

RUBY

Memoria de desarrollo de RUBY - El videojuego

Índice de contenido

1. Introducción	2
2. Equipo de desarrollo	2
3. Ficha técnica	2
4. Historia	2
5. Objetivos del juego	3
6. Controles	3
7. Mecánicas	3
8. Estados del juego	4
9. Análisis del código	6
10. DIAGRAMA DE CLASES	7
11. Tecnología, herramientas y fuentes utilizadas	11
12. Problemas encontrados en el desarrollo.	12

1. Introducción

Esta es la memoria de nuestro proyecto para la asignatura de Tecnología de Videojuegos, el videojuego RUBY.

Hemos creado un videojuego RPG inspirado tanto en juegos antiguos como en juegos modernos, y hemos dado nuestro particular punto de vista a un género con el que todos nosotros hemos crecido.

2. Equipo de desarrollo

Jose M^a Oliet Villalba : Jefe de proyecto/ diálogos

Marina Fernández Gutiérrez : diseñadora arte y sonido/ diseño de nivel

Enrique Coronado Barco: programador/ diseño web/ diseño de interfaces/ diseño de nivel

Javier de la Peña García : programador/ diseño de interfaces

Jesús Villafranca Martínez : testeador/ diseño de nivel

3. Ficha técnica

Equipo de desarrollo: TEAM RUBY (Equipo C)

Género: RPG por turnos, dungeon crawler, simulador de granja

Plataforma: PC

Idioma: Español

Modo de juego: Un jugador

Público objetivo: niñas de 12 a 15 años

4. Historia

Ruby está pasando el verano en casa de su abuela, pero los días son largos y no hay mucho que hacer, por lo que se propone averiguar el mayor misterio de la zona: ¿Hacia va donde va el gato Manchas cada día?

Para ello debe aventurarse en el Bosque de Piedra y enfrentarse a los numerosos enemigos que se interpongan en su camino.

Para esa tarea cuenta con la ayuda de un misterioso poder mágico que le permite formular hechizos a partir de las flores que lleva.

5. Objetivos del juego

El objetivo del juego es resolver el misterio, para ello debes explorar una mazmorra, enfrentarte a los enemigos que la vigilan y llegar al final del nivel.

Para ello la jugadora debe prepararse en la fase previa, la granja, donde siembra y cosecha los que serán sus hechizos.

Al final del nivel o al ser derrotada se presenta la posibilidad de recomenzar el ciclo de la mazmorra, conservando todo el progreso obtenido y volviendo a la granja, en la que con nuevos recursos fortalecemos aún más a nuestro personaje y lo preparamos mejor.

El juego invita a la jugadora a cumplir con la meta principal y además a mejorar al personaje todo lo que quiera.

6. Controles

Movimiento: teclas WASD/ flechas de teclado

Interacción con objetos/NPC's: ratón estando próximo

Menú de pausa: tecla ESC

Abrir inventario: tecla e

Interfaces y menús: movimiento de ratón y click izquierdo

7. Mecánicas

Las tres mecánicas principales son la gestión de recursos, la exploración de la mazmorra y el combate.

Gestión de recursos: en la granja cultivamos las plantas que serán nuestros hechizos; además podemos comerciar con la abuela para obtener más recursos a cambio de dinero.

Exploración de la mazmorra: cada día hay una diferente, repleta de enemigos y tesoros, en ella te espera un temible jefe final al que derrotar. Cuando acabas la mazmorra comienza un nuevo día y, además de cambiar su estructura, incrementa la dificultad (un máximo de tres veces).

Combate: el combate contra los enemigos está basado en un sistema por turnos, cada acción cuesta un turno, en el que el enemigo también emprende una acción.

