las que creímos suficientes para la estimación del *sprint* y que fueron necesarias para la conclusión del mismo con todas las tareas terminadas. Ver la figura **??** como referencia.

5.3.2.2. Qué hemos conseguido en esta iteración

Con esta última iteración tenemos la parte básica del juego completada. Hay un objetivo, enemigos con IA, diferentes objetos, puertas que conectan distintas habitaciones y varias acciones que se pueden realizar.

5.3.2.3. Qué queremos conseguir en la próxima iteración

Con el juego en sí completado (o por lo menos la parte primordial), es hora de empezar a analizar y diseñar la parte de la generación automática del lenguaje.

5.3.3. 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.

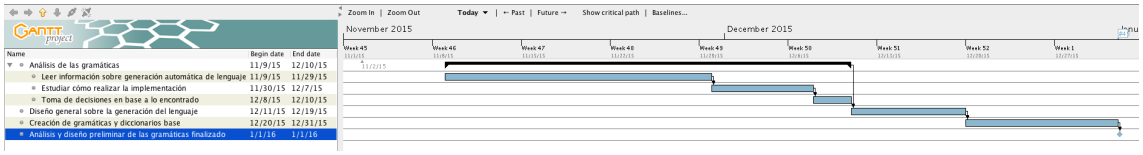


Figura 5.8: Diagrama de Gantt de la tercera iteración de la tercera etapa

Análisis de las gramáticas Una parte muy importante del juego es crear frases que sean generadas de forma automática y pseudo-aleatoria en base a una serie de gramáticas y diccionario dado. Si conseguimos realizar esto, crear diferentes idiomas sería trivial (solamente tendríamos que cambiar la gramática en caso de que en dicho idioma sea diferente y traducir el diccionario). Para ello tendremos que investigar cómo podemos hacer esto en nuestro caso y cómo han solventado este problema otra gente.

Diseño general sobre la generación del lenguaje Una vez hemos analizado y decidido lo que debemos de realizar, crearemos un diseño lo más simple y adaptable posible para la generación de dichas frases en base a las gramáticas dadas.

Creación de gramáticas y diccionarios base Para empezar el desarrollo necesitamos tener una gramática y diccionaro base con lo que poder ver los diferentes resultados obtenidos.

5.3.3.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

WBS 3.1 Análisis de las gramáticas.

- **WBS 3.1.1** Leer la información que existe sobre la generación automática de lenguaje.
- **WBS 3.1.2** Estudiar cómo lo podemos implementar en nuestro proyecto con lo ya existente.
- **WBS 3.1.3** Tomar la decisión en base a lo encontrado y sentar las bases sobre ello.

WBS 3.2 Diseño general sobre la generación del lenguaje y cómo las gramáticas interactuarán con nuestro programa.

WBS 3.3 Creación de gramáticas y diccionarios base para tener una base con lo que testar lo que implementaremos.

Hemos reservado 130 horas para esta iteración y hemos conseguido terminar todas las tareas asignadas a tiempo. Ver la figura **??** para más información.

5.3.3.2. Qué hemos conseguido en esta iteración

Hemos sentado las bases de lo que queremos implementar con las gramáticas.

5.3.3.3. Qué queremos conseguir en la próxima iteración

Empezar con los tests y desarrollo de lo investigado en esta iteración.

5.3.4. 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, tiene 2 semanas.

Análisis sobre cómo crear las gramáticas NP Las gramáticas NP[1] son básicas, pero no iguales en todos los idiomas. Debemos de estudiar qué planteamientos existen que faciliten su implementación de la mejor forma posible.

Diseño de las gramáticas NP Las frases de sintagma nominal serán las más usadas dentro de nuestro juego. No solamente serán utilizadas para la generación de las frases, pero también a la hora de representar los elementos en la interfaz de usuario. Por ello debemos de crearlas de la forma más extendible y accesible que podamos.

Creación de los tests para las gramáticas NP Al igual que en otras iteraciones, realizamos los tests antes de nada.

Implementación de las gramáticas NP Con los tests realizados, debemos de empezar con la implementación. Tenemos que tener en cuenta que hay que ser capaces de sustituir las clases de palabras por las palabras en sí que se encuentra en el diccionario. También hay que tener en cuenta las restricciones posibles (por el momento solamente contamos con las restricciones de inglés. Es decir, número).

5.3.4.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

[1] Sintagma nominal.

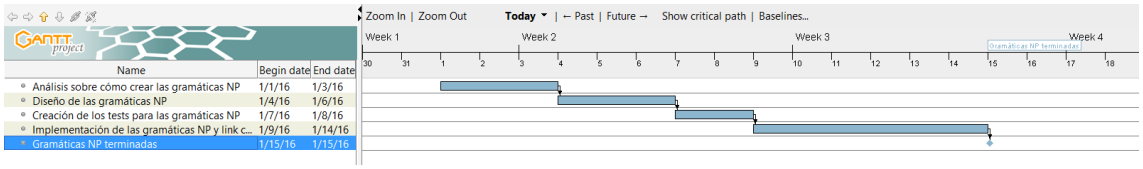


Figura 5.9: Diagrama de Gantt de la cuarta iteración de la tercera etapa

WBS 4.1 Análisis sobre cómo crear las gramáticas NP de forma genérica y fácilmente extendibles para otros idiomas.

WBS 4.2 Diseño de las gramáticas NP.

WBS 4.3 Creación de los tests para las gramáticas NP.

WBS 4.4 Implementación de las gramáticas NP y *link* con el diccionario.

Hemos reservado 40 horas para esta iteración y terminamos todas las tareas a tiempo. Ver la figura **??** para más información.

5.3.4.2. Qué hemos conseguido en esta iteración

Al finalizar esta iteración hemos conseguido generar sintagmas nominales de forma pseudo-aleatoria que basan su información en las gramáticas y diccionarios dados.

5.3.4.3. Qué queremos conseguir en la próxima iteración

Aumentar estas gramáticas para que lo generado no solamente sean sintagmas nominales, pero también frases que usen estos sintagmas nominales.

