Asignatura: Ingeniería del Conocimiento

Curso 2011/12

Curso: 3º Ingeniería **Grupo de Teoría: A**

Grupo de prácticas: Lunes 16:00-18:00

Nombre del ejecutable: 53590723

Datos de los Alumnos: (ordenar alfabéticamente por apellidos)**.**

Apellidos: Costela Sanmiguel

Nombre: Ángel

D.N.I.: 75170462E

e-mail: acosté@correo.ugr.es

Apellidos: Medina Godoy

Nombre: David

D.N.I.: 53590723X

e-mail: asce88@gmail.com

Índice

1. Introducción

El objetivo de la práctica consiste en la elaboración de un jugador inteligente de BattleTech. BattleTech es un juego de estrategia donde varios robots gigantes o mejor dicho Mechs se enfrentan.

De esta forma, el problema que se nos presenta es el de diseñar e implementar un programa capaz de jugar a este juego eligiendo la jugada que le sea más favorable, como si de una partida de ajedrez se tratara, que deberá escoger entre una serie de movimientos cual es la mejor siguiendo unos criterios. Aunque en el "BattleTech" también debemos de tener en cuenta otro factor que es el azar, ya que se trata de un juego de rol. En cierto modo es como enseñarle a la máquina a jugar a dicho juego de la mejor manera posible.

Esta práctica está orientada a la resolución del problema presentado mediante la construcción de un agente, usando los conocimientos aprendido en la asignatura Ingeniería del Conocimiento de 3º de Ingeniería Informática. Nuestro programa deberá interactuar con un entorno de simulación de BattleTech mediante la lectura y escritura de ficheros.

En el presente documento realizaremos una visión general a los conceptos teóricos aplicados en nuestra práctica, así como el concepto de agente y los tipos de estos. Las abstracciones utilizadas y finalmente nuestro diseño.

2. Descripción del problema

2.1 Descripción general

El juego BattleTech es un juego de estrategia por turnos en ambientado en un universo futurista ficticio, desarrollado por FASA Corporation, en el que unas máquinas con forma humanoide y equipadas con armas, llamadas Mechs o BattleMechs, intentan aniquilarse entre ellas. El juego se desarrolla en un tablero formado por casillas hexagonales, cuyas casillas tienen diferentes características. Como en cualquier juego de rol, como es este, el azar influye en las jugadas, ya que para impactar a un jugador, decidir el orden del turno, determinar caídas, etc., se hace una tirada de dos dados de 6 (2D6). Ésta es sólo una versión reducida del juego, ya que en el original, también hay unidades de infantería y distintos vehículos, aunque en este proyecto sólo se tendrán en cuenta el manejo de los Mechs.

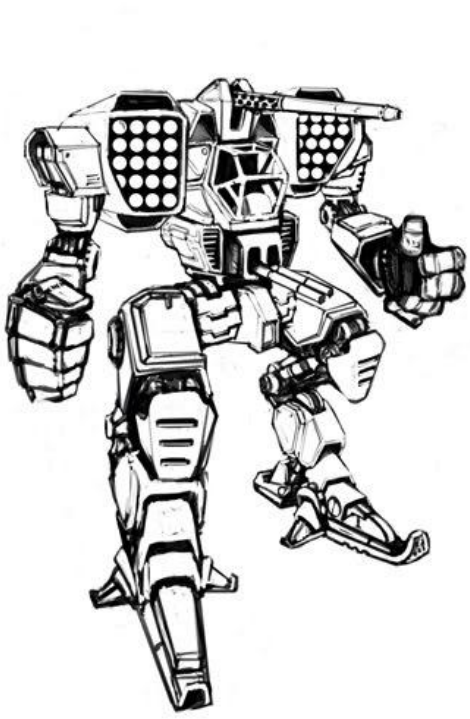


Figura 1: BattleMech

El sistema a implementar debe ser capaz de decidir y realizar los movimientos y acciones que vea oportunos en cada momento. Para ejecutar dichas acciones se creará un fichero en un formato determinado para que el entorno sea capaz de leerlo y que el Mech realice las acciones que se indiquen. Y para que el sistema sepa el estado actual del juego, leerá unos archivos que genera el entorno y los guardará en memoria, para su futura interpretación.

.