8. Estados del juego

Casa



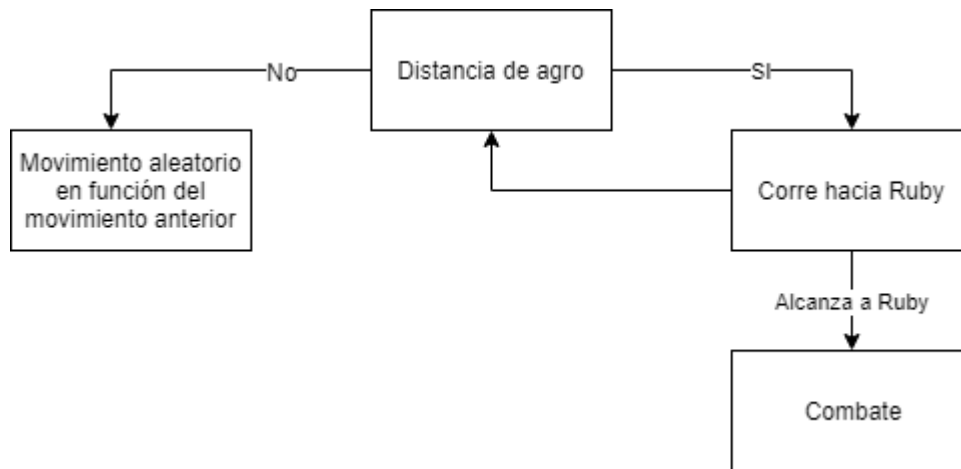
Es el estado principal del juego, en él se desarrollan las acciones de cultivo y comercio, así como las conversaciones entre Ruby y su abuela, que sirven como tutorial, para situar al jugador en el juego y explicarle el fin de este.

Mazmorra

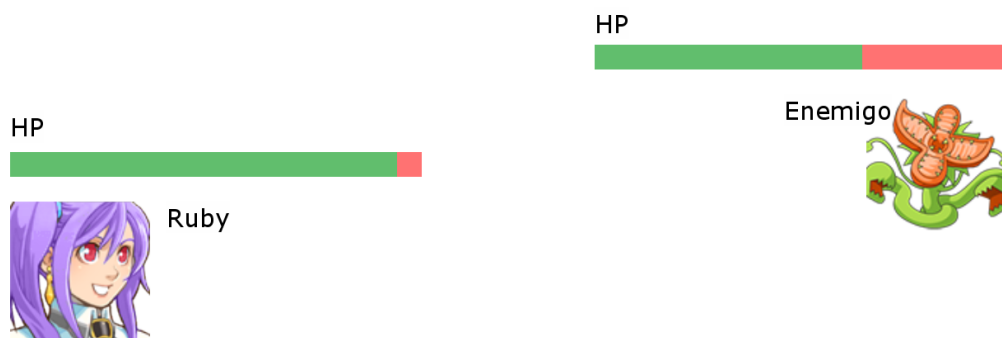


Para que Ruby pueda encontrar a su gato ha de ser capaz de acabar con un boss dentro de la mazmorra, el cual se ha apropiado de él. En este escenario encontraremos además del boss




muchos enemigos distribuidos por el mapa y que dificultarán finalizarlo, estos enemigos tienen capacidad de movimiento respecto a una IA por estados para simular un movimiento inteligente en el cual aumentaran su velocidad e irán en dirección al jugador en caso de tener agro:



Combate



Selecciona ataque:

 Agua x14
  Fuego x2
  Rayo x0

ATAQUES

POCIONES

HUIDA

Cuando Ruby colisiona con un enemigo o un boss salta este escenario, en él se realiza un combate por turnos, en los que se utilizarán los recursos obtenidos del estado Casa. Este estado puede resolverse de tres formas distintas:

- **Intentando huir**, en el caso de conseguirlo se volverá al estado mazmorra y los enemigos en agro con Ruby se detendrán momentáneamente para dar oportunidad al jugador de escapar de una situación desfavorable.
- **Ganar el combate**, si el jugador acaba con el enemigo recogerá los tesoros que este tiene (dinero, semillas y plantas); si se trata de un enemigo normal desaparecerá del mapa de mazmorra y el jugador podrá continuar, en el caso de ser el boss de la mazmorra, Ruby recupera su gato y vuelve a casa.

- Si pierdes el combate serás enviado a casa perdiendo una cantidad de dinero y reiniciándose la mazmorra.

9. Análisis del código

CLASES MAS IMPORTANTES

Colision_Service.java

Es una clase a modo de servicio, todas sus funciones son static y es usada a lo largo de todo el juego. En ella se tratan las colisiones con los muros, las colisiones con personajes o enemigos y las colisiones por salto de escenario.