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

Esta segunda iteración comienza el 15 de enero y termina el 29 de enero. Es decir, tiene 2 semanas y media de duración, igual que la iteración anterior.

Análisis sobre cómo crear las gramáticas complejas de forma genérica Una vez somos capaces de crear sintagmas nominales, deberemos de ser capaces de combinarlos con otros elementos del lenguaje como verbos para crear frases que sean capaces de describir todo lo que realmente deseamos. Deberemos de analizar las diferentes formas de hacer esto.

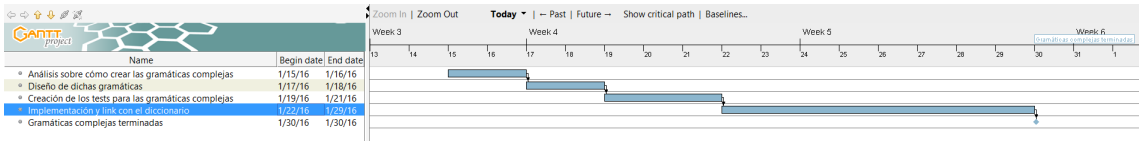


Figura 5.10: Diagrama de Gantt de la quinta iteración de la tercera etapa

Diseño de dichas gramáticas Una vez hemos 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.

Implementación y *link* con el diccionario Implementar las gramáticas más complejas que se comuniquen con las gramáticas NP y nos permitan generar frases que describan lo que ocurre en el juego de forma aleatoria y automática.

5.3.5.1. 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, al igual que su relación con las NP y diccionario.

WBS 5.4 Implementación y *link* con el diccionario.

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

5.3.5.2. Qué hemos conseguido en esta iteración

Tener gramáticas que generen frases de manera pseudo-aleatoria para todas las descripciones que contemplamos actualmente en el juego.

5.3.5.3. Qué queremos conseguir en la próxima iteración

Las frases son generadas, pero todavía no las mostramos en ninguna parte de la interfaz gráfica, por lo que el jugador no puede ni verla ni escucharla. En el siguiente sprint tenemos que añadir esta opción, además de añadir otros idiomas (es decir, añadir las gramáticas y diccionarios) como el gallego y español, lo que significará añadir las restricciones adicionales que tienen de género.

5.3.6. 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 dará comienzo el 30 de enero y terminará el 6 de febrero. Es decir, tiene solamente 1 semana de desarrollo, pero fue una semana 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 las mostramos. Ahora es el momento de mostrar en una ventana la frase generada.

Implementación de las restricciones para el resto de idiomas En inglés solamente tenemos la restrición en número, pero otros idiomas, como el castellano o gallego, tienen un restricciones adicionales como el 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 en tener el proyecto en la mayor cantidad posible de idiomas, por lo que añadirlo en gallego y castellano es importante.

Crear primer vídeo para recibir *feedback* Al terminar esta iteración tendremos la base del proyecto completada. Por ello crearemos un vídeo mostrando el punto en el que estamos y con el que recibiremos comentarios con lo que podremos crear mejoras en las futuras iteraciones [2].

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 de realizar en base a lo que está especificado en el diagrama de casos de uso.

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

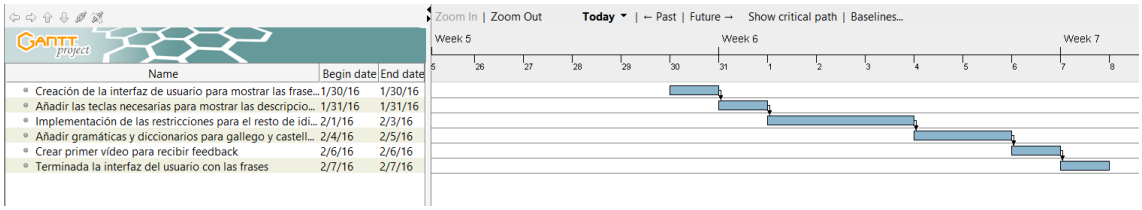


Figura 5.11: Diagrama de Gantt de la sexta iteración de la tercera etapa

5.3.6.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

WBS 6.1 Creación de la interfaz de usuario para mostrar las frases 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 Implementación de las restricciones para el resto de idiomas.

WBS 6.4 Añadir gramáticas y diccionarios para gallego y castellano.

WBS 6.5 Crear primer vídeo para recibir *feedback*.

En la figura **??** se muestra el diagrama de Gantt de esta iteración. Hemos reservado, tal y como ya hemos comentado, 56 horas, que fueron suficiente para terminar todas las tareas mencionadas.

5.3.6.2. Qué hemos conseguido en esta iteración

Terminar lo básico del juego, generando las frases necesarias al usuario y añadiendo más idiomas disponibles.

5.3.6.3. Qué queremos conseguir en la próxima iteración

Debemos de esperar por los comentarios recibidos en el vídeo y, mientras, podremos mejorar diferentes aspectos y características del juego haciendo las tareas que hay en el *backlog*.

5.3.7. 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 y 40 horas en total para la

iteración.

Solucionar bug donde 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.

Cambiar las teclas que usamos por defecto Las teclas que tenemos por defecto no son del todo intuitivas. Deberemos de cambiarlo para que sí lo sean.

Hacer los adjetivos que definen los personajes variables Si un enemigo tiene poca vida y el personaje que controla el usuario tiene mucha más vida, habrá la posibilidad de que el adjetivo a usar sobre el enemigo sea "pequeño", "insignificante"... mientras que si es al revés, se usará "grande", "poderoso", de forma que se verá al enemigo de forma diferente.

Añadir opción para que las descripciones sean descriptivas o numéricas Cuando escuchamos una descripción, a veces queremos que dicha descripción sea numérica, es decir, que mencione exactamente la vida o posición de los personajes. Otras veces deseamos que esto no sea así y que todas las descripciones se hagan de una forma más poética y se usen palabras para las definiciones. Crearemos una opción en la que el usuario será capaz de seleccionar lo que prefiera.