Cabe destacar en este juego que el ganador de la iniciativa es el último en llevar a cabo sus movimientos, obteniendo así la ventaja de reaccionar a los movimientos y acciones de sus rivales. Además uno de los factores más determinantes y que más nos limitará las acciones de un BattleTech será el calor. Pues si una unidad se encuentra sobrecalentada tendrá penalizaciones al movimiento, la

precisión al disparar o incluso podría sufrir daño si explota la munición almacenada a causa de la temperatura

2.2 Descripción del juego

Como se ha dicho anteriormente el problema consiste en crear una I.A. para el juego de estrategia BattleTech, el cual consiste en que los Mechs deben luchar entre ellos, y cuyo objetivo es ser el último que quede en pie, aunque antes de centrarnos a fondo en qué hace el sistema, habrá que explicar los diferentes elementos que componen el juego:

- **Tablero**: El tablero está formado por casillas hexagonales. Dichas casillas están a alturas diferentes, son de terrenos distintos (pantano, agua, llano…) y tienen unos determinados elementos en el terreno, como bosques, edificios, escombros…, todos estos elementos influyen en la cantidad de casillas que puede moverse el Mech y en la probabilidad de recibir un disparo.

- **Mechs**: Los Mechs o BattleMech son máquinas gigantes con forma humanoide tripuladas por humanos, a los que denominamos BattleWarriors. Cada Mech se compone de varios slots o ranuras en las diferentes partes del cuerpo (piernas, brazos, torso y cabeza). En cada slot se introduce un tipo de arma u objeto que puede usar el mech. Además el mech tiene dos partes de armadura, la externa y la interna. Si la armadura interna del torso central o de la cabeza es destruida, el mech queda inoperativo para el resto de la partida. Dependiendo de la armadura y de las armas, cada mech tendrá un tonelaje, y se dividirá en ligeros, medios, pesados y de asalto. También poseen un calor interno, que puede penalizar a las acciones del mech si es muy elevado.

- **Dinámica**: El juego se va desarrollando por turnos, y cada turno se divide en seis fases, que en orden son:

- Movimiento, el Mech puede situarse en otra casilla del mapa, ya sea andando, corriendo o saltando, usando sus puntos movimiento.

- Reacción, el Mech puede girar el torso a la derecha o la izquierda, para poder disparar y recibir los diparos desde otro ángulo diferente a donde se ha quedado mirando en la fase de Movimiento
- Ataque con armas, el Mech elige las armas que tiene equipadas para atacar a un Mech rival o una casilla del mapa.
- Ataque físico, el Mech ataca a otro Mech o una casilla usando sus brazos o sus piernas.
- Fase de temperatura, se calcula el calor disipado por el Mech y se le resta al calor acumulado en las fases anteriores, lo que produce un determinado nivel de calor en el Mech que se va sumando.
- Fase Final, el Mech puede soltar munición y/o apagar o encender radiadores.

En un turno cada Mech realiza la misma fase antes de que se pase a la fase siguiente, es decir, todos los Mechs realizan la fase de Movimiento antes de que se pase a la fase de Reacción, y así sucesivamente. La realización de cada fase se hace de forma aleatoria en cada turno, se hace una tirada de dados, y el que mayor puntuación saque es el primero que realiza la fase. Una vez todos los Mechs han realizado la fase final, se pasa al turno siguiente.

- **Entorno**: Para simular una partida de tablero de BattleTech se usa el simulador realizado por Helio Huete López de las Huertas. En dicho simulador se muestra gráficamente el mapa y la posición de los mechs en él, asimismo la ficha con los datos de cada mech en juego, y un log donde se va informando sobre cada suceso de la partida.

- **Comunicación**: En cada fase, se llama a nuestro programa para que genere un archivo de acción. Como hay varias fases y siempre se llama al mismo programa, se le pasa como argumento al programa el número del Mech que queremos que realice la acción y la fase en la que se encuentra. De esta manera según el argumento que se le pase por entrada devolverá un conjunto de acciones diferentes. Estas acciones dependen del estado actual del juego. Para saber el estado del juego, el propio entorno crea unos archivos de las diferentes partes (mapa, Mechs…), y nuestro programa lee estos

archivos y almacena los datos en memoria, para finalmente operar con ellos y producir una respuesta.

.

2.3 Objetivo del sistema

El objetivo del sistema es la destrucción de todos los Mechs enemigos, sin ser destruido en la batalla. Para este fin, el sistema debe comunicarse con el entorno de juego a partir de varios archivos de texto plano con un formato determinado, construyendo un archivo diferente dependiendo de la fase en la que se encuentre.