Es mencionada en este apartado por el manejo de las colisiones con los muros tanto por parte del jugador como de los enemigos. El programa, en el caso de una colisión, te devuelve a la posición anterior y, en el caso de que el vector **movimiento** esté compuesto por valores distintos de 0 para X e Y, es decir, se pulsen varias teclas al mismo tiempo de movimiento, se recalculan las distintas posiciones cercanas a ese vector en las cuales dicha colisión no se produce. Esto se traduce en una colisión que no bloquea el movimiento de los personajes en el caso de un movimiento diagonal y son arrastrados en la dirección en la que no hay posibilidad de colisión.

Mapa.java

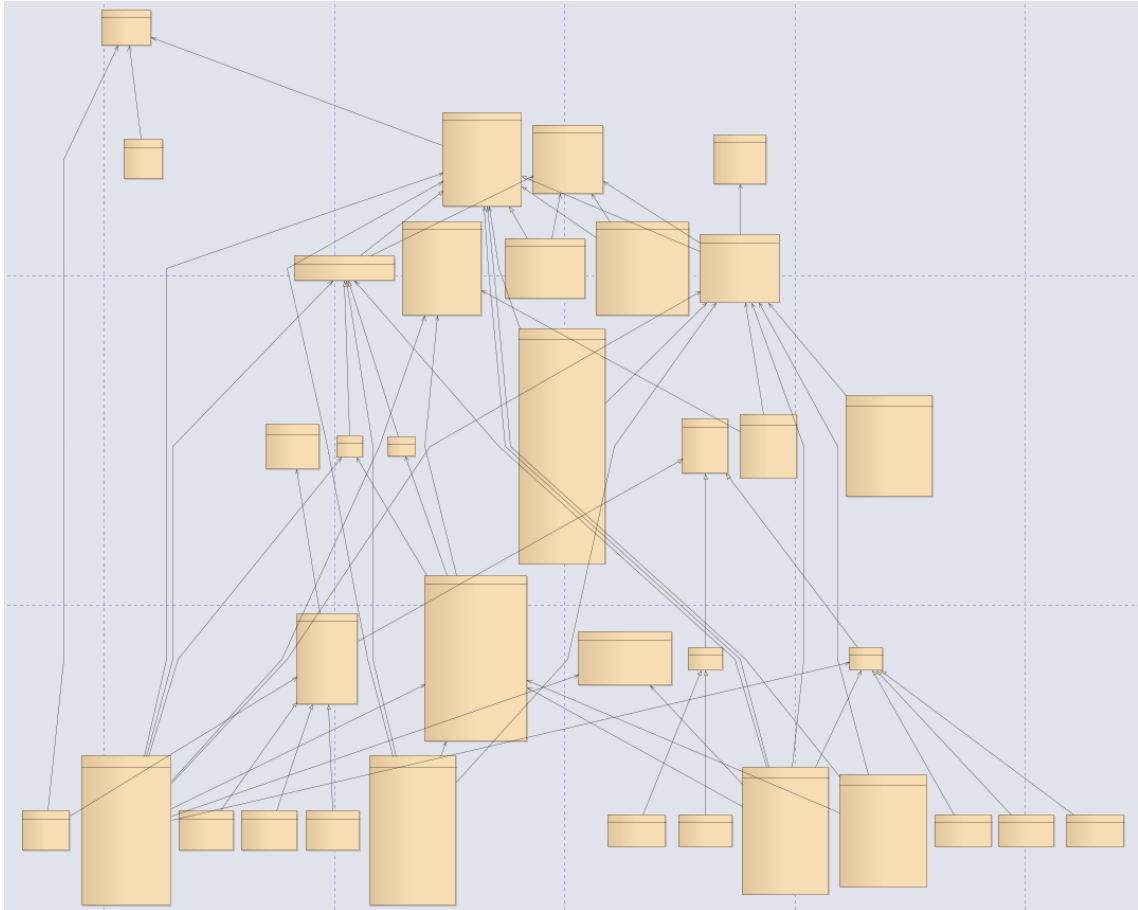
Se encarga de cargar los mapas de los distintos estados, es totalmente independiente del mapa que se le mande por parámetro (la ruta del mapa) y genera los distintos elementos del mapa a partir de las capas que implementa TiledMap y sus nombres. Además, se encarga de actualizar el posicionamiento de los elementos respecto al movimiento del jugador y por último de renderizar los elementos que contiene; de esta forma los mapas y sus elementos son cargados, actualizados y renderizados de forma muy sencilla en todos los estados que los usan.

