Jugador Inteligente BattleTech

BORJA MOLINA ZEA – BORJA.MZ8@GMAIL.COM

INGENIERÍA DEL CONOCIMIENTO

Contenido

[1. Introducción 2](#_Toc366535416)

[1.1. Consideraciones iniciales 2](#_Toc366535417)

[1.2. Organización del material entregado 2](#_Toc366535418)

[2. Descripción del problema 2](#_Toc366535419)

[2.1. BattleTech 2](#_Toc366535420)

[2.2. Elementos de BattleTech 3](#_Toc366535421)

[2.3 Objetivo 3](#_Toc366535422)

[2.4. Fases de juego 3](#_Toc366535423)

[2.5 El simulador 3](#_Toc366535424)

[3. Abstracción del problema 4](#_Toc366535425)

[4. Propuesta y justificación de un modelo teórico 4](#_Toc366535426)

[5. Descripción y justificación de la solución 5](#_Toc366535427)

[5.1. Representando el entorno 5](#_Toc366535428)

[5.2. La fase de movimiento 7](#_Toc366535429)

[5.2.1. ¿Cómo moverse? 7](#_Toc366535430)

[5.2.2. ¿Dónde moverse? 8](#_Toc366535431)

[5.2.3. Autocrítica a la solución adoptada 9](#_Toc366535432)

[5.3. La fase de reacción 9](#_Toc366535433)

[5.4. La fase de ataque con armas 9](#_Toc366535434)

[5.4.1. ¿A quién atacar? 9](#_Toc366535435)

[5.4.2. ¿Cómo atacar? 9](#_Toc366535436)

[5.5. La fase de ataques físicos 10](#_Toc366535437)

[5.6. La fase de final de turno 11](#_Toc366535438)

[6. Conclusiones 11](#_Toc366535439)

# 1. Introducción

## 1.1. Consideraciones iniciales

**1.** El modelo teórico se encuentra totalmente especificado para todas las fases pero las fases de ataque con armas y final de turno no están totalmente implementadas. Son funcionales –no producen errores de log- pero triviales.

El resto de fases: movimiento, reacción y ataques físicos se encuentran tanto especificadas como totalmente implementadas.

**2.** La práctica ha sido programada en C#, con Visual Studio, en Windows 7 y usando la versión de .NET 2.5 para evitar incompatibilidad en XP.

En el ordenador descrito la práctica es completamente funcional y no da ningún error. Si a la hora de ejecutarla en los ordenadores de la facultad con XP da algún tipo de error pido encarecidamente que se ponga en contacto conmigo: Borja.mz8@gmail.com, antes de evaluar con un 0 una práctica que me ha llevado muchas horas de trabajo.

**3.** La práctica ha sido realizada por una sola persona. No encontré nadie que la quisiera hacer conmigo.

## 1.2. Organización del material entregado

Dentro de la carpeta principal encontrará 3 carpetas.

1. **Carpeta ejecutables**: los dos ejecutables necesarios para ejecutar el jugador inteligente, 76422722.exe que se corresponde con el jugador inteligente y LDVyC.exe usado por 76422722.exe para el cálculo de la línea de visión y que tienen que estar en la misma carpeta.

2. **Carpeta fuentes**: incluye el código fuente de todas las clases.

3. **Carpeta proyecto**: incluye el proyecto entero en formato visual studio 2012.

# 2. Descripción del problema

## 2.1. BattleTech

BattleTech es un **juego de rol por turnos** de tablero en el que Mechs –robots gigantes como los de Pacific Rim o Evangelion- se enfrentan entre sí. Los Mechs son pilotados por los Mechs-Warriors, humanos que van en su interior y los dirigen.

El juego no es totalmente determinista, cómo por ejemplo el ajedrez, sino que cuenta con cierto componente aleatorio ya que muchas de las acciones requieren de una tirada de dados y que salga cierto número para que consigan llevarse a cabo. A pesar de la existencia de dicho componente aleatorio, el juego requiere de un **complejo y completo estudio del entorno** para poder llevar a cabo una buena estrategia.

