Infinity Gates

Project: Infinity Gates

Project: Infinity Gates
Juan José Verdera Infante
Andrew Nauzet Beltrán Pérez
Proyecto de desarrollo de aplicaciones web

2. Índice de contenidos

3. Índice de imágenes

4. Introducción

4.1. Resumen del proyecto

Infinity Gates es un juego del género Roguelike, en este proyecto hemos juntado varias ideas de distintos juegos del género para crear nuestro juego con nuestro estilo combinado; pensado principalmente para usuarios de móvil(Aunque también estará disponible para otras plataformas), para la elaboración de este proyecto utilizamos tecnologías que nos permiten elaborar nuestra aplicación en múltiples plataformas simultáneas, usando React Native. Todo esto con el objetivo obtener un producto que pueda satisfacer las necesidades de entretenimiento de un público en concreto, aquellas personas que le puedan importar nuestro producto.

4.2. Abstract

Infinity Gates is a game of the genre Roguelike, in this project we have merge various ideas from different games of the genre to create our own with our own combined style; thinking mainly for the mobile users(Although it will be available for other platforms), for the making of this project we use technologies that allow us to develop our app in different platforms simultaneously, using React Native. Everything with the main goal of making a product that could satisfy the needs of entertainment for the customers that will concern us about our product.

5. Análisis

5.1. Identificación del problema/necesidad

La necesidad es el entretenimiento humano y la creación de un juego de un estilo único multijugador que no existe en el mercado actual; que pueda satisfacer a una variedad de públicos que puedan estar buscando nuestro juego para su disfrute.

5.2. Búsqueda de información

Para la inspiración de nuestro proyecto hemos estado valorando anteriormente distintas opciones y posibilidades que realizar para este trabajo, al final nos quedamos con esta idea y para su consecuente diseño y desarrollo nos hemos basado en otros juegos populares del género, tales como The Binding of Isaac, Slay the Spyre, Pokémon y Final Fantasy; a continuación explicaremos las influencias de cada uno de estos en nuestra idea.

5.2.1. The Binding of Isaac

De este juego hemos sacado la idea de que cada una de las "runs" sea independiente la una de la otra, es decir, se nos ocurrió que en una run podrías obtener objetos o herramientas que se puedan utilizar solo dentro de la propia run, pero conservaremos el valor que hacer runs a pesar de no vencerlas, es decir, fuera de la propia dungeon el jugador tendrá su propio equipo con el que poder influenciar cómo va de preparado a la mazmorra, dando importancia a la preparación antes de hacer una run y a que se puedan utilizar herramientas dentro de esta.

5.2.2. Slay the Spire

Este juego nos inspiró enormemente en cómo diseñar un Roguelike disfrutable para móvil, hablando en general de esta inspiración, es cierto que hemos sacado bastantes cosas de este juego que no tiene otro similar, ni en estilo ni en éxito, pero, el problema es que este es un juego pensado exclusivamente para una persona, al contrario que nuestro propio juego, siguiendo el razonamiento de "Si una persona lo disfruta, con dos personas disfrutas el doble".

En específico de este juego hemos sacado nuestro sistema de combate que será explicado posteriormente, está basado en ese mismo juego y recoge influencias de otros juegos de cartas de combate como turnos como puedan ser Hearthstone o Legends of Runaterra. El sistema de mazmorras es muy parecido al de Slay the Spire, pero con un par de adiciones y mejoras para hacer la aventura más dinámica y cada run distinta la una de la otra y por último también vamos a utilizar habilidades pasivas que se desarrollan fuera de la mazmorra, esto de nuevo le dará importancia a las decisiones del jugador, pero tampoco planeamos que hayan decisiones claramente mejores que otras.

5.2.3. Pokémon

Aunque en los juegos anteriores existen diferentes personajes para el uso de cada run, hay un elemento que no se emplea en los anteriores y es usar diferentes personajes con mecánicas distintas en dungeons, Pokémon es el ejemplo más claro de este estilo, en Pokémon tu puedes hacer combates individuales e ir cambiando de "personajes", pero también puedes hacer combates de 2vs2 o 1vsN, hemos pensado en el sistema de combate y lo hemos descartado pero gracias a pensar en ello se nos ocurrió la idea de que cada Dungeon puede tener diferentes modalidades como por ejemplo: 1, 2 ,3 ,4 personajes simultáneamente y el nivel de desafío o el diseño de la dungeon cambiaría de acorde al número de personajes, además de que esto nos permitirá añadir interacciones entre personajes, esto sobretodo satisface la necesidad de los usuarios coleccionistas, es decir, aquellos usuarios que prefieren coleccionar personajes o tener un equipo que desarrollar en un juego, tal y como es Pokémon.

5.2.4. Final Fantasy

Final Fantasy es un juego que no se puede considerar como Roguelike por carecer elementos importantes del género, aún así todos los géneros tienen partes de otros géneros, en este caso escogemos el elemento RPG de esta saga, los personajes de nuestros jugadores podrán gozar de diferentes habilidades según su clase y que podrán ser desbloqueadas por nivel además, el jugador podrá equipar diferentes piezas de equipo a su personaje, el equipo podrá dar diferentes propiedades o ayudas al jugador en sus aventuras y ayudarán a definir a qué se puede enfrentar y dar la posibilidad de establecer "Builds" empleando habilidades y equipo que puedan satisfacer al público softcore que les guste este tipo de cosas, haciendo que puedas optimizar tu personaje, pero también simplemente puedes escoger lo que más te gusta e ir con ello, sin obligar a seguir reglas predeterminadas hechas por la comunidad.

5.3. Estado del arte (situación actual)

El género Roguelike consiste en una gran variedad de juegos distintos que acaban siendo muy influyentes a nivel personal y bastante queridos, pero el género tiene una deficiencia y es que no hay casi elecciones en cuando se trata de jugar con amigos a este tipo de juegos, que los juegos multijugador sean competitivos o cooperativos mantienen la vida de un juego sin necesidad de grandes actualizaciones o paquetes de contenido de algún tipo sean gratis o de pago, en nuestro caso creando un juego que junta el elemento online con la aventura solitaria del Roguelike y poder compartir la experiencia con amigos alargaría la vida del juego y podría atraer una buena parte del público por los motivos anteriores, además pensando en que nuestro juego está pensado para dispositivos móviles y en los jugadores casuales que conforman la inmensa mayoría del mercado al que queremos llegar, es decir, teniendo en cuenta lo anterior, haciendo un juego de partidas cortas pero disfrutable es la clave para ganarnos a este público tan grande.

5.4. Razones que justifican el proyecto

Nuestro proyecto se cimenta en las bases de que no existe un juego igual al que nosotros planteamos, eso significa que estamos creando un producto innovador, como se ha explicado en apartados anteriores, creemos que puede tener éxito por el amplio mercado al que apuntamos y el público objetivo que pueda estar interesado en nuestro producto que consideramos como una mejora de varios juegos creando uno distinto de ellos, la gente que ya está familiarizada con esta idea simplemente podrá cambiarse a usar más nuestro producto del que ya usaran anteriormente y algo que planeamos con este juego añadiendo la interacción entre jugadores es crear una comunidad integrada sin necesidad de encontrar grupos por internet con los que poder jugar o hablar del juego, haciendo esto dentro del propio juego cimenta la relación entre jugadores y su relación con el juego.