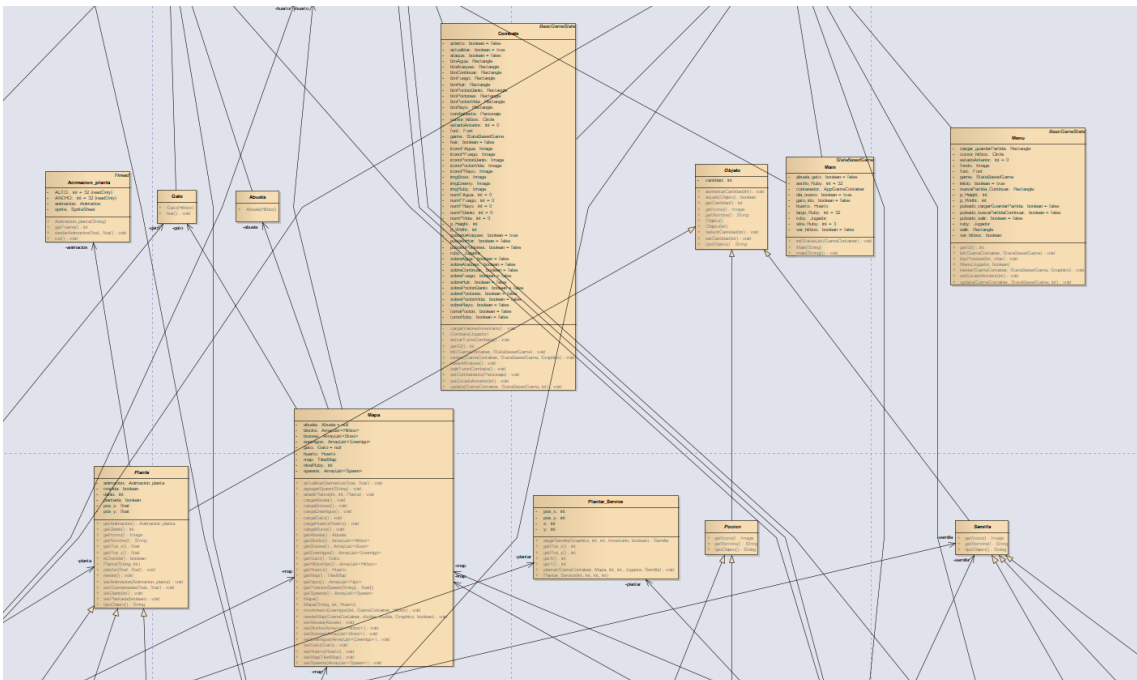
Plantar_Service.java y las clases que componen las mecánicas de plantar

