TP Nro2

AlgoFormers

**Informe**

Grupo 14

Integranes:

* Casal Joaquin
* Cruz Pablo
* Farfan Nicolas
* Repetto Pablo

Corrector: Massuh Pablo

**Supuestos**

1. El movimiento de los personajes será de a un casillero (el jugador elige paso a paso a que casillero moverse). La cantidad máxima de movimientos será la indicada por la velocidad de cada personaje.
2. Para poder combinarse, el líder del equipo (Optimus Prime en el caso de los Autobots y Megatron para los Decepticons) deberá estar como máximo a dos casilleros de cada uno de los demás integrantes del equipo.
3. El tablero será rectangular, de NxM casilleros.
4. La distribución de las superficies y los bonus en el tablero será aleatoria.
5. Cuando los Algoformers se combinan (o separan) o se transforman se pierden los bonus que se pudieran tener.
6. Al separarse los Algoformers (habiéndose combinado anteriormente), la vida se repartirá en partes iguales para cada personaje.
7. La posición de los personajes al separarse los Algoformers será aleatoria (en alguna de las ocho casillas alrededor de donde se encontraba el Algoformer combinado).
8. La forma de aparición de los Algoformers en el tablero siempre será en su modo humanoide.

**Modelo de dominio**

Objetos vistos en el dominio del enunciado: AlgoFormer, Forma, Tablero, Celda, Superficie, Bonus, Efecto, Posición, Movimiento, Jugador y Juego.

AlgoFormer en realidad es una clase abstracta, de la cual heredan dos clases hijas concretas: AutoBot y Decepticon.

Cada AlgoFormer tiene dos estados posibles: estado humanoide y estado alterno (el estado alterno puede ser tanto del tipo terrestre como aéreo); es por eso que cada instancia de AlgoFormer está asociada a una instancia Forma. Si bien el objeto AlgoFormer es uno solo, puede cambiar su comportamiento dinámicamente gracias a este estado que guarda, ya que Forma sabe cambiar de estado.

Jugador representa a cada uno de los jugadores del juego. Contiene a todos los AlgoFormers de su equipo. Al igual que AlgoFormer, tiene varios estados posibles. Entre ellos: estado activo, estado inactivo y estado combinándose.

La clase Tablero representa el escenario en el cual se va a desarrollar el juego; es una cuadrícula rectangular de instancias de Celda.

Celda contiene a un AlgoFormer (si hubiese uno en la celda), un Bonus y a la chispa suprema (si estuviese en la celda). Las celdas tienen asociada una instancia de Posición (que representa a las coordenadas de cada celda dentro del tablero).

Además, cada Celda tiene dos Superficies (una SuperficieAerea y otra SuperficieTerrestre), que a su vez contienen un Efecto (el efecto que contienen depende de qué tipo de superficie son). De igual manera, la celda también contiene un Bonus que también tiene un efecto.

Movimiento es un enum que representa a los movimientos que un AlgoFormer puede realizar, esto es las posibles direcciones en el tablero.

Juego es la clase encargada de utilizar a todas las clases descriptas recién (directa o indirectamente) para llevar cabo la ejecución del juego. Es decir, inicializa a todas las clases del juego y luego las utiliza para llevar adelante la ejecución.

**Detalles de implementación**

Los puntos más conflictivos que encontramos durante el desarrollo son la creación de los AlgoFormers y el movimiento de los mismos.

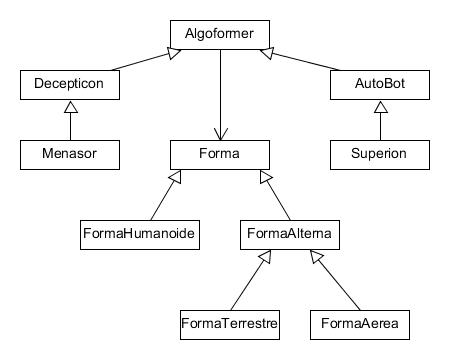
Con la creación nos referimos a que existen seis personajes específicos que tienen que existir en el juego, y si bien la solución que primero se nos ocurrió fue hacer una clase que herede de AlgoFormer para cada personaje, nos pareció que era una forma de implementación muy difícil de mantener (si se quisiera agregar más personajes, por ejemplo). Por lo que decidimos que los AlgoFormers se creen a partir de una clase AlgoFormerFactory, que contenga toda la información de los AlgoFormers existentes y que, cuando se lo requiera, devuelva un AlgoFormer ya creado.