6. Definición

6.1. Alternativas y propuestas de solución

Como se ha mencionado anteriormente, nuestra idea es una que no existe actualmente específicamente como la hemos desarrollado, pueden existir juegos parecidos pero no uno igual, pero pueden ser de géneros diferentes o con objetivos distintos, como por ejemplo Divinity:

Original Sin 2, puede ser visto como un juego RPG en el que pueden jugar hasta 4 jugadores al mismo tiempo en el que cada uno lleva su personaje y van de aventuras, pero si como contraste ponemos nuestra idea nos damos cuenta que una vez que acabas la aventura de Divinity: Original Sin 2 no hay nada más, el cooperativo es solo para 4 personas y no interactúas con gente por fuera de tu grupo con el que juegas, estas son diferencias a grandes rasgos de nuestro juego con otro, que hay más diferencias específicas como que nuestro juego se basa en el combate por turno de cartas y no en el combate del juego que hemos usado de ejemplo que se basa en turnos por puntos de acción.

Como otras propuestas para solucionar el vacío que cubre la idea que desarrollamos hemos pensado en básicamente la misma realización pero con diferentes componentes y cambios en cosas como el sistema de combate, el sistema de combate por turnos con cartas es algo por lo que nos decidimos al final dada la comodidad de móvil, la simpleza y el toque de aleatoriedad que le presta al juego haciendo que tengas que pensar qué debes hacer.

6.2. Requisitos

Aunque nuestro objetivo y requisitos distan bastante uno del otro, creemos que podríamos cumplir con los requisitos con un par de meses de tiempo, pero lo que hemos planeado para nuestro juego es lo siguiente:

- Sistema de identificación y registro.
- Sistema de creación de personajes.
- Sistema de mazmorras (1, 2, 3, 4 jugadores).
- Sistema social (Chat global, amigos, guilds, mazmorras cooperativas).
- Sistema de administración de cuentas.
- Sistemas de administración de personajes.
- Plataforma de pago.
- Despliegue en plataformas móviles y de escritorio.
- Interfaz multilenguaje(Inglés, Chino simplificado, Español, Alemán, Japonés).

6.3. Objetivos y alcance del proyecto

Nuestro objetivo en este proyecto es una versión funcional del sistema de cliente, servidor, api y base de datos, esta versión contendrá los siguientes objetivos:

- Despliegue de API online.
- Despliegue de base de datos.
- Login y registro funcional.
- Menú principal de navegación de usuario.
- Mapa de dungeon.
- Combate.

Creemos que nuestro proyecto es bastante grande y complejo de realizar especialmente porque estaremos aprendiendo un workframe del lado del cliente con el que no hemos trabajado anteriormente y todavía en nuestro diseño inicial para empezar el desarrollo tenemos varios puntos con los que todavía dudamos si cambiarlos incluso a mitad de proyecto, como el motor

de base de datos, en principio usamos MongoDB, pero quizá al ser una base de datos no relacional nos hace replantearnos nuestra elección, dado que sí que se pueden hacer relaciones en MongoDB pero no son unas relaciones reales.

Nuestros propios objetivos están sujetos a cambios principalmente por los factores de los que dependemos, que es el tiempo que tengamos disponible para poder realizar la ejecución hasta la presentación, es por ello que esta lista es por lo menos lo mínimo que se podrá ver en la demo y será ampliable si podemos plantearnos nuevas metas después de cumplir las actuales.

7. Diseño

7.1. Mecánica de juego

7.1.1. Flujo de Mazmorra

Al darle a jugar entraremos en una run, tendremos diferentes rutas que seguir hasta llegar al boss final de la run, en las rutas habrán salas, las cuales pueden ser: **combate**, **combate élite**, **tesoro**, **tienda**, **hoguera**, **misión**, **sala misteriosa**, **boss**.

Combate: será una lucha contra enemigos normales que darán como recompensa por acabar con ellos las siguientes cosas:

- Cartas para añadir al mazo actual.
- Oro para la run actual.

Combate élite: será una lucha contra enemigos más difíciles con una mecánica especial, ejemplo: cada vez que recibe daño aumenta su ataque. Darán como recompensa las siguientes cosas:

- Cartas para añadir al mazo actual.
- Oro para la run actual.
- Aolita(dinero global de la app).
- Pasiva aleatoria.

Tesoro: sala donde se conseguirá:

- Pasiva aleatoria.
- Oro para la run actual.

Tienda: sala donde se podrá comprar con el oro de la run actual lo siguiente:

- Cartas para la run actual.
- Pasivas.
- Pociones.
- Aolita(dinero global de la app).

Hoguera: permitirá realizar alguna de las siguientes opciones:

- Eliminar carta del mazo actual.
- Curar un porcentaje de la vida del personaje.
- Aumentar la vida máxima del personaje esta run.

- Mejorar una carta del mazo.

Misión: sala donde se encomendara al usuario realizar una misión en una mazmorra definida pues al completarlas darán recompensas:

- Aolita(dinero global de la app).
- Desbloquear equipo.
- Desbloquear pasivas.

Sala misteriosa: en esta sala podremos encontrarnos…

- Eventos especiales, por ejemplo un goblin amigable que nos ofrece un trato.
- Una sala de tesoro.
- Una sala de tienda.
- Un combate.

7.1.2. Flujo de combate

Inicio de combate

Un combate será iniciado cuando el jugador tenga un encuentro con un enemigo que podría ser dado por casillas de: combate, misión, evento especial, sala misteriosa, combates de élite o llegando a la parte final de la mazmorra.

Combate por turnos

El combate será dado por turno, para jugador/es, turno para enemigos.

En el turno de jugador que será el primero, el jugador tomará un número aleatorio de cartas, cada carta tendrá un valor de punto de acción, es decir, el jugador está limitado por el número de puntos de acción (Cada clase podría llamarse de otra forma ej: Guerrero-Stamina), los puntos de acción se utilizan para utilizar cartas/acciones.

Se cambiará de turno automáticamente cuando el jugador no puede utilizar cartas por los siguientes motivos:

- No tener más cartas.
- El coste de las cartas supera los puntos de acción disponibles.

Una vez finalizado el turno, si quedan puntos de acción disponibles, se irán a un pool de puntos de acción extra con un máximo de puntos guardables definido.

Después del turno del jugador, tomará turno de los enemigos, los enemigos en su turno dependiendo de sus habilidades podrán:

- Atacar.
- Colocar un debuff.
- Preparar una acción de múltiples turnos.

