Informe

Supuestos

- Los AlgoFormers se mueven de a un casillero por vez, siendo el limite de casilleros seguidos que se puede mover la Velocidad de los mismos. De todos modos, la acción mover es limitante en el turno. Se mueva un casillero o siete, es lo único que puede hacer.
- Los AlgoFormers siempre deben atacar a otros AlgoFormers enemigos, en caso de atacar al 'aire' o a un amigo, se levantará una excepción.

Modelo de dominio

Propusimos un modelo en que la clase principal es el Juego, que será la interfaz con la que más adelante interactuará el controlador. El Juego se encarga de delegar todas las tareas que son necesarias para que la aplicación funcione.
Además, existen las clases Jugador y Mapa, cada Jugador tiene sus AlgoFormers y se encarga de darle las órdenes (que recibe a través del Juego) y el Mapa modela al mundo del juego, con sus coordenadas y casilleros. Estos casilleros almacenan tanto un AlgoFormer como un objeto Capturable (hasta aquí, la Chispa).
Los AlgoFormers tienen un EstadoDeTransformación, el cual representa a los estados Alterno y Humanoide. Cada estado tiene su Vida, Ataque, Velocidad y Distancia de Ataque.

Diagramas de clases

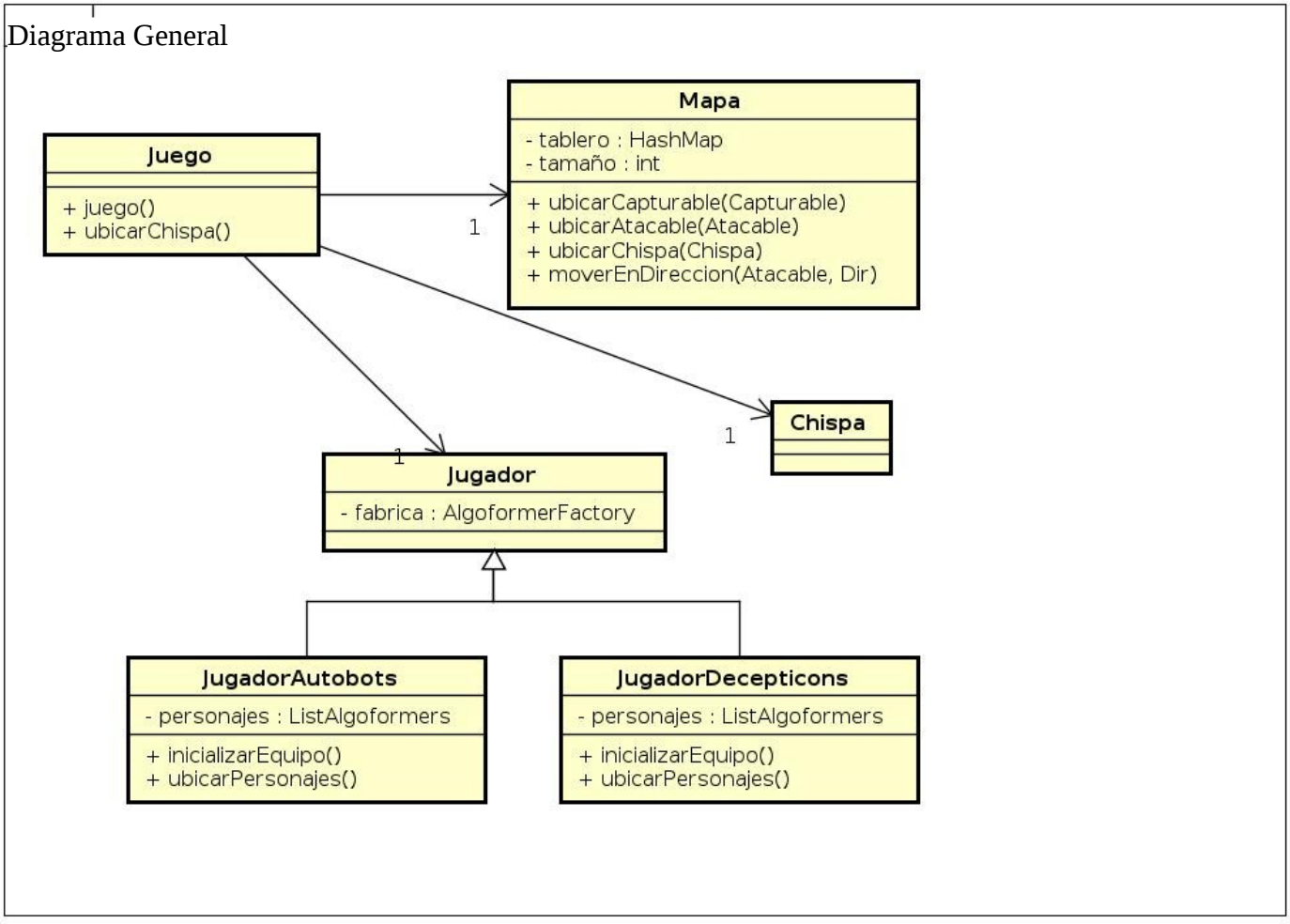


Diagrama de Mapa

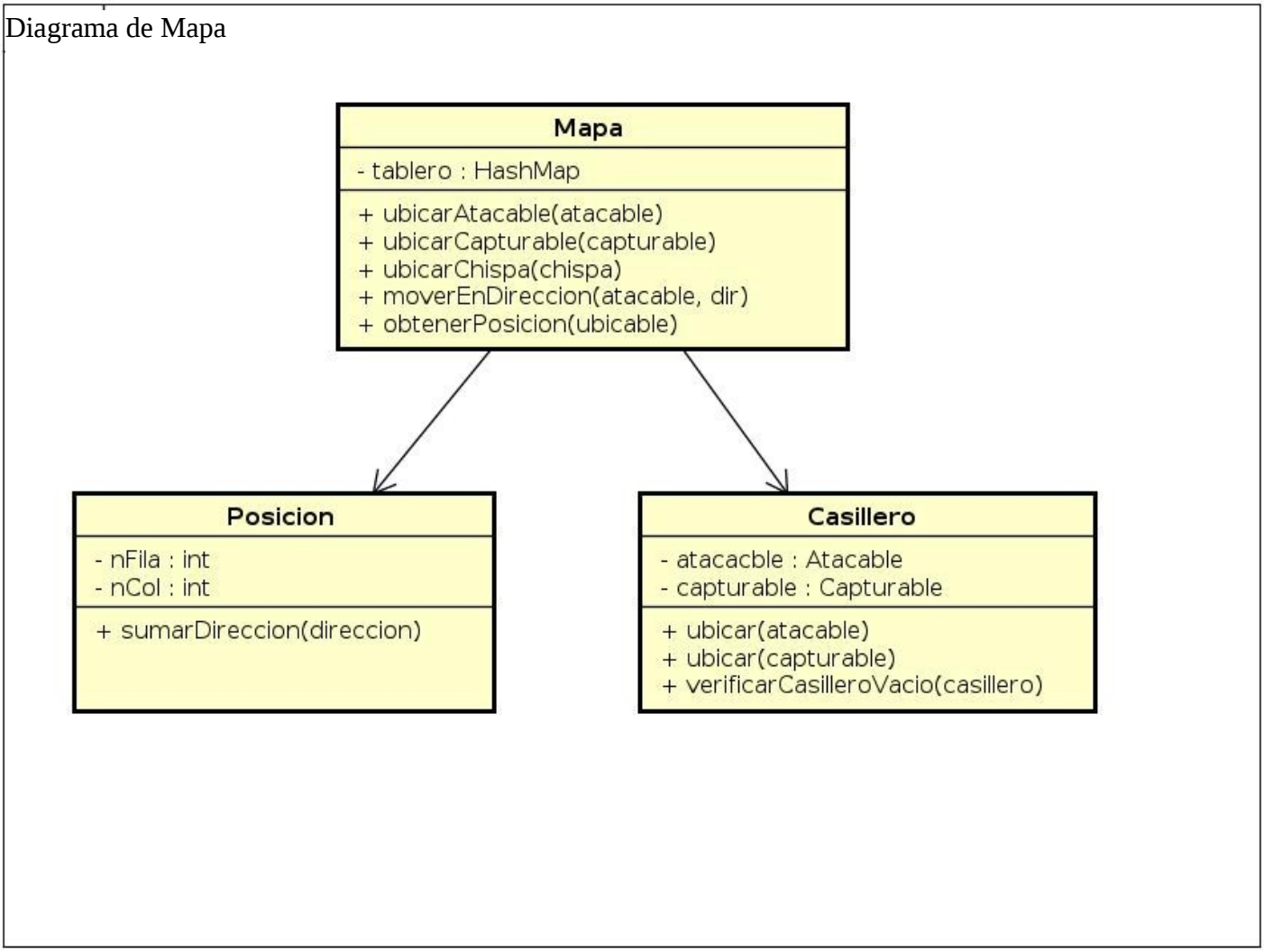
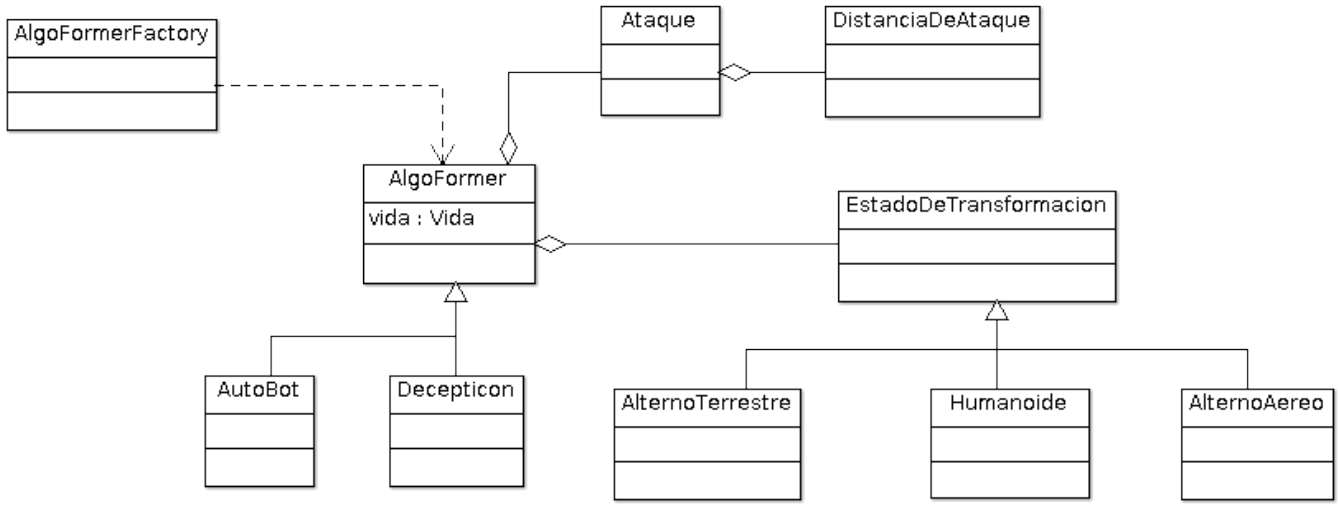
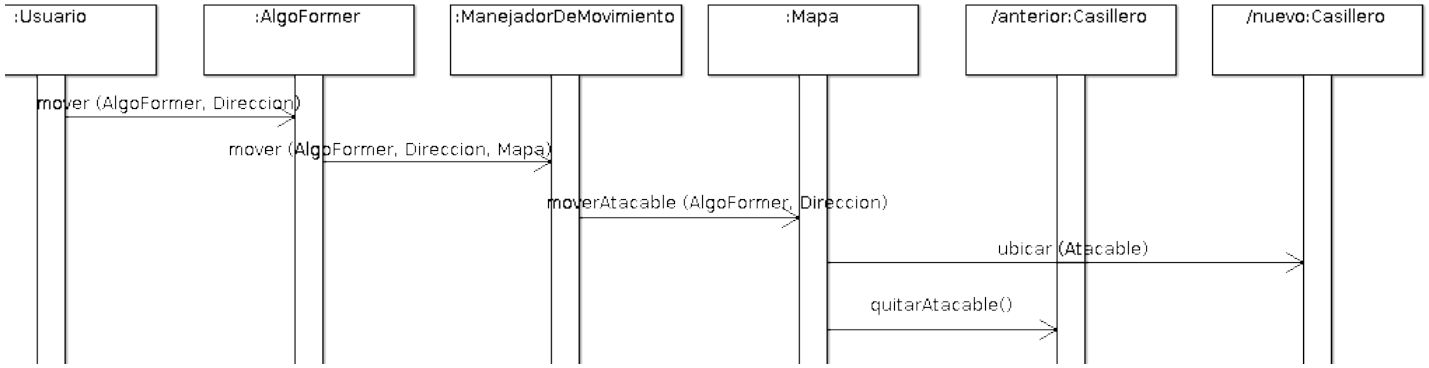


