**FIUBA - 75.07**

**Algoritmos y programación III**

*Trabajo práctico 2: AlgoCraft*

1er cuatrimestre, 2015

(trabajo grupal)

***Alumnos:***

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Padrón** | **Mail** |
| Matias Iñaki Otegui | 97263 | mati\_otegui@hotmail.com |
| Daniel Sorichetti | 97152 | me@dancodes.com.ar |
| Alexander Villa Jimenez | 95428 | Ale.92xx@yahoo.com |

***Fecha de entrega final***: Miércoles 24/06/2015 - Jueves 25/06/2015

***Tutor***:

***Nota Final***:

Informe

Supuestos

## 

1. El Jugador gana visión a medida que va explorando el mapa o construyendo unidades/construcciones con visión.
2. El jugador en iniciar será el primero que fue agregado al juego.
3. Un enemigo puede atacar una edificación aún en construcción.
4. Si un enemigo ataca a la construcción y la destruye, esta no continuará construyéndose y el jugador que gastó recursos en esa construcción no recuperará los mismos.
5. Una unidad no puede caminar por una zona de recursos.
6. Los recursos recolectados se contabilizan al finalizar el turno del jugador que posee los edificios recolectores.
7. La construcción se actualiza siempre al final del turno del jugador que la está construyendo (el tiempo de construcción se incrementa al finalizar el turno del jugador que dio la orden de la construcción, no siempre que pasa un turno).
8. El límite máximo de población de un jugador es 200, pero esto no evita que pueda seguir creando depósitos de suministro (para el caso del terran) o pilons (para el caso del protoss). Esto puede ser útil en caso de que el enemigo ataque y destruya hospedables, si tiene cierta cantidad de más puede mantener su nivel de población elevado por más tiempo.
9. Si hay una unidad terrestre sobre una celda, no se pueden construir edificios.
10. El escudo regenera por turno el 5% de su valor máximo, si está dañado, por lo que es diferente la regeneración según la unidad/construcción.
11. Si una unidad finaliza su entrenamiento permanece en el edificio hasta que el jugador la mueva de su ubicación (la "active")
12. Si una celda está ocupada por otra unidad o construible la unidad no se moverá hacia ella.
13. Un jugador enemigo no puede ni mover, ni ordenar que ataque, ni activar ninguna unidad/construcción de otro jugador.
14. Un jugador puede mover/atacar una vez por turno, si el ataque o el movimiento fue fallido por alguna razón, por ejemplo coordenada inválida, no se contabiliza la acción como ataque/movimiento, es decir puede volver a realizar la acción hasta que la ejecute correctamente.
15. La radiación afecta de forma diferente a cada unidad, decrementa la vida en un 20% del valor de la vida original, cuando fue creada la unidad.  
    Es decir si poseía 100 de vida al crearse pierde un 20%, es decir 20, pero en el siguiente turno (que poseerá 80 de vida), perderá nuevamente un 20% del valor original, es decir 20 nuevamente.
16. Un edificio militar puede ubicar una unidad entrenada a una distancia igual o menor a su visión.  
    Es decir, si posee una visión de 7, entonces podrá colocar la unidad en un rango radial de 7 desde su ubicación.
17. Una unidad alucinada no consume suministros, necesita 0.
18. Una unidad alucinada se ubica, luego de ser creada, en un rango radial disponible de 5 de la unidad a alucinar. El algoritmo busca una ubicación disponible automáticamente.
19. Un usuario inicia en una base pero puede crear más, su costo es (150 minerales, 100 gas vespeno, la construcción dura 10 turnos y posee una visión de 10).
20. Si una unidad “sube” a una unidad de transporte, o carro, “desaparece” del mapa pero continúa como aliada en los registros del jugador y está ubicada en el carro.
21. Si el carro es destruido, se pierden las unidades que contenía.

## Modelo de dominio

Empezando por lo básico, tenemos la clase **Juego**. Esta es la encargada de contener el Mapa, los Jugadores y manejar el transcurso de los turnos. Es un singleton, porque solo puede haber una instancia de un Juego a la vez.