7.2. Esquemas

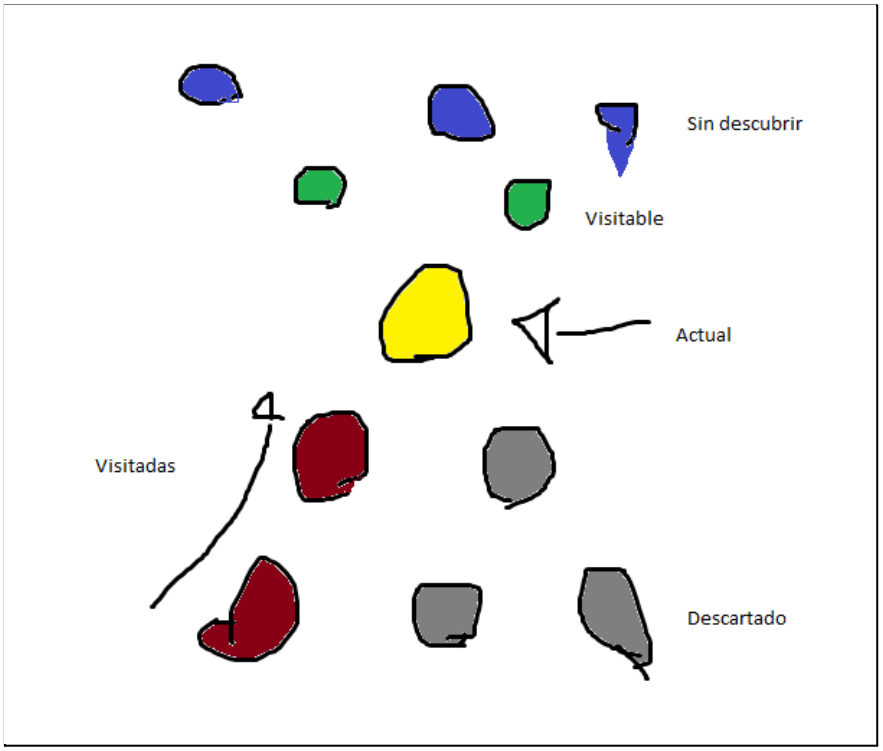

Figura 1. Esquema representativo del mapa

Este esquema fue utilizado para ilustrar los posibles estados del mapa

Casilla	Probabilidad
Combate	50%
Combate élite	10%
Tesoro	15%
Misión	1 por partida
Tienda	10%
Sala misteriosa	5%
Hoguera	10%
Boss	1 por nivel

Con esta tabla pensamos cómo podría funcionar el factor aleatorio del juego y además recopilar las posibles salas del mapa.

7.3. Sketches (bocetos)

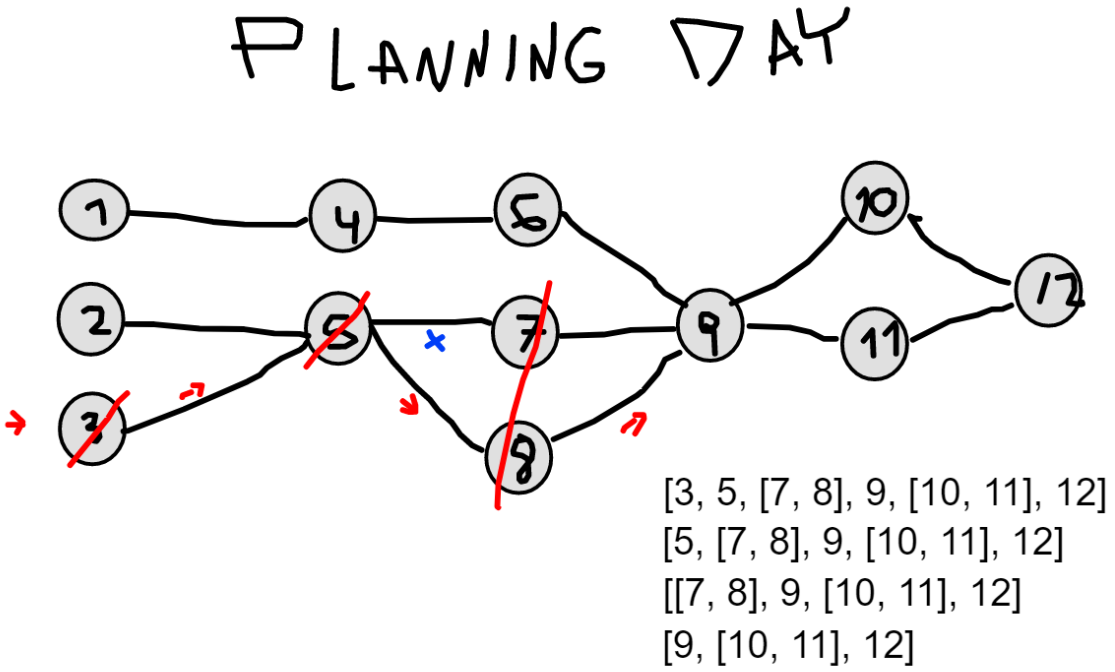

Figura 2. Sketch representativo de funcionalidad del mapa

Sketch representativo de cómo se vería y en parte funcionaria el mapa de cuando estábamos decidiendo cómo iba a funcionar.

Se puede apreciar la representación de un array, al empezar se elige una ruta inicial(1, 2 o 3), al elegir '3', tendríamos como resultado el segundo array que se muestra en la imagen que viene a representar las siguientes opciones que tenemos al seguir el mapa, solo sería seleccionable el primer elemento del array, si es tipo 'number' significa que solo hay una opción, si es un 'array' significa que todas las casillas representadas dentro de este son seleccionables, al seleccionar una simplemente se eliminaría el primer elemento del array y nos dejaría con el mapa restante.

Esta idea fue desechada rápidamente ya que en un mapa más amplio la lógica tiene lagunas importantes. La idea de hacer todos los *pathins* posibles no era algo viable, en su lugar acabamos usando un modelo de matriz para nuestro mapa con comprobaciones de los estados de casillas anteriores y siguientes.

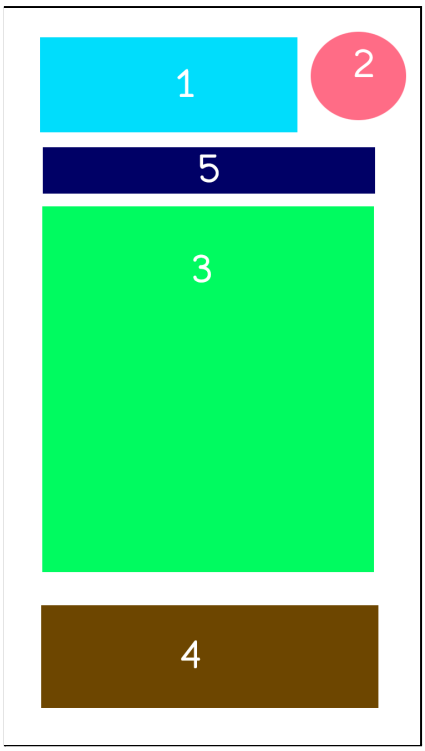

Figura 3. Boceto

Boceto de selección de Dungeon

1-Título de la pestaña dónde se encuentra
2-Salir
3-Lista de dungeons
4-Entrar a dungeon seleccionada
5-Seleccionar número de personajes

Figura 4. Boceto logo

Boceto de logo de compañía.

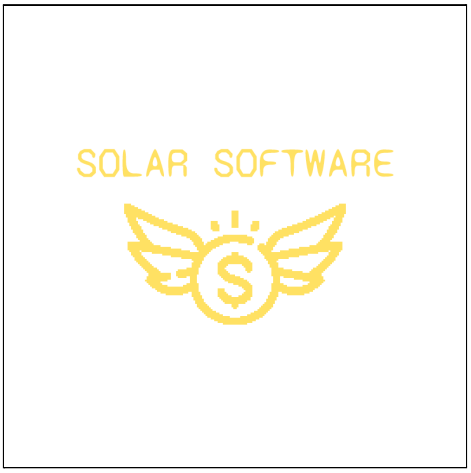

Figura 5. Boceto logo

Boceto de logo de compañía.

Es similar al actual pero el actual tiene un trazado más grueso y está a mayor resolución.

