**FIUBA - 75.07**

**Algoritmos y programación III**

*Trabajo práctico 2: AlgoFormers*

1er cuatrimestre, 2016

(trabajo grupal de 4 integrantes)

Alumnos:

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Padrón** | **Mail** |
| Alfaro Miguel | 97743 | miguel.alfaro.a95@gmail.com |
| Mendez Agustina | 98338 | abmendez25@hotmail.com |
| Peirano Guido | 98187 | guidopeirano11@hotmail.com |
| Valdivia Rodrigo | 98213 | rodrivaldivia10@gmail.com |

***Fecha de entrega final***: Lunes 20 de Junio del 2016

***Tutor***: Carlos Fontela

***Comentarios***:

# 

# 

Introducción

# Objetivo del trabajo

Aplicar los conceptos enseñados en la materia a la resolución de un problema, trabajando en forma grupal y utilizando un lenguaje de tipado estático (Java)

# Consigna general

Desarrollar la aplicación completa, incluyendo el modelo de clases e interfaz gráfica. La aplicación deberá ser acompañada por pruebas unitarias e integrales y documentación de diseño. En la siguiente sección se describe la aplicación a desarrollar.

Descripción de la aplicación a desarrollar

Se deberá desarrollar una aplicación que implemente un juego relacionado con el clásico cómic de los Transformers. Todos los algoformers presentan las siguientes características en cualquiera de sus modos:

* Nombre
* Puntos de vida
* Ataque
* Distancia de ataque
* Velocidad de desplazamiento

Todos los algoformers tienen un modo humanoide (que siempre es terrestre) y su modo alterno, que será distinto para cada uno.

### Autobots

1. Optimus Prime, líder de los Autobots. Su modo alterno es un Peterbilt 379 azul con llamas rojas (unidad terrestre)

|  |  |  |
| --- | --- | --- |
| **OPTIMUS** | Humanoide | Alterno |
| Ptos de Vida | 500 | |
| Ataque | 50 | 15 |
| Distancia de ataque | 2 | 4 |
| Velocidad | 2 | 5 |

1. Bumblebee, el joven explorador de los Autobots y guardián de Sam. Su modo alterno es un reluciente Chevrolet Camaro Concept de 2006. (unidad terrestre)

|  |  |  |
| --- | --- | --- |
| **Bumblebee** | Humanoide | Alterno |
| Ptos de Vida | 350 | |
| Ataque | 40 | 20 |
| Distancia de ataque | 1 | 3 |
| Velocidad | 2 | 5 |

1. Ratchet, Su modo alterno es un F22 raptor (unidad aérea)

|  |  |  |
| --- | --- | --- |
| **Ratchet** | Humanoide | Alterno |
| Ptos de Vida | 150 | |
| Ataque | 5 | 35 |
| Distancia de ataque | 5 | 2 |
| Velocidad | 1 | 8 |

1. Los 3 combinados forman un Superion (unidad terrestre), la transformación dura 2 turnos propios hasta completarse. Queda a criterio del grupo definir la distancia mínima a la que tienen que estar los algoformers entre sí para formar un Superion.

|  |  |
| --- | --- |
| **SUPERION** | modo único |
| Ptos de Vida | ⅀ Ptos de vida de los algoformers que lo forman |
| Ataque | 100 |
| Distancia de ataque | 2 |
| Velocidad | 3 |

### 

### 

### Decepticons

1. Megatron, líder de los Decepticons. Su modo alterno es un jet cibertroniano. (unidad aérea)

|  |  |  |
| --- | --- | --- |
| **MEGATRON** | Humanoide | Alterno |
| Ptos de Vida | 550 | |
| Ataque | 10 | 55 |
| Distancia de ataque | 3 | 2 |
| Velocidad | 1 | 8 |

1. Bonecrusher, el desbocado buscaminas de los Decepticons. Su modo alterno es un vehículo blindado Force Protection Industries Buffalo HMPCV buscaminas. (unidad terrestre)

|  |  |  |
| --- | --- | --- |
| **Bonecrusher** | Humanoide | Alterno |
| Ptos de Vida | 200 | |
| Ataque | 30 | 30 |
| Distancia de ataque | 3 | 3 |
| Velocidad | 1 | 8 |

1. Frenzy, un pirata informático de los Decepticons. Su modo alterno es una Renault Duster. (unidad terrestre)

|  |  |  |
| --- | --- | --- |
| **Frenzy** | Humanoide | Alterno |
| Ptos de Vida | 400 | |
| Ataque | 10 | 25 |
| Distancia de ataque | 5 | 2 |
| Velocidad | 2 | 6 |

1. Los 3 combinados forman un Menasor (unidad terrestre), la transformación dura 2 turnos hasta completarse. Queda a criterio del grupo definir la distancia mínima a la que tienen que estar los algoformers entre sí para formar un Menasor.

|  |  |
| --- | --- |
| **MENASOR** | modo único |
| Ptos de Vida | ⅀ Ptos de vida de los algoformers que lo forman |
| Ataque | 115 |
| Distancia de ataque | 2 |
| Velocidad | 2 |

### Ataques y distancia de ataques

Los algoformers de un mismo equipo **no** pueden atacarse entre ellos.

|  |  |  |  |
| --- | --- | --- | --- |
| ***SIMPLIFICACIÓN***: Los algoformers pueden atacar en cualquiera de las formas, es todo igual: | | | |
| tierra-tierra | tierra-aire | aire-tierra | aire-aire |

La distancia de ataque se mide en casilleros. Por ejemplo OPTIMUS en modo humanoide posee distancia de ataque = 2 significa que podrá atacar a cualquier otro algoformer que se encuentre en un casillero verde, no así en los celestes.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  | **OPTIMUS** |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

### Superficies:

Tierra

1. **Rocosa**: Todas las unidades pueden atravesarla.
2. **Pantano**: En modo humanoide no es posible atravesarlo. En modo alterno las unidades terrestres tardan el doble que una superficie rocosa.
3. **Espinas**: Causa un 5% de daño a quien la atraviese, independientemente del modo en que se halle.

Aire

1. **Nube**: Todas las unidades aéreas pueden atravesarla.
2. **Nebulosa de andrómeda**: Las unidades aéreas quedan atrapadas 3 turnos.
3. **Tormenta psiónica**: Al pasar por una tormenta psiónica las unidades aéreas pierden poder de ataque (sólo del modo alterno) y el mismo queda disminuido en un 40 % **para siempre**. (En el contexto de una partida). El daño no es acumulable en caso de que el algoformer ya afectado vuelva a pasar por una tormenta, no lo afecta.

### Bonus[[1]](#footnote-1)

Una vez que el algoformer (en cualquier modo) captura un bonus el mismo es consumido por el algoformer y desaparece del mapa.

1. **Doble** **Cañón**: El algoformer que se tope con este bonus, duplica su capacidad de ataque durante 3 turnos propios.
2. **Burbuja inmaculada**: El algoformer que se tope con este bonus, no recibe ningún daño por ningún tipo de ataque de otro algoformer durante 2 turnos propios.
3. **Flash**: El algoformer que se tope con este bonus triplica su velocidad de desplazamiento durante 3 turnos propios.

#### Ejemplo de Doble cañón:

*Turno Autobots*:

**Optimus Humanoide captura doble cañón**

*Turno Decepticons*

**….**

*Turno Autobots:*

**Optimus Humanoide ataque = 100 ( 2 x 50 )**

*Turno Decepticons*

….

*Turno Autobots:*

**Optimus Humanoide ataque = 100 ( 2 x 50 )**

*Turno Decepticons:*

….

*Turno Autobots:*

**Optimus Humanoide ataque = 100 ( 2 x 50 )**

*Turno Decepticons:*

….

*Turno Autobots:*

**Optimus Humanoide ataque = 50**

Queda a criterio de cada grupo definir qué pasa cuando se combinan los 3 algoformers para formar un Menasor o Superion y alguno de ellos tiene un bonus activo.

### Jugabilidad

Hay 2 jugadores, cada uno debe elegir un equipo antes de iniciar una partida. Cada jugador comienza la partida con sus 3 algoformers.

Es un juego por turnos. En cada turno el jugador debe elegir UN algoformer y solicitarle que realice una actividad (moverse, transformarse, atacar, combinarse, capturar chispa, etc…). Luego pasará el turno al contrincante y así sucesivamente hasta la captura de la chispa suprema.

El juego elige al azar qué jugador comienza. Cada jugador inicia en el extremo opuesto al otro con sus 3 algoformers juntos.

#### Tablero

El juego tiene lugar en un tablero compuesto de casilleros. El tamaño, forma y cantidad de casilleros del tablero queda a definir por cada grupo y acordado con su ayudante.

