**PROGRAMACIÓN**

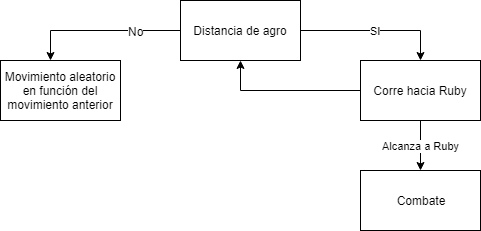
**ESTADOS PRINCIPALES**

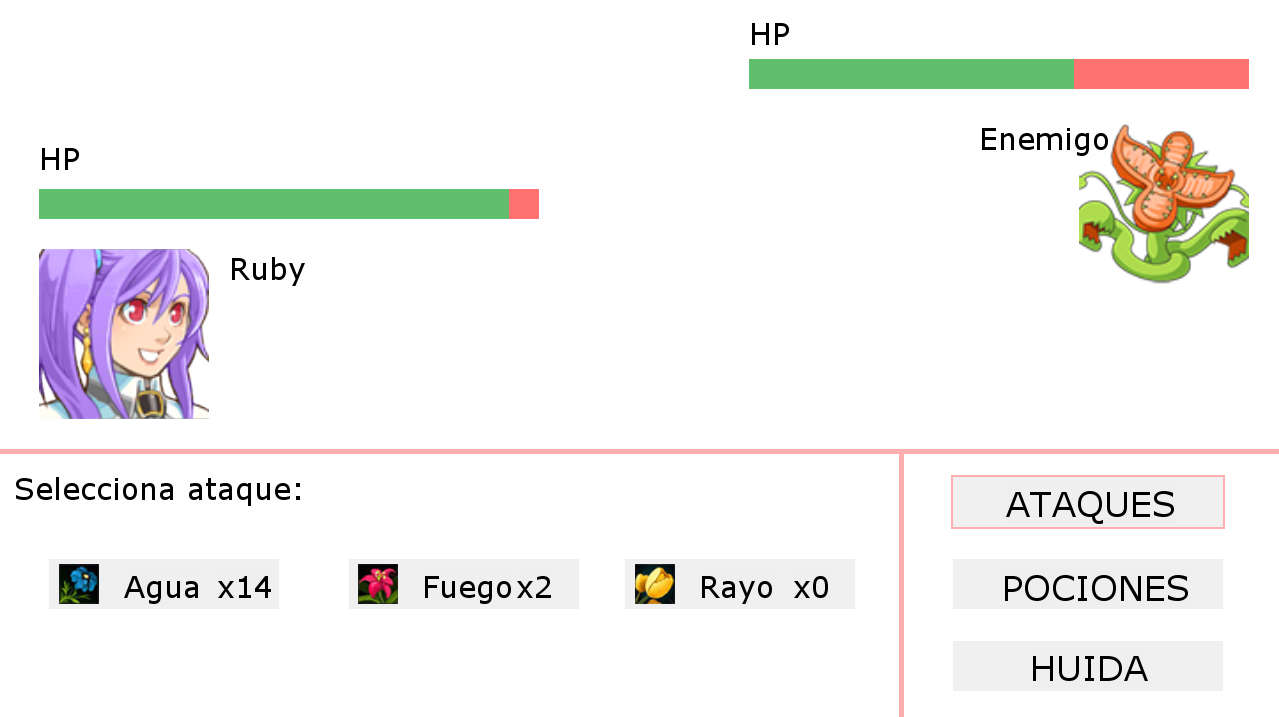
**Casa**

Es el estado principal del juego, en él se desarrolla las acciones de cultivo y comercio, así como las conversaciones entre Ruby y su abuela que sirven como tutorial y 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 dificultaran 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**

Cuando Ruby colisiona con un enemigo o un boss saltara este escenario, en el se realiza un combate por turnos, en los que se utilizaran 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.