Luego **Mapa** es la clase responsable de organizar las **Celdas**, calcular distancias, y enumerar elementos que cambian la dinámica del juego solo por existir (recolectores, proveedores de suministros, unidades). Las Celdas se ubican y se asocian con **Coordenada**, la clase que representa puntos en el espacio.

**Celda** es la clase que almacena unidades y construibles (de ahora en más, Controlables). Es responsable de saber si el piso es espacial o no, si tiene recursos, y si tiene controlables ya instalados en ella.

Por otro lado está **Jugador**, la clase que representa a un usuario que está jugando. Almacena información cómo su nombre y color, y por herencia se separa en JugadorTerran y JugadorProtoss con acciones específicas de construcción de edificios.

**Jugador** también, al finalizar su turno, itera sobre sus Unidades, sus Construcciones, los Entrenamientos efectuados y las Magias presentes, efectuando todas las acciones que estos requieran por turno (entrenar un Marine, quemar por radiacion etc).

Dichas construcciones se basan de **Construccion**, una clase abstracta de la cual salen **ConstruccionMilitar, ConstruccionHabitable** y **ConstruccionRecolectora**, cada una con sus respectivas herramientas para lograr su trabajo.

Finalmente tenemos **Unidad**, aquella entidad que representa todo aquello que se mueva y ataque. Provee visión y ocupa suministros.

Datos faltantes completados:

*Construcciones*

*Terran:*

- Centro de Mineral: ocupa 1 celda

- Refineria: ocupa 1 celda

- Deposito de suministro: ocupa 2 celdas

- Barraca: ocupa 4 celdas

- Fabrica: ocupa 6 celdas

- Puerto Estelar: ocupa 6 celdas

*Protoss:*

- Nexo Mineral: ocupa 1 celda

- Asimilador: ocupa 1 celda

- Pilon: ocupa 2 celdas

- Acceso: ocupa 4 celdas

- Puerto Estelar: ocupa 6 celdas

- Archivo Templario: ocupa 6 celdas

*Unidades*

\*todas las unidades ocupan 1 celda

*Terran:*

- Marine: su velocidad es de 2 celdas por turno

- Golliat: su velocidad es de 1 celdas por turno

- Espectro: su velocidad es de 3 celdas por turno (es volador)

- Nave ciencia: su velocidad es de 3 celdas por turno (es volador)

- Nave Transporte: su velocidad es de 5 celdas por turno si posee hasta 4 de capacidad

si posee 5 o mas de capacidad disminuye a 3 celdas por turno. (agregado al enunciado, no es tan complicado de hacer)

*Protoss:*

- Zealot: su velocidad es de 2 celdas por turno

- Dragon: su velocidad es de 1 celda por turno

- Scout: su velocidad es de 3 celdas por turno (es volador)

- Alto Templario: su velocidad es de 2 celdas por turno.

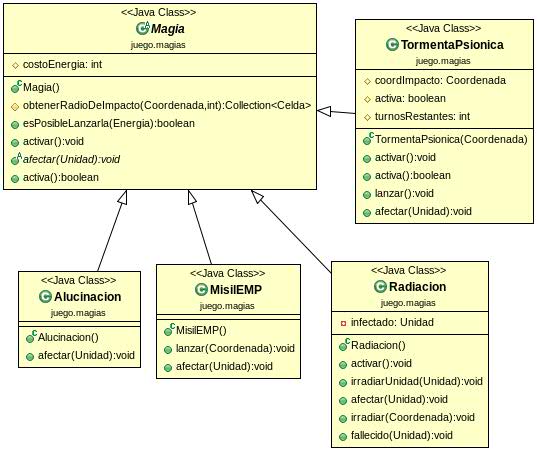
- Nave de transporte: su velocidad es de 5 celdas por turno si posee hasta 4 de capacidad