5.3.7.1. Tareas y seguimiento

La descomposición de las tareas es la siguiente:

WBS 7.1 Solucionar bug donde la pantalla no se refresca cuando es necesario.

WBS 7.2 Cambiar las teclas que usamos por defecto.

WBS 7.3 Cambiar los adjetivos que definen los personajes dependiendo de la vida de dichos personajes.

WBS 7.4 Añadir opción para que las descripciones sean descriptivas o numéricas, dependiendo de lo que el usuario desee.

En la figura **??** se muestra el diagrama de Gantt de esta iteración.

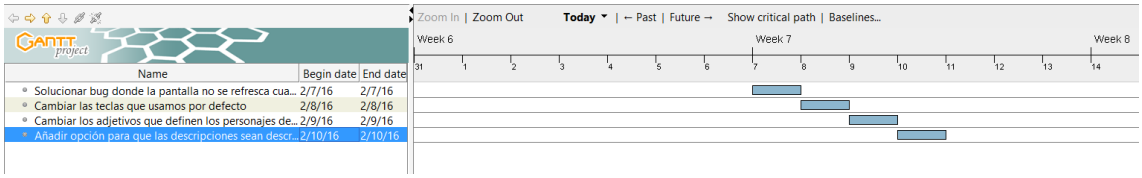


Figura 5.12: Diagrama de Gantt de la séptima iteración de la tercera etapa

5.3.7.2. Qué hemos 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 detectados mientras jugábamos.

5.3.7.3. Qué queremos 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 analizarla e implementar los cambios que nos piden.

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

En esta iteración implementaremos los cambios más importantes sobre el *feedback* recibido. Dado que estos comentarios pueden llevar bastante tiempo (quizás haya más cambios o los realizados desencadenan más ideas a implementar), es mejor terminarlos en menor tiempo posible para poder recibir *feedback* más rápidamente. Por este motivo esta iteración contará solamente con 4 días y 20 horas en total, empezando el 22 y acabando el 26, que es cuando crearemos el vídeo.

5.3.8.1. Feedback recibido

Todo este feedback es el recibido por parte de mis supervisores que a su vez se basaron en lo que ellos mismos y otros usuarios les han comentado.

Tener en cuenta la persistencia del tiempo Si matamos a un enemigo, sería una buen adición que las descripciones lo tuvieran en cuenta. Por ejemplo a la hora de enumerar los elementos que hay alrededor del usuario, se podría comentar que también se encuentra un “goblin” muerto.

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 algo que ha sucedido en el juego), mostrábamos una nueva ventana con dicha frase, que tendríamos que cerrar en cada ocasión. Esto es un inconveniente para aquellas personas que sí pueden ver y no quieren ser molestados por este tipo de ventanas y para los usuarios invidentes tampoco es la mejor solución disponible, dado que no es lo que se suelen encontrar en otros juegos del género. Lo que mayoría de juegos usan es 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 servirá como un log. Lo importante está en cambiar el "foco" del juego a esta ventana cada vez que una frase sea generada para que el lector sea capaz de leerla.

Pequeños cambios en los adjetivos usados En algunas ocasiones usábamos adjetivos poco comunes a la hora de definir ciertos nombres.

Adición de niveles y experiencia Los enemigos, armas y el propio usuario deberían de tener niveles para que el juego escale en dificultad. Cada vez que se destruye un enemigo, el personaje del usuario ganará una serie de puntos que serán usados para subir niveles.

Cambio en la aleatoriedad Hasta ahora, todo lo generado era aleatorio (el tipo de enemigos que nos encontrábamos, los objetos que soltaban, los propios objetos encontrados...). 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: **??**

Evitar la repetitividad Algunas de las frases que generamos tienden a ser bastante repetitivas (el héroe desequipa la espada, el héroe desequipa la armadura...), en vez de tener un lenguaje más natural (el héroe desequipa la espada, la armadura...). Esto sucede en un par de casos y debemos de tratarlo.

Darle nombre al héroe Siempre nos referimos al personaje que controla el usuario como "héroe" o por su pronombre ("él"). Podríamos darle un nombre o dejar que el usuario elija para dar una mayor variedad.

Mostrar una serie de estadísticas al cambiar de mazmorra Cuando pasamos de una mazmorra a otra usando el portal, podríamos mostrar una serie de estadísticas de los enemigos batidos, la cantidad de experiencia obtenida, el nivel actual del usuario...

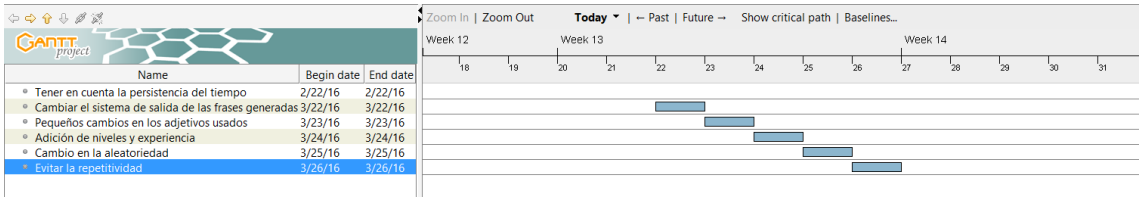


Figura 5.13: Diagrama de Gantt de la octava iteración de la tercera etapa

A mayores, crearemos otro vídeo informando sobre los cambios que hemos realizado en base a estos comentarios [3].

5.3.8.2. Tareas y seguimiento

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

WBS 8.1 Tener en cuenta la persistencia del 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 Evitar la repetitividad.

En la figura **??** se muestra el diagrama de Gantt de esta iteración.

5.3.8.3. Qué hemos conseguido en esta iteración

Hemos conseguido implementar los cambios más importantes en base a los comentarios recibidos.

5.3.8.4. Qué queremos conseguir en la próxima iteración

Seguir implementando el resto de cambios y tener en cuenta el *feedback* del resto de usuarios.