## 2.2. Elementos de BattleTech

El juego de tablero está compuesto por dos elementos principales:

1. Los **Mechs**: como ya hemos comentado son los robots gigantes que manejamos y a los que nos enfrentamos. Están equipados con armas, algunas de ellas de energía –como los láseres- y otras que requieren munición para funcionar –ametralladoras, lanzadores de misíles-

2. **Mapa**: la partida se desarrolla en un tablero de casillas hexagonales, cada hexágono es la unidad mínima de movimiento por lo que nos encontramos con un entorno de movimiento discreto, al igual que el ajedrez. Cada hexágono tendrá una altura, un tipo de terreno y otros elementos como bosques o edificios.

## 2.3 Objetivo

Matar a todos los enemigos y no morir en el intento.

### 2.4. Fases de juego

El juego se divide en 5 etapas:

**1. Fase de moviento**: todos los Mechs se mueven –o intentan hacerlo- para dar caza a otros mechs, huir de una muerta segura, posicionarse en un lugar estratégico y en definitiva conseguir con su movimiento tomar ventaja al resto de Mechs.

**2. Fase de reacción**: los Mechs pueden girar su torso para estar bien posicionados para su ataque con armas.

**3. Fase de ataque con armas**: los Mechs utilizan sus armas de distancia –láseres, misiles, ametralladoras…- contra otros Mechs para causarles daños o contra el terreno para devastarlo.

**4. Fase de ataque físico**: los Mechs golpean a otros mechs adyacentes con sus brazos, piernas o arma blanca –espadas, garrotes- de tenerlos.

**5. Fase de fin de turno**: se apagan radiadores y se expulsa la munición que ya no se quiera.

## 2.5 El simulador

Para que el alumno pueda desarrollar la práctica se le da el **simulador de BattleTech**, un programa que simula una partida de BattleTech con todos sus elementos y que **automatiza los cálculos y tiradas de dados** por lo que el alumno solo ha de preocuparse de comprender el juego y buscar una buena estrategia ya que el trabajo más tedioso ya se da hecho.

El simulador puede ser manejado por una persona humana usando su interfaz o por un jugador inteligente que mediante ficheros de texto se comunicará con el simulador.

# 3. Abstracción del problema

El problema que debemos resolver, **crear un jugador inteligente que controle un mech del simulador de la mejor forma posible**, es un claro ejemplo de problema que puede ser resuelto mediante un agente inteligente.

Se ha de manejar un **considerable cantidad de información** –estado del mapa, estado de nuestro Mech y de los Mechs rivales y opciones de juego de la partida-, en un **entorno complejo** –reglas de juego bastante extensas sobre todo para el movimiento y el ataque con armas y cierta aleatoriedad con las tiradas de dados-, el problema está bien diferenciado en **distintas etapas** –movimiento, reacción, ataque con armas, ataque físico y fin de turno- y se requiere un respuesta acotada en el tiempo -10 segundos para cada etapa- todo lo descrito hace que el problema sea conveniente afrontarlo desde el punto de vista de los agentes inteligentes.

# 4. Propuesta y justificación de un modelo teórico

Un agente inteligente es una entidad capaz de **percibir** su entorno mediante sensores y **actuar** **racionalmente** sobre dicho entorno mediante actuadores.

En nuestro caso el agente es el ejecutable que programemos, el entorno está compuesto por todos los elementos de BattleTech anteriormente descritos y que son generados y manejados por el simulador de BattleTech. Nuestro agente percibirá el entorno a través de los ficheros de texto .sbt que el entorno genere sobre el estado del juego y actuará sobre el entorno a través de fichero accion.sbt que generará de una manera racional, entendiendo por racionales que siempre buscará maximizar su beneficio en el objetivo de la partida: ser el último mech con vida del tablero.

El entorno en el que nos movemos tiene una serie de características que determinarán qué tipo de agente tendremos que programas. Veamos cuales son:

**1. Accesible**: el entorno es totalmente accesible, mediante los ficheros de información del mapa, mechs y opciones de juego tenemos acceso a toda la información necesaria sobre el entorno.