7.4. Mockups (prototipos)

Diseño de vista inicial(descartado)

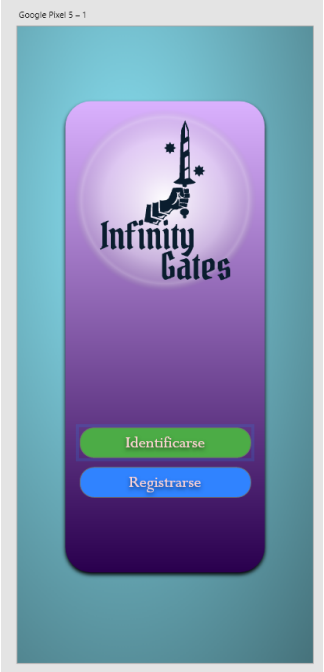

Figura 6. Mockup inicial

Diseño registro v1(descartado)

Figura 7. Mockup registro

Diseño de vista inicial

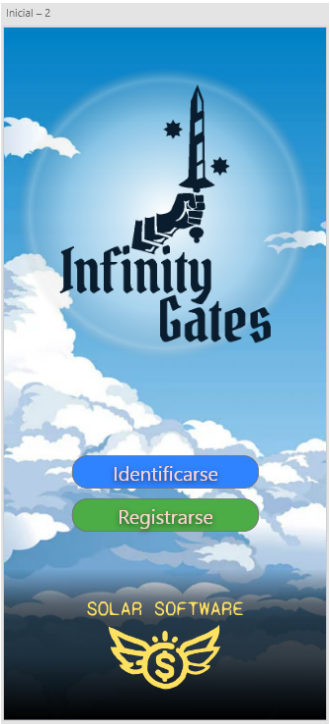

Figura 8. Mockup inicial

Diseño registro v2

Figura 9. Mockup registro

Diseño login

Figura 10. Mockup login

Página de inicio

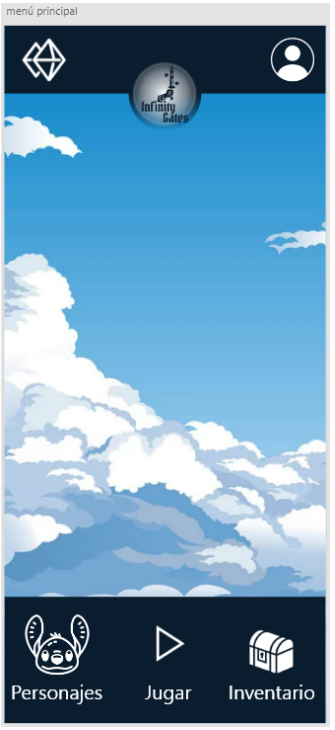

Figura 11. Mockup login

Lista de personajes

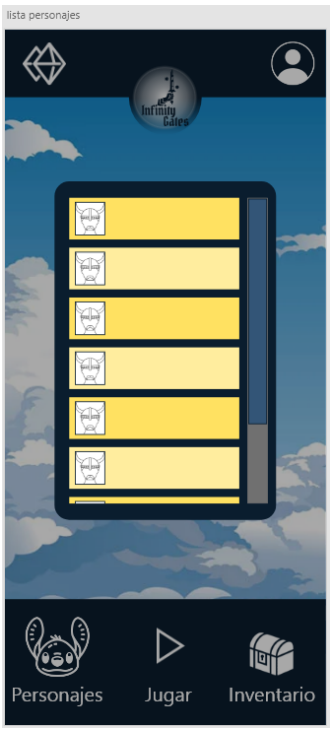

Figura 12. Mockup personajes

Diseño provisional de vista edición de pj

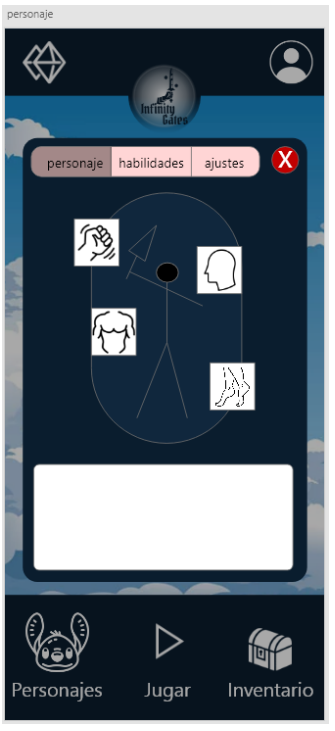

Figura 13. Mockup edición de personaje

7.5. Diagramas UML (de casos de uso, de flujo, de clases, despliegue, componentes, etc.)

7.5.1. Diseño provisional de la base de datos relacional de infinite gates

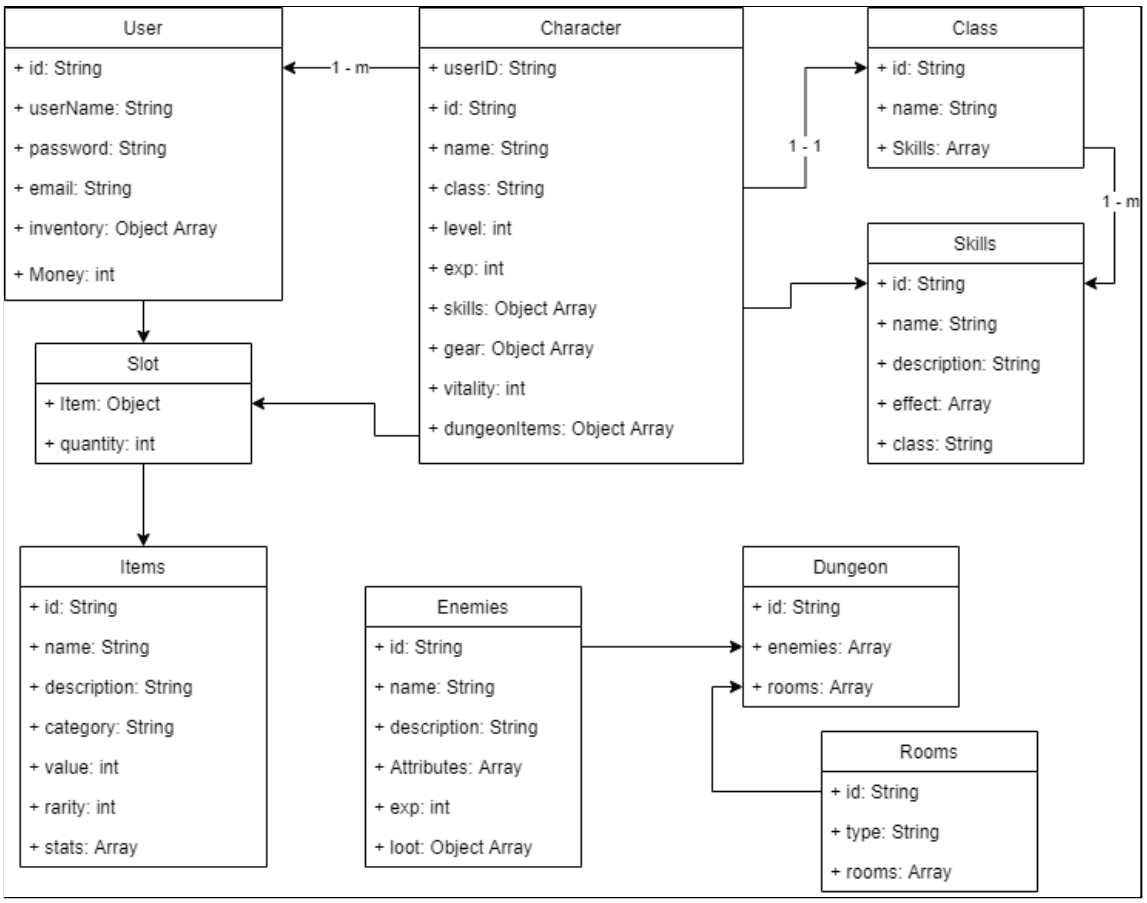

Figura 14. Diagrama base de datos

7.5.2. Diagrama de flujo de combate

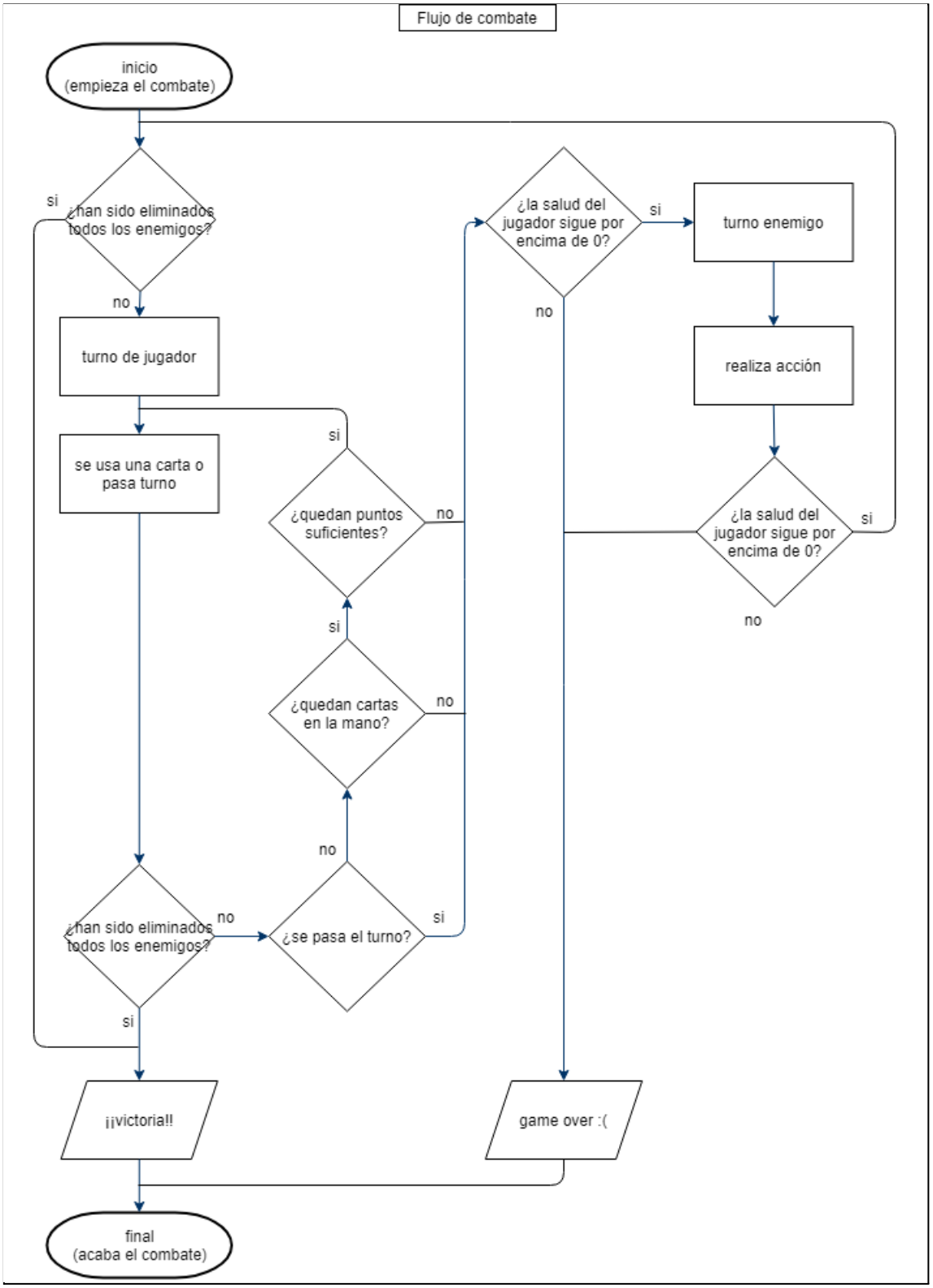

Figura 15. Diagrama de flujo de combate

8. Planificación

8.1. Recursos

8.1.1. Materiales (equipos informáticos, etc.)

- 2 Equipos informáticos: Serán necesarios para que nuestro equipo de programadores puedan trabajar en paralelo sin necesidad de depender de tener solo una máquina.

- 1 Dispositivo móvil para pruebas: En una prueba de despliegue real será necesario utilizar al menos un dispositivo móvil de sistema operativo Android, sería opcional obtener otro dispositivo móvil iOS para pruebas.

- Raspberry Pi 4B.

8.1.2. Herramientas (editores, IDEs, *software*, etc.)

- Visual Studio Code: VSCode será nuestro editor de código debido a su simpleza, estilo visual, la capacidad de adaptar nuestro entorno de trabajo a las necesidades del proyecto usando addons o personalizando el propio.

- Android Studio: Se ha utilizado para la emulación en dispositivos móviles de la aplicación y su propio testeo.

- Photoshop: Se ha utilizado para dibujar los primeros bocetos de diseño sobre las vistas de la aplicación, creación de logo entre otras expresiones de arte.