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

Capítulo 6

Análisis de Requisitos globales

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

6.1. Consultas con la comunidad

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

Ambas consultas, así como el resultado obtenido de las mismas, se han especificado en la sección **??**

6.1.1. Resumen de las peticiones recibidas

Fácil cambio de color de la interfaz La interfaz gráfica tendrá diferentes colores que diferencien los distintos tipos de enemigos y elementos del juego. Tal y como hemos mencionado en la sección **??**, deberíamos de tener esto en cuenta a pesar de que dichos elementos son fácilmente diferenciables.

Descripciones automáticas de todos los elementos posibles 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 de generar estas descripciones de mejor manera posible para que no sean repetitivas.

Diferentes idiomas Al ser un videojuego en el cual el idioma es esencial, debemos de tener en cuenta la inclusión de otras lenguas.

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

6.2. Análisis de los elementos del juego

Al ser un *roguelike*, debemos de incluir lo básico del género (aleatoriedad, dificultad, progreso, etc.). Los detalles de los mismos, así como lo que hemos introducido en el proyecto sobre ellos, se encuentran en la sección **??**.

6.3. Requisitos del aplicativo

Con toda la información obtenida y analizada, creamos una lista con los casos de uso que nuestro proyecto debe de cumplir.

- **Movimiento** Un usuario siempre será capaz de moverse con su personaje a una posición válida.
- **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 a una distancia determinada del enemigo, éste será capaz de realizar un ataque mágico, siempre y cuando tenga sufiencience mana para el mismo.
- **Coger elemento** En las habitaciones del mapa habrá varios objetos que se pueden recolectar, así como enemigos que soltarán diferentes objetos.
- **Equipar elemento** Poder equipar un objeto que está en el inventario a nuestro personaje, siempre y cuando no tengamos un mismo objeto 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 no tendrá suficiente espacio para almacenar nuevos elementos, por lo que se podrá tirar objetos del inventario para hacer hueco a aquellos nuevos que queramos recoger.

- **Descripciones** Durante el transcurso del juego podremos generar diferentes descripciones dependiendo de lo que queramos saber. Por ejemplo lo que el usuario tiene en el inventario, las posiciones a las que nos podemos mover, la descripciones de los enemigos a los que nos enfrentamos, etc.
- **Activación de descripciones numéricas** Algunas de las descripciones nos comentarán posiciones o características de los enemigos que podremos querer escuchar numéricamente (por ejemplo la cantidad de vida de un cierto monstruo) o con palabras (mucha, poca, bastante...), dependiendo de lo que prefiramos.
- **Cambio de colores** Para gente que es daltónica hemos incluido una opción para cambiar el color de la interfaz gráfica.

Al realizar algunas de estas acciones (coger, equipar, desequipar y tirar un objeto, moverse y atacar), un turno será consumido. Al consumir un turno los enemigos realizarán el suyo, por lo que el jugador puede ser atacado o enemigos pueden acercarse a él.

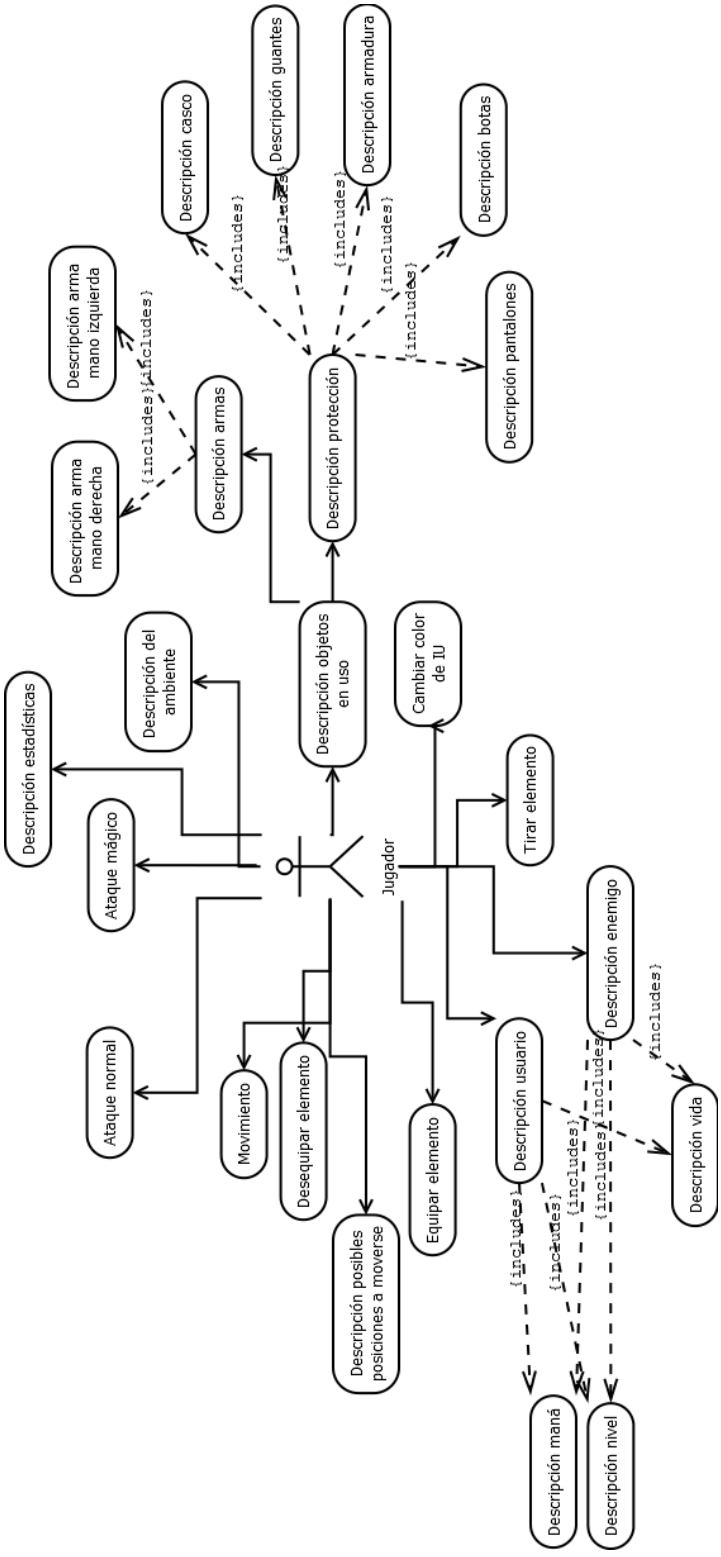


Figura 6.1: Diagrama UML de casos de uso del proyecto

Capítulo 7

Diseño e Implementación

En este capítulo mostraremos los detalles del diseño y la implementacion de diferentes partes del proyecto.