**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 la trata 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 este 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 produzca. 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**

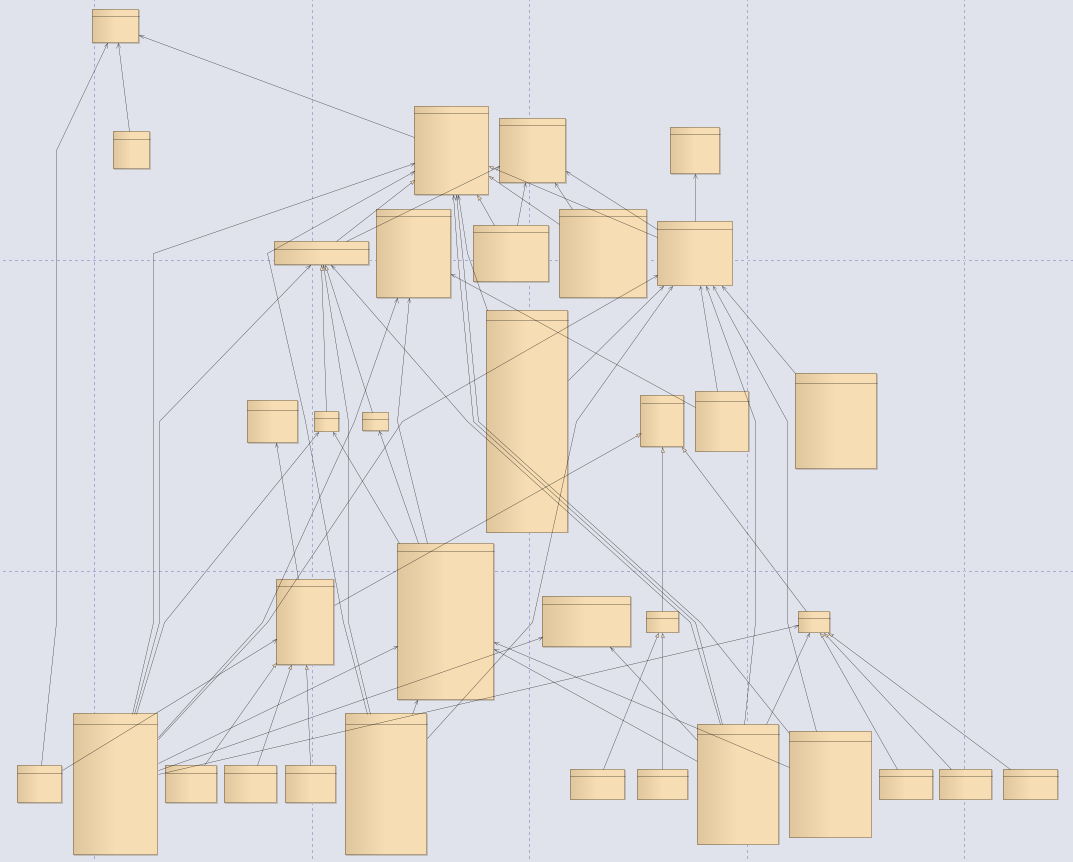
Se encarga de cargar los mapas de los distintos estados, es totalmente independiente del mapa que se le mande por parámetro (la ruta del mapa), genera los distintos elementos del mapa a partir de las capas que implementa TileMap y sus nombres. Además, se encarga de actualizar el posicionamiento de los 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.

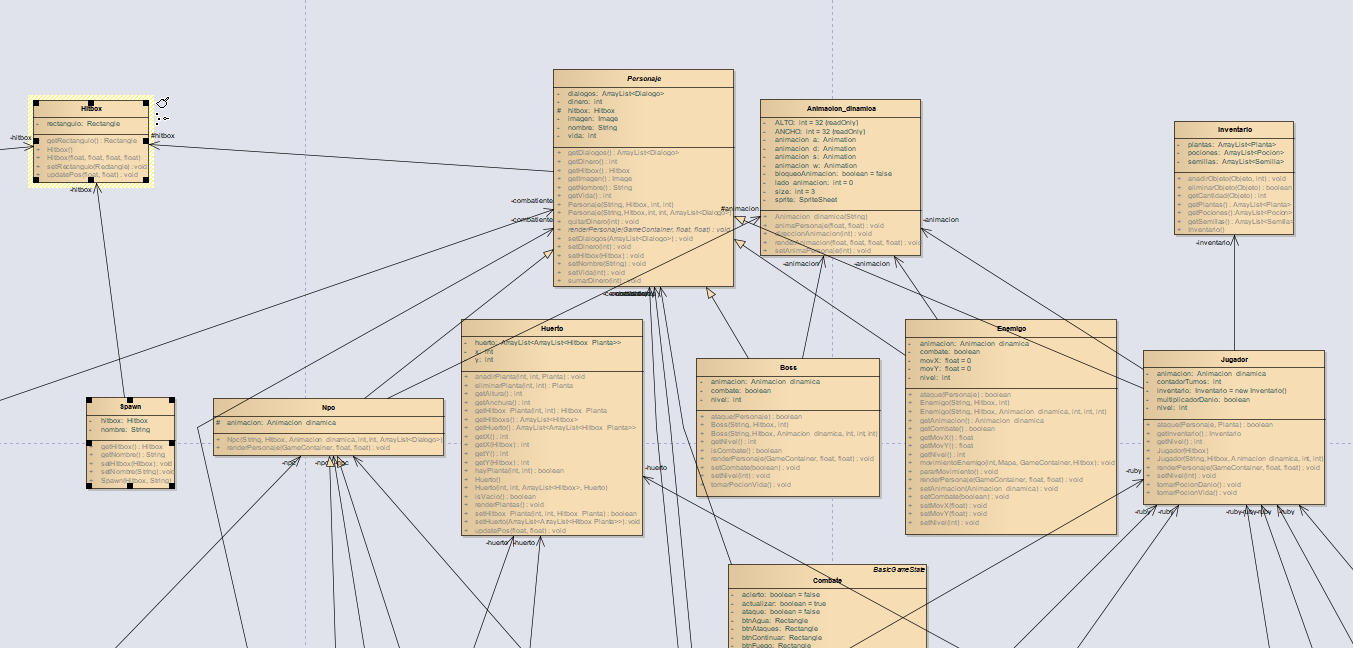
**Plantar\_Service.java y las clases que componen las mecánicas de plantar**