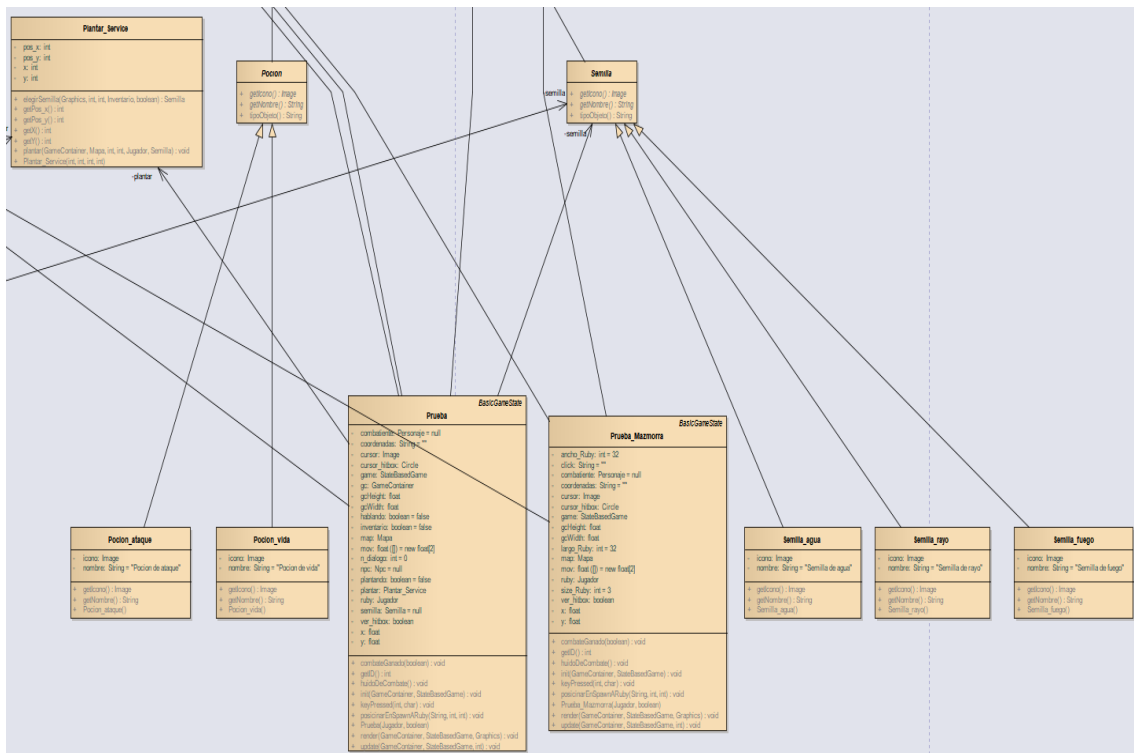
Uno de los pilares más importantes del juego es la mecánica de plantar y como actúa el huerto a lo largo del juego como un medio de recursos. La forma de acceder a esta mecánica es mediante la clase **plantar**, la cual despliega una interfaz en la que se muestran las distintas semillas que podemos Plantar_Service.java. La clase cargará en un objeto **Huerto** perteneciente al mapa del estado Casa un objeto **planta** con su animación. Esta planta será renderizada por medio del render del huerto y tendrá un periodo de crecimiento que funcionará en paralelo del resto del juego. Concretamente su animación extiende al objeto **Thread** de Java para que el crecimiento de las plantas no dependa de que Ruby esté en el estado y así poder ir a las mazmorras y volver para recolectar las plantas ya crecidas.