**2. No determinista**: en el juego hay tiradas de dados que incluyen componente aleatorio que hace que el entorno no sea determinista. Imaginemos que tenemos un arma con munición y que la disparamos hacia un mech sobre el que tenemos LDV y llegamos a pesar de todo lo descrito no es seguro que le demos pues antes deberemos tirar unos dados y conseguir cierto número.

**3. Estático**: el entorno está dividido en turnos y mientras que nuestro mech piensa que realizar en dicho turno el entorno no cambia, solo lo hace al final de cada turno.

**4. Episódico**: el comportamiento de nuestro agente puede ser dividido en etapas, en concreto en 5 que se corresponden con las mismas fases que posee un turno en el juego.

5. **Discreto**: el entorno posee un número limitado de estados, el mapa es finito, el número de mechs es finito y los estados de los distintos elementos que conforman el entorno están determinados siempre por valores discretos.

Dada la **naturaleza episódica del problema** utilizaremos una arquitectura horizontal, el agente estará dividido en 5 módulos: movimiento, reacción, ataque con armas, ataque físico y fin de turno, todos ellos tendrán acceso a la percepción del entorno que tengamos y una función de selección determinará que modulo está en funcionamiento en cada momento.

Nuestro agente tomará las decisiones siguiendo una **arquitectura deliberativa** en cada etapa el agente pensará que acciones ha de llevar a cabo para cumplir su objetivo final.

No poseerá ningún tipo de creencia que influya en su comportamiento. Todas las decisiones que tome estarán basadas en la percepción objetiva del entorno.

No habrá comunicación con otros agentes toda la información será extraída de los ficheros generados por el simulador, no habrá paso de mensajes con otros agentes.

Además de todas las propiedades ya citadas nuestro agente cumplirá con las características implícitas que se le suponen a todo agente:

-**Autónomo**, funcionará sin la necesidad de un humano.

-**Reactivo**, podrá responder a los cambios que se den en el entorno.

# 5. Descripción y justificación de la solución

# 5.1. Representando el entorno

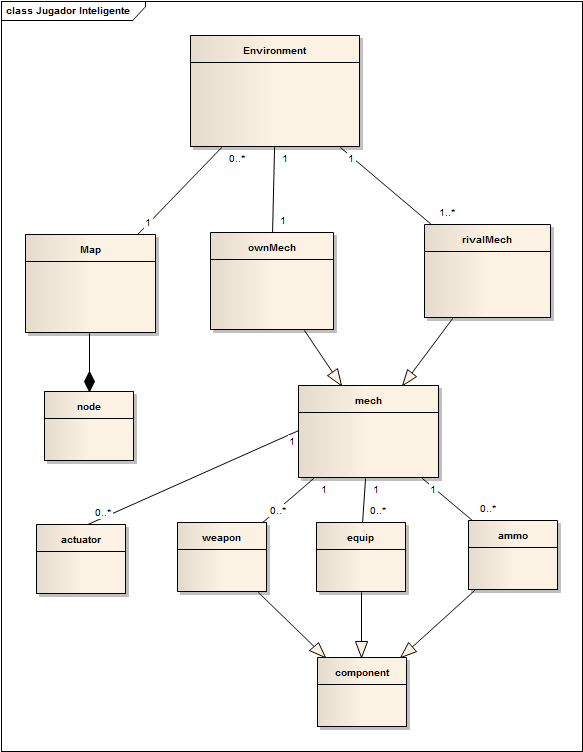
El primer problema al que nos enfrentamos es el de **cómo representar internamente toda la información** que el simulador nos proporciona. En los ficheros está toda la información necesaria pero no usaremos toda y por supuesto no está organizada de la mejor forma posible para los cálculos que debamos realizar.

Distinguiremos pues **dos partes en nuestro agente**, una de ellas que será la encargada de **procesar y estructurar la información** que nos cede el entorno y otra parte que se encargara de **deliberar cual es la mejor acción posible** y que será nuestro agente propiamente dicho.

Centrémonos primero en cómo estructurar la información del entorno para que sea de fácil acceso por los módulos de decisión en cada etapa.

El simulador, a través de los ficheros “mapaJ*x*.stb”, “mechsJ*x*.stb”, “configJ*x*.stb” y “defMech*i*”… “defMech*n*” dónde x = nº del jugador inteligente, n = número total de mechs y 0 <= i < n, nos muestra toda la información necesaria del entorno.