Por otro lado, el movimiento fue un poco conflictivo también ya que un AlgoFormer tiene una cantidad de movimientos máximos por turno que no puede sobrepasarse, además de que hay muchos caminos para llegar desde la celda A hasta la celda B, con distintas consecuencia cada uno. Para tratar estos problemas, decidimos que el usuario se irá moviendo de a un casillero, eligiendo paso a paso por donde moverse, por lo que será su responsabilidad elegir con que camino llegar hasta la celda B; al mismo tiempo, se irá contando la cantidad de movimientos que se hicieron en el turno y verificando que no supere la velocidad del personaje, y si esto ocurriera, el AlgoFormer ya no se moverá por más que el usuario lo requiera.

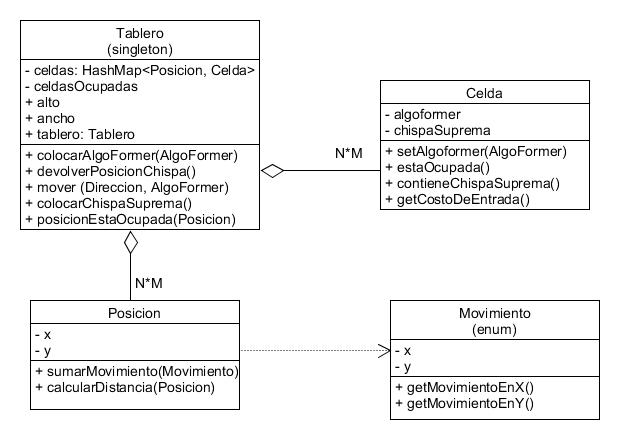
El tercer punto conflictivo que encontramos fue al momento de aplicar los efectos de las Superficies o de los Bonus sobre los algoformers. Luego de debartirlo y consultarlo con nuestro corrector, decidimos que sea el AlgoFormer el que recibe el efecto y sepa cómo este debe modificarlo (algunas veces, delegando en su Forma).

**Diagramas de clases:**

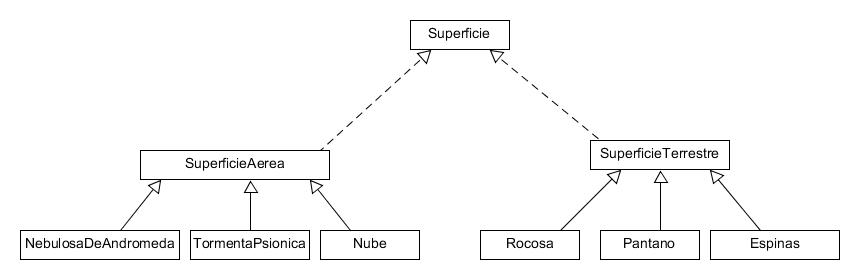
**Diagrama de clases del paquete algoformers**

****

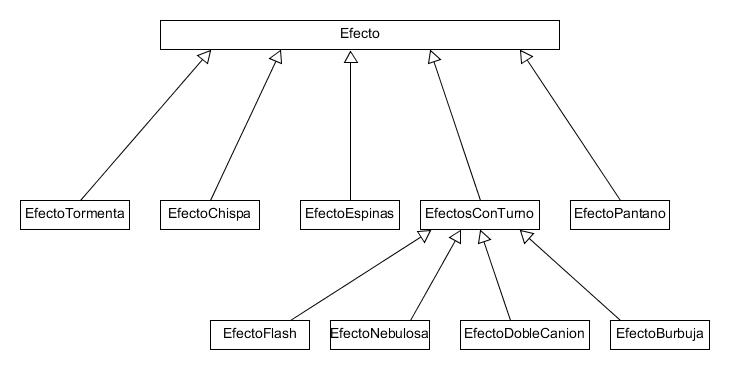
**Diagrama de clases del paquete escenario**