10. DIAGRAMA DE CLASES

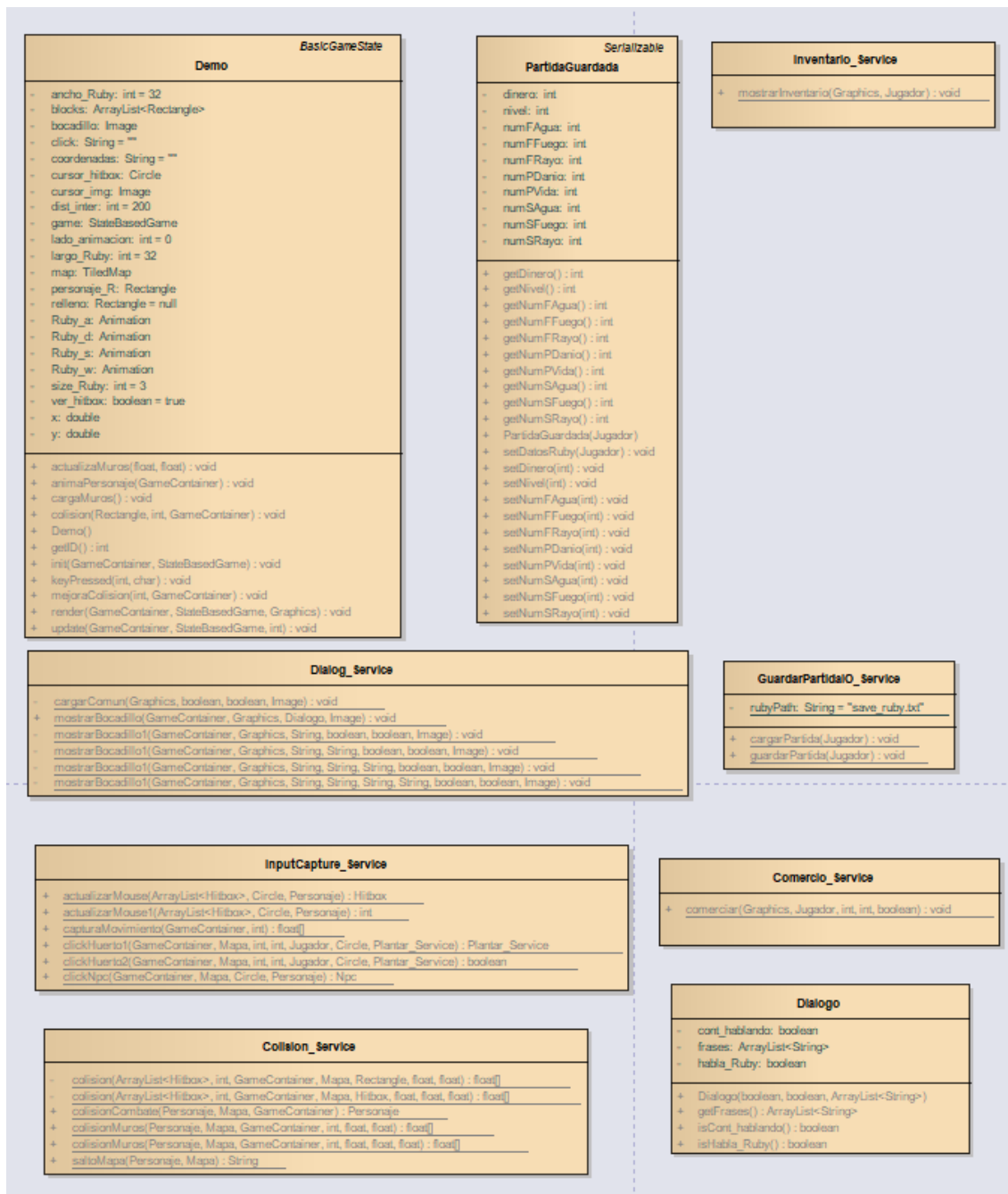
Respecto a las clases con referencias vamos a incluir un mapa genérico para facilitar el entendimiento de las distintas capturas que lo forman, esto debido al tamaño del diagrama.







Clases como servicios o independientes



11. Tecnología, herramientas y fuentes utilizadas

Tecnología y herramientas

- **Netbeans:** entorno de desarrollo empleado en la programación del videojuego.
- **Java:** lenguaje de programación
- **Librería Slick2D:** librería utilizada en el entorno de desarrollo.
- **Github y GitHub Desktop:** control de versiones del proyecto.
- **Adobe Xd:** diseño de interfaces.
- **Adobe Photoshop CS6:** diseño gráfico y tratamiento de imágenes
- **colors.co:** web utilizada para definir la paleta de colores de la interfaz del estado combate.
- **svgtopng.com:** web empleada para la conversión del formato **svg** a **png**
- **online-video-cutter.com:** web para la edición y conversión de audio.
- **TiledMapEditor:** construcción de mapas.
- **imageenlarger.com/es:** web usada para redimensionar texturas y crear nuevos mapas
- **donjon.bin.sh/5e/dungeon/:** web usada en la creación de los layouts aleatorios.
- **Paint:** Edición

Fuentes

- <http://slick.ninjacave.com/>: tanto para el javadock de la librería como para los ejemplos, muy útiles en el desarrollo del juego.
- www.youtube.com/user/MakiGAS93: para aprender a usar la librería Slick2d
- www.youtube.com/user/CodingQuickTips: para aprender a usar e implementar TiledMap en Slick y la aplicación por capas de elementos del escenario.
- opengameart.org/content/tiled-terrains: asset del mapa sandbox
- www.flaticon.com para el icono de Github de la web que sirve de enlace al repositorio.
- Sprites de Ruby y su abuela sacados de RPG Maker VX (<http://www.rpgmakerweb.com/products/programs/rpg-maker-vx>) y editados posteriormente.
- Los sprites de los enemigos son de otras versiones de RPG Maker, también editados posteriormente.
- Paquetes de texturas sacados del RPG Maker (todas las versiones) y de los foros. En especial los árboles son de Celianna (<https://pixanna.nl/>).
- Los sonidos sacados de YouTube:
 - Mazmorras: <https://www.youtube.com/watch?v=YJJikZxnLVQ>
 - Sonido de ambiente: <https://youtu.be/IKEHYRkShBY>
 - Iconos pociones: <https://graphicriver.net/item/magic-bottles-icons/19679073?ref=KlitVogli>
 - Iconos plantas: <https://graphicriver.net/item/herbal-icons/19649778?ref=KlitVogli>
 - Icono semillas sacadas de la wiki de Stardew Valley: https://es.stardewvalleywiki.com/Stardew_Valley_Wiki