Todos los algoformers ocupan 1 casillero en cualquiera de sus modos. No puede haber más de 1 algoformer en un casillero. Hay 1 o 0, nunca 2, 3, etc.

Los algoformers se desplazan por el tablero de casillero en casillero. Cada punto de su velocidad de desplazamiento representa 1 casillero. Por ejemplo, OPTIMUS en modo alterno posee una velocidad de desplazamiento = 5 lo que nos lleva a la siguiente configuración:

*Estado inicial:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **OPTIMUS** |  |  |  |  |  |
|  |  |  |  |  |  |

**Optimus se mueve ⇒**

*Estado final:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  | **OPTIMUS** |
|  |  |  |  |  |  |

### Otro ejemplo

Megatron en modo humanoide posee una velocidad de desplazamiento = 1. Con lo cual desde dónde está ubicado actualmente se puede mover a cualquiera de los casilleros verdes. Como puede verse, aplica la misma lógica que en distancia de ataque.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  | **MEGATRON** |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Fin del juego

Para ganar se debe capturar la chispa suprema. que será ubicada de forma aleatoria cerca del centro del tablero. Cualquier algoformer en estado Humanoide es capaz de capturar la chispa suprema, no así en su modo alterno. Los superion y menasor también pueden atrapar la chispa suprema. Si un jugador logra destruir a todos los algoformers del jugador contrario, también gana en ese caso.

[OPCIONAL] Fin del juego alternativo (suma puntaje extra)

Para ganar el juego el algoformer que captura la chispa suprema debe transportarla hasta el monte de la perdición donde es arrojada. ¡ Pero cuidado ! Si en el camino el algoformer que la transporta es destruido, la chispa suprema puede ser recapturada por el jugador contrario.

## Interfaz gráfica

Se debe desarrollar una interfaz visual para la interacción entre los jugadores. En la misma se pondrá mucho énfasis y se evaluará como parte de la consigna la **USABILIDAD** de la misma.

Cada vez que le toque el turno a cada jugador la vista del mapa debe centrarse en el lugar donde utilizó a su último algoformer.

Entregables

1. Código fuente de la aplicación completa, incluyendo también: código de la pruebas, archivos de recursos
2. Script para compilación y ejecución (ant)
3. Informe, acorde a lo especificado en este documento

# Formas de entrega

Habrá **4 entregas formales.** Las mismas tendrán una calificación de **APROBADO o NO APROBADO** en el momento de la entrega.

Aquél grupo que acumule 3 no aprobados, quedará automáticamente desaprobado con la consiguiente pérdida de regularidad en la materia. En cada entrega se debe traer el informe actualizado.

# Evaluación

El día del vencimiento de cada entrega, cada ayudante convocará a los integrantes de su grupo, solicitará el informe correspondiente e iniciará la corrección mediante una entrevista grupal.

**Es imprescindible la presencia de todos los integrantes del grupo el día de cada corrección.**

Se evaluará el trabajo grupal y a cada integrante en forma individual. El objetivo de esto es comprender la dinámica de trabajo del equipo y los roles que ha desempeñado cada integrante del grupo. Para que el alumno apruebe el trabajo práctico debe estar aprobado en los dos aspectos: grupal e individual.

Dentro de los ítems a chequear el ayudante evaluará aspectos formales (como ser la forma de presentación del informe), aspectos funcionales: que se resuelva el problema planteado y aspectos operativos: que el TP funcione integrado.

# Casos de prueba para cada entrega

1er Entrega Lunes 30/05/2016 - *1 semana y 4 dias desde fecha inicio*

Turnos, Jugadores, Unidades, Tablero, casillero

1. Se ubica un algoformer humanoide en un casillero, se pide que se mueva, se verifica nueva posición acorde a su modo.
2. Se ubica un algoformer humanoide se lo transforma, se verifica que se pueda transformar en ambas direcciones.
3. Se ubica un algoformer en su modo alterno y se pide que se mueva y se verifica que su nueva posición sea acorde.
4. Crear una prueba de integración en la cual se pueda crear un juego, con 2 jugadores cada uno de ellos con sus 3 algoformers distribuidos en el tablero según el enunciado y la chispa suprema por el centro del tablero.
5. Combinaciones en modos de: Ubicar un autobot, ubicar un decepticon, pedir que se ataquen respetando ( y no ) las distancias verificando los daños ( o no daños ).

2da Entrega Lunes 6/06/2016 **-** *2 semanas y 4 dias desde fecha inicio*

1ra entrega + Interacción con superficies

1. Llenar una zona rocosa, verificar que todos los algoformers en todos sus modos la atraviesen sin problemas
2. Llenar una zona pantano, verificar que en modo humanoide no se pueda atravesar.
3. Llenar una zona pantano, verificar que en modo alterno las unidades terrestres tardan el doble que rocoso
4. Llenar una zona pantano, verificar que las unidades aéreas las atraviesan sin problemas
5. Llenar una zona de espinas verificar que todas las unidades terrestres pierden un 5% de sus vida por cada casillero de estos que atraviesen
6. Llenar una zona de espinas, verificar que unidades aéreas no tienen problemas al atravesarlas.
7. Llenar una zona con nubes, verificar que las unidades aéreas las atraviesan sin problemas
8. Llenar una zona de nebulosa de andrómeda, pasar una unidad aérea, corroborar que quede 3 turnos atrapada, sin moverse
9. Llenar una zona de tormenta psiónica, pasar un algoformer alterno aéreo, ver que baje su capacidad de ataque
10. test 9 + volver a pasar y ver que no bajó su capacidad de ataque.

3er Entrega: Jueves 16/06/2016 - Lunes 20/06/2016 - *3 semanas y 4 dias desde fecha inicio*

2da entrega + Bonus, Interfaz gráfica inicial

1. Ubico un algoformer, ubico un bonus doble cañón, ubico otro algoformer enemigo, el algoformer captura el bonus y ataca al enemigo verificando que causa el doble de daño durante 10 turnos.
   1. Repetir para el modo alterno.
2. Ubico un algoformer, ubico un bonus burbuja, ubico un otro algoformer enemigo, el algoformer captura el bonus, el otro algoformer ataca al primer algoformer, este no recibe daños, repetir hasta 2 turnos propios, continuar y verificar que en el 3ro sí reciba daño.
   1. Realizar el mismo test en modo alterno
3. Ubico un algoformer, ubico un bonus flash, verifico que se mueve 3 veces más rápido durante 3 turnos propios.
   1. Repetir en modo alterno
   2. Repetir en modo humanoide-alterno-humanoide
4. Test boundary cases (Si ya tiene un bonus de un tipo que no pueda agarrar otro del mismo tipo, Atrapar 2 bonus distintos verifcar ambos comportamientos, etc…)

4ta y última Entrega: Lunes 20/06/2016 - *4 semanas y 4 dias desde fecha inicio*

Trabajo Práctico completo funcionando, con interfaz gráfica final, sonidos e informe completo.

|  |
| --- |
| **Tiempo total de desarrollo del trabajo práctico**: 4 semanas y 4 días |

Informe

**Supuestos:** consideraciones tomadas, en cuanto al funcionamiento del Juego, que no eran aclaradas por el enunciado.

Supuestos de los Algoformers:

* Para que los tres AlgoFormers puedan combinarse debe cumplirse que la distancia de los del costado, con el que queda en el medio, sea máxima de un casillero.
* Cuando se combinan los Algoformers, el Algoformer combinado (Superion o Menasor) ocupa la posición en donde se encontraba el líder del equipo (Optimus o Megatron).
* Cuando se descombinan los Algoformers, el líder (Optimus o Megatron) ocupa la posición que ocupaba el combinado (Superion o Menasor), los otros dos Algoformers se ubican en la posición mas próxima disponible.
* Si cuando se combinan los Algoformers, un Algoformer tiene un bonus activo, este queda activo pero su efecto no influye en el Algoformer combinado. Solo cuando se descombine se vera afectado por el bonus, los turnos del bonus se consumen aun si este AlgoFormer esta combinado.

Supuestos del Tablero:

* El tamaño del tablero es: 20 casilleros hexagonales de ancho por 30 de alto.

Supuestos de los Bonus:

-Los Bonus solo pueden ser recolectados por un AlgoFormer cuando este se para encima de él, es decir, no se puede agarrar si solamente cruza por encima del Bonus.