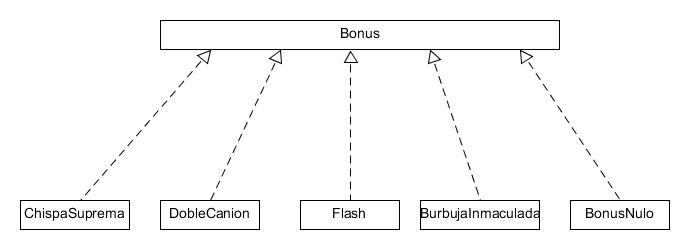
****

**Diagramas de clases del paquete superficies**

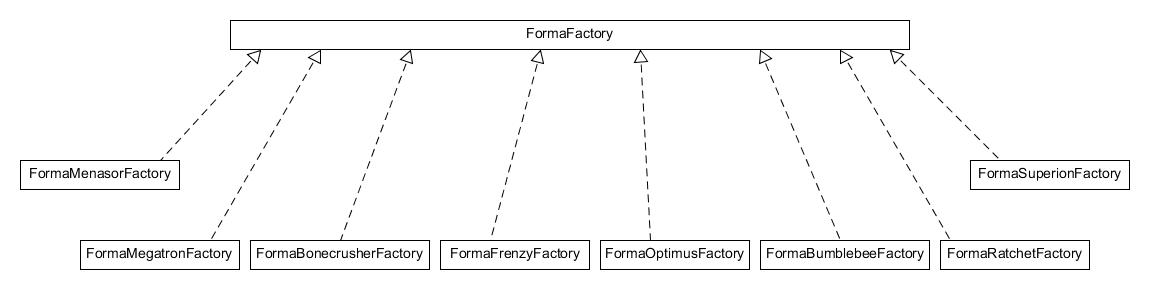
****

**Diagramas de clases del paquete efectos**

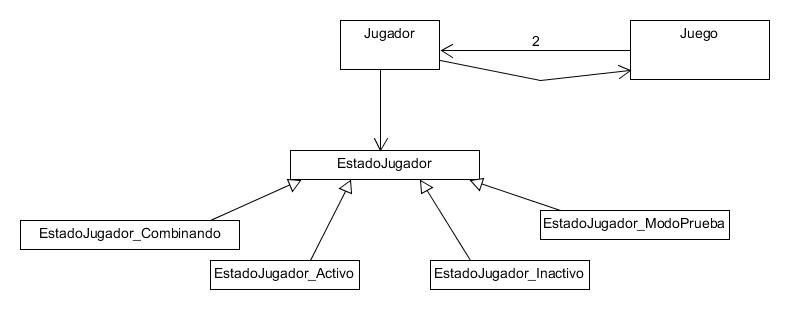
****

**Diagrama de claes del paquete bonus**

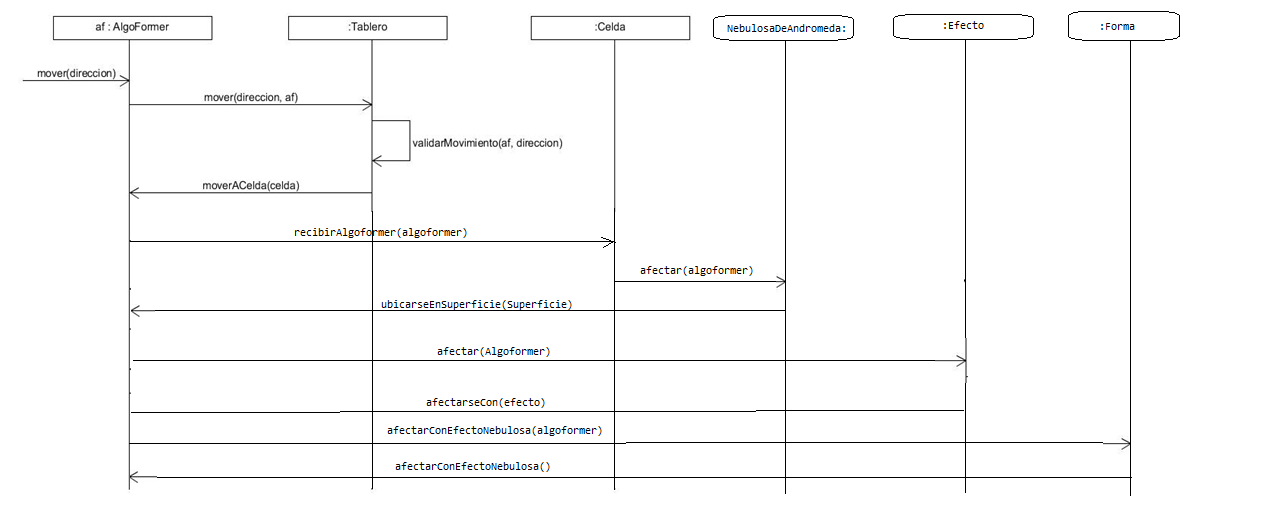