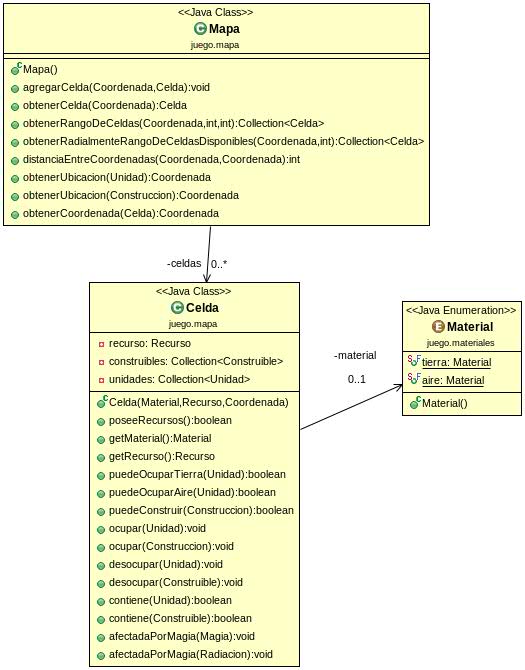
si posee 5 o mas de capacidad disminuye a 3 celdas por turno. (agregado al enunciado, no es tan complicado de hacer)

Diagramas de clases

## 

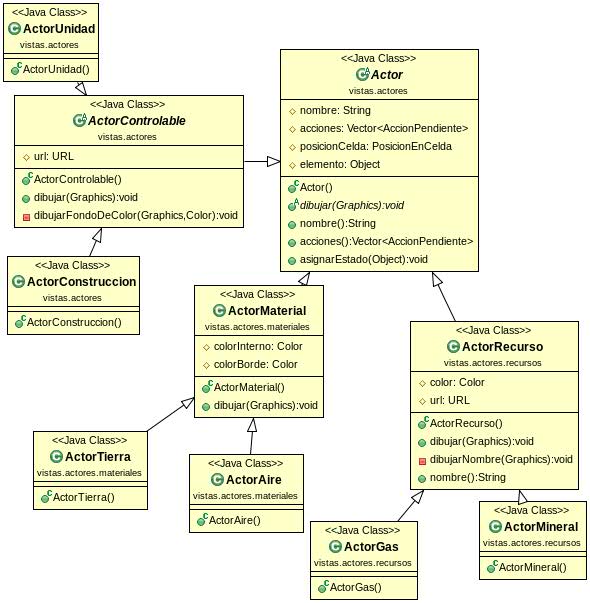


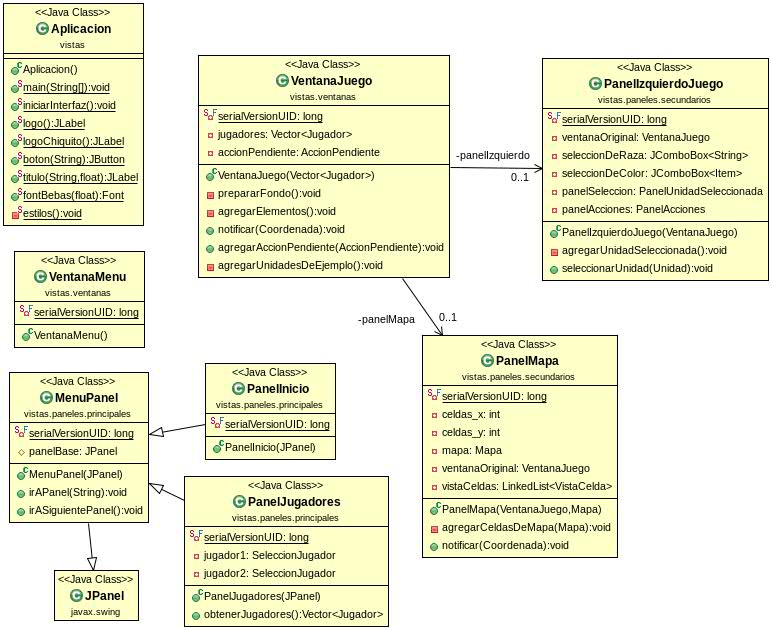




## ENMmLzw.jpgzxn8Cc2.jpg

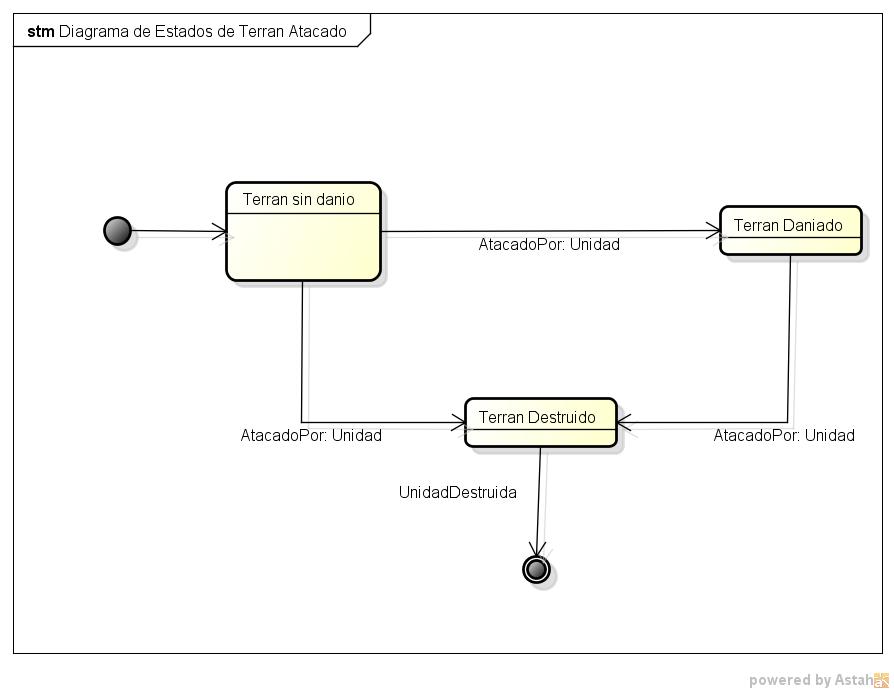
## VentanaJuego.jpg

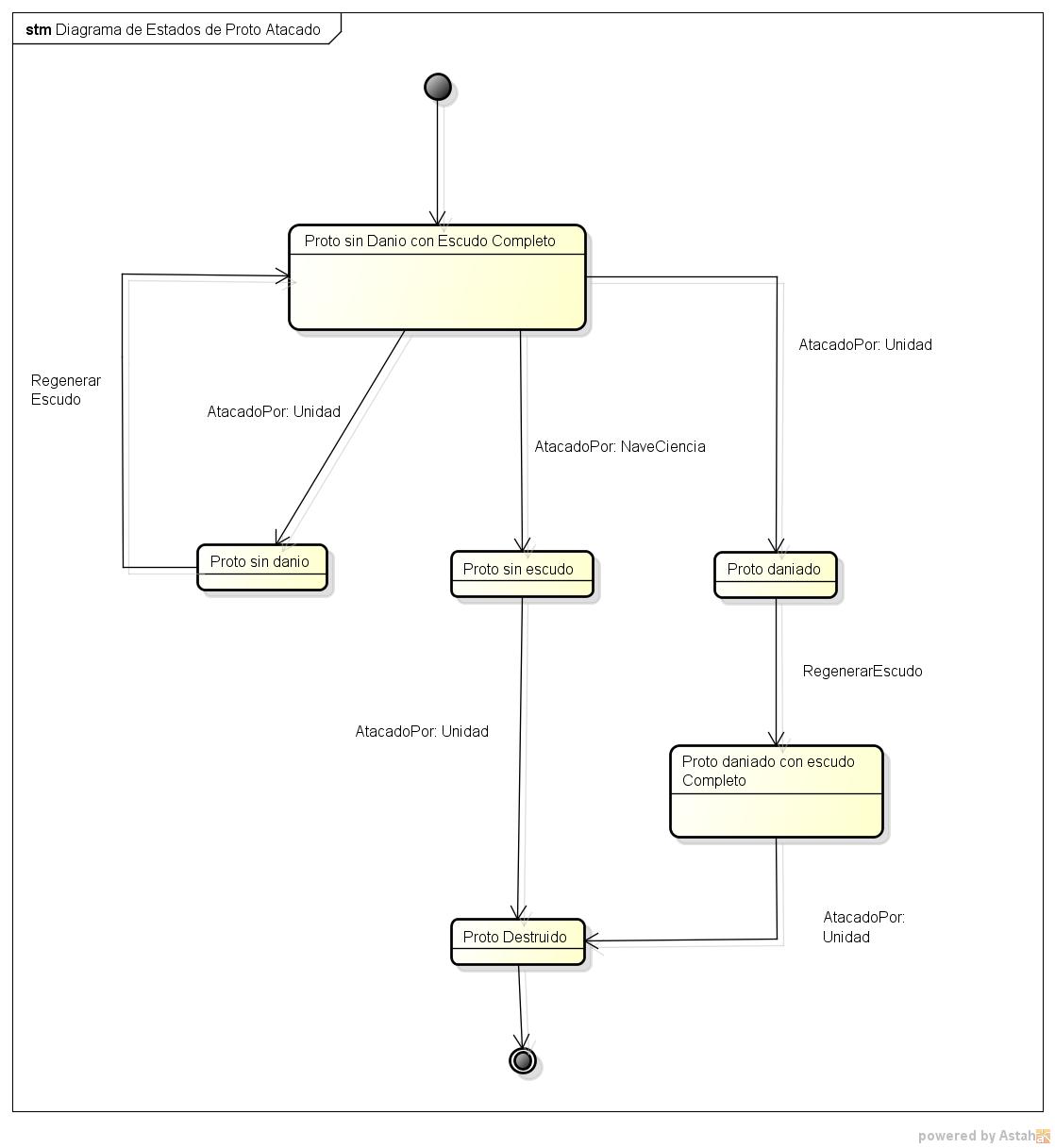




Diagramas de estado

## 



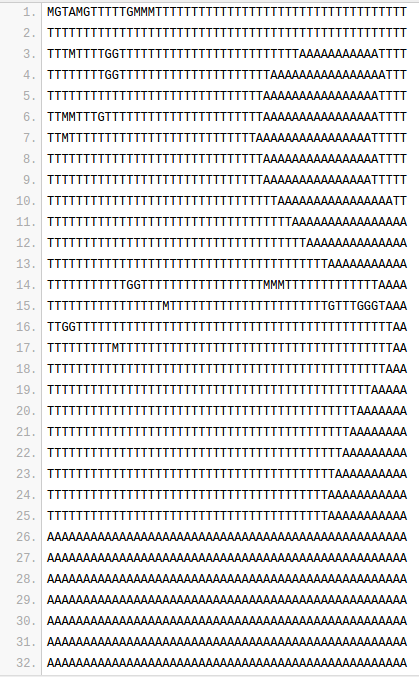


Detalles de implementación

## 

En primer lugar, decidimos que el mapa esté compuesto por **Celdas**, y que cada una tenga la capacidad de contener multiples controlables. Asimismo, cada celda contiene un material **base** y opcionalmente un **recurso**, que ocupan una celda al igual que sus recolectores.

Los **Mapas** los generamos a partir de archivos con la clase **GeneradorMapa**. Esta entidad devuelve un mapa a partir de un archivo .map. Un ejemplo de este archivo es test.map:



(este mapa no contiene bases bien delimitadas, lo usamos solo para probar combates y peleas en las pruebas)

En este archivo, cada caracter simboliza a una celda, definiendo implícitamente su posición, material base y recurso.

Las **Celdas** pueden contener una cantidad infinita de Unidades - siempre y cuando no se interpongan entre sí. Esto se determina usando la **EstrategiaMovimiento** de cada Unidad, donde actualmente tenemos dos: **MovimientoTerrestre** y **MovimientoVolador**. Las estrategias mencionadas deciden cómo moverse a una celda y como recibir daño de un ataque.

Como mencionamos antes, las **Unidades** tienen su estrategia de movimiento, y aparte tienen su **Vida**, que puede ser decorada con **Escudos** que cambian la manera de recibir daño.