El sistema no debe infringir las reglas del juego, y deberá ser capaz de funcionar para cualquier Mech, aprovechando las ventajas que pueda tener, y en cualquier mapa, por lo que tendrá en cuenta todas los tipos de terreno y objetos en el mismo.

Como resultado de tantas variables cada Mech debe seguir una estrategia de juego diferente, dependiendo además de su estado y del estado del mapa. Dichas estrategias las explicaremos más adelante.

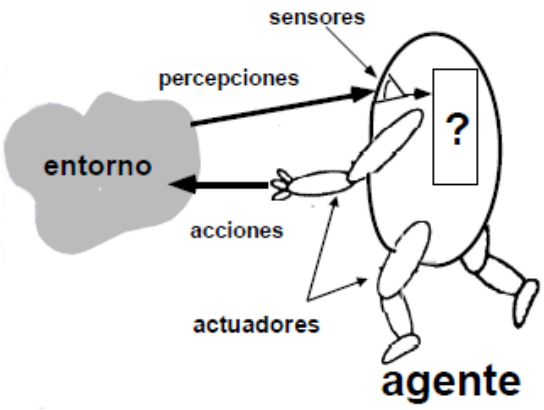
3. Abstracción del problema

3.1 Abstracción del jugador

La resolución del problema que nos planteamos en la presente práctica encaja perfectamente en el marco de los llamados agentes inteligentes. Si tomamos prestada la definición de agente del libro "Inteligencia Artificial, un enfoque moderno, Stuart J. Russell y Peter Norvig":

"Un agente es cualquier cosa capaz de percibir su medioambiente con la ayuda de sensores y actuar en ese medio utilizando actuadores"

Vemos que esta definición es inmediatamente aplicable al problema que nos concierne, debemos realizar una entidad software capaz de percibir el medio en el que está, el cual no es otro que el propio tablero de juego junto con la situación y el estado de los otros jugadores que participan en la partida, y actuar en consecuencia a esas percepciones.



En este caso los sensores citados en la definición podemos tomarlos como los ficheros que genera el entorno del juego, ya que son estos los que nos ofrecen información sobre el

estado actual de la partida. Debemos aclarar aquí que en general los agentes toman sus decisiones basándose en el historial de percepciones recibidas, nuestro agente (a excepción de algunas percepciones de pequeño calibre que se tratarán más tarde) sólo tiene en cuenta las últimas percepciones recibidas del ambiente.

Los actuadores también mencionados serían los ficheros que debe generar nuestro programa y gracias a los actuales interactuamos con el entorno, el cual aplica nuestras decisiones al entorno de juego, produciéndose así cambios en el medio.

Además para justificar aún más este punto de vista, veamos como la entidad que nos disponemos a exponer cumple con las propiedades fundamentales de cualquier agente:

- **Reactivo**: Ante unos impulsos de entrada (el estado del mapa, nuestro estado y el de los demás jugadores) nuestro jugador emitirá una secuencia de pasos a seguir a la salida, acordes con las reglas del juego, aunque esta salida puede ser eventualmente no hacer nada.
- **Autónomo**: Si tenemos en cuenta la abstracción realizada anteriormente, podemos considerar que el agente tiene sus propios sensores y por lo tanto que no necesita de intervención externa para llevar a cabo su tarea.
- **Orientado por metas**: Nuestro software tiene un objetivo claro, sobrevivir a los demás jugadores e intentar que ellos no sobrevivan y toma las decisiones acorde a esta meta.
- **Temporalmente continuo**: En este caso podríamos tener una discusión sobre si nuestro software cumple o no esta propiedad. Si bien es cierto que nuestro software es llamado por el entorno en distintos "turnos", entre los cuales no se está ejecutando, como abstracción podríamos tomar que es continuo y que los estímulos se reciben en el momento de cada "turno".

Para ser aún más precisos en la definición de la abstracción que realizamos con el software, diremos que nuestro agente es un agente racional, que podríamos definir como:

"Un agente racional es aquel que ante cualquier posible secuencia de percepciones debe emprender aquella acción que supuestamente maximice su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en el conocimiento que el agente mantiene almacenado"

Si bien podemos definir a nuestro agente como del anterior tipo, debemos recordar que nuestro agente mantiene poco conocimiento almacenado de sus estados anteriores. Esto se tratará más adelante. Además nuestro agente tendrá poca autonomía, ya que se apoyará siempre en el conocimiento aportado en el momento de la implementación, sin que sea capaz de aprender.

