Informe: Trabajo Práctico Final
Algoritmos y Programación III

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

1 de Julio
1er. Cuatrimestre 2015

1. Supuestos

- Los recursos son infinitos.
- Está permitido crear estructuras no explotadoras en parcelas con recursos.
- Las unidades en construcción mueren si se destruye la estructura creadora.
- La unidad mágica Alto Templario permite clonar cualquiera de las unidades ordinarias del jugador.
- Las unidades embarcan en la unidad de transporte más cercana. De no haber unidades de transporte en su radio de pasos por turno, no podrán subirse a ninguna.
- 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 y su ConstructionQueue está vacía.

2. Modelo de dominio

Para modelar el juego 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 y el Acceso, 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 DiagramaDeClasesStructureTemplate.png y DiagramaDeClasesUnitTemplate.png adjuntos), estas plantillas dictan cuáles son las características de todos los objetos del juego. Lo beneficioso de ellas es que, si queremos agregar una nueva estructura u objeto, sólo 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 (Ver DiagramaDeClases-Map.png adjunto). 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 métodos 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, esta tiene referencia al mapa, los jugadores y constantes que determinan el tamaño de las bases, la densidad de recursos y parcelas de aire en el mapa y los recursos iniciales de los jugadores. Posee métodos para generar el juego, comenzar, y pasar turnos.

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

Los estados reconocidos en el juego son el pase de turnos entre jugadores y la vida de las unidades y estructuras. Los usuarios juegan realizando acciones a gusto y pasando su turno hasta que todos, a excepción de uno, se hayan quedado sin unidades, estructuras y recursos. Allí se anuncia el nombre del ganador y finaliza la partida. Las unidades y estructuras viven hasta que reciben suficiente daño para terminar con su vida.
Se los obviaron diagramas debido a la simplicidad de estas transiciones.

7. Detalles de implementación

7.1. Mover

El metodo move:

debido a que las parcelas no delimitan un mata discreto en cuanto a posiciones de parcelas, se debio implementar un metodo para moverlas que sea independiente de las parcelas.

La manera implementada remite a integrales vectoriales, en donde se calcula el tramo a recorrer y luego se lo dividie en porciones ïnfinitecimales", que son vectores de longitud muy chica.

Se itera sumando esos vectores chicos para que la suma total de el recorrido a mover por la unidad e instantaneamente verifica que la unidad este en una posicion valida que depende de la parcela en la que esta parada. De no poder seguir moviendos, esta se deposita en el vector infinitecimal anterior a no poder pararse.

7.2. Atacar

En un principio, ataque se intentó modelar como se vió en el StarCraft original pero simplificado. En el juego real, las unidades se mueven y atacan al encontrarse con enemigos en su camino, aquí las unidades se movían, llegaban a destino y atacaban a la unidad enemiga más cercana provista por el mapa. Al finalizar el trabajo nos dimos cuenta que el juego era muy difícil, cambiamos las

reglas de forma que las unidades de un jugador ataquen a la unidad o estructura enemiga más cercana al comienzo del turno.
Cabe aclarar que la lógica del ataque se encuentra dentro del método attack de la clase Player, lo que significa que en caso de querer cambiar el momento de ataque, sólo se debe cambiar el lugar de la llamada a este método. Incluso se podría agregar un botón para ejecutar esta acción en la botonera ActionsView.

7.3. Embarcar

El trabajo de embarcar lo tiene la unidad que se transporta, el método embark de la clase Player recibe a la unidad Transportable y busca a la TransportUnit más cercana para entrar. En el caso de que no haya una unidad de transporte en el radio de pasos por turno que tiene la unidad transportable, se levanta una excepción.

7.4. Desembarcar

El trabajo de desembarcas lo tiene la unidad transportadora, el método disembark de la clase Player recibe la TransportUnit y el Transportable que debe salir, simplemente lo busca en su lista de pasajeros y lo retira.

7.5. Poderes

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.

7.6. Generador de escenarios

Cada instancia de la clase StarCraft posee un generador de escenarios con el cual genera su mapa. ScenarioGenerator tiene métodos para distribuir aire y recursos en las parcelas del mapa contenidas en un cuadrado dado (Para ello, recibe un punto que representa la esquina superior izquierda del cuadrado y un entero que representa el lado). Estos métodos además reciben un flotante que representa la densidad de distribución, sea de aire o recursos, que se desea. El mapa instanciado por la clase StarCraft se crea completo con parcelas de tierra, y luego se pasa por el Scenario generator para decorar con parcelas de aire. Por último se generan las bases, el generador tiene un método para distruibuir las bases uniformemente en el mapa, lo que hace es crear una circunferencia de radio $R = \frac{Lado_{MAPA}}{2} - \frac{Lado_{BASE}}{2}$, obtener una n cantidad de puntos distribuídos equidistantemente en el perímetro de la misma y devolver una lista con los mismos. Con esta lista de puntos se generan las bases de los jugadores, utilizando el método para decorar las parcelas contenidas en un cuadrado.