Cada Unidad provee **Vision**, una entidad (perteneciente a Jugador) que determina qué celdas puede ver un jugador, que es un factor al ver si una unidad puede atacar a otra.

Las habilidades especiales de los Altos templarios y las Naves de Ciencia las representamos con un objeto **Magia**, que permite almacenar las magias existentes en el juego y aplicar sus efectos.

Luego, ya desde la primera entrega nos surgió el problema de cómo validar y ejecutar **la creación de construcciones**. Hacerlo en el Juego involucraba darle demasiadas responsabilidades a esta, y hacerlo en la construcción misma hacía que cada construcción accediera demasiado al resto de las clases. Como solución a esto, hicimos que las construcciones mismas tuvieran su método *construir*.

Excepciones

## 

# Juego

* **ColorInvalido**: Se realiza cuando dos jugadores intentan elegir el mismo color para desarrollar la partida.  
    
   *Ejemplo: Si el jugador 1 elige color "rojo" y el jugador 2 elige "rojo" también.*
* **FaltanJugadores**: Esta excepción se realiza cuando no se cumplen la cantidad necesaria de jugadores para comenzar una partida  
    
  *Ejemplo: Se registra un jugador, y este quiere comenzar la partida sin haberse registrado un segundo jugador. Análogamente, para el caso de que se quiera comenzar una partida sin haber registrado ningún jugador.*
* **NombreInvalido**: Esta excepción se realiza cuando un jugador se registra con un nombre, y posteriormente el otro jugador quiere registrarse utilizando el mismo nombre.  
    
  *Ejemplo: Si el jugador uno se llama "Terran92", y el jugador dos también se quiere llamar "Terran92".*
* ***BasesInsuficientes****: Se realiza cuando se crea una cantidad de jugadores mayor a la que el mapa puede soportar, entonces las bases serían insuficientes.*
* ***InicioInvalido****: Se ejecuta cuando se lanza alguna de las dos excepciones anteriores.*

*Ejemplo: Cuando el jugador elige un color inválido (el color similar al otro jugador) o un*

*nombre inválido, (un nombre similar al otro jugador) se ejecutara alguna de las dos*

*excepciones mencionadas arriba, y esto dará lugar a la excepción InicioInválido.*

Juego Interfaces

* **CeldaOcupada**: Indica que una celda ya está ocupada por otra entidad (ya sea una construcción o alguna unidad de combate) que puede ser propia o ajena al jugador, entonces no es posible utilizar dicha celda.   
    
  *Ejemplo: Es trivial, si un jugador construye alguna edificación en la posición (1,0), y luego el mismo o otro jugador quiere realizar otra construcción ahí se lanzará dicha excepción.*
* **SobrePoblacion**: Se efectúa cuando se intenta construir una nueva unidad, pero se ha alcanzado el límite de la población.

Ejemplo: Si el limite de poblacion es 200 y el jugador tiene ocupados 199 cupos de

población. Entonces si se quisiera crear un Goliat (ocupa 2 de población) dicha excepción

me lo impediría.

* **ImposibleConstruir**: Se realiza en el caso de que alguna de las dos razas quiera construir una edificación que no le pertenece.  
    
  *Ejemplo: Si el jugador, juega con la raza Terran no debería poder construir un asimilador, o un pilón, etc. Ya que estas construcciones solo le pertenecen a los Protoss.*
* **UbicacionInvalida**: Se ejecuta cuando se quiere construir una edificación que recolecta recursos, en un lugar inapropiado o inválido.  
    
  *Ejemplo: Se crea un mapa, y en la posición (0,0) existe un nodo de minerales, por lo tanto en ese lugar solo se puede crear un centro mineral. Pero si el jugador intenta construir una refinería, esta acción no se realizará pues las refinerías solo se construyen sobre los géiseres de gas vespeno.*
* **RecursosInsuficientes**: Indica que no se puede construir alguna edificación o alguna unidad de combate, debido a que no se cuentan con los recursos suficientes (minerales o gas vespeno) para satisfacer la creación de los mismos.   
    