**Dicha información la he estructurados de la siguiente forma:**



Una clase entorno compuesta por un mapa, un mech propio y una serie de mechs rivales. El mapa estará compuesto por elementos de la clase nodo que contendrán toda la información relativa a un hexágono del mapa, las clases mech propio y mech rival heredan de la clase abstracta mech y todo mech tendrá un equipo dividido en actuadores y componentes a su vez un componente podrá ser una arma, munición o una pieza de equipo.

Cada una de las fases tendrá su propia clase que será la encargada de determinar qué acciones llevar a cabo y de escribir el fichero “accionJ*x*.sbt” correspondiente.

## 5.2. La fase de movimiento

La fase de movimiento, aunque compleja, se resuelve dando respuesta a dos preguntas:

**1. ¿Cuál es la mejor casilla a la que movernos?**

**2. ¿Cómo es la mejor manera de movernos hasta esa casilla?**

Aunque el orden lógico de responder dichas preguntas es responder primero el dónde y luego el cómo, supongamos que el dónde está resuelto y respondamos el cómo, que aun siendo quizá más técnico que el dónde es más simple.

### 5.2.1. ¿Cómo moverse?

Dependerá del método de desplazamiento que elijamos.

**Saltando**

Trivial –previamente respondiendo a la pregunta de dónde movernos en se han calculado que casillas se pueden saltar y cuál es la mejor-, tan solo hay que decirle al mech que salte a esa casilla y el lado que queremos que encare cuando aterrice.

**Andando/Corriendo**

Tres son los objetivos:

1. Que el camino sea válido.

2. Que el camino pueda realizarse en dicho turno.

3. Que el camino sea el mejor que existe para ir de A a B.

Para conseguir todo eso utilizaremos el **algoritmo A\***.

Los elementos principales que necesitamos para implementar A\* son -siendo *start* el origen, *goal* el destino y *node* una casilla cualquiera del mapa-:

1. Una función heurística h(*start*, *goal*) que estime el coste

2. Una función g(*start*, *goal*) que nos de el coste real.

3. Una función neigs(*node*) que nos devuelva los nodos vecinos accesibles de x.

Además de dichas función también definiremos las siguientes variables:

1. **Lista de nodos abiertos**, nodos que van a ser evaluados

2. **Lista de nodos cerrados**, nodos que ya han sido evaluados, entendiéndose por evaluados que ya se calculado cual es el coste de ir de todos sus vecinos al destino pasando por dicho nodo

3. g[node], diccionario dónde las entradas son los nodos del mapa y el valor es el coste acumulado que cuesta llegar a ese nodo desde *start*.

4. f[node], diccionario dónde las entradas son los nodos del mapa y el valor es g[node]+h(node, goal)

5. came\_from[node], diccionario donde las entradas son los nodos del mapa y el valor es el nodo vecino a node desde el que se llega a él en el camino actual calculado.

Una vez tengamos todos esos elementos implementar A\* no es difícil, la implementación concreta puede ser encontrada en la clase movement de mi práctica, la idea general del algoritmo es:

Mientras haya nodos en la lista abierta escoger aquel nodo cuyo f(node) sea menor, si dicho node es la meta ya hemos acabado.

Si no entonces lo quitamos de la lista abierta y lo incluimos en la cerrada y calculamos sus vecinos, aquellos que no estén en la lista cerrada los insertamos en la lista abierta y para cada uno de ellos calculamos su g() llegando a él desde el nodo actual, si dicha g() es menor a la que ya tenía entonces la actualizamos.

Si la lista abierta se queda sin nodos y no hemos llegado al destino entonces no existe camino de *start* a *goal*, si el algoritmo consigue terminar se puede obtener el camino a partir de la variable *came\_from[]*

### 5.2.2. ¿Dónde moverse?

Si no tenemos armas la mejor casilla será aquella de las casillas que podamos llegar en el turno actual, ya sea andando o saltando.