Supuestos del Juego:

* Al iniciarse el juego por default Optimus y Megatron estan seleccionados en ambos equipos como algoformers a realizar las acciones
* Una vez combinados los algoformers, si se vuelve a tratar de combinar el jugador perdera su turno

## **Modelo de dominio**

*Introducción*

El desafío propuesto es diseñar un juego por turnos inspirado en el conocido comic Transformers, como se explica anteriormente, el juego consiste básicamente en dos equipos con 3 AlgoFormers, cada uno los cuales lucharán a muerte, atravesando por el tablero diferentes dificultades u obteniendo ayudas, hasta alcanzar la chispa suprema o matar a todos los AlgoFormers enemigos, y así derrotar al otro equipo.

*En que consiste el diseño y como se pensó.*

Esta sección pretende dar un panorama general de como se diseñó la estructura del juego y las entidades que tienen lugar, sus responsabilidades y como se relacionan. Para luego explicar en profundidad cada una de ellas.

La pelea entre los dos equipos de AlgoFormers tiene lugar sobre un tablero donde estos se pueden mover por tierra o por aire dependiendo de su modo.

Por lo que se debe pensar en una entidad que contenga todos los elementos que hacen al juego, además permita recibir y mandar distintos mensajes a estos objetos.

Tanto en la tierra como en el aire se encuentran diferentes superficies que causan diversos efectos sobre el AlgoFormer que la atraviesa o directamente no permite que la atraviese bloqueándolo.

Esto nos da una idea de que los AlgoFormers tienen que tener una manera de reaccionar a estas superficies cuando se encuentren sobre ellas.

También existen los Bonus que se encuentran sobre el tablero y que pueden ser recolectados por los AlgoFormers para expandir sus poderes, también debe ser recolectada por ellos, la chispa suprema.

Por lo que el AlgoFormer tiene que permitir que se le modifiquen sus características en tiempo de ejecución debido que agarra un Bonus, y poder recolectar la chispa suprema que llevara su equipo a la victoria.

Esto nos da un indicio de los dos tipos de elementos que vamos a tener en el tablero, los cuales van a ser Accionables o Recolectables.

Por ultimo tener una entidad responsable de coordinar todas las llamadas a los diferentes objetos que componen el Juego, tener a los jugadores con sus equipos y saber cuando se terminó el juego y quién ganó.

AlgoFormer

Empezando por cada AlgoFormer, estos se diferencian por su vida máxima, el ataque que realizan, la distancia máxima a la cual pueden atacar y la velocidad con la que se mueven.

Además de que cada uno tiene dos modos posibles ya sea Humanoide o Alterno.

Para poder diferenciar estos personajes primero se crea la entidad “AlgoFormer” abstracta ya que nunca vamos a tener un AlgoFormer propiamente dicho en el tablero e implementa las interfaz Accionable (que se explicará luego).

Para después poder tener referencia a que equipo pertenece cada AlgoFormer, heredan de AlgoFormer clases que representan un “Autobot” o bien “Decepticon”.

Y finalmente se crean los diferentes personajes de cada equipo, heredando de Autobot: Optimus, Ratchet y Bumblebee. Y de Decepticon: Megatron, Frenzy y Bonecrusher.

Lo anterior se diseño de esta manera para poder tener referencia a la clase y no tener que andar preguntando ni validando que AlgoFormer representa ni de que equipo es, así podemos aplicar métodos polimórficos que usan para comunicarse entre ellos, así también para que instanciar un AlgoFormer sea mas fácil y directo.

Además si en un futuro se quiere agregar, borrar o cambiar alguna característica de estos es mas sencillo teniendo las clases por separado.

Modos

Las diferentes características antes nombradas varían según el modo en el que se encuentra el AlgoFormer ya sea Humanoide o Alterno, es por eso que se tuvo en cuenta en el diseño una entidad mas que es su modo en donde estarán todas estas características, excepto la vida ya que es indiferente del modo, el modo es también un atributo de los AlgoFormers.

Se creo la interfaz “Modo” donde se declaran todos los métodos que deben tener.

Existen dos modos (se diferencian si los AlgoFormers se mueven por tierra o por aire), Humanoide (es siempre terrestre) y Alterno (puede ser terrestre o aéreo).

Es por esto que las clases ‘ModoTerrestre’ y ‘ModoAereo’ implementan la interfaz modo.

De ‘ModoTerrestre’ hereda ‘ModoHumanoide’ (Ya que es siempre terrestre) y ‘ModoTerrestreAlterno’ que representa el modo de los AlgoFormers que cuando están en modo alterno son terrestres.

De ‘ModoAereo’ hereda ‘ModoAereoAlterno’, luego simplemente cada modo de cada AlgoFormer hereda de la clase que corresponda según la característica del AlgoFormer.

***Por ejemplo***: el modo humanoide de Optimus hereda de ‘ModoHumanoide’ (como todos) y el modo alterno de Optimus hereda de ‘ModoTerrestreAlterno’.

En cambio Megatron que también su modo humanoide hereda de ‘ModoHumanoide’ pero su modo alterno hereda de ‘ModoAereoAlterno’.

En cada subclase de los modos se definen las diferentes caracterisitcas del AlgoFormer, también es el modo el que recibe el mensaje de cambiar el modo y el sabe a que modo tiene que cambiar.

En el caso de la instanciación de los modos se aplico el patrón de diseño llamado ‘Singleton’ ya que necesitamos que solo exista una instancia de cada modo, para que cuando se cambie de modo se mantenga el estado del otro modo ya que como es singleton, la instancia es la misma.

*Métodos y Atributos del Algoformer.*

*Cada AlgoFormer tiene como atributo lo siguiente:*

***Vida:*** es un atributo del AlgoFormer que no depende del modo, sino de que AlgoFormer se trata.

***Modo:*** Representa el modo en el que se encuentra y contiene todas las otras caracterisitcas del AlgoFormer, es así como parece que el AlgoFormer cambia de clase en tiempo de ejecución de la aplicación, este patrón de diseño se llama “Strategy”.

Muchas de las responsabilidades a la hora de interactuar entre dos AlgoFormers se delegan a su modo ya que este sabe como manejar y modificar su estado.

Además, como existen las superficies y los bonus que modifican el comportamiento del AlgoFormer (velocidad, ataque, recibir ataque, etc.) cada AlgoFormer tiene sus diferentes ‘multiplicadores’ que representan cual porcentaje pueden usar de sus habilidades.

***Agilidad:*** modifica la velocidad con la que se puede mover.

Velocidad del AlgoFormer = Agilidad \* Velocidad del Modo

***Potencia:*** modifica el ataque con el que puede atacar

Ataque del AlgoFormer = Potencia \* Ataque según el Modo

***Escudo:*** modifica el daño recibido por otro AlgoFormer

Daño neto Recibido = Escudo \* Daño recibido

También cada AlgoFormer tiene una lista de afectadores que son los que modifican estos valores y esto nos lleva a explicar una nueva entidad en el juego

*Afectadores*

Debido a que en el juego existen diferentes elementos que cambian el comportamiento de los AlgoFormers como son las superficies y los bonus.

Estos efectos se pueden producir por ejemplo cuando un AlgoFormer atraviesa una superficie de Espinas (efecto por posición), una Nebulosa de Andrómeda (efecto por turnos) o bien una Tormenta Psiónica (efecto permanente) o cuando agarra algún bonus.

Los afectadores son los responsables de manejar estos efectos y producírselos a los AlgoFormers modificando su vida, agilidad, potencia o escudo.

Es por esto que existe la clase ‘afectador’ y heredan de esta los tres tipos: por turnos, por posición, y permanente.

Estos tienen como atributo su efecto, pueden afectar y desafectar a un AlgoFormer, también se le avisa cuando pasa un turno para que los afectadores por turnos lleven cuenta de cuantos turnos restantes le quedan para afectar.

*Métodos y atributos de los modos*

Los atributos que tiene un modo como ya se explico anteriormente son el ataque, la distancia de ataque y la velocidad de un AlgoFormer.

En cuanto a las responsabilidades delegadas son, atacar, moverse, cambiar de modo, reaccionar a las superficies por la cual pasa el AlgoFormer y ser desafectado.

*Generador de caminos*

Una parte importante para que los AlgoFormers puedan moverse es encontrar el mejor camino para llegar de su coordenada actual a la deseada. De esto se encarga una clase llamada GeneradorDeCaminos que genera el mejor caminos dadas las condiciones del tablero, de las cuales se detallará más adelante. Teniendo este camino a transitar por el AlgoFormer resulta fácil saber si con su paso actual podrá recorrerlo en un turno o no.

Para la implementación de la clase se utilizó el algoritmo de Dijkstra, que se utiliza para encontrar el camino más corto entre dos puntos.

*Tablero*

Los Algoformers, las superficies y los bonus se encuentran sobre un tablero, estos elementos se pueden referenciar a través de coordenadas.

Es por esto que se crea la clase tablero que tiene como atributos el alto y el ancho del tablero, también (aunque ya es mas un detalle de implementación) 3 Hash maps para tener referencias de Accionables, Superficies y Recolectables.