**Diagrama de clases de FormaFactory**

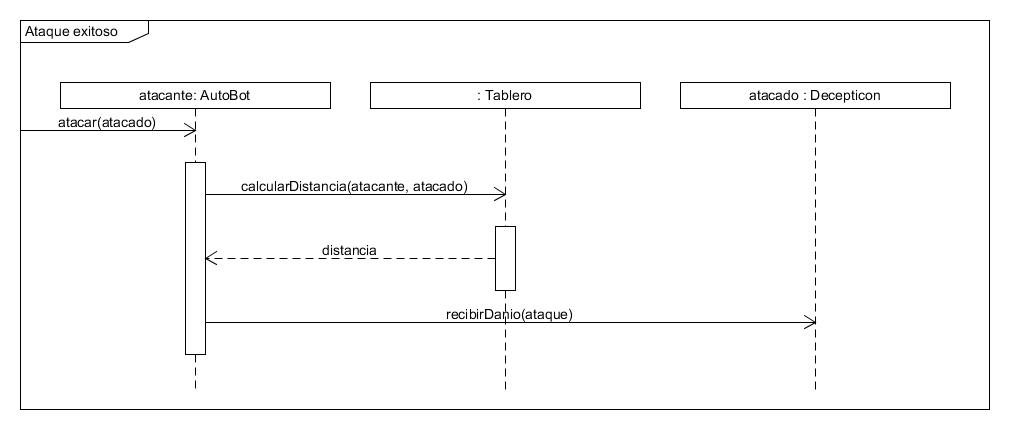
****

**Diagrama de clases del paquete juego**

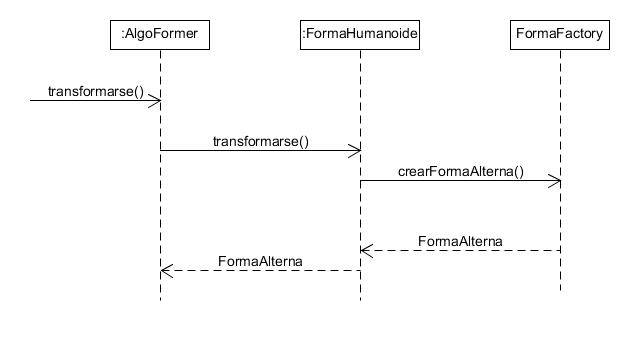
****

**Diagramas de secuencia**

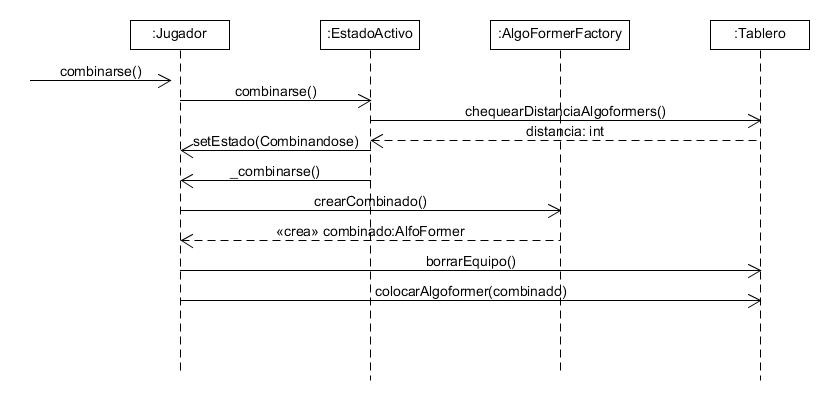
**Diagrama de secuencia de AlgoFormer.mover**