Si tenemos armas la mejor casilla será aquella que **nos proporcione LDV, cobertura y a distancia suficiente como para disparar al enemigo**, de no existir dicha casilla nos quedaremos con al que nos ofrezca LDV y este a una distancia suficiente como para disparar al enemigo. Si ninguna de esas condiciones de cumple, seguiremos la misma estrategia que cuando no tenemos armas, **ir a la casilla que más nos acerque a nuestro enemigo**.

Como la fase de ataque con armas no ha sido completamente implementada, en nuestra clase movimiento siempre asumimos que nºArmas = 0, por lo tanto todo mech se comporta como un mech sin armas. Cuando dicha fase quede totalmente implementada no se hará esa suposición.

### 5.2.3. Autocrítica a la solución adoptada

Sobre el cómo no tengo ninguna mejora pensada, de todo lo que he leído el algoritmo A\* me parece sin duda el más apropiado para resolver el problema de que camino llevar del punto A al punto B.

El dónde moverme me gustaría mejorarlo, sobre todo por el hecho de que no tengo en cuenta el estado de mi mech y del mech enemigo, por ejemplo si estoy peor que el otro mech sería bueno tomar una estrategia más defensiva priorizando buscar áreas del mapa de difícil acceso para el enemigo que me den ventaja posicional más allá de tener LDV y cobertura.

## 5.3. La fase de reacción

Se pueden realizar tres acciones:

1. Quedarse igual.

2, 3. Girar el torso a la derecha o izquierda.

Calculamos cual es el mejor lado de nuestro nodo para ir desde nuestro nodo al nodo enemigo.

Si ya estamos en el mejor lado no hacemos nada.

Si el mejor lado queda a nuestra derecha/izquierda giramos el torso a la derecha/izquierda.

## 5.4. La fase de ataque con armas

Como ya he comentado esta fase no se encuentra totalmente implementada, en concreto me he quedado atascado en calcular los ángulos de tiro, es decir dado un enemigo saber con qué armas –las del brazo derecho/izquierdo, torso derecho/izquierdo, pierna derecha/izquierda o traseras- le puedo disparar según su posición con respecto a mí.

### 5.4.1. ¿A quién atacar?

Si tenemos algún mech en nuestro rango **atacaremos al más cercano** pues será al que podamos atacar con más armas y será más fácil de impactarle.

### 5.4.2. ¿Cómo atacar?

Una vez tenemos nuestro objetivo primero debemos saber cuáles son las a**rmas que podemos usar contra él**. Para poder usar un arma contra un mech se debe de cumplir:

1. Tener LDV.

2. Que esté dentro de la distancia máxima de alcance del arma.

3. Que el arma esté operativa.

4. Que el arma este en una parte del cuerpo que tenga Angulo con el mech rival, por ejemplo si lo tenemos detrás no podemos dispararle con un arma frontal. Es aquí, por falta de tiempo, dónde me he quedado atascado a la hora de implementarlo.

5. Para impactar antes se necesitará lanzar un dado y que lo que salga sea mayor o igual a lo requerido. Si lo requerido es mayor de 12 sabemos que nunca le dará, pues la tirada es de dos dados de 6 caras. Si la tirada es cercana a 12 puede que tampoco nos interese tirar.

Una vez tenemos objetivo y sabemos que armas podemos usar solo queda aplicar el **algoritmo de la mochila**. La capacidad de la mochila vendrá dada por el calor que podamos generar en dicho turno y el beneficio de cada arma vendrá dado por la razón entre su poder destructivo y el calor que generé al usarse.

## 5.5. La fase de ataques físicos

En esta fase la estrategia será infligir el **máximo de daño posible** teniendo en cuenta las restricciones de cuando se puede atacar.

**No tenemos en cuenta el uso de garrotes**, solo usamos nuestros brazo y piernas. El máximo de acciones permitidas en un mismo turno son: puñetazo derecho, puñetazo izquierdo y patada (derecha o izquierda) ya que no pueden usarse las dos piernas a la vez.

Si el mech enemigo no está en un hexágono vecino o su nivel es superior o inferior al nuestro en dos unidades entonces no podemos hacerle nada-ni el a nosotros-.