Por cada coordenada en el tablero, es decir, por cada posición tenemos dos superficies, una de tierra y otra de aire. Por lo que necesitamos una entidad que contenga estas dos superficies para luego poder saber que superficies se encuentran en una determinada coordenada.

Esta responsabilidad es de la clase ‘Casillero’ que además de contener las superficies se encarga de mandarle el mensaje a ellas para que afecten a los AlgoFormers que se encuentren en ese casillero.

Uno de las ideas mas notables en cuanto al diseño del tablero es la forma hexagonal de cada casillero, es por este cambio en diseño que se deben tener diferentes coordenadas a que si tuviéramos un tablero rectangular normal, es por esto que existe una clase coordenada con todas estas características.

Cuando se instancia un Tablero, se llama a la entidad ‘GeneradorDeCoordenadas’ este lo que hace es inicializar un tablero vacío, esto es:

* Las superficies de Tierra son todas Rocosas
* Las superficies de Aire son todas Nubes
* En donde se encontrarían los accionables se coloca Vacío
* En donde se encuentran los bonus se coloca un “Bonus Vacío”

Los mensajes que puede recibir el tablero, son todos aquellos que tienen relación con colocar o sacar algo de este, como un AlgoFormer, superficies, bonus. Y poder obtener alguna coordenada de donde se encuentra alguno de estos elementos.

También puede mover algún Accionable a otra posición, o coordinar un ataque entre dos Accionables validando que estén a una distancia valida.

En el caso particular, cuando los tres AlgoFormers se combinan o descombinan, como es el Tablero el que sabe la posición de estos, es este el encargado de saber si se encuentran a distancia correcta para combinarse, o colocar los individuales cuando se descombinan.

*Superficies*

Aunque ya se hablo anteriormente sobre las superficies, aquí se pretende profundizar y dar una mejor idea de como estas fueron pensadas para cumplir su función.

Las superficies se dividen en dos grupos, de **tierra** o de **aire**.

Y dentro de estos grupos se tiene a las diferentes superficies del juego.

Cada una de ellas tiene su ‘Afectador’ que este es el que va a modificar el comportamiento del AlgoFormer como se explico con anterioridad.

El ‘Afectador’ tiene el Efecto que provoca la superficie, estos efectos son entidades que sencillamente modifican los atributos del AlgoFormer cuando este se aplica.

Como el efecto que tiene una superficie sobre un AlgoFormer depende del Modo de este ultimo, en el Modo se definen métodos polimórficos donde dependiendo de que superficie reciba el AlgoFormer será afectado de diferentes maneras o directamente no va a ser afectado.

Cuando al Modo se le pide que sea afectado por una superficie se produce un “double dispatch”, ya que luego se le dice a la superficie que afecte al modo y es este ultimo el que sabe como ‘ser afectado’ por las diferentes superficies.

De una forma similar a lo anterior funciona cuando se quiere desafectar a un AlgoFormer, esto es porque algunas superficies afectan por una cantidad de turnos solamente y luego revierten su efecto.

*Bonus*

Dado que los bonus tienen un comportamiento muy parecido a las superficies, en pocas palabras son entidades que producen algún efecto en el ‘Accionable’ (AlgoFormer), la lógica con la que funcionan los bonus es muy parecida a las superficies.

Se tiene la clase ‘Bonus’ que implementa la interfaz ‘Recolectable’, que tiene un Afectador (el cual tiene dentro el efecto del bonus) y los métodos son simplemente dos: afectar y desafectar a un AlgoFormer.

En el caso de los bonus cuando afecta a un AlgoFormer lo afecta por igual, independientemente del modo en el que se encuentre.

Es por esto que no se tiene en cuenta el Modo del AlgoFormer. Cuando un AlgoFormer se encuentra en la misma coordenada de un Bonus, este es recolectado y el Bonus le dice a su Afectador que afecte al AlgoFormer que lo acaba de recolectar, validándose luego que este ultimo no tenga un bonus del mismo tipo.

*Chispa Suprema*

Este es el objeto por el cual los AlgoFormers pelearan a muerte, como tiene que ser recolectado por ellos, esta clase implementa la interfaz ‘Recolectable’.

Y el “efecto” que produce sobre el AlgoFormer, es justamente el de capturar la chispa y notificar que el equipo al cual pertenece ganó el juego.

*Jugador y Equipo*

Todas las entidades antes explicadas deben ser controladas e instanciadas por otra. En este caso es un juego de dos jugadores, donde cada uno tiene su equipo.

***Equipo***

Un equipo esta compuesto por sus 3 AlgoFormers y una referencia al AlgoFormer seleccionado, ya que todos los mensajes que se le manden al equipo serán aplicados sobre el seleccionado.

Por supuesto se permite que cambie de AlgoFormer, y los mensajes de atacar, mover, combinar y descombinar pasan por el equipo.

Además lleva cuenta de los AlgoFormers enemigos que va matando, esto se diseño de esta manera para saber que si un equipo mato a todos los AlgoFormers del equipo contrario este equipo es el ganador. Se actualiza cuando el equipo mata a un AlgoFormer del otro equipo.

***Jugador***

Volviendo al Jugador, este tiene un Equipo y una referencia al tablero en el cual se disputa la batalla, esta referencia es necesaria ya que los métodos necesitan el tablero para poder ser llamados.

El jugador le dice a su equipo que coloque los AlgoFormers en el tablero y este le dice a la entidad ‘Ubicador de personajes’ (esa es su responsabilidad), cuando el juego se inicia.

*Juego*

Es la entidad que integra todo el juego y se encarga de reunir las partes anteriores para hacer al funcionamiento del juego.

Es a esta clase la única con la que se comunica el usuario desde afuera, esta clase conoce como funcionan las otras y que función tiene cada una.

Se encarga de instanciar el Tablero, crear los jugadores con sus equipos, ubicar los Accionables y Recolectables en el Tablero

Elije el primer jugador de manera aleatoria, y se encarga de avisarle a los objetos correspondientes de como van pasando los turnos.

Todos los métodos principales del juego, que el usuario puede llamar, como: mover, atacar, cambiar de modo, combinar, descombinar, etc. Son delegados a los otros objetos por esta clase.

*Fin del Juego*

El fin del Juego se puede dar en dos situaciones:

***Cuando un AlgoFormer recolecta la chispa suprema***

La chispa suprema es un recolectable que se encuentra en el tablero y puede ser juntado por un AlgoFormer.

Cuando un AlgoFormer se coloca en una coordenada en el tablero, este obtiene el recolectable que se encuentra en esa posición y es recolectado por el AlgoFormer (esto funciona así también para las bonus), cuando se trata de la chispa, esta produce su efecto sobre el AlgoFormer y este ‘efecto’ es el de capturar la chispa.

Una vez que esto ocurre se le notifica a los observadores (que se explican luego) que un AlgoFormer ya tiene la chispa por lo tanto el equipo al cual pertenece ganó el juego

***Cuando un equipo mata a todos los AlgoFormers del otro equipo***

Cuando un AlgoFormer de un equipo ataca a otro del equipo contrario y se le acaba la vida (muere), el equipo lo cuenta como AlgoFormer vencido.

Una vez que esta cuenta alcanza la cantidad de AlgoFormers que tenia el equipo contrario originalmente, se le notifica al Juego de que este equipo venció al otro, por lo tanto ganó el juego.

*Observadores*

Anteriormente se nombraron a Observadores, esto es que en diferentes situaciones del juego se utiliza el patrón Observador-Observado para saber si ocurrió algún cambio de estado en algún objeto.

Para eso se crea una interfaz Observador, y existe el sujeto el cual va a ser observado. El observador se subscribe en el sujeto y es este el que notifica al observador que algo cambió.

*Esto también se usa para la vista, en la interfaz grafica. Pero en este caso la funcionalidades son otras:*

**Saber cuando se muere un AlgoFormer:**

Un AlgoFormer solo puede morir cuando este es atacado por otro, es por eso que el Sujeto observado es el AlgoFormer que recibe el ataque, y el observador es el Equipo del AlgoFormer atacante. Si el AlgoFormer una vez que recibe el daño su vida es menor o igual a 0, significa que murió y notifica a sus observadores.

**Saber cuando gana un equipo:**

* Porque mató a todos los Algoformers del otro equipo:

El equipo lleva cuenta de cuantos AlgoFormers del equipo contrario va matando, es por esto que además de ser observador es un Observado, y el observador es el Juego, que le interesa saber cuando termino el Juego.

Cuando el equipo mata a todos los AlgoFormers del equipo contrario significa que ganó el juego y ahí notifica al Observador, en este caso el Juego, de que el juego debe terminar y el ganador es ese Equipo.