7.1. Gramáticas y frases

La parte de la generación automática de frases y definición de gramáticas es la más relevante del proyecto y la que la diferencia del resto de juegos de similares caracteríticas. Por ello consideramos esencial mostrar su funcionamiento.

Tal y como se muestra en la figura **??**, la parte esencial de esta sección consta de seis clases, cuyo funcionamiento detallarremos a continuación.

7.1.1. Explicación general del funcionamiento

Tenemos dos clases (GrammarSelectorS y GrammarSelectorNP) que se encargan de generar las frases de manera aleatoria en base al nombre o nombres sobre los que queremos obtener dicha frase. La clase GrammarSelectorNP crea sintagmas nominales mientras que GrammarSelectorS crea frases usando dichos sintagmas nominales. Ambas usan diccionarios y gramáticas en un idioma en concreto especificado en el archivo de configuración *language.properties*:

```
language=EN
```

Parte de la dificultad de generar estas frases está en que todos los elementos de la misma deben de coincidir en base a las restricciones del propio idioma. Por ejemplo, los nombres, verbos y determinantes de una frase en español deben de coincidir en género y

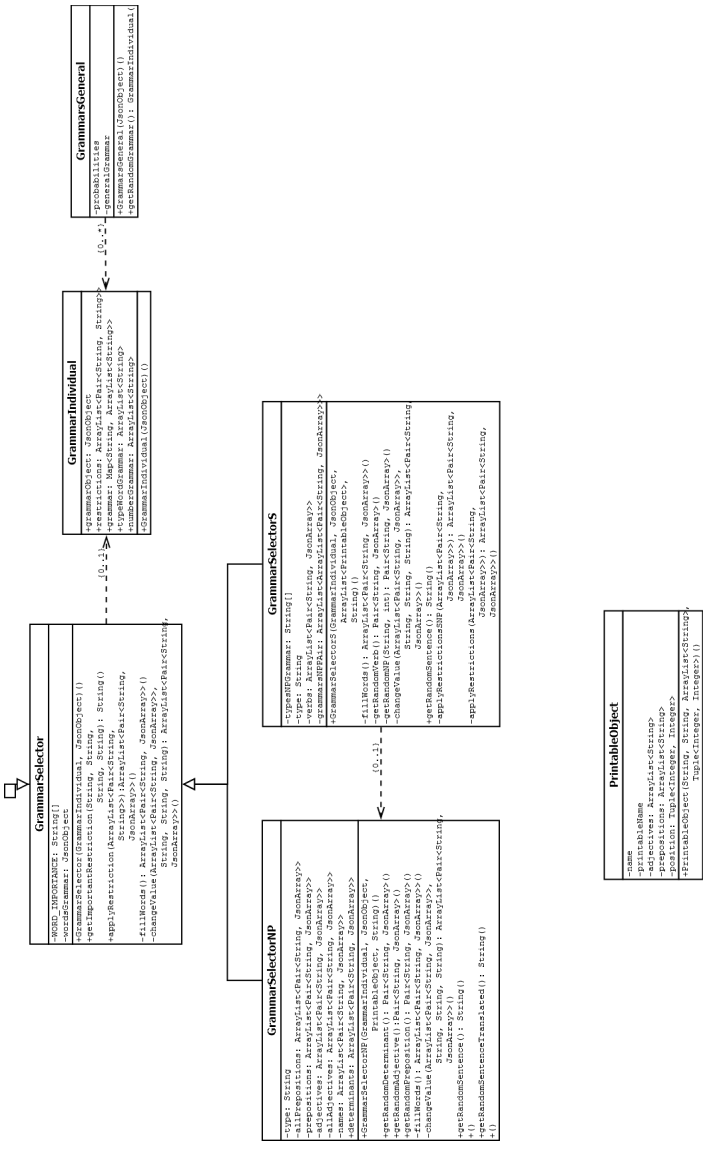


Figura 7.1: Diagrama de clases de las gramáticas

número. Este tipo de restricciones vienen dadas en 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 que tenga menor relevancia por la contraria dependiendo del tipo que sea (género o número):

```
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 algunas de las palabras de una frase, iteraremos entre toda ella buscando una solución hasta que la frase sea coherente.

La clase GeneralGrammar, por su parte, obtiene la información de varias gramáticas (que es como vienen dadas en el fichero, dado que para describir una misma situación puede haber varias gramáticas que funcionen de la misma forma) y dispone de una función que devuelve una gramática individual (de la clase GrammarIndividual), que es la que usaremos a la hora de generar la frase en sí. Es decir, cuando queremos generar una frase de una gramática en concreto (por ejemplo cuando un personaje ataca a otro), seleccionaremos una de estas gramáticas al azar de todas las que estén disponibles en el idioma que tenemos (esto lo realiza la clase GrammarsGeneral) y luego usamos esa gramática en particular con GrammarIndividual. De esta forma no siempre usamos la misma gramática y dependerá de la seleccionada aleatoriamente en GrammarsGeneral.

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 (usado a la hora de describir lo que hay alrededor de un personaje).

Cabe destacar que estas gramáticas, al igual que los diccionarios usados para cada uno de los idiomas, se obtienen en base a ficheros JSON dados.

7.1.2. *Input* de gramáticas y diccionario

Tal y como hemos mencionado anteriormente, tanto las gramáticas como los diccionarios son archivos JSON que el usuario puede cambiar y que afectará al resultado de las frases generadas por el juego.

Para las gramáticas tenemos dos archivos por idioma. Uno de ellos son las gramáticas con las que creamos sintagmas nominales, mientras que las otras creamos frases usando, en su mayor parte, estos sintagmas nominales.

7.1.2.1. Gramáticas: Sintagmas nominales

Ejemplo de gramáticas de sigtagma nominal en inglés:

```
"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 de la gramática (en este caso, DETADJN). Este nombre es usado por el programa y ayudará al usuario a entender el tipo de gramática o grupo de gramáticas que son.