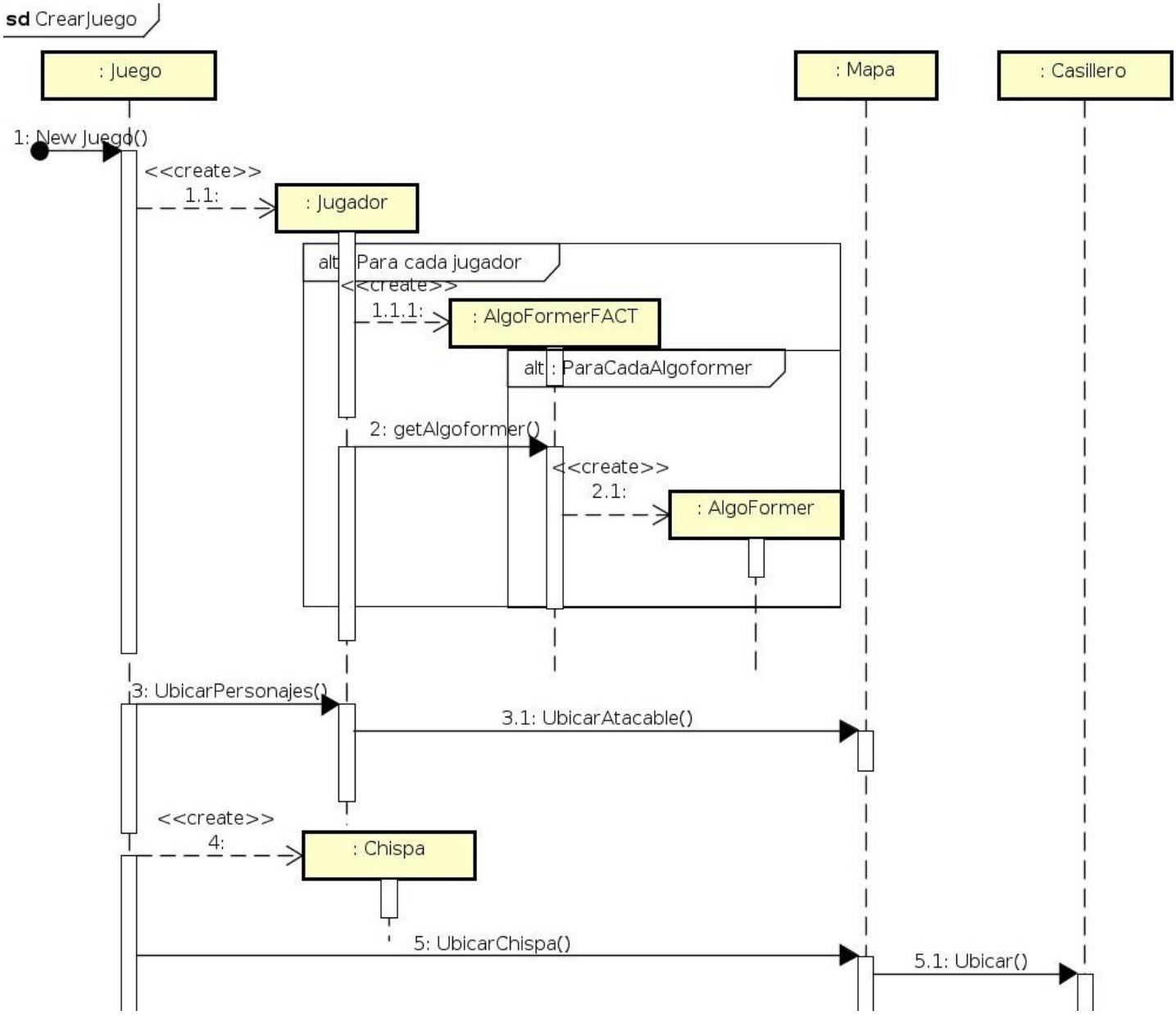
Diagrama de AlgoFormer

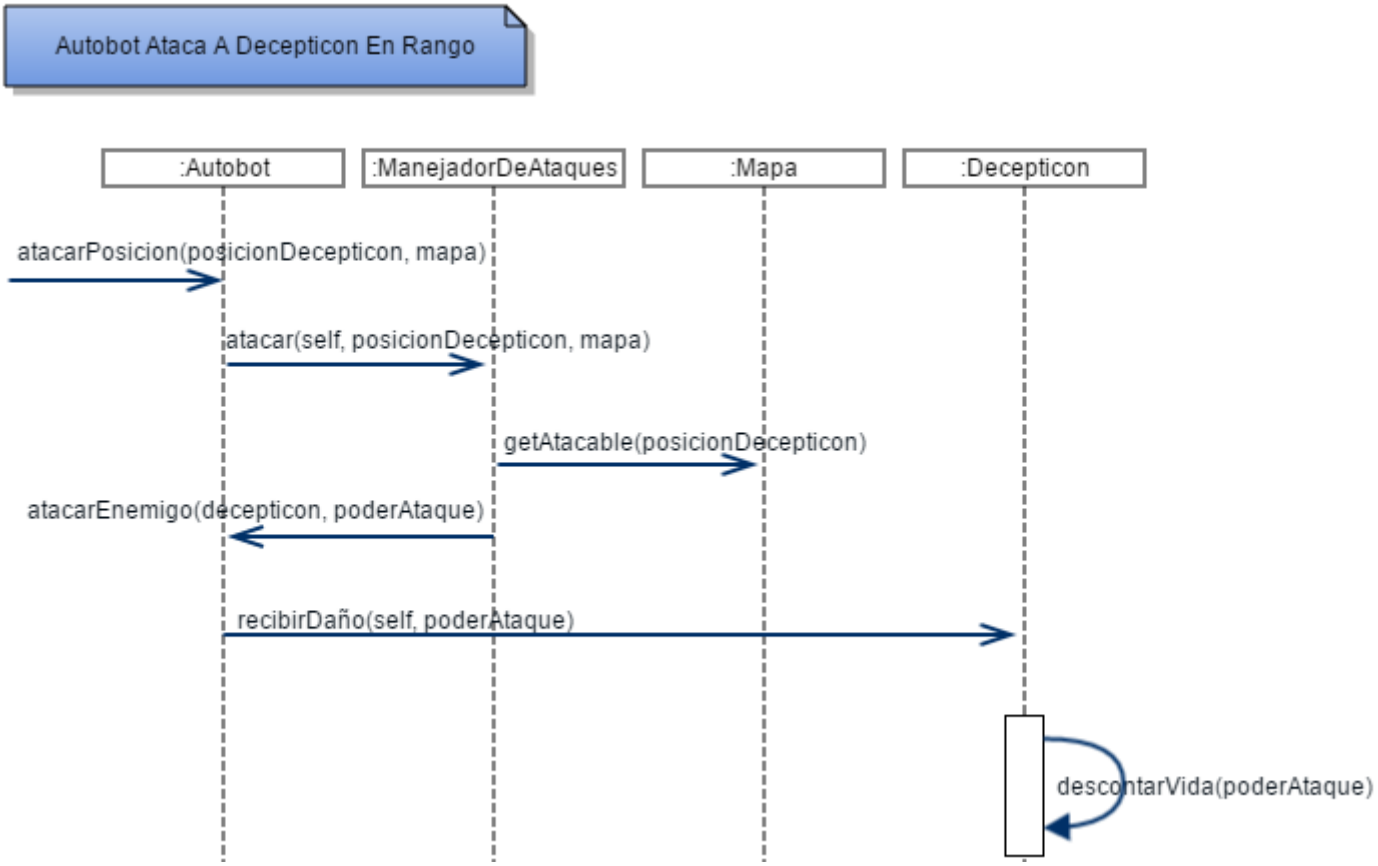


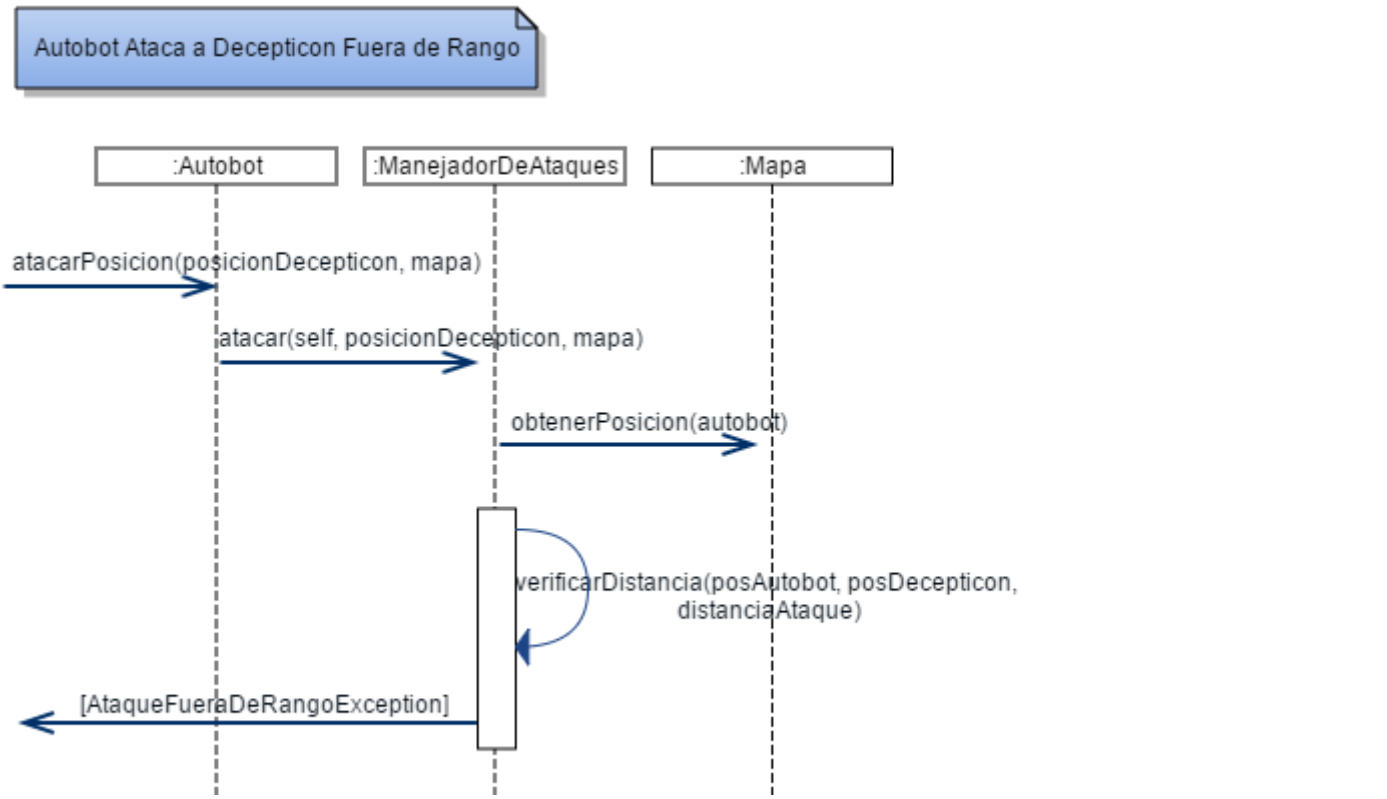
Diagramas de secuencia

Diagrama de secuencia mover AlgoFormer






Autobot Ataca A Decepticon En Rango
:Autobot
:ManejadorDeAtaques
:Mapa
:Decepticon
atacarPosicion(posicionDecepticon, mapa)
atacar(self, posicionDecepticon, mapa)
getAtacable(posicionDecepticon)
atacarEnemigo(decepticon, poderAtaque)
recibirDaño(self, poderAtaque)
descontarVida(poderAtaque)