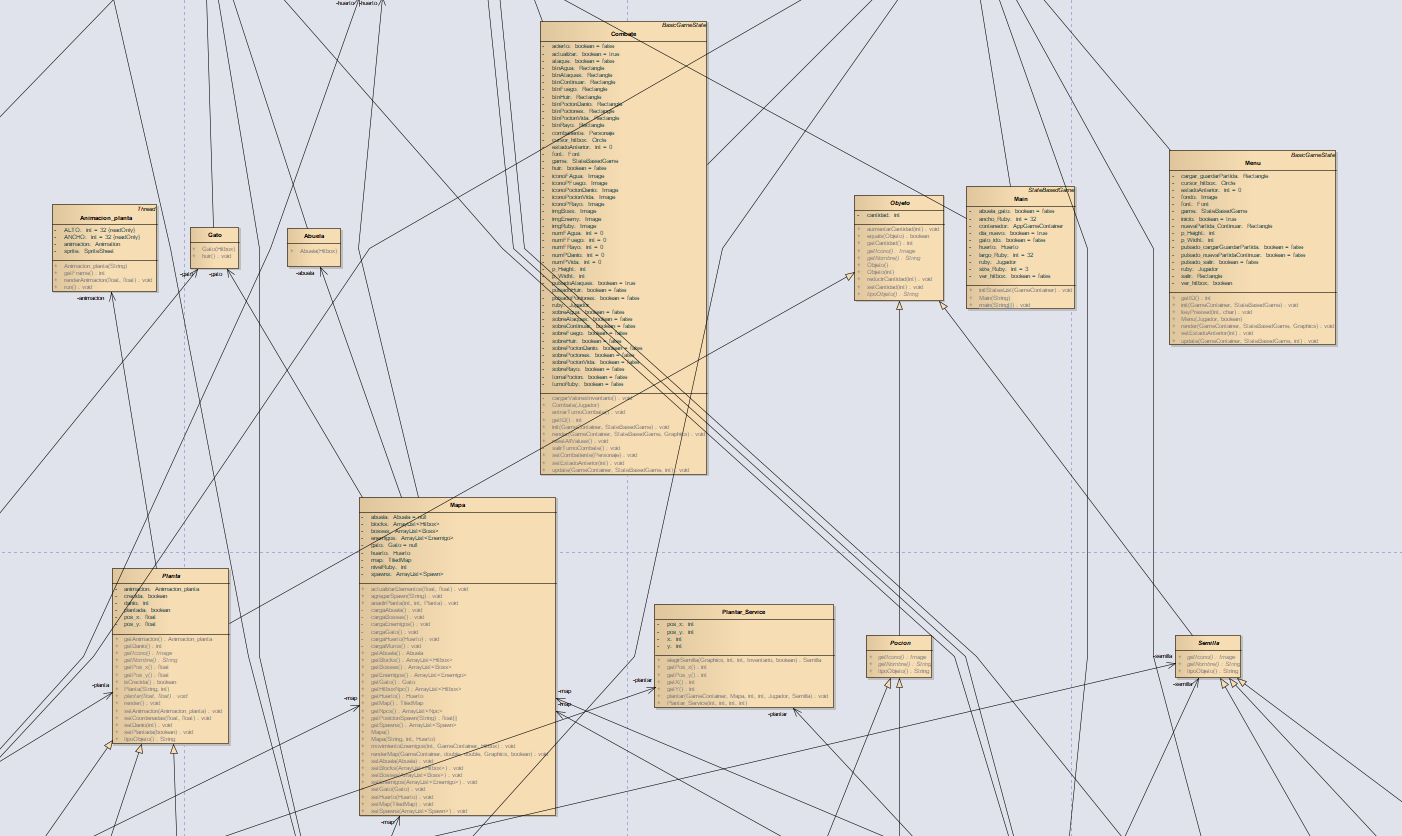
Uno de los pilares mas importantes del juego es la mecánica de plantar y como actúa el huerto a lo largo del juego como un medio de recursos. La forma de acceder a esta mecánica es mediante la clase plantar, la cual despliega una interfaz en la que se muestran las distintas semillas que podemos Plantar\_Service.java. 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 tendrán un periodo de crecimiento que funcionara en paralelo del resto del juego, concretamente su animación extiende al objeto Thread de Java para que el crecimiento de las plantas no dependiese de que Ruby estuviese en el estado y así poder ir a las mazmorras y volver y recolectar las plantas ya crecidas.