**Diagrama de sucuencia de AlgoFormer.atacar**

**Diagrama de secuencia de AlgoFormer.transformarse**

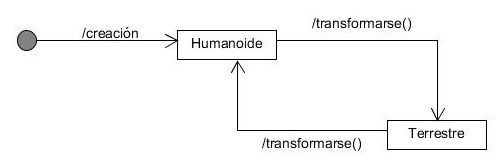
****

**Diagrama de secuencia de Jugador.combinarse**

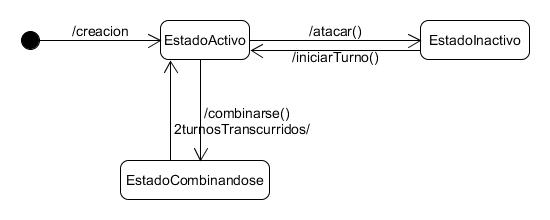
****

**Diagramas de estado**

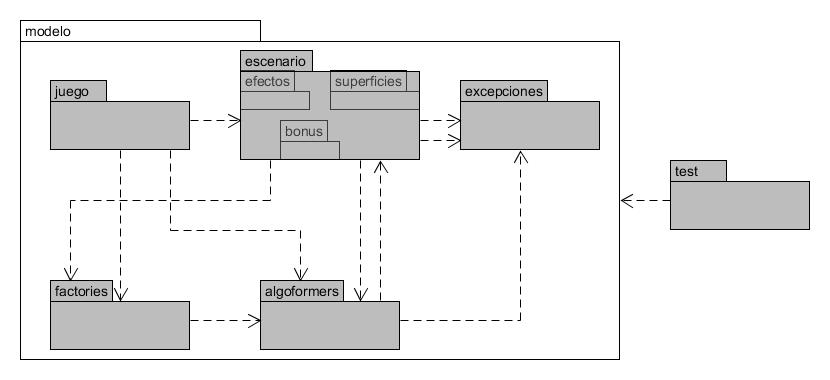
**Diagrama de estado de cambio de estado de la Forma del AlgoGormer**

****

**Diagrama de estado de cambio de estado de Jugador**

****

**Diagramas de paquetes (del modelo)**

****

**Excepciones**

**FueraDeAlcanceException:** es lanzada cuando un AlgoFormer intenta a atacar a otro que está más lejos que lo que su distancia de ataque le permite alcanzar. Si es lanzada, es atrapada por el EventHandler del BotonAtacarAlgoformer,.

**NoHayMasMovimientosException:** el movimiento de los AlgoFormer es de a un casillero por vez, por eso, cuando se intenta hacerlo avanzar más casilleros que lo que su velocidad le permite en un solo turno, esta excepción es lanzada. Es atrapada por el manejador del movimiento (MovimientoHandler).

**CeldaOcupadaException:** es lanzada cuando se intenta colocar o mover a un AlgoFormer en una celda del tablero que ya está ocupada por otro AlgoFormer. Es atrapada por MovimientoHandler.

**PosicionInvalidaException:** es lanzada cuando se intenta colocar o mover a un AlgoFormer a una posición inválida, es decir, que no existe en el tablero. Es atrapada por MovimientoHandler.

**FuegoAmigoException:** es lanzada cuando se intenta atacar a un AlgoFormer del mismo equipo. Es atrapada por el EventHandler de BotonAtacarAlgoformer.

**AlgoformerMuyLejosException:** es lanzada cuando un jugador intenta combinarse sin que sus algoformers no estén lo suficientemente cerca. Es atrapada por BotonCombinarseEventHandler.

**EquipoIncompletoException:** es lanzada cuando un jugador intenta combinarse teniendo uno o más integrantes del equipo muertos, es atrapada por BotonCombinarseEventHandler

**SinLugarParaDescombinarseException:** es lanzada cuando un jugador intent descombinarse sin tener los casilleros suficientes alrededor del algoformer combinado para poder alojar a todos los miembros del equipo. Es atrapada por BotonCombinarEventHandler

Las excepciones hasta aquí descriptas son atrapadas durante la ejecución del programa, y el manejo de todas es exactamente el mismo: levantar una ventana de alerta indicando lo sucedido.

Sin embargo, las excepciones descriptas a partir de aquí no son atrapadas en ningún lugar, ya que por la naturaleza de la interfaz grafica, es imposible que sean lanzadas (por ejemplo, si un jugador esta combinado, el botón de combinarse se deshabilita, por lo que no podría lanzarse la excepción YaEstaCombinadoException)

**NoEstaCombinadoException:**  es lanzada si un jugador se intentara descombinar sin estar combinado con anterioridad.

**YaEstaCombinadoException:** es lanzada si un jugador se intentara combinar, estando ya combinado.

**AlgoformerNoTieneIntegrantesException:** es lanzada si se le pidiera a un algoformer (no un combinado) que devuelva sus integrantes. Esta es solo una propiedad de los algoformers combinados.

**CombinadoNoPuedeTransformarseException:** es lanzada si se intentara transformar a un algoformer combinado, lo cual no es posible, ya que solo tienen una forma.

**CombinadoNoTieneFormaHumanoideException:** es lanzada si se le intentara pedir la forma humanoide de un algoformer combinado a la factory encargada de crearlo.