- Adobe xd: La herramienta seleccionada para la creación de wireframes/mockups.

8.1.3. Tecnologías (lenguajes de programación, SO, SGBD, etc.)

- Windows 10 profesional: Sistema operativo en el que se programará el código, debido a su disponibilidad y familiaridad de los programadores con el sistema operativo.

- JavaScript: El lenguaje de programación que se utiliza en todas las webs del mundo, es por ello por lo que nuestra aplicación web debe utilizar JavaScript, además de ser un lenguaje con el que los programadores web deben de estar familiarizados, lo mejor de JavaScript es estar orientado a eventos, la cantidad de workframes y repositorios de paquetes de la comunidad con los que podemos trabajar.

- Node.js: Workframe para JavaScript del lado del servidor, en un juego el punto principal de la relación cliente-servidor es la interacción entre ambos, es por eso que Node.js al venir de JavaScript nos hace pensar que esta es nuestra mejor opción dado que parece encajar perfectamente con nuestra idea.

- React Native: React Native es la versión de lenguaje híbrido del workframe React, el workframe más utilizado de JavaScript, ha sido escogida con el workframe de Expo por un motivo que nos ahorra tiempo de programación y es que React Native es un lenguaje de programación que sirve para cualquier plataforma, dado que cuando se compila se convierte en código html, css y JavaScript que puede leer un navegador web o se traduce a Java que lo puede leer un dispositivo Android.

- MongoDB: Este motor de base de datos es el más utilizado en cuanto a proyectos realizados con los objetivos que nosotros tenemos, es decir, es especialmente bueno para juegos, debido a su funcionamiento, aunque encontramos varios inconvenientes y es que nos gustaría más un modelo relacional y con schemas, es por eso que nos planteamos cambiar de motor de base de datos.

8.1.4. Servicios (*hosting*, control de versiones, *quality assurance* o *testing*)

- Github: El control de versiones más utilizado en el mundo de la programación que se podría considerar standard también como repositorio para alojar el control de versiones.