Cabe aclarar que las densidades de aire y recursos en el mapa, así como el tamaño del mismo y de sus bases se encuentran declaradas como constantes en la clase StarCraft pero, en un futuro, se podría adaptar fácilmente el juego y su vista para darle a los jugadores opciones para personalizar el mapa de juego.

7.7. Cancelación de construcción de unidad

El jugador puede pedirle a una determinada estructura que le construya una nueva unidad, pero, si la estructura muere, esta construcción se cancela. La forma de implementar este caso fue que cuando se pide la creación de la nueva unidad se guarde una referencia a la estructura que la creó, tanto la unidad como la referencia a la estructura que la pidio van a estar encapsuladas en una Construction hasta el momento en que se termine de construir la unidad. Cuando la unidad está lista para ser entregada, ConstructionQueue se fija si la estructura que la mando a a crear sigue viva, si no lo esta, se descarta la unidad.

7.8. Vista

La vista esta separada en dos partes, la primera es StartUpView, un frame con paneles dinámicos que guía a los usuarios hasta el inicio del juego. Este objeto se ocupa de preguntar la cantidad de jugadores y sus configuraciones

(nombre, color y raza). Al finalizar, crea una instancia de la clase StarCraft y le pasa la lista de objetos PlayerSetup con los datos de los jugadores.
Una vez instanciado el juego, la vista principal es StarCraftView, un frame con múltiples componentes:

PlayerStatusView
: Un panel que muestra un ícono característico de la raza del jugador, su nombre y sus recursos.

MessageBox
: Un panel con un área de texto en la que se muestran mensajes de error descriptivos.

ActionsView
: Un panel que presenta los botones de acciones de juego, los mismos se habilitan y deshabilitan acorde al objeto de juego que se seleccione.

MapView
: Un componente que representa dos listas de objetos DrawableView; ParcelView y UnitView.

7.8.1. Unidades

UnitView extiende DrawableView y guarda una referencia a la Unit que representa, muestra en pantalla su nombre (con el color correspondiente del jugador), su vida y su escudo, en caso de ser una unidad Protoss. Esta clase implementa un MouseListener y le da las acciones que puede realizar su unidad a ActionsView en caso de ser clickeada, para que esta última habilite los respectivos botones.

A continuación se encuentran las imágenes de las unidades del juego con sus respectivos nombres:

Figura 1: Marine

Figura 2: Golliat

Figura 3: Espectro

Figura 4: Scout

Figura 5: Dragon

Figura 6: Zealot

Figura 7: Nave Ciencia

Figura 8: Alto Templario

Figura 9: Nave Transporte Protoss

Figura 10: Nave Transporte Terran

7.8.2. Parcelas

ParcelView extiende DrawableView y guarda una referencia a la Parcela que representa. Esta clase implementa un MouseListener y le da las acciones que puede realizar a ActionsView en caso de ser clickeada, para que esta última habilite los respectivos botones. En el caso de que la parcela sea vacía, se habilitará el botón de construcción en el panel, en el caso de que tenga una estructura, se habilitará el botón de creación.

A continuación se encuentran las imágenes de las parcelas del juego:

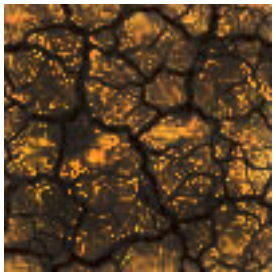

Figura 11: Parcela de tierra

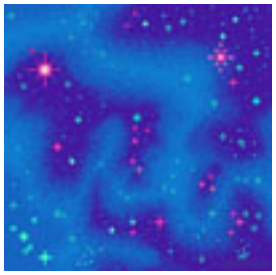

Figura 12: Parcela de aire

Figura 13: Parcela con mineral

Figura 14: Parcela con gas vespeno

Figura 15: Parcela con construcción en curso

7.8.3. Estructuras

Debido a que las estructuras se posicionan en una parcela y la ocupan completa, se decidió no hacer una clase StructureView, ya que es la vista de la parcela la que cambia y sólo en su forma de representación. Para representar a las estructuras, se colocó una figura de una casa, con su nombre corrrespondiente, sobre la parcela de tierra.

Figura 16: Figura de estructura

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.

UnitAlreadyMovedThisTurn
Es creada cuando se desea mover una unidad que ya se movio durante el turno actual.

UnitCannotBeSetHere
Es creada cuando se desea colocar una unidad en una parcela de espacio o con una estuctura.

StructureCannotBeSetHere
Es creada cuando se desea colocar una estructura en una parcela ocupada por otra anterior.

NoReachableTransport
Es creada cuando no existe una unidad de transporte en el radio de pasos permitidos por turno que tiene la unidad.

ConstructorIsDead
Es creada cuando la ha muerto la estructura cradora de la construcción que se quiere liberar.

UnitCantGetToDestination
Es creada cuando una unidad no puede llegar al destino que se le ha indicado.

ColorIsTaken
Es creada cuando el color elegido por el jugador ya está tomado.

NameIsTaken
Es creada cuando el nombre elegido por el jugador ya está tomado.

NameIsTooShort
Es creada cuando el nombre elegido por el jugador tiene menos de cuatro caracteres.

PlayerSetupIncomplete
Es creada cuando la configuración del jugador está incompleta.