Autobot Ataca a Decepticon Fuera de Rango
:Autobot
:ManejadorDeAtaques
:Mapa
atacarPosicion(posicionDecepticon, mapa)
atacar(self, posicionDecepticon, mapa)
obtenerPosicion(autobot)
verificarDistancia(posAutobot, posDecepticon, distanciaAtaque)
[AtaqueFueraDeRangoException]

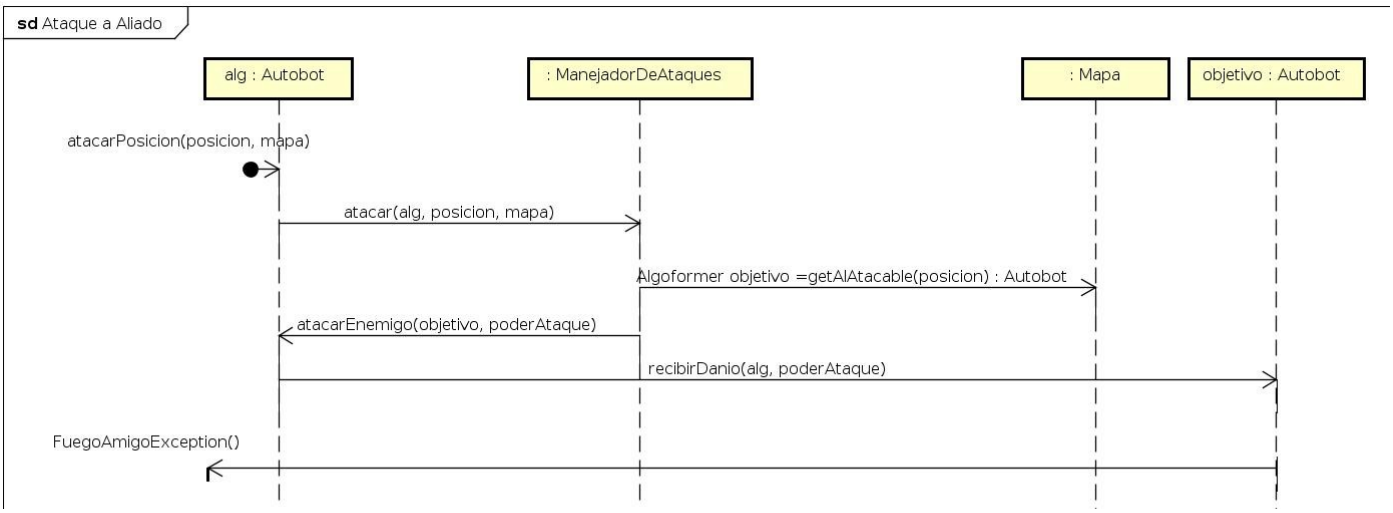
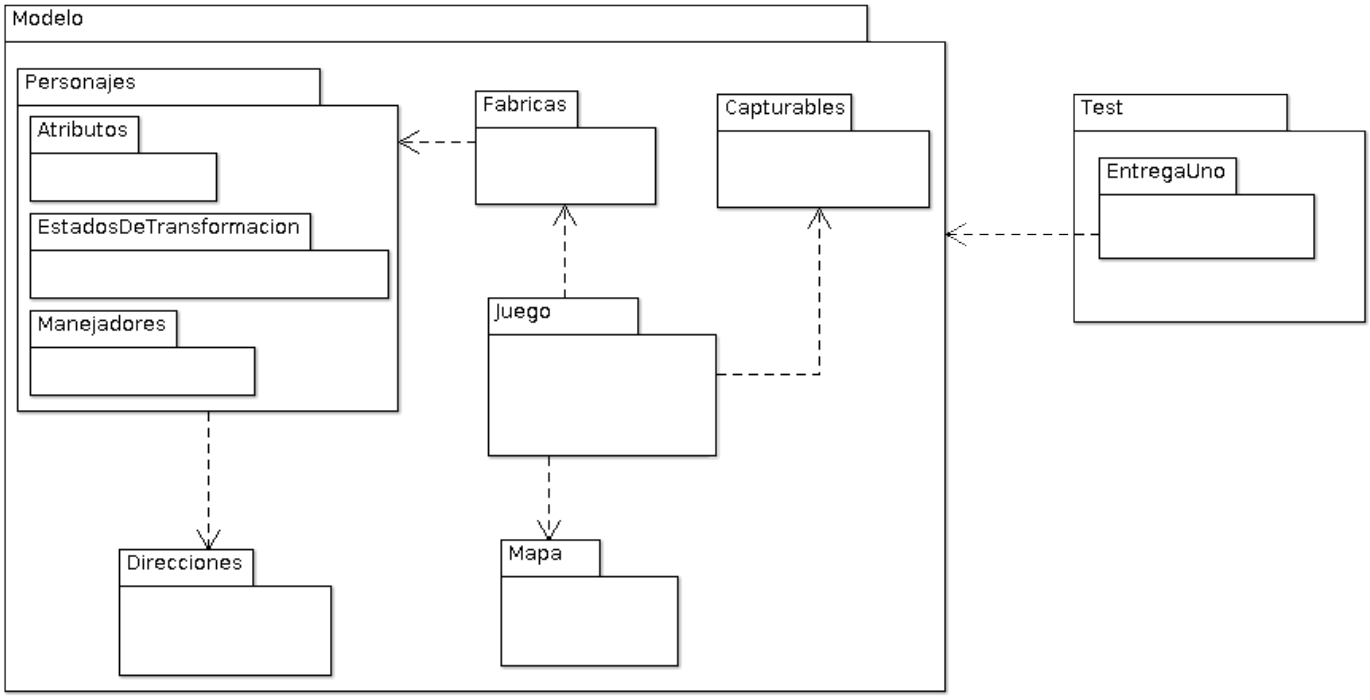
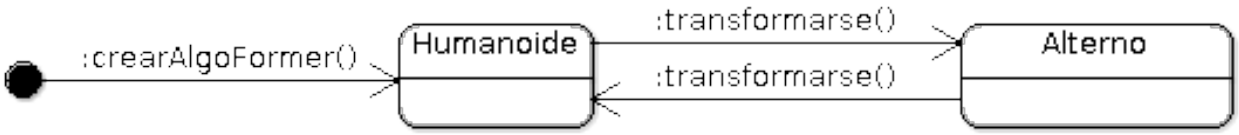