El siguiente nombre es el nombre de la gramática en particular. No es usado en el programa, pero su uso es necesario para poder definir varias gramáticas sobre el mismo árbol (si queremos varias gramáticas de tipo DETADJN, necesitaremos que se especifiquen el nombre de cada una de ellas).

Luego viene el contenido relevante. Primero nos encontramos con *S*, que es donde encontraremos la definición de la gramática en sí; y luego *restrictions* que, como su nombre indica, es especificaremos las restricciones.

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 detallaremos las partes de la gramática que tienen que coincidir. En esta gramática solamente tenemos que el determinante, nombre y adjetivo tienen que ser iguales en número. Es decir, que "the red swords" no sería generada, sino que generaría "the red sword".

Este es un ejemplo en inglés, cuyas restricciones son más sencillas que en otros idiomas como el español o el gallego, donde el nombres, determinante y adjetivo no solamente tiene que coincidir número, pero también en género. Por este motivo, para generar el mismo tipo de gramática para estos idiomas, necesitaremos hacer lo 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"), pero también hemos cambiado el orden del nombre y adjetivo para que se adapten a ambos idiomas.

A la hora de cambiar las palabras disponemos de un array donde ordenamos las palabras por importancia. Un sustantivo es más importante que un determinante, por lo que si una de las restricciones no se cumple por culpa de algunos de estos dos elementos, el determinante será en que cambiará para adaptarse al nombre.

De esta manera podemos adaptar gramáticas a idiomas diferentes de una forma muy fácil y sin necesidad de tocar nada de código.

7.1.2.2. Gramáticas: Frases

Las gramáticas para la generación de frases también usa las gramáticas de sintagma nominal:

```
"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 idéntica a la mencionada anteriormente. En este caso vemos que dentro de "ATTACK" disponemos de dos gramáticas distintas (este 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 como "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.

7.1.2.3. Diccionarios

Al igual que las gramáticas, los diccionarios también están divididos por idiomas y son archivos JSON. Éste es un ejemplo de parte de una gramática en 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, en la clave, se menciona la palabra en sí y el resto de elementos serán un array, aunque en este caso no importa el orden. El primer elemento en esta lista es "num", es decir, si el elemento es singular o plural (para esta palabra esta información es innecesaria, por eso es un string vacío) y en "numopposite" guardaremos 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 de estar en otros idiomas. En este valor almacenaremos la traducción de esta palabra en el idioma del diccionario que estaremos definiendo. "gender" y "genopposite", al igual que con "num" y "numopposite", almacenan el género de la palabra y la palabra en el género contrario.

Con los determinantes en inglés no hay mucho cambio, pero en español sí que es bastante diferente:

```
"DET": {
        "the": [
            {"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 seguimos definiendo “the”, dado que es la clave en inglés y la palabra que se tiene que mantener. Sí que vemos que “numopposite” y “genopposite” apuntan a “la”y “los”, que son las nuevas entradas de determinantes en español. De esta forma siempre obtendremos la palabra que queramos siempre y cuando el programa quiera obtener un tipo de palabra concreta en base a las restricciones que contenga la gramática.

7.2. Otras partes del sistema

Las gramáticas juegan un papel muy importante en el proyecto, pero hay otras partes también relevantes que requieren una atención especial.

7.2.1. Interfaz de usuario para el texto generado

Los invidentes usan diferentes programas que leen el texto que se muestra en la pantalla. Para ello nuestro juego debe de mostrar texto de alguna forma y ser capaz de que el lector 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 (TODOOOOOOOOO PONER LO DE JAVA THINGY).