* Porque algún AlgoFormer capturó la chispa suprema:

Cuando la chispa suprema es capturada el juego termina inmediatamente y el ganador es el equipo al cual el AlgoFormer que la capturo pertenece.

Es por esto que el Observable es la chispa suprema y es Observada por el Juego, cuando esta es capturada se le notifica al Juego que el equipo que esta jugando en ese turno ganó el juego.

*Vista*

La vista consiste en una vista principal, en donde se eligen los nombres y equipos de los jugadores y la vista del juego propiamente dicho.

En esta vista se destacan 4 secciones: En los costados se pueden ver barras con los personajes que sirven para seleccionarlos desde cualquier lugar del mapa, y se puede ver información como la vida, ataque o velocidad actual del AlgoFormer.

En la parte de abajo del programa se muestran todas las opciones que tiene disponible un jugador, tales como moverse o transformarse. Para agregar usabilidad al juego, se crearon “shortcuts” para que en lugar de hacer click al botón, el jugador pueda apretar la tecla con la letra con la que empieza el botón, a excepción de pasar turno que se hace con enter, para lograr esa funcionalidad.

En el centro de la pantalla se ubica el juego en sí mismo. Aquí es donde se ven a los AlgoFormers en el tablero, y se interactúa con ellos. Desde esta ventana también se puede seleccionar a los personajes, haciendo click sobre ellos.

Una vez que el jugador decidió que AlgoFormer realizará una acción, la interfaz gráfica mostrará cuales son las opciones disponibles: Cuando se quiera atacar se mostrará cuales son los casilleros a los que se llega a atacar, y cuando se quiera mover se pintará un camino hasta la coordenada sobre la cual se encuentre el mouse, aclarando si el jugador puede moverse hasta ahí, utilizando verde para los caminos posibles y gris para los que no se pueden.

Por último, cuando un jugador gana el juego, se abre una ventana informando el jugador y se pregunta si se desea seguir jugando, para intentar ganar el juego con el final que no haya sucedido (si se recolecto a la chispa se puede intentar matar a todo el equipo contrario y viceversa), o si se desea cerrar la aplicación.

*Controlador*

Para completar el patrón de MVC el controlador se ocupa de interpretar la entrada del usuario sobre la interfaz gráfica y comunicársela al modelo.

La parte más importante del controlador es la clase controlador en sí misma. A esta es a quien le llegan todas las interacciones del usuario (a excepción de los shortcuts que tienen una clase aparte). Esta clase diferencia que es lo que el jugador pretende hacer con distintos modos en los que se encuentra, como ModoSeleccionar y ModoMover.

Para cambiar entre los distintos modos se utiliza el patrón de diseño State, que logrará que el controlador pueda hacer distintas acciones y decidir cual usar en runtime.

Por defecto se utiliza al ModoSeleccionar y el usuario podrá cambiarlo cuando se decida por un personaje. Cada Modo tiene un método que responde a el click de una nueva coordenada por el usuario, y cuando el mouse pasa por encima de alguna, que se utiliza para generar caminos.

*Comunicación MVC*

Para hacer las comunicaciones entre la vista y el controlador, se le asigna a este último la responsabilidad de manejar los eventos que ocurren en la vista, aunque es necesario notar que el controlador no puede interactuar con la vista sino que solo lo puede hacer con el modelo, ya que contiene una referencia al juego para actualizarlo.

Para hacer la comunicación entre la vista y el modelo se utilizó el patrón de diseño Observer pero modificado ya que el modelo perdía varias referencias que no son importantes para el mismo pero si para la interfaz, por lo que estas se pasan como parámetro al actualizar alguna vista cuando sea necesario.

De esta forma logramos que la vista se pueda actualizar de una forma sencilla y que el modelo no dependa de las vistas que se le agreguen.

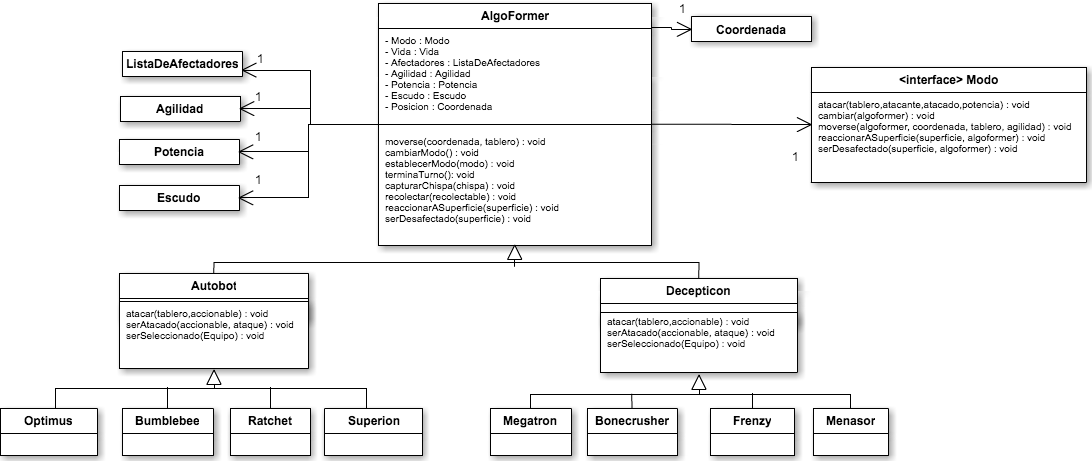
**Diagramas de clases**

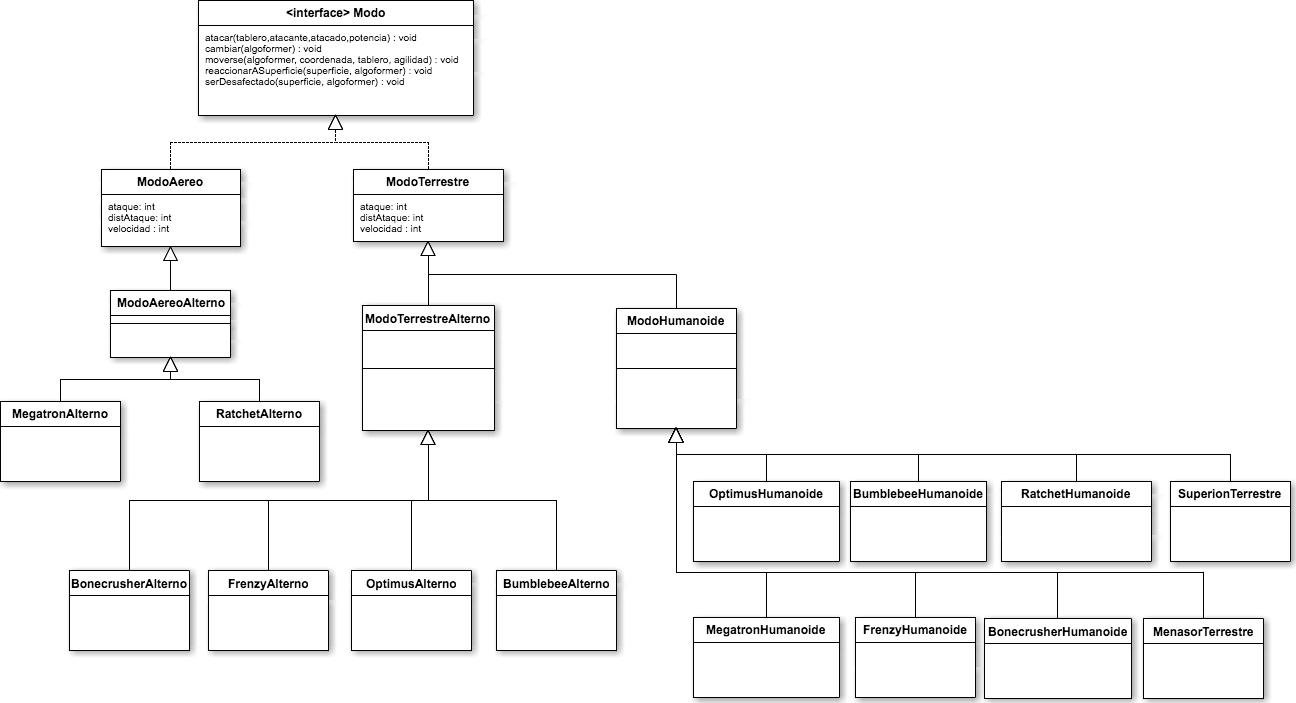
Aquí se encuentran los diagramas de clase que representan de una manera simplificada las relaciones estáticas entre los principales objetos que hacen al Modelo. Los diagramas se encuentran separados y en algunos se hacen referencia a Clases que están bien detalladas en otro diagrama.