- Servicio de hosting para base de datos: El servicio a elegir dependerá de qué motor de base de datos utilizamos para nuestro proyecto ya que muchas veces el gestor de uno ofrece el propio servicio para hostear una base de datos y utilizarla, al contrario que nuestro servidor intermediario, si que creemos que un servicio externo sería más seguro, aunque al final decidimos utilizar MongoDB como gestor y como host para nuestra base de datos.

8.1.5. Recursos humanos

- 2 Programadores: Sin programadores no podemos ni empezar a hacer un prototipo para el proyecto, podrían incluirse otros tipos de especialidades como música, modelaje, diseño… pero para el alcance del proyecto no planeamos emplear ninguno de los anteriores.

- Reviewers: En cualquier cantidad serán necesarios para obtener un feedback de nuestro proyecto.

8.1.6. Estimación temporal

- Despliegue modelo funcional: 1 mes.
    - desarrollo de documentación inicial: 1 semana.
    - backend: 4 días.
    - front end 2 semanas.
- Despliegue de alpha: 2 meses.
- Despliegue de Beta: 3 meses.
- Despliegue versión completa: 3 meses.

8.2. Tareas a llevar a cabo

8.2.1. Elaboración de documentación del proyecto y de la aplicación

Será necesario terminar el documento actual para obtener una imagen clara del diseño y desarrollo de nuestra aplicación con la que empezaremos a trabajar, además de esta documentación del proyecto en sí mismo vamos a necesitar elaborar una documentación sobre la propia aplicación y cómo funciona cada método, el manejo de la api, la estructura de la base de datos.

8.2.2. Desarrollo de aplicación cliente

Para la elaboración del proyecto será necesario hacer una aplicación del lado del cliente que pueda funcionar en dispositivos de escritorio y móviles, además de que se pueda poder descargar o instalar de alguna manera, para ello se tendrá que hostear la versión web en un dominio y las versiones de móvil se podrán descargar utilizando una apk, por ejemplo por lo menos para la versión de Android.

8.2.3. Creación de servidor

La creación de un servidor y su API para la comunicación entre base de datos y cliente es algo que debe ser construido en orden para que toda la aplicación funcione correctamente, la API servirá para definir la comunicación y como se trata esta misma, además de ser capaces de poder asegurarnos que la información enviada a nuestro servidor y base de datos es fiable y utilizable en caso de tener usuarios con intenciones malignas sobre nuestro programa.

El servidor como se ha escrito anteriormente en las tecnologías que se van a utilizar va a ser node debido a que su lenguaje base está muy preparado para los eventos y la reacción antes estos lo cual en un juego nos parece bastante importante.

8.2.4. Servidor de base datos

Un lugar donde poder hostear nuestra base de datos será imperativo tenerlo dado que es donde se guardará la información de usuarios, en este caso sí que usaremos un servicio externo, dado que el servidor lo podemos migrar en cualquier momento, una base de datos es más complicado. Será fácil migrar la base de datos o cambiarla durante el proceso de desarrollo inicial dado que todavía no tendremos grandes volúmenes de datos guardados, lo cual nos haría replantearnos si cambiar de motor de base de datos y servicio de hosting para la misma.

8.2.5. Cronograma

Para nuestra planificación utilizamos un sistema de tareas u objetivos por día que se iba modificando según la necesidad, siempre es necesario poder adaptarse cuando no se completa algo a su fecha límite, además a estas tareas específicas habría que sumarles los formateos de códigos de cada mañana antes de las prácticas y los cambios refactorizaciones después de acabar las sesiones de programación, que se estiran más tiempo del que se había planeado, especialmente cuando se estaba cerca de terminar una funcionalidad.

Fecha	Tarea
18/03/2021	Diagrama de flujo sobre las vistas de nuestra App.
18/03/2021	Realizar los diseños de cada vista
18/03/2021	Diagrama de flujo combate
18/03/2021	Refactorizaciones de documentación y limpieza
19/03/2021	Repositorios Github, Replit, Bitbucket y MongoDB
19/03/2021	Diagrama de casos Mazmorra
21/03/2021	Fecha límite para finalización de diseño de la app
21/03/2021	Realizar cursos de React + Aprender React
22/03/2021	Crear el repositorio
22/03/2021	Pruebas de código y funcionamiento
23/03/2021	Página inicial
25/03/2021	Formulario de registro y login
26/03/2021	Refactorización de página inicial, registro y login
29/03/2021	Incorporación de menú principal de usuario
30/03/2021	Cambiar "singup" por "signup" en el código y nombre de carpetas del proyecto
31/03/2021	Cambiar en la API las constantes con información sensible, en los ficheros "middleware/auth.js", "routes/index.js" y "app.js", con nombre "st" y "mySecret"
30/03/2021	seguridad de que el usuario deba estar autentificado
31/03/2021	Implementación de la autentificación
01/04/2021	Control de errores de autentificación + guardado en local(localStorage) de token
01/04/21	limpieza de código
02/04/21	Diseñar funcionamiento del mapa básico
02/04/2021	Planificación y layout de la vista de combate
03-23/04/2021	Implementación del mapa dungeons
24-30/04/2021	Rework completo a la implementación del mapa dungeons
03-05/05/2021	Trabajo completo en el documento

8.3. Metodologías

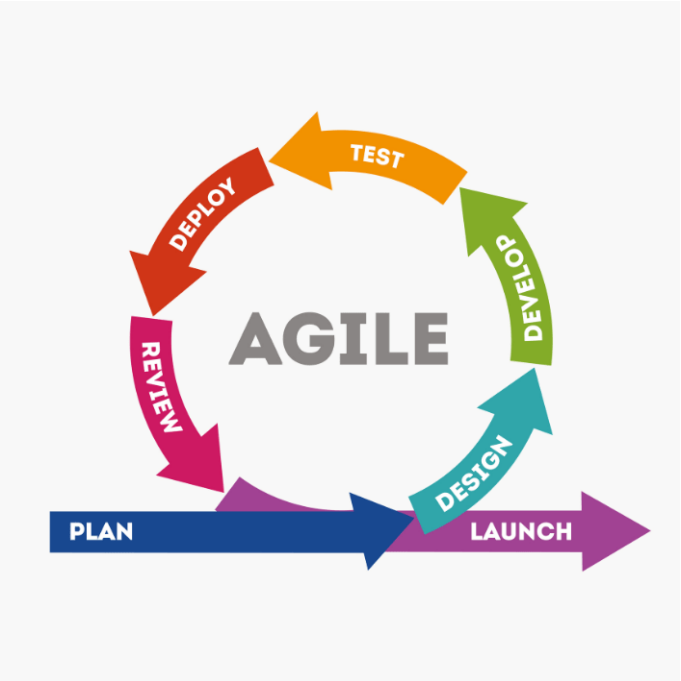

Figura 16. Metodología

Este esquema representa completamente el cómo vamos a desarrollar nuestro proyecto, nos basamos en la programación funcional para ir construyendo modelos funcionales cada vez más funcionales.

Explicando nuestra metodología con más profundidad, nosotros tenemos un plan, un objetivo que hacer y al que llegar como mínimo, diseñamos el cómo va a ser, los desarrollamos, vamos probando los módulos, después probamos cómo funciona en conjunto y revisamos nuestro proyecto, así continuamente hasta llegar a una aplicación que se pueda lanzar al mercado, nos gusta tener un plan flexible e introspectivo dado que al ser un proyecto tan grande debemos asegurarnos que el conjunto siga funcionando y pueda soportar nuevas funcionalidades.

