Informe: Trabajo Práctico Final
Algoritmos y Programación III

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

10 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.
- La "Tormenta Psionica" se divide en dos turnos de 50 puntos de daño cada uno.
- 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 y son capaces de generar poderes), 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 actualizan los atributos del jugador turno a turno, aumentan 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, 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), y una lista de poderes en curso. Este jugador tiene métodos para crear nuevos objetos de juego, mover, atacar, utilizar un poder, entre otros.

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

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 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 implementar los poderes de las unidades mágicas. Cada poder es único y el conjunto es difícil de generalizar. Lo que hicimos fue reconocer las etapas que tienen todos e implementar una clase abstracta de la cual heredamos para redefinir los metodos que ejecutan estas etapas.

De esta forma creamos la clase Power con los métodos activate(), execute(Unit), itsFinished() y auxiliares para obtener el costo energético y rango. Power tiene una clase hija por poder, por ejemplo; Radiación. Radiacion fija su objetivo principal en activate() guardando una referencia a la unidad más cercana al punto que clickeó el jugador cuando accionó el poder. Luego de activarse, el poder será guardado en la colección de poderes activados de Player, al pasar los turnos, los poderes de la colección se actualizarán corriendo el método execute() que, en caso de Radiacion, llama al método executeRadiacion() que tiene la clase Unit (Disminuye en 40 la salud de la unidad). Finalmente, los poderes se eliminarán de la colección de activos si itsFinished() devuelve true. En caso de Radiacion, esto ocurre cuando la unidad que es objetivo principal esta muerta.

Los poderes se inicializan con el llamado del metodo usePower de Player, el cual recibe una referencia a la MagicalUnit que debe generar el poder, el nombre del poder deseado y un punto correspondiente al destino del poder. Para obtener una instancia del poder se llama al metodo usePower de la MagicalUnit y esta recurre a su PowerGenerator. El generador de poderes recibe un nombre y la energía de la unidad. Si la energía le es suficiente para crear el poder que se le pide, devuelve una instancia del mismo, caso contrario, crea una excepción descriptiva. Cuando el jugador obtiene esta instancia de poder, consigue las unidades que se encuentran alrededor del punto recibido por parámetro y dentro del rango de acción. Le pasa al poder las unidades afectadas, lo activa, lo ejecuta por primera vez y lo agrega a la lista de poderes activos.

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.

NoResourcesToExtract
: Es creada cuando se intenta extraer recursos de una superficie de tierra sin recursos.

ConstructionNotFinished
: Es creada cuando se intenta recolectar una unidad o estructura en construcción antes de que el tiempo de construcción haya pasado.

InsufficientEnergy
: Es creada cuando se intenta generar un poder, desde una unidad mágica, sin la energía requerida.

NoMoreSpaceInUnit
: Es creada cuando se intenta subir una unidad a una nave de transporte que no tiene espacio suficiente para llevarla.

NonexistentPower
: Es creada cuando se intenta generar un poder de nombre desconocido.

NoUnitToRemove
: Es creada cuando se desea bajar de la nave de transporte a una unidad que no se encuentra dentro de la misma.

StepsLimitExceeded
: Es creada cuando se desea mover una unidad a una distancia superior a su rango de movimiento permitido por turno.