Clases principales:

## 

AlgoFormer, el personaje principal del Juego



Relaciones de los Modos:

Superficies:

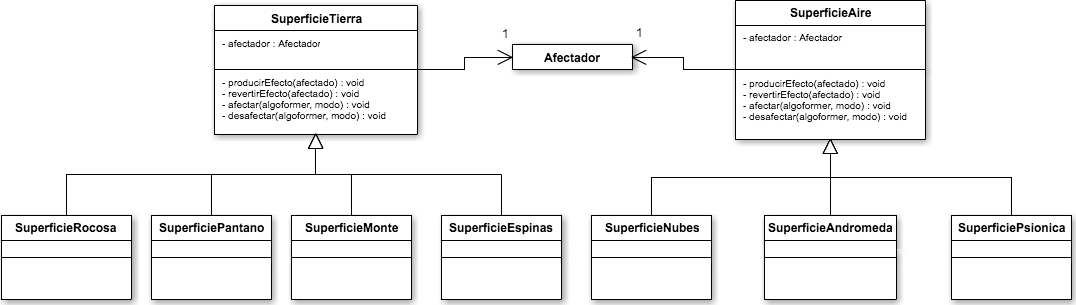
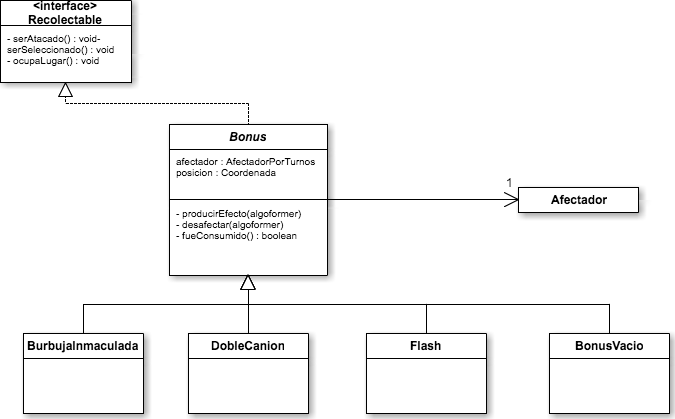
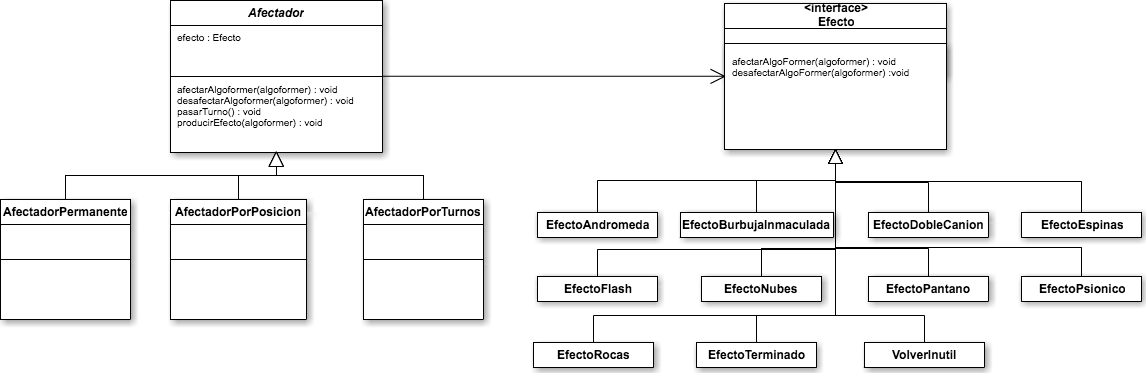


Diagrama de Bonus:



Afectadores y Efectos:

**Diagramas de secuencia**

## Diagrama de Ataque de un Algoformer:

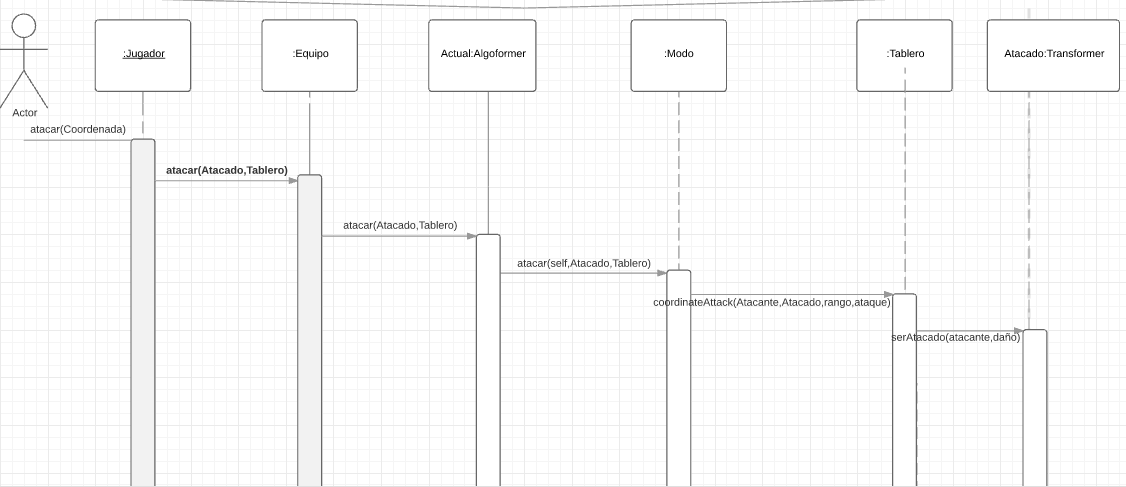


Diagrama de movimiento de un Algoformer:

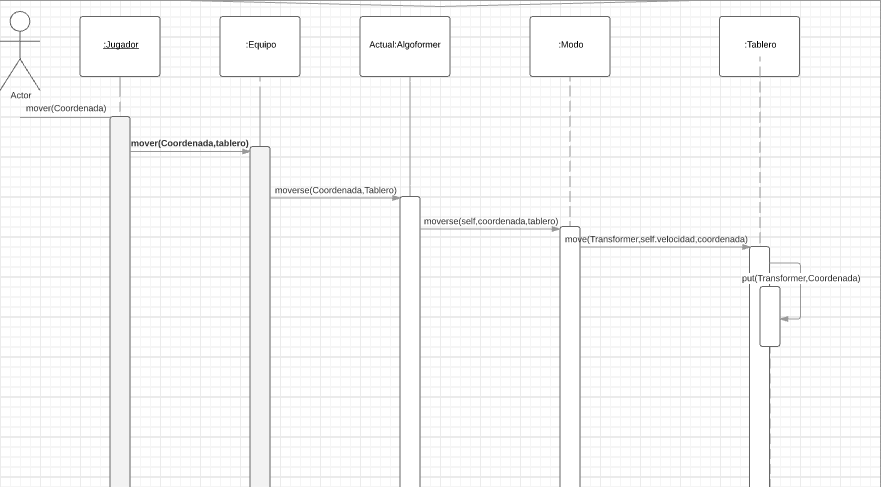


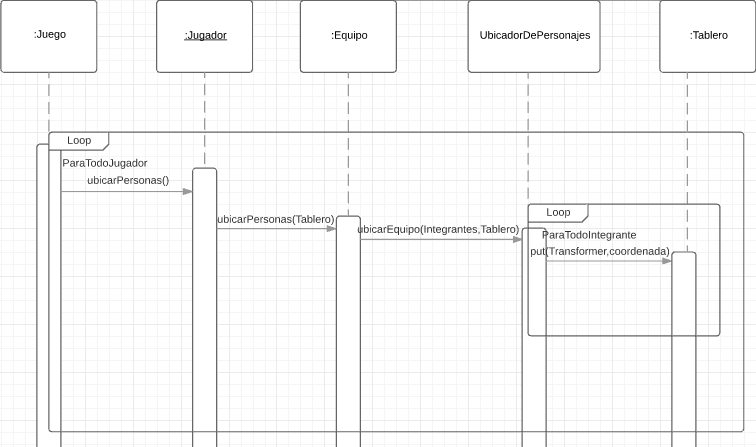
Diagrama de ubicación de un Algoformer:

Diagrama de Controlador:

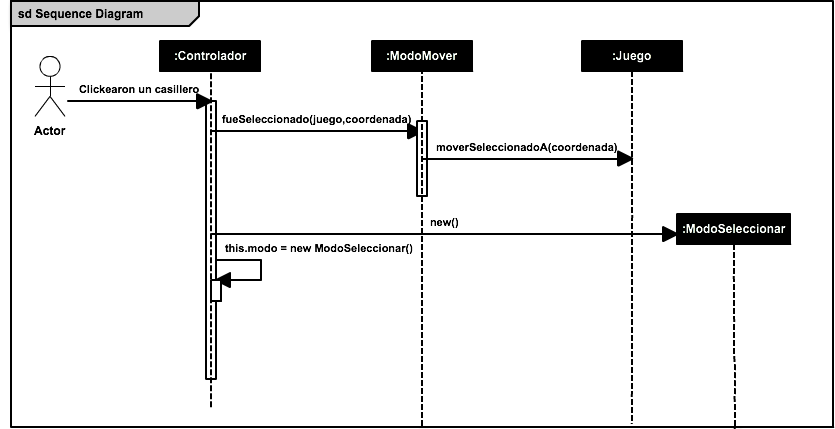


Diagrama seleccionar AlgoFormer correcto:

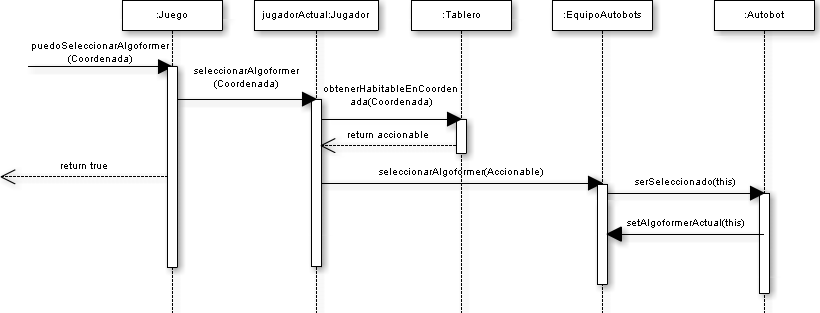


Diagrama de seleccionar AlgoFormer equipo correcto:

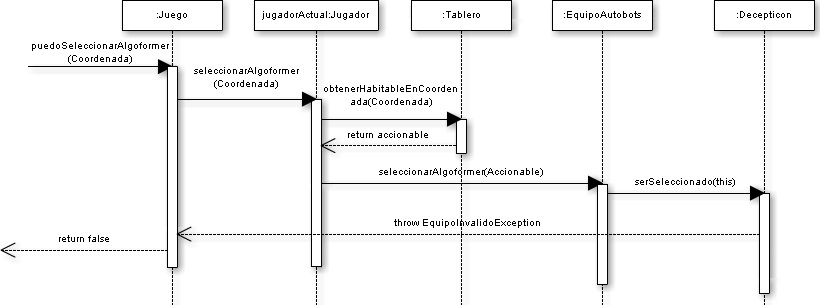


Diagrama seleccionar vacio:

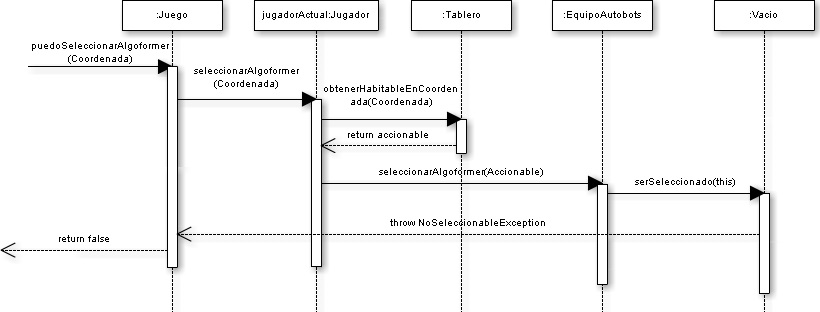


Diagrama setear imágenes en interfaz grafica:

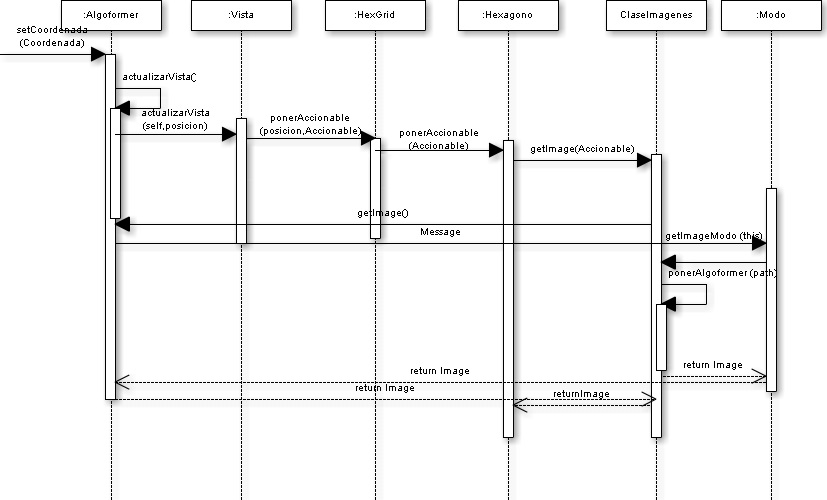
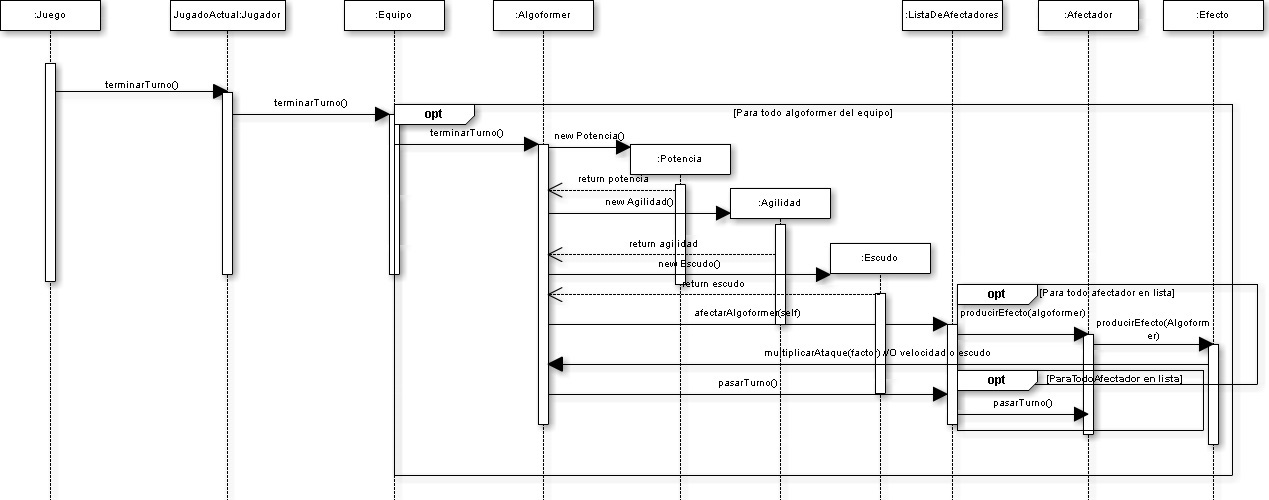
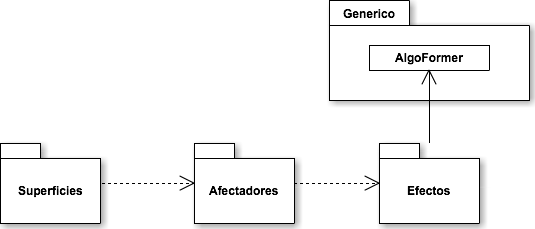


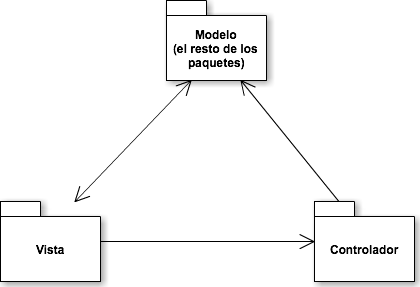
Diagrama de terminar turno:



**Diagrama de paquetes**

## 





**Diagramas de estado**

Diagrama de estados de afectador por turnos:

## 

Diagrama de estados de AlgoFormerActual:

