Informe: Trabajo Práctico Final
Algoritmos y Programación III

Barbetta, Agustina 96528 Lazzari, Santiago 96735
Ordoñez, Francisco 96478

3 de Junio
1er. Cuatrimestre 2015

1 Supuestos

- La unidad mágica "Alto Templario" permite clonar cualquiera de las unidades del jugador, no sólo a ella misma.
- El clon no ocupa espacio en la población.
- El juego finaliza cuando un jugador no tiene ni estructuras, ni unidades ni recursos suficientes para construir los anteriores.

2 Modelo de dominio

Para modelar el StarCraft hemos creado familias de unidades, estructuras, templates y clases como jugador, mapa, juego y auxiliares. Lo primero que hicimos al recibir el trabajo fue tratar de identificar la verdadera responsabilidad que podrían tener los objetos de juego presentados en el enunciado.

Se nos ocurrió implementar la siguiente familia para las unidades (Ver DiagramaDeClasesUnit.png adjunto) habiendo reconocido los datos comunes a todas, los cuales ahora son los atributos de la clase Unit. Las unidades que existen en el juego pueden ser mágicas (Estas tienen energía, poderes y habilidades), regulares (Sólo tienen un ataque con un determinado daño y rango) o de transporte (Estas pueden trasladar otras unidades, tienen una capacidad y carecen de ataques). Las unidades que pueden ser transportadas (Mágicas y regulares) implementan la interfaz Transportable y responden a si pueden volar y cuánto ocuparían en caso de subirse a una unidad de transporte.

De la misma forma implementamos la familia para las estructuras (Ver DiagramaDeClasesStructure.png adjunto), las mismas sólo pueden ser comunes o constructoras. Las comunes no tienen una responsabilidad real, son una simple figurita que debe tener el jugador para aumentar sus recursos o cupo poblacional.

Las constructoras crean las unidades del jugador y su existencia, a excepción de la Barraca, depende de estructuras de menor nivel. Las estructuras constructoras, además de las cualidades de una estructura regular, permiten la creación de unidades, poseen referencias a plantillas que indican como crearlas. Se les pide una nueva unidad invocando el método 'create' en el cual el jugador debe indicar el nombre de la unidad deseada, sus recursos y espacio poblacional. Estas estructuras verifican que haya espacio para la unidad que van a crear, cobran y devuelven un objeto de clase Construction, un paquete con la unidad que sólo se podrá abrir pasado el tiempo de construcción correspondiente a la unidad.

Así como las estructuras de construcción pueden crear nuevas unidades, existe una clase capaz de construir las estructuras del juego (Ver DiagramaDeClasesBuilder.png adjunto). TerranBuilder y ProtossBuilder son singletones que poseen referencias a plantillas que indican cómo crear las estructuras de sus razas. Se les puede encargar una nueva estructura invocando al método 'create' en el cual el jugador debe indicar el nombre de la estructura deseada, sus recursos y dejar asentado las estructuras que posee en el momento. El Builder se encarga de verificar si puede construir lo que se le pide (Recordando que existe una dependencia entre estructuras y, en algunos casos, necesita de una estructura anterior para construir), cobra y devuelve, al igual que las estructuras de construccion, un paquete de clase Construction.

El trabajo de los builders y las estructuras de construcción es posible debido a la existencia de la familia Template (Ver DiagramaDeClasesTemplate.png adjunto), estas plantillas dictan cuáles son las características de todos los objetos del juego. Existe una instancia de plantilla única por objeto (Son singletones) y lo beneficioso de ellas es que, si queremos agregar una nueva estructura u objeto, creamos una nueva heredando de la clase abstracta de plantillas correspondiente y le damos su referencia al builder o estructura constructora. Los templates tienen un método 'create' en el cual crean y devuelven un objeto del tipo del cual son plantilla. Ejemplo: MarineTemplate hereda de la clase abstracta MuggleTemplate y su método create devuelve una instancia de MuggleUnit, en estado válido, con las caracteristicas de un Marine (vida, daño, vision, rango de ataque, etc. respectivos al Marine).

Además de estas familias creamos la clase Map. Map maneja la lógica del terreno. El terreno se pensó como un conjunto de parcelas y sobre una grilla flotante. Esto quiere decir que un punto cualquiera es identificable en el mapa. Por otro lado, el mapa esta subdividido en parcelas que son las unidades básicas de separación de los terrenos. Ejemplo, en una parcela se puede construir una Barraca, es decir que esta no se puede construir en cualquier punto del mapa, su origen y lado van a coincidir con el de la parcela. En resumen, el mapa es una combinación de una grilla discreta de parcelas y a su vez una de puntos flotantes.