  *Ejemplo: En el caso de que, el jugador cuente con 50 de mineral y quiera crear un asimilador. Como este cuesta 100 de minerales, no se podrá realizar la acción y se lanzará dicha excepción.*
* **YaFueDestruido**: Indica que una unidad( ya sea construcción o guerrero) ya fue destruida, cuando se la intente atacar.  
    
  *Ejemplo: En el caso de que una entidad (por ej un guerrero zealot) quiera atacar a otro que ya está en estado de destruido, es decir, que este último tenga "vida" o "defensas" cero.*

Mapa

* **CoordenadasFueraDeRango**: Se emplea en el caso de que el jugador utilice coordenadas que se encuentran fuera del rango preestablecido.  
    
  *Ejemplo: Al iniciar el juego se crea un mapa de 100x100. Posteriormente el jugador intenta crear un centro mineral en la posición (-3,4), o (101,5). Ambas posiciones se encuentran fuera de rango.*
* ***RecursosInsuficientes****: Ocurre cuando se quiere construir una unidad (construcción o unidad de ataque) y no se cuentan con los recursos suficientes( ya sea gas o mineral) para*

*hacerlo.*

*Ejemplo: Si el jugador quiere crear un Acceso (cuesta 150 mineral) pero solo cuenta con 140*

*de mineral.*

Recursos

* **RecursoAgotado**: Señala que un recurso (ya sea gas o mineral) en un determinado lugar (sea nodo o volcán) ya no puede recolectarse pues se ha agotado totalmente.  
    
  *Ejemplo: En el caso de que el jugador se encuentra recolectando alguno de los dos recursos (nodos o gas) y este se acabe de manera inesperada de su correspondiente lugar( nodo o volcán). Entonces esta excepción le indicará al jugador que debe dirigirse a otro lugar, para poder seguir obteniendo recursos.*
* **EnergiaInsuficiente**: Se efectúa cuando alguna de las unidades mágicas intenta lanzar alguno de sus ataques, pero no cuenta con la energía suficiente para hacerlo.  
    
  Ejemplo: Si un Alto Templario desea lanzar una alucinación(cuesta 100 de energía)

contando solamente con 95 de energía

.

* **FueraDeRangoDeAtaque**: Se ejecuta cuando una unidad de batalla intenta atacar a otra unidad enemiga, pero esta no se encuentra en su rango de ataque.  
    
  Ejemplo: Si un zealot (tiene rango de ataque 1) intenta atacar a una unidad enemiga que se encuentra a 2 celdas de distancia.
* **UnidadAliada**: Se ejecuta cuando se intenta atacar a una unidad propia.  
    
  Ejemplo: Si un jugador es Terran y el otro es Protoss, entonces el jugador terran le ordena a un marine que ataque a un goliat.
* **YaAtacoEnEsteTurno**: Como un jugador solo puede atacar una vez por turno, entonces si intenta atacar dos veces a una unidad enemiga se ejecutará la excepción.

Ejemplo: Si un marine ataca a un zealot, y acto seguido intenta atacar a un dragón, la

excepción se mostrará

* **YaSeMovioEnEsteTurno**: Se efectua cuando un jugador intenta mover una misma unidad dos veces en un mismo turno.
* **UnidadEnEntrenamiento**: Se ejecuta cuando se quiere dar de alta a una unidad, sin que esta haya terminado su entrenamiento en su respectiva unidad de creación.  
    
  Ejemplo: Si se quisiera dar de alta a un Marine, pero este todavía continuará entrenando en la Barraca, es decir, todavía no tendría forma física.
* **RequierePuertoEstelar**: Se ejecuta cuando se intenta construir un Archivo Templario, sin tener previamente al Puerto Estelar construido.
* **RequiereAcceso**: Se efectúa cuando el jugador intenta construir un Puerto Estelar, pero no cuenta con el acceso construido previamente.

* **RequiereBarraca**: Se realiza cuando se intenta crear una Fábrica, sin haber creado la barraca previamente.
* **RequiereFabrica**: Se ejecuta si un jugador desea construir un Puerto Estelar sin haber construido la Fábrica previamente.