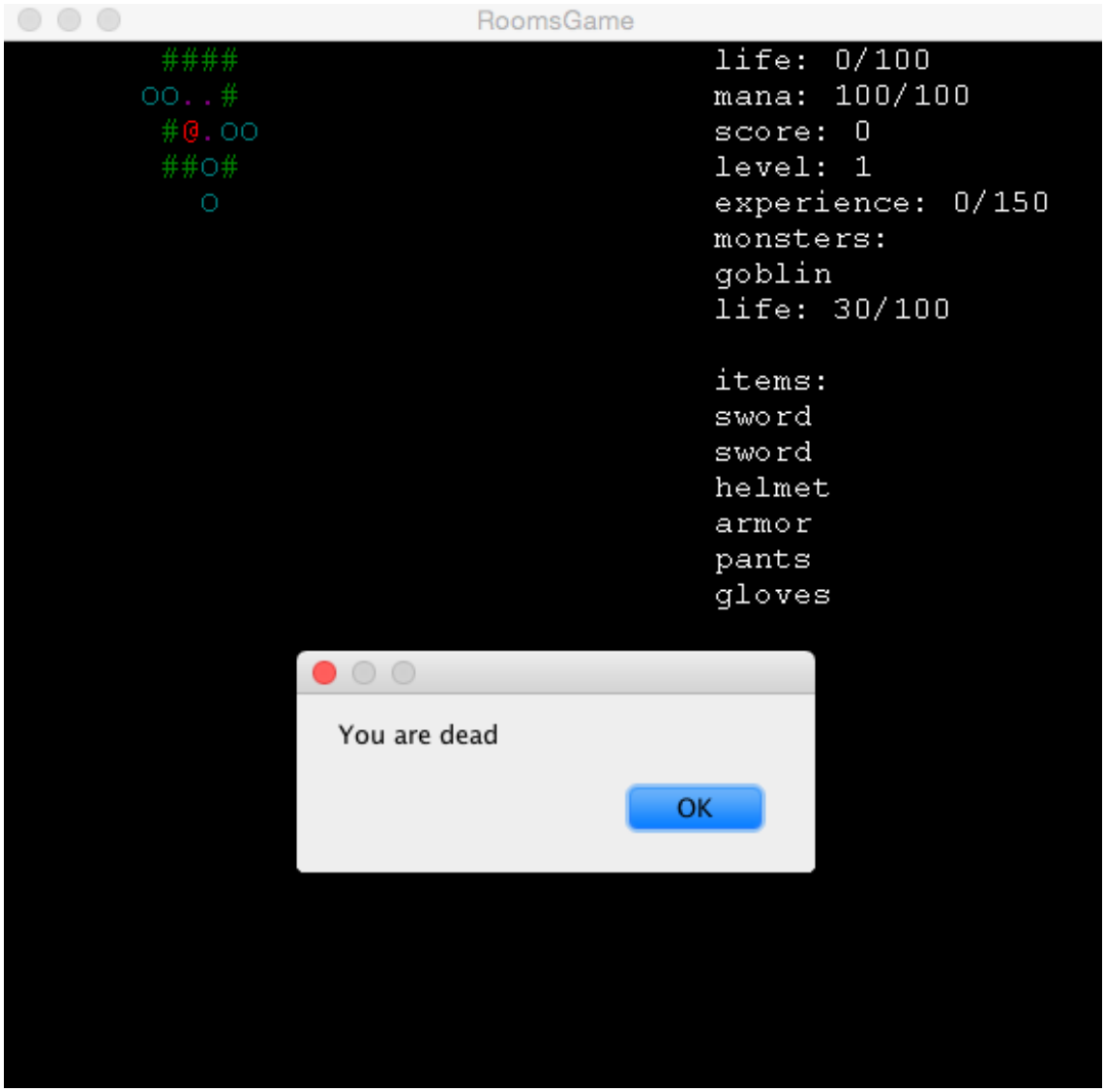


Figura 7.2: Versión definitiva de la interfaz de usuario para el texto generado

7.2.1.1. Primera idea: pop-up

La forma más sencilla de mostrar este texto es generar 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 que reproducirla la cantidad de veces que se quiera.

La imagen **??** muestra el aspecto de nuestro *roguelike* con esta idea implementada.

7.2.1.2. Segunda idea: *TextArea*

Tras recibir el feedback decidimos cambiar la forma en la que mostramos al usuario estas frases por tres razones. La primera es que resulta desconcertante para el jugador vidente, dado que no es necesario para él o ella. Esto se podría solventar añadiendo una opción que se permita activar o desactivar esta opción, pero no es una solución óptima. La segunda razón es que estos "pop-ups" no son un mecanismo de salida óptimo para las salidas rutinarias, como en nuestro caso, y rompe el "flujo de trabajo". La tercera y última razón es que los usuarios invidentes no suelen usar elementos de este tipo, si no que están más acostumbrados a usar areas de texto donde se almacenan las últimas frases generadas.

En base a estas ideas decidimos crear el área de texto seleccionada, que permanecerá abierta siempre y cuando el jugador no decida cerrarla (permitiendo a los usuarios videntes no usarla). Una vez que algo suceda en el juego que requiera que se genere una descripción, dicha descripción será enviada al área de texto, el "foco" de las aplicaciones cambiará para que se sitúa en esa área de texto y el lector de pantalla que usemos leerá dicha descripción. Para que el "foco" vuelva a situarse en el juego, deberemos de pulsar cualquier tecla. De esta forma solucionamos todos los inconvenientes que los "pop-ups" causaban a los usuarios y mejoramos la interfaz gráfica y accesibilidad del proyecto.

En **??** se puede encontrar una captura de pantalla donde se muestra el estado final de la parte de la interfaz de usuario que muestra las frases generadas.

7.2.2. Generación aleatoria de mapas y elementos

La generación aleatoria de mapas y elementos dentro del mismo es muy importante en un juego de género *roguelike*. A continuación explicaremos las decisiones tomadas en base a este tema.

7.2.2.1. Primera idea: 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. En el mundo del rol, tener en cuenta ciertas características del usuario para generar elementos externos se denomina *generación*