- Los layouts de la mazmorra se han sacado de <https://donjon.bin.sh/5e/dungeon/> y se han editado posteriormente.

12. Problemas encontrados en el desarrollo.

- Para el correcto uso de la librería Slick2D ha sido necesario un proceso de aprendizaje por parte de los programadores.
- Para la investigación del proceso de carga del mapa y la creación de los mapas de prueba ha sido necesario también aprender a usar la herramienta TiledMapEditor.
- Aprendizaje de distintas herramientas de edición gráfica.
- Problemas de compatibilidad Tiled-Slick2D, solucionados con edición manual de los archivos .tsx
- **Problemas en el salto de estados entre mapas:** el personaje saltaba de forma directa sin moverse de la hitbox del spawn y se detectaba un salto continuo entre escenarios de forma continuada e infinita. Para resolverlo se estableció un reposicionamiento por desplazamiento respecto al spawn: si el spawn se encuentra en la zona norte se reposiciona el personaje unas casillas más hacia el sur del spawn para no causar la colisión y tener una transición entre escenarios correcta.
- **Realización e implementación de la IA:** el movimiento de los enemigos está ligado al valor de entero *i* de la función *update()* del bucle del juego que nos aporta Slick y dicha función no siempre es consistente, dándoles unas velocidades muy altas en cargas del juego entre estados. La solución, aunque un poco tosca, ha sido poner un límite a la velocidad que pueden adquirir estos enemigos.
- **Guardado de la partida:** ya que no era posible el guardado por serialización, la solución ha sido agregar una clase con los valores más importantes que se guardará entre partidas.
- Ha sido necesaria una modularización del código según el proyecto crecía, por lo que se ha requerido un tiempo corto para adaptarse al nuevo código.
- **Detección de colisiones con los muros del mapa,** sobre todo evitar que al colisionar el personaje no se salga del mapa y lograr que el personaje se pueda "deslizar" por el muro. Para ello se ha ajustado el sistema de colisiones de manera que al chocar el personaje con el muro, este vuelve a la posición anterior al instante del golpe.
- **Implementación de la acción de plantar las plantas:** al plantar, las plantas se tienen que renderizar en una posición diferente cada vez que Ruby se mueve, la solución pasó por la creación de un objeto huerto que contenía las hitbox de las posiciones del huerto y las plantas allí plantas, todo ello guardado en el mapa en forma de una matriz que contenía en cada posición la hitbox y la planta.
- **Guardado en cada uno de los personajes los diálogos que usarían y método para usarlos** a partir de las funciones ya creadas para la interfaz de diálogo, para

ello se creó un array con las frases para cada persona, quién hablaba en cada momento y si la conversación continuaba.

- **Implementación de los diálogos según el estado de juego:** transición del estado hablar al estado comerciar, activación de diálogo al colisionar con el boss, entre otros ejemplos. Para esto se crearon booleanos, un entero para el control del estado en el que se encontraba el tutorial, frases clave para el paso de entre el dialogo y otras acciones y controladores que indicaban que frase tenía que decir cada personaje en cada momento.
- **Escala del mapa:** se tuvo que cambiar la escala del mapa para ajustarse a la del personaje varias veces.
- **Fase granja problemática:** el mapa que contiene la granja funcionaba a una tasa de frames alarmante. Para solucionarlo se unificó la decoración del mapa en una misma capa, cuidando de que las partes de que siguiesen funcionales todos los sistemas implementados.
- **Trabajo en equipo:** ha habido problemas de comunicación entre los distintos miembros del equipo a lo largo del desarrollo, lo que ha dificultado el proceso de asignación de tareas y el correcto seguimiento del cumplimiento de hitos a lo largo del proceso.