UNCLASSIC
TRABAJO PRACTICO
PROGRAMACION ORIENTADA A OBJECTOS

ALUMNOS

- 52056 – BELLINI, JUAN MARCOS
- 54132 – GEORGESCU, FRANCO
- 55291 – RODRIGUEZ NICASTRO, JULIAN
- 55824 – LI PUMA, JUAN FRANCISCO

BREVE DESCRIPCIÓN

El trabajo consiste en un juego de rol, sin ningún objetivo, en el que el jugador puede recorrer una serie de mundos, y realizar distintas actividades.

TOMA DE DECISIONES

Las clases más importantes en las que hay interacción son *entity*, *game* e *ítems*. *Game* controla todo el comportamiento del juego, y funciona como *controller* de la aplicación. En cuanto a *entity*, sus instancias son las que pueden ser representadas en la vista del juego. Toda *entity* puede ser “dibujada” en pantalla.

Una subclase de entity es *player*. La misma representa al jugador. Los ítems son aquellos que pueden modificar el estado del jugador al ser usados (mediante su método *use*).

Para simplificar el desarrollo, existe una interfaz llamada *interactable*, que contiene un método – *respond* – el cual es llamado dentro de un método de *player*, *interact*. Cada clase que implemente dicha interfaz, responde a su manera, sabiendo como responder por sí misma.

De forma similar, el *player* tiene un método que es *useItem*, que llama al método *use* del ítem con el que fue llamado. Por ejemplo, al usar una comida, la misma cura al jugador por si misma. De esta forma no hace falta hacer diferentes métodos de uso de ítems para el jugador.

En cuanto a funcionamiento, se decidió crear un sistema de clases *wrapper* de las clases representables. La mismas contienen la clase en cuestión, y su imagen para ser representada. También, aunque no tenga imagen propia, la clase world tiene su *wrapper*, que contiene las dos colecciones de las clases wrapper de *entity* y *tile*. Esto se hizo así para simplificar el *rendering*, y para evitar errores de modificación de colecciones.

Algo importante a aclarar es el hecho de que durante toda la ejecución del juego, sólo existe una instancia de la clase *game,* un *singleton*. No tendría sentido que existan dos instancias de *game* en el juego, sino que cada vez que se ejecute la aplicación, se crea una nueva, y se utiliza la misma siempre.

Similar a lo anterior, todos los *ítems* que hay en el juego están cargados dentro de clases *factory* (*ConsumableItemFactory* y *EquipmentFactory*). De esta forma se evita crear nuevas instancias de objetos que son siempre igual. Cuando, por ejemplo, se obtiene una espada, se le pide una referencia a la misma a la *factory* correspondiente para poder ejecutar los distintos tipos de métodos (incrementar el inventario, equipar, etc.). Gracias a esto se evitar instancia muchas veces algo igual, ahorrando memoria, y haciendo más eficiente el programa.

Las *factories* contienen un *HashMap* en el que se mapea el nombre del ítem, con su correspondiente instancia. Para pedir el objeto, basta con conocer su nombre. En caso de que no contenga un ítem pedido, el método que se encarga de entregar la instancia lanza una excepción (Esto pasaría únicamente por un error de programación).

Este mismo funcionamiento se aplica al *SkillSet* del jugador.