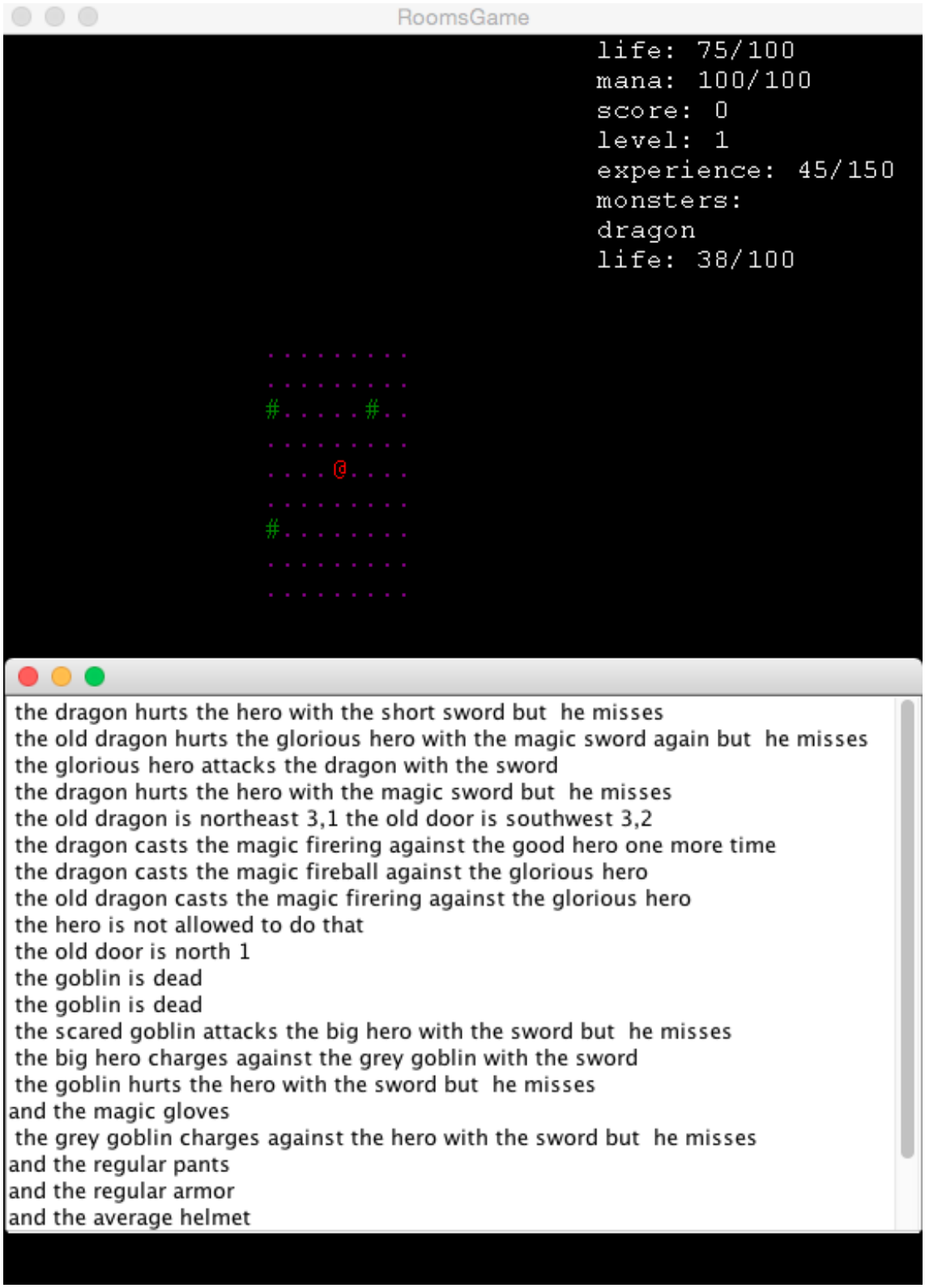


Figura 7.3: Versión definitiva de la interfaz de usuario para el texto generado

de encuentros, que es lo que tuvimos en cuenta.

7.2.2.2. Segunda idea: Generador de encuentros

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 enemigos que se encuentran deben de ser de un nivel, por ejemplo, entre 8 y 12, mientras que los elementos que los enemigos sueltan al morir (como las espadas y armaduras) también deben de tener un nivel similar para que la progresión tenga sentido y eliminar enemigos tenga un cierto nivel de recompensa.

De esta forma podremos, en base al nivel dado, podremos generar enemigos con diferentes características que se adapten a lo que necesitamos:

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

7.2.3. Comportamiento de los enemigos

Como cabe suponer, los enemigos 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 en el juego actualmente son los siguientes:

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

FollowingMovementDumb Los personajes que tengan este tipo de movimiento son "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 movimiento es el más complejo. El enemigo se moverá de manera precisa para llegar lo antes posible al usuario y usará todo tipo de herramientas para derrotarlo (como el uso de magia o atacándole cuerpo a cuerpo).

Crear nuevos tipos de movimientos es una tarea trivial, ya que solamente tendríamos que implementar la interfaz *Movement* con la única función que posee e insertar el código del nuevo tipo de movimimiento en él.

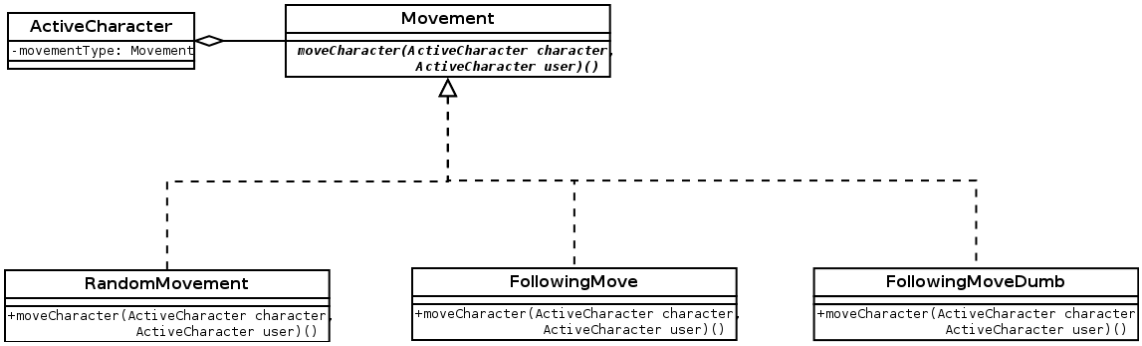


Figura 7.4: Diagrama de clases sobre el comportamiento de los enemigos

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

Capítulo 8

Recepción, conclusión y trabajo futuro.

En este último capítulo detallaremos la recepción y el *feedback* recibido tras mostrar nuestro producto a la comunidad, la conclusión sacadas de la elaboración del proyecto y el posible trabajo futuro del mismo.

8.1. Recepción y *Feedback*

8.2. Conclusión

8.3. Trabajo Futuro

Blabla. Blabla

Apéndice A

Escoger la licencia

Apéndice B

Instalación e Instrucciones

El *roguelike* es software libre y el código fuente se puede encontrar en Github[1], al igual que varias versiones ejecutables. 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 o EN para obtenerlo en español, gallego o inglés, respectivamente. También es posible cambiar las teclas del propio juego. Las que vienen por defecto se puede encontrar en la wiki del proyecto.[4]

[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