**Podremos usar el brazo derecho si:**

1. Tenemos hombro derecho.

2. La diferencia de altura es 0 o 1.

3. Tenemos al enemigo justo en frente o a nuestra derecha según la dirección de nuestro torso y no nuestras piernas.

4. No hemos usado armas del brazo derecho.

**Podremos usar el brazo izquierdo si:**

1. Tenemos hombro izquierdo.

2. La diferencia de altura es 0 o 1.

3. Tenemos al enemigo justo en frente o a nuestra izquierda según la dirección de nuestro torso y no nuestras piernas.

4. No hemos usado armas del brazo izquierdo.

**Podremos usar nuestra pierna derecha si:**

1. Tenemos ambas caderas.

2. La diferencia de altura es 0 o -1

3. Tenemos al enemigo justo enfrente o a nuestra izquierda o derecha según la dirección de nuestras piernas.

4. No hemos usado armas de la pierna derecha.

5. No hemos usado nuestra otra pierna

**Podremos usar nuestra pierna izquierda si:**

1. Tenemos ambas caderas.

2. La diferencia de altura es 0 o -1

3. Tenemos al enemigo justo enfrente o a nuestra izquierda o derecha según la dirección de nuestras piernas.

4. No hemos usado armas de la pierna izquierda.

5. No hemos usado nuestra otra pierna.

Una vez determinado que extremidades podemos hacer uso queda resuelto el problema pues la respuesta será utilizar todas las utilizables.

## 5.6. La fase de final de turno

Esta fase, al igual que fase de ataque con armas, tampoco se encuentra implementada.

Se puede realizar dos operaciones apagar/encender radiadores y expulsar munición.

No veo ningún beneficio en apagar radiadores por lo tanto nunca se haría, al no apagarlo no hay que encenderlos por lo tanto de la opción de apagar/encender radiador no hay que preocuparse.

La munición a ser expulsada serán aquellas cuyas armas que las usan ya no están operativas. Será lo que implemente cuando termine la práctica.

# 6. Conclusiones

0. He subestimado la dificultad de la práctica, le he dedicado un mes entero, del cual dos semanas han sido enteras y no ha sido suficiente, si bien es verdad que tres de las fases ya están terminadas y de las otras dos una es muy sencilla, aún queda mucha tela por cortar para que lo implementado sea lo mismo que lo diseñado.

1. Ha sido la práctica de la carrera que más tiempo me ha llevado comprender lo que se pide, si bien el problema en sí, programar un agente inteligente, es fácil de comprender; el llegar a saberse todas las reglas del entorno BattleTech es una tarea ardua. Cuando uno cree saber todo lo que tiene que saber para realizar la práctica encuentra una regla nueva que le hace revisitar el código que ya creía funcional y terminado.

2. A pesar de faltar la fase de ataque con armas creo que de los 7 puntos, sobre los que se evalúa la práctica, 3 de ellos los pasa correctamente. En concreto: movimiento de largo recorrido, saltos y ataque físico. Además la gestión de calor también ha sido tenida en cuenta, por lo tanto el punto de gestión de calor si el mech llega a entablar combate también lo pasará, pero dudo que sin usar sus armas, solo atacando físicamente, consiga llegar al otro mech sin morir en el intento.

3. Los agentes inteligentes episódicos si son episódicos son muy buenos para que muchas personas trabajen sobre ellos, al estar las etapas tan bien diferenciadas distintas personas pueden ocuparse de las distintas etapas. En esta práctica por ejemplo cada una de las distintas fases podría haber sido fácilmente implementarle por personas distintas y que luego todo funcionase.

4. Si suspendo ahora en septiembre espero que la práctica se pueda volver a presentar en diciembre, habiéndola terminado ya entera, no quiero que todo el trabajo sea en vano.

7. Bibliografía

[1] ICO - Guión de práctica

[2] Manual de Usuario de BattleTech

[3] Manual de Reglas de BattleTech

[4] <http://bg.battletech.com/forums/>

[5] <http://www.policyalmanac.org/games/aStarTutorial.htm>

[6] <http://www.webwargaming.org/lineofsight.shtml>

[7] <http://www.programminglogic.com/knapsack-problem-dynamic-programming-algorithm/>