Diagrama de paquetes



Diagramas de estado



Detalles de implementación

- Para la creación de los AlgoFormers usamos AbstractFactory, hay una gran AlgoFormerFactory que tiene sus AutoBotFactory y DecepticonFactory. De esta manera para, por ejemplo, crear a Optimus Prime, el Jugador dice algoFormerFactory.getOptimus() y esta hace this.autoBotFactory.getOptimus().
- Para el movimiento, creamos la clase Dirección (y sus subclases DirArriba, DirAbajo, …) que se utiliza para calcular la nueva posición del AlgoFormer en un movimiento. Entonces, la idea aquí es que cuando se elija una dirección en la cual mover al personaje (ya en el flujo del juego) se cree el objeto Direccion y se le diga al personaje algoFormer.mover(direccion).
- En cuanto a los estados de transformación, los AlgoFormers tienen dos estados almacenados. El ‘estado actual’ y el ‘estado opuesto’. Todas las acciones que se delegan al estado siempre se delegan al actual. Cuando se pide al personaje que se transforme, se intercambian las referencias de esos dos estados.

Excepciones

Todas las excepciones están creadas para más tarde notificar al usuario con un ‘cartel de error’. Por ejemplo, si trata de atacar a un amigo, verá una notificación que dice que el fuego amigo no esta permitido. Las excepciones son:

- AtaqueEspacioVacioException: se lanza cuando se trata de atacar a un espacio vacío.
- AtaqueFueraDeRangoException: se lanza cuando se trata de atacar a una posición fuera del rango de ataque del personaje.
- CasilleroOcupadoException: se lanza cuando se trata de ubicar un personaje en un casillero que ya tiene a otro personaje ubicado.
- FuegoAmigoException: se lanza cuando se trata de atacar a una posición que tiene un personaje aliado.
- MovimientosAgotadosException: se lanza cuando ya no quedan puntos de velocidad y un personaje trata de moverse.