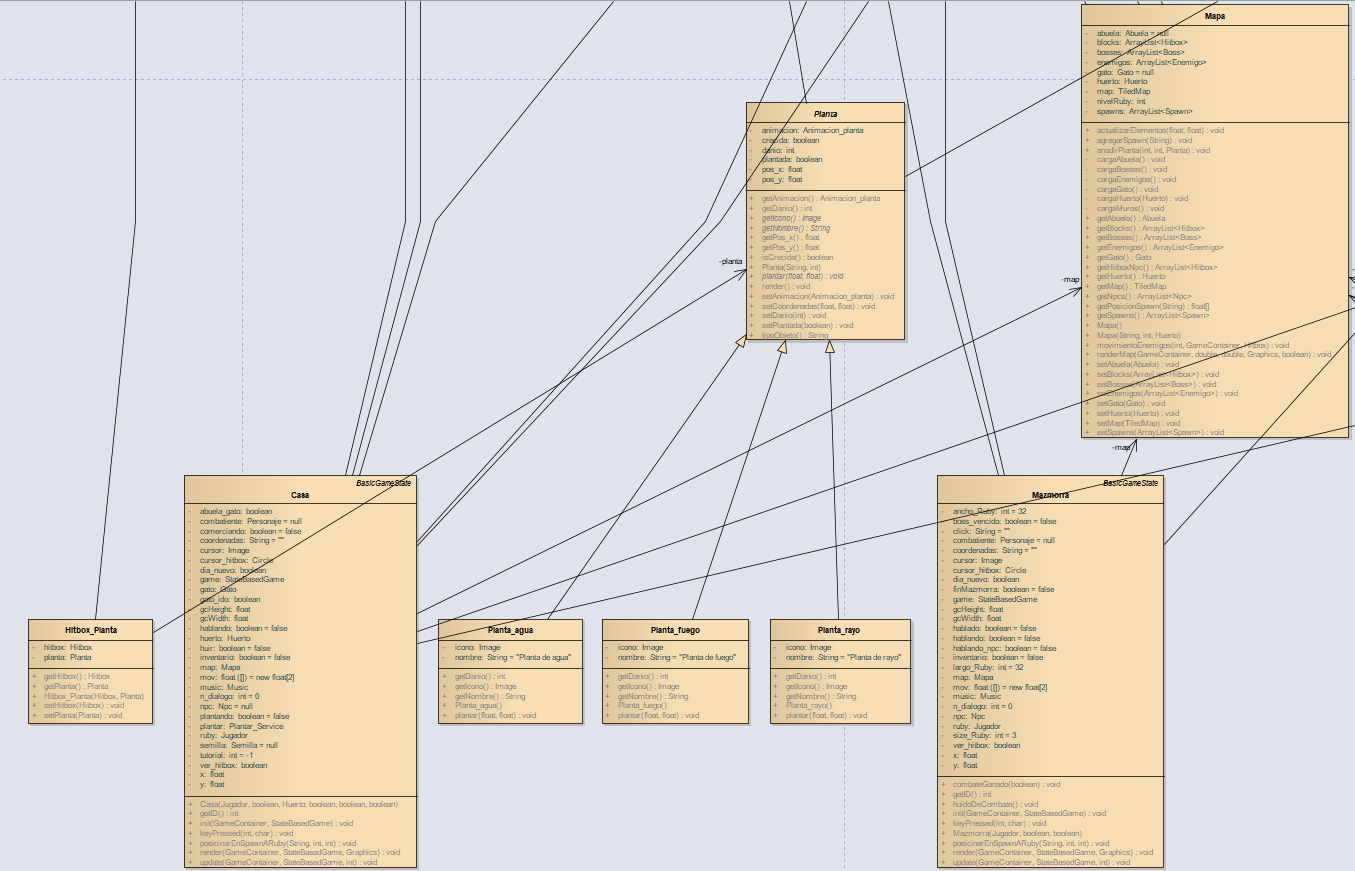
**DIAGRAMA DE CLASES**

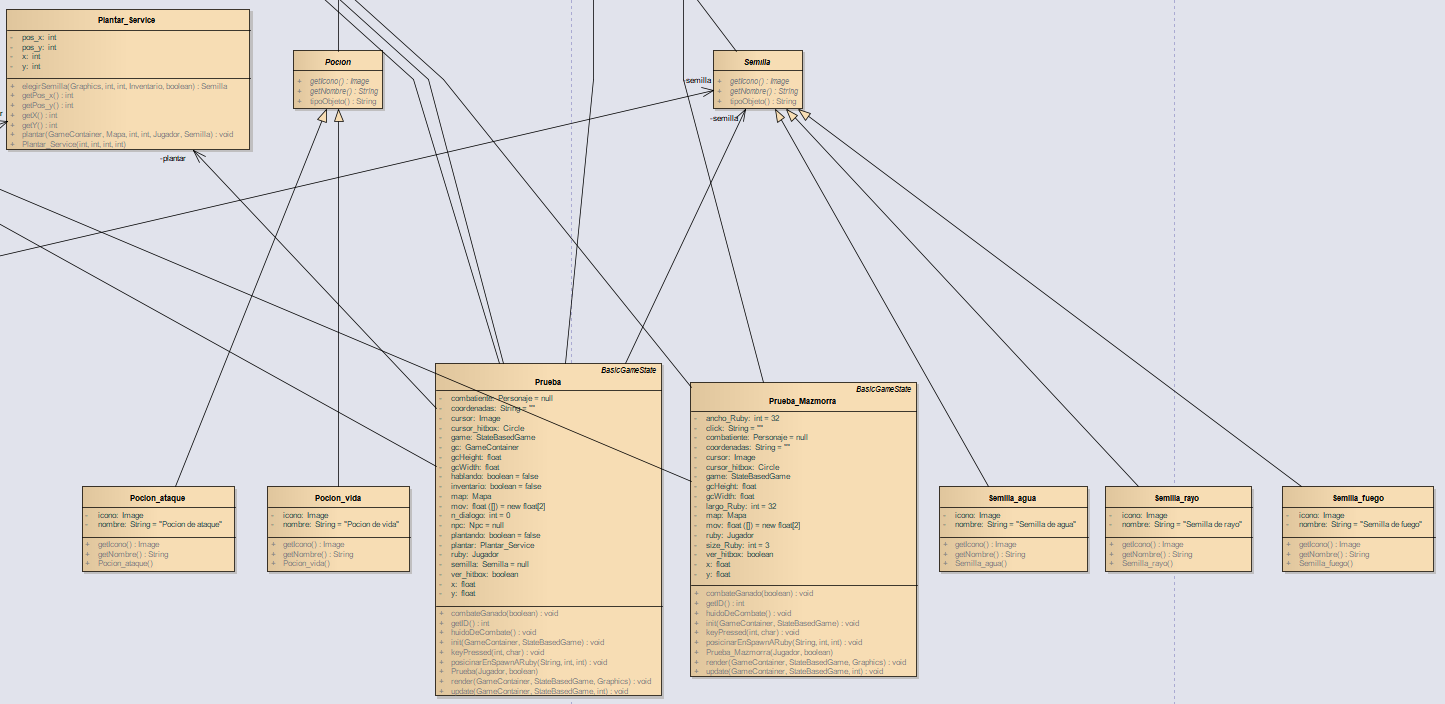
Respecto a las clases con referencias vamos a incluir un mapa genérico para facilitar el entendimiento de las distintas capturas que lo forman, esto debido al tamaño del diagrama.











Clases como servicios o independientes