Las parcelas van a tener una superficie, esta puede ser de tipo aire o tierra. Estas, a su vez, van a ser explotables, es decir, si la superficie posee algun

recurso, este va a ser explotado, de no ser así, se lanza una excepción de tipo no hay recursos para explotar. Evidentemente, las parcelas de aire van a responder siempre con excepciones mientras que las de tierra no (siempre y cuando se tenga una superficie extractora asociada para extraer un mineral). Una ventaja de esta forma de representar las parcelas en capas, es que cuando uno crea la parcela, esta se puede ir setteando en capas, por ejemplo, se puede comenzar con un mapa de todas parcelas de tierra, luego, se podrían repartir los recursos de manera aleatoria, aleatoria-dirigida, o directamente donde se desee, luego trazar los lugares donde haya aire (pisando la tierra existente y reemplazandola con aire), y finalmente las bases podrían ser apoyadas en algun lugar válido.

Finalmente, la clase generadora de escenarios, tiene metodos para poder armar cualquier tipo de mapas, con distribuciones aleatorias, aleatorias con densidades de minerales por cantidad de parcelas, etc. Esto desacopla el decorado del mapa y las parcelas delegandoselo a la clase generadora que tiene un papel más estético.

Por último comenzamos a implementar las clases Player y Starcraft. Player tiene nombre, color, una referencia al builder de su raza, recursos, una colección de estructuras, una colección de unidades y una ConstructionQueue en la que guardará todos los paquetes Construction que potencialmente serán nuevas estructuras y unidades (ConstructionQueue le puede entregar estos objetos una vez terminado su tiempo de construcción). Este jugador tiene métodos para crear nuevos objetos de juego, perder la referencia a los muertos, recolectar los recursos que le corresponden en un turno y devolver el espacio de la población.

En cuanto a la clase StarCraft, por el momento sólo tiene el main y referencias a un mapa y dos jugadores.

3 Diagramas de clases

Los diagramas de clases elaborados se encuentran en formato .png dentro de la carpeta /Diagramas UML del repositorio.

4 Diagramas de secuencia

Los diagramas de secuencia elaborados se encuentran en formato .png dentro de la carpeta /Diagramas UML del repositorio.

5 Diagrama de paquetes

El diagrama de paquetes elaborado se encuentra en formato .png dentro de la carpeta /Diagramas UML del repositorio.

6 Diagramas de estado

En esta primera entrega, no encontramos situación que se necesite aclarar por medio de un diagrama de estado.

7 Detalles de implementación

Por el momento el punto más conflictivo del trabajo fue darnos cuenta cuáles eran las verdaderas funciones que cumplen los objetos de juego.
Así logramos diferenciar a las unidades en mágicas, no mágicas (muggles) o de transporte. Las mágicas tienen energía y en un futuro resolveremos cómo implementar sus poderes y habilidades, las no mágicas un ataque y las de transporte tienen una collección de las anteriores que pueden llevar a distintos puntos del mapa.
En el caso de las estructuras, comentamos anteriormente que los depósitos y explotadores son sólo figuras que tiene el jugador para incrementar la ganancia de recursos por turno y el cupo poblacional. No es este el caso de las estructuras de construcción, las cuales guardan una colección de plantillas y son capaces de devolver paquetes con nuevas unidades.
Bajo esta misma idea creamos los builders, existe uno por raza y sólo se pueden instanciar una vez (Crear múltiples carece de sentido, no son objetos de juego. Caso contrario son las estructuras de construcción, ya que pueden ser destruidas por los enemigos).
Con el propósito de almacenar los datos de cada estructura y unidad, creamos templates. Estos solo se pueden instanciar una vez. Lo único que saben hacer es crear a la unidad o estructura a la que representan (Por ejemplo: GolliatTemplate puede crear una MuggleUnit con las características de un golliat. Por otro lado, un AsimiladorTemplate va a crear un GasExploiter con las caracteristicas de un asimilador).

8 Excepciones

Se han creado las siguientes excepciones heredando de la clase Exception:

TemplateNotFound
: Es creada cuando no se encuentra una plantilla correspondiente al nombre recibido.

InsufficientResources
: Es creada cuando el valor de la unidad o estructura que se desea construir supera los recursos del jugador.

QuotaExceeded
: Es creada cuando la unidad que se desea construir ocupa más espacio poblacional del disponible.

MissingStructureRequired

Es creada cuando la estructura, que se le ha pedido crear al Builder, depende de una anterior que no posee el jugador.