3.2 Abstracción del medio

Como medio tomaremos el tablero de juego con cada una de sus posiciones y sus correspondientes atributos además del estado de los otros jugadores.

Este ambiente posee las siguientes características encajadas dentro de la teoría de agentes:

- Podríamos decir que el ambiente es **semi-accesible** pues si bien conocemos muchos datos acerca de él como para tomar decisiones en cada etapa, también es cierto que hay algunas informaciones que nos podrían resultar importantes para refinar la inteligencia de nuestro agente y que no se nos proporcionan en los ficheros que genera el entorno.
- **Episódico**, tal y como se presentan las acciones que debe realizar nuestro agente el entorno se divide en episodios independientes entre sí.
- Es un entorno **no-determinista** ya que la inclusión del azar al tirar los dados hace que no siempre las acciones que decidimos realizar se produzcan en el juego.
- El ambiente podemos definirlo como **estático**, ya que hay ocasiones en las que se producirán cambios mientras nuestro agente piensa, pensemos por ejemplo en el caso de que algún mech se mueva antes de nosotros en la fase de movimiento, nosotros no podríamos tener en cuenta ese movimiento.
- Además será un ambiente **discreto**, ya que tanto el número de percepciones (los datos obtenidos de los ficheros generados por el entorno) como las acciones a realizar (los posibles ficheros escritos por nuestro programa) son finitas.

El ambiente será abstraído como un grafo ponderado, representando cada nodo una posible posición del Mech en el tablero, tal como se puede observar en la figura 3.1. El peso de las aristas se determinará en función del coste de pasar de un nodo a otro, es decir, de una posición a otra, además de incluir otros modificadores, como por ejemplo la posible presencia de fuego o humo, que modificarán el peso de las aristas correspondientes a esos nodos.

Se elige esta abstracción por el hecho de que al trabajar con grafos podemos disponer de varios algoritmos eficientes ya creados y de dominio público.

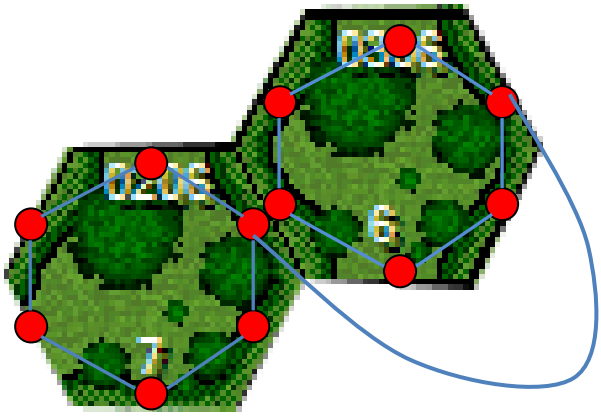


Figura 3.1 Abstracción del tablero

En cuanto a la abstracción de los demás jugadores, nos basaremos en una estructura de datos que contendrá el estado (puntos de blindaje, armas disponibles, etc) de los demás mechs.

Esta estructura la consultaremos para obtener información valiosa que nos ayude en las distintas fases del juego.

4. Propuesta del Modelo Teórico

En este apartado echaremos un pequeño vistazo a los posibles modelos teóricos de los que nos provee la teoría de agentes, observándolos con poco detalle y realizando finalmente una propuesta para la implementación del jugador inteligente la cual se justificará en el siguiente apartado.

4.1 Arquitecturas de agentes.

Dentro del campo de la ingeniería del conocimiento disponemos de un amplio abanico de opciones para elegir el tipo de agente que solucione nuestro problema.
Sin entrar en muchos detalles podemos clasificar las distintas arquitecturas atendiendo a dos criterios distintos:

- Punto de vista topológico
- Puntos de vista del conocimiento empleado

4.2 Agentes por clasificación topológica (niveles de abstracción).

En esta sección haremos una liviana introducción a los distintos tipos de agentes basándonos en su topología. Podemos distinguir tres tipos de arquitectura esencialmente:

- Arquitectura horizontal
- Arquitectura vertical

4.2.1 Arquitectura horizontal

Es un tipo de arquitectura en capas en la que cada capa está directamente conectada con los sensores y los actuadores. Se utiliza una función mediadora para elegir qué capa actúa en cada momento.