El cómo hacemos nuestro diseño se basa en nuestro plan, para rediseñar algo nos basamos en pruebas y opiniones externas además de influencias que podamos obtener o nuevas ideas.

Nuestro método de desarrollo se basa en el peer-2-peer para la escritura de código del cuál no estamos familiarizados, pensamos que con este método podemos sobrepasar los obstáculos que se nos ponen en el camino relacionados con aprender un nuevo lenguaje o en nuestro caso un nuevo workframe.

Para el desarrollo regular de otros componentes de nuestra aplicación simplemente trabajaremos de forma paralela en partes distintas de la aplicación, ya que estaremos mucho más familiarizados con los objetivos que tenemos que llegar y cómo realizarlos y para la review del código que escribamos por separado simplemente lo comprobará el otro además de que individualmente tendremos que asegurarnos que un módulo hace y cumple lo que debe de hacer aunque sea de forma mínima.

El despliegue de la actualización en las fases de desarrollo consistirá en el alojamiento virtual de servidor, base de datos y la ejecución en emuladores de la versión de cliente y la aplicación en el cliente debe de estar en un repositorio al que podamos acceder de estar forma podremos trabajar a distancia y con diferentes ritmos de trabajo, y en caso de no estar trabajando con nuestros equipos habituales podremos acceder a todo nuestro proyecto en distintas localizaciones sin ningún tipo de problema.

El desarrollo de la aplicación en cuanto a fecha y progreso sigue la tabla del apartado anterior, siendo un esquema muy flexible de trabajo y muy introspectivo, eso no has hecho adaptarnos a la metodología explicada anteriormente de una forma muy natural, es por ello que íbamos prácticamente, implementando funcionalidades, ver que falla, adaptar la idea, ver que falla y así continuamente, el método de trabajo de asemeja al de un escultor trabajando con mármol.

8.4. Estimación de costes

Los costes han sido estimados a la fecha de Marzo de 2021.

8.4.1. Gastos iniciales

Para desarrollar el código no se requieren equipos informáticos de gran potencia, por ello podríamos prescindir de tarjeta gráfica dedicada, cada uno nos podría costar unos 550€.

El dispositivo móvil Android para pruebas ha de ser de gama media como mucho, queremos que todo el mundo pueda jugar sin problemas a nuestro juego, el cual no debería pedir mucho para funcionar correctamente, su coste debería rondar los 150€.

En cuanto al dispositivo móvil iOS, en caso de querer realizar pruebas nos aumentaría bastante el coste así que obtener uno de hace un par de años podría ser buena idea, el modelo más barato actualmente que vende Apple oficialmente cuesta 449,01€.

La Raspberry Pi 4B tiene varios modelos que se diferencian en la ram, 2GB con un coste de 28,88€, 4GB con un coste de 45,38€ y la de 8GB con un coste de 61,88€.

El coste inicial estimado es desde 1.128,88€ a 1.960,89€.

8.4.2. Mensualidades

(Se podría mirar el crear otro apartado para pagos únicos, ejemplo, un equipo de traductores o diseñador gráfico para la interfaz)

(Photoshop, Adobe XD) Para empresas cada una 29,39€, para estudiantes 19,66€.

Artista musical, disponer de un profesional para el sonido del juego suena muy bien pero para este proyecto vamos a prescindir de ello ya que no lo creemos muy importante en un juego de este estilo.

Artista gráfico, aunque no es un proyecto muy grande, necesitaríamos tener una interfaz propia bonita y diseños de cartas únicos, disponer de un artista gráfico sería primordial para una versión avanzada del proyecto, contratar uno al comienzo del desarrollo de la beta para trabajar junto al equipo sería una opción a considerar.

Traductores, serían necesarios traductores dedicados para traducir el juego a los diferentes idiomas objetivos.

9. Conclusiones

9.1. ¿Se han cumplido los objetivos?

Hemos cumplido parcialmente el alcance de los objetivos, hemos logrado para la fecha de entrega hacer la mayoría de objetivos planteados en la planificación de tiempo estimada. Con grandes dificultades y saliendonos mucho de la planificación inicial.

9.2. Dificultades encontradas

Como dificultades que hemos encontrado podemos destacar las siguientes, aunque algunas ya habíamos anticipado desde la planificación que iban a ser un obstáculo que superar.

9.2.1. Un workframe nuevo

Trabajar sin un guía con algo completamente desconocido aunque divertido es complicado y tiene sus momentos, el desconocimiento implica muchos puntos en contra algunos peores que otros, principalmente no conocemos las capacidades totales de lo que estábamos haciendo, cuál era el límite o la mejor forma de realizar algo, aunque nos enorgullece el resultado al que hemos llegado.

Literalmente hemos encontrado una dificultad a cada paso, a cada nueva funcionalidad que hemos querido introducir dentro del juego, han habido muchos problemas en su incorporación, por ende, muchos retrasos y frustraciones.

9.2.2. No somos diseñadores

El cómo nos hemos llegado a adaptar a esta situación de que no somos expertos en materia ni mucho menos ninguno de los integrantes del equipo se podría decir que tiene un buen nivel en cualquiera de las áreas de dibujo, música y modelaje, pero con las herramientas adecuadas, fallo y error además de sumar la experiencia previa que aunque no nos hiciese profesionales en los campos sí que facilita que podamos llegar a una resolución mejor cuando se trata del apartado gráfico de nuestra aplicación.

9.2.3. Trabajar con móviles

Ha sido la primera vez que trabajamos con móviles o un despliegue multiplataforma y para solventar este problema pues decidimos utilizar caminos que nos facilitaban esto mismo, como por ejemplo el workframe del lado del cliente que utilizamos para el despliegue que nos permite compatibilidad con el mismo código en distintas plataformas y al principio la verdad es que nos encontrábamos perdidos de cómo empezar a escribir el lado del cliente pensando en un móvil, no resultó ser tan complicado pero no conocíamos el primer paso a dar.

Además de trabajar con android studio, un programa nuevo para nosotros y pudimos ver algunos fallos relacionados con el programa, aunque sí que nos adaptamos bastante al flujo de trabajo con el emulador, cada vez que veíamos algo que debería ser imposible, el "recargar" se convirtió en algo recurrente.

9.3. Posibles mejoras

Obtener un diseñador gráfico, dibujante, músico, modelador y traductores. Tener alguien con quien poder consultar dudas de React Native sería fantástico, sigue pareciendonos increible que con todo el handicap del workframe llegásemos tan lejos, y estábamos dando palos de ciego mucho tiempo.