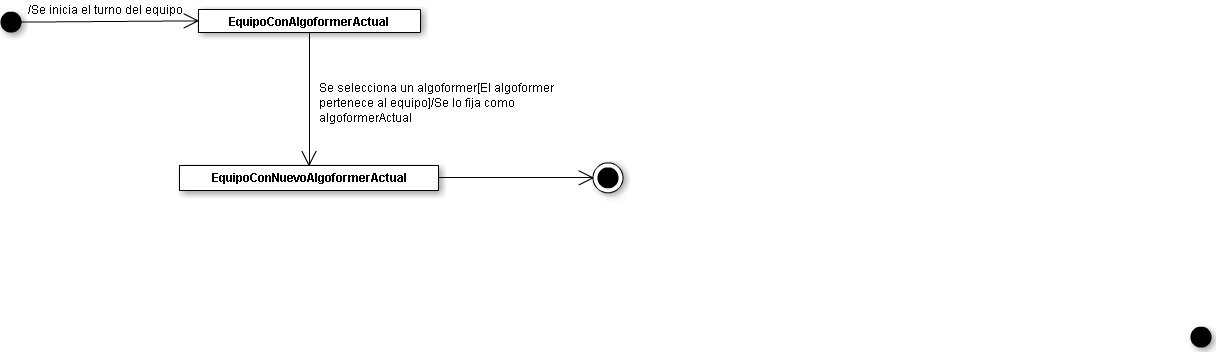


Diagrama de estados de la agilidad:



Diagrama de estados del AlgoFormer:

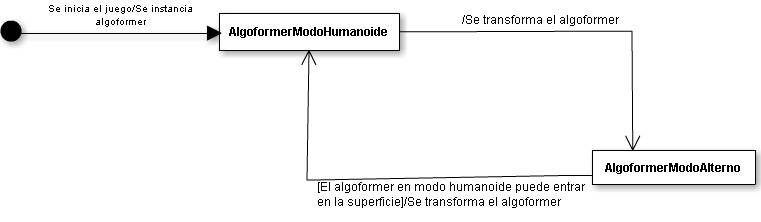


Diagrama de estados de la potencia:



Diagrama de estados de un AlgoFormer combinado:

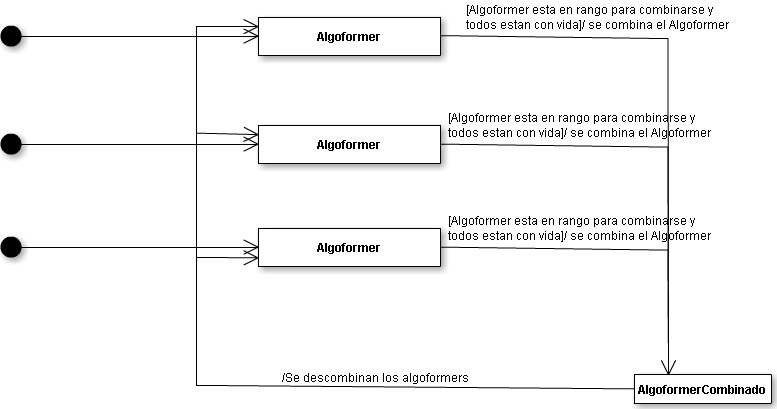
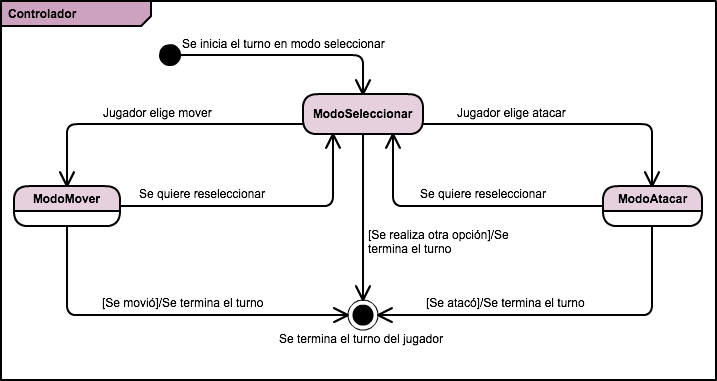


Diagrama de estados del controlador:



**Detalles de implementación****: Patrones de diseño utilizados**

En cuanto a los patrones de diseño utilizamos, para nombrar por encima los mas destacados se pueden mencionar:

**Strategy**, los modos de los AlgoFormers son strategy ya que estos cambian a decisión del jugador pero el AlgoFormer nunca deja de ser el mismo.

**State**, para los afectadores por turnos elegimos utilizar state para que una vez cumplida la vida útil del afectador este se vuelve automáticamente inactivo.

**Abstract factory**, para la tarea de manejar las imágenes decidimos implementar una pseudo abstract factory donde a la clase "ClaseImagenes" se le pasa un accionable o una superficie o un bonus y la misma devuelve una instancia de la imagen correspondiente a este objeto

**Facade**, en la clase Vista utilizamos facade ya que esta simula ser la interfaz grafica cuando en verdad solo una clase que contiene una interfaz grafica y de forma simplificada el modelo interactua con la clase "Vista" y esta le comunica a la verdadera interfaz grafica todas las acciones que debe llevar a cabo

**Excepciones****:**

* **NoCombinableException:**

Se lanza cuando los 3 AlgoFormers no cumplen las condiciones para combinarse, como son estar a una distancia mínima y que los 3 estén vivos.

* **DistanciaInvalidaException:**

Esta excepción se lanza cuando no se cumple la distancia mínima entre los tres AlgoFormers para combinarse y se agarra en el método que se encarga de combinar los AlgoFormers cuando esto ocurre se lanza una excepción indicando que no se pueden combinar los AlgoFormers esta es NoCombinableException.

* **ElementoNoExisteException:**

Para saber si los AlgoFormers pueden combinarse es necesario saber sus coordenadas para luego poder calcular la distancia entre ellos. Esta excepción se lanza cuando uno o mas AlgoFormers no existen en el Tablero o también cuando se busca la coordenada de un Recolectable que no existe en el Tablero.

Esta excepción se agarra en el mismo caso que DistanciaInvalidaException y se lanza NoCombinableException.

* **EquipoInvalidoException:**

Esta excepción se lanza cuando el usuario quiere seleccionar un AlgoFormer del equipo contrario. Por lo tanto se lanza en el método que tienen los AlgoFormers para ser seleccionados.

Y es agarrada por el Juego para saber si se pudo seleccionar en esa coordenada del tablero.

* **FueraDeRangoException:**

Es lanzada cuando el tablero coordina el ataque entre dos AlgoFormers y uno de estos se encuentra fuera del rango de ataque del otro.

* **MismoEquipoException:**

Se lanza cuando un AlgoFormer quiere atacar a otro de su mismo equipo.

* **ModoUnicoException:**

Se lanza cuando un AlgoFormer que solo tiene un modo, quiere cambiar su modo. En este caso solamente son los AlgoFormers combinados Superion y Menasor.

* **MovimientoInvalidoException:**

Se lanza cuando un AlgoFormer se quiere mover sobre una superficie donde no puede, también cuando quiere cambiar su modo sobre una superficie que no admite a ese AlgoFormer en ese modo. Es por esto que esta excepción se agarra en el método de transformar AlgoFormer que se encuentra en Jugador, si se agarra el paso siguiente es reposicionar al AlgoFormer a su posición valida anterior.

* **NoColisionableException:**

Se lanza cuando un AlgoFormer quiere moverse hacia la misma posición donde hay otro AlgoFormer y esta no es colisionable, colisionar significa que lo que se encuentra en esta posición si ocupa lugar y no puede haber otro AlgoFormer encima.

Esta excepción se agarra cuando se quiere saber el camino que puede tomar un AlgoFormer, y saber que no se puede colisionar con lo que lance esta excepción.

* **NoSeleccionableException:**

Esta excepción se lanza cuando se quiere seleccionar un elemento en el tablero que no es seleccionable por ejemplo una coordenada Vacia.

Se atrapa en el método de Juego, que se enecarga de devolver si se pudo seleccionar algo en esa coordenada y se devuelve falso.

# **Checklist de corrección**

Esta sección es para uso exclusivo de los docentes, por favor no modificar.

## **Carpeta**

**Generalidades**

* ¿Son correctos los supuestos y extensiones?
* ¿Es prolija la presentación? (hojas del mismo tamaño, numeradas y con tipografía uniforme)

**Modelo**

* ¿Está completo?¿Contempla la totalidad del problema?
* ¿Respeta encapsulamiento?
* ¿Hace un buen uso de excepciones?
* ¿Utiliza polimorfismo en las situaciones esperadas?

## **Diagramas**

**Diagrama de clases**

* ¿Está completo?
* ¿Está bien utilizada la notación?

**Diagramas de secuencia**

* ¿Está completo?
* ¿Es consistente con el diagrama de clases?
* ¿Está bien utilizada la notación?

**Diagrama de estados**

* ¿Está completo?
* ¿Está bien utilizada la notación?

# **Código**

**Generalidades**

* ¿Respeta estándares de codificación?
* ¿Está correctamente documentado?

1. Si por alguna razón existe un grupo de 3 integrantes, el mismo puede no implementar los bonus. [↑](#footnote-ref-1)