10. Webgrafía

React Router: Declarative Routing for React. (2021). Retrieved 27 March 2021, from https://reactrouter.com/native/guides/quick-start

Dobby, C. (2021). text-highlight. Retrieved 23 April 2021, from https://github.com/ChrisDobby/text-highlight

SecureStore - Expo Documentation. (2021). Retrieved 16 April 2021, from https://docs.expo.io/versions/latest/sdk/securestore/

11. Videografía

Fireship. (2021). *Raspberry Pi versus AWS// How to host your website on the RPi4* [Video]. Retrieved from https://youtu.be/QdHvS0D1zAI

React Native Tutorial - How To Create A Simple Responsive Layout For Beginners. (2019). [Video]. Retrieved from https://www.youtube.com/watch?v=EOcMS2yDpwY&ab_channel=FullstackDevelopment

Scalable Scripts. (2020). *React Login, Logout and Handling JWT Token | React Authentication #2* [Video]. Retrieved from https://www.youtube.com/watch?v=XWj18K4Uhg8&ab_channel=ScalableScripts

Fireship. (2020). *7 Database Paradigms* [Video]. Retrieved from https://www.youtube.com/watch?v=W2Z7fbCLSTw&ab_channel=Fireship

Scalable Scripts. (2021). React JWT Authentication [Video]. Retrieved from https://www.youtube.com/watch?v=OUP-urBy1k4

12. Anexos

12.3. documentación

Nuestro proyecto se encuentra dividido en 2 ramas principales, en primer lugar en la carpeta "doc", encontramos la documentación, en la carpeta src, encontramos el código de nuestro proyecto

12.3.1. doc

Esta carpeta contiene un pdf con esta misma documentación del proyecto.

12.3.2. src

Esta carpeta se encuentra dividida en dos apartados: cliente y servidor(react y node, respectivamente) a continuación empezará la documentación del código del proyecto.

12.3.2.1. react

El código de nuestro proyecto y recursos utilizados para el apartado del cliente se encuentra en la carpeta "app". Dentro de la carpeta app tenemos nuestro código organizado por clase.

components: Esta carpeta contiene clases reutilizables, sus objetos cumplen una funcionalidad concreta que puede verse utilizada varias veces. Un componente puede ser declarado de dos formas distintas, en nuestro proyecto, solo se encontrará una única forma, declarar componentes como clases, para mantener el mismo formato.

Cada componente, tendrá la siguiente estructura:

1. Importación de componentes de React y React Native.
2. Declaración de estilo.
3. Declaración de la clase.

- LogIn: Clase que devuelve un modal que da visión a un cuestionario de *login,* este método recibe de su llamador las variables y métodos "visible", "onCloseModal", "login" y tiene las variables locales "username", "password". Su función es enviar los datos de los inputs al servidor y comprobar en la base de datos si el usuario y contraseña son correctos.

- SignUp: Clase que devuelve un modal que da visión a un cuestionario de *sign up,* este método recibe en su llamada las variables y métodos "visible", "onCloseModal", "login" y tiene las variables locales "username", "password". Su función es enviar los

datos de los inputs al servidor y ya el servidor se encarga de todas las comprobaciones necesarias para lograr meter los nuevos datos en el servidor.

- Tiles: Clase que devuelve un botón, estos botones son el bloque básico de cómo funciona nuestro mapa jugable, recibe de su mapa padre “tile”, “updateMap”, “updateTile”.

    - “tile” es un subconjunto de funciones y propiedades que están íntimamente relacionadas con el funcionamiento de la clase “Tile” que deben de ser declarados desde el mapa por motivos dinámicos.

    - updateMap y updateTile, son métodos usados para actualizar el estado de la propia casilla y actualización de mapa.

- Map: Clase que devuelve una vista ordenada de Tiles, esta clase tiene una adicción de otra clase llamada “TileModel”, se usa para la declaración de “Tiles” y la aplicación de las funciones utilizadas dentro de la misma.

    - Como variables locales lo más importante es la matriz, la matriz es nuestro propio mapa, hecho con declaraciones de “TileModel”.

    - La creación del propio mapa se realiza leyendo el propio array de tileMap y con bucles anidados y algo de estilo, logramos crear una forma de creación de mapas reutilizable.

screens: Las screens son las vistas de la aplicación, en un periódico online, por ejemplo, una vista se podría considerar como un enlace en la barra de navegación, como “noticias internacionales” o “deportes”.

Su declaración es muy similar a la de un componente siguiendo la siguiente estructura:

1. Importación de elementos de React y React Native.
2. Declaración de hoja de estilo.
3. Declaración de clase que hereda de “Component”.
4. Exportación de la clase para app.js.

Cada vista puede contener múltiples componentes, estas tienen propiedades heredadas de “Component”, pueden declararse funciones dentro y variables con la adicción de algunos métodos extra

- start: Vista de inicio de la aplicación, tiene las variables locales “loginVisible” y “signupVisible” usadas para controlar la visibilidad de los componentes SignUp y LogIn, además tiene los siguiente métodos:
    - setLoginVisible & setSignupVisible: Métodos para la visibilidad de los modales de esta pantalla.

    - createAccount: Función que usa la API para enviar los datos necesarios de creación de cuentas.

    - isLogged: Función que comprueba si el usuario ya está *logueado* por comprobación en el "AsyncStorage".

    - login: función que usa la API para comprobar si los datos de inicio de sesión son correctos.

    - componentDidMount: Función nativa de React, es utilizada para comprobar si cuando el usuario accede al menú de inicio y ya está logueado, que lo mande directamente al menú principal de juego.

- menu: Vista del menú principal de jugador, esta vista es la principal forma de navegación entre las funcionalidades de la aplicación, contiene enlaces a otras vistas y será la forma principal de navegación.

    - requireAuth: Método encargado de la comprobación del acceso a esta vista, básicamente busca si el usuario tiene credenciales, si no las tienes o si son erróneas no podrá acceder a esta pantalla.

    - disconnect: Función para destruir la autorización.

    - componentDidMount: Función nativa de React, que se ejecuta antes de cargar y devolver la propia vista, hace ejecutar la función requireAuth.

- game: Vista del primer y único mapa de juego, como métodos tiene "requireAuth" y "componentDidMount", de la misma forma que la vista de "menu". Además de eso esta vista solo devuelve un botón para volver y el componente de Map.

12.3.2.2. node

En esta carpeta se encuentra definido el código que utiliza nuestro servidor, de aquí podemos denotar 2 archivos y una carpeta como los más importantes:

- index.js: Donde se monta el servidor html en el puerto indicado.

- app.js: Se compone de distintos apartados que pueden ser resumidos como: importación de dependencias, declaración de la instancia de la app, asignación de dependencias a la app, conexión a la base de datos, indicación de *headers*, declaración de ruta principal, control de errores para las rutas y exportación de app.

- api: La propia API se subdivide en 3 carpetas, "middleware"(comprobaciones de seguridad), "models" (Modelo de datos), "routes" (rutas de peticiones disponibles para la API).

- middleware: Contiene auth.js, este fichero tiene un conjunto de funciones y métodos para la comprobación de la autenticidad de la autenticación, se utiliza JWT(JSON Web Token) para la codificación de las credenciales, las propias credenciales están encriptadas por bcrypt con una sal de 10 capas, aunque en este fichero no se encriptan, se van a codificar o descodificar. Cada vez que se utiliza una ruta que requiera de credenciales, este fichero será llamado siempre.

- models: Contiene User.js que es el esquema principal de usuario, contiene la forma que adoptan los datos de este modelo, sus requerimientos y cómo se guardan, además contiene los métodos de comparación de contraseña, al utilizar bcrypt como método de encriptación, la contraseña introducida en la base de datos no siempre será la misma que mande el usuario cuando se vuelva a encriptar, es por ello por lo que cuando se usan contraseñas encriptadas con bcrypt la forma de saber si una contraseña es correcta es por la comparación entre ellas, si son suficientemente similares el servidor lo tomará por válida. Finalmente se exporta el modelo.

- routes: Dentro se encuentra index.js que tiene las rutas de GET y POST para registro y autenticación, además de conceder el acceso al menú principal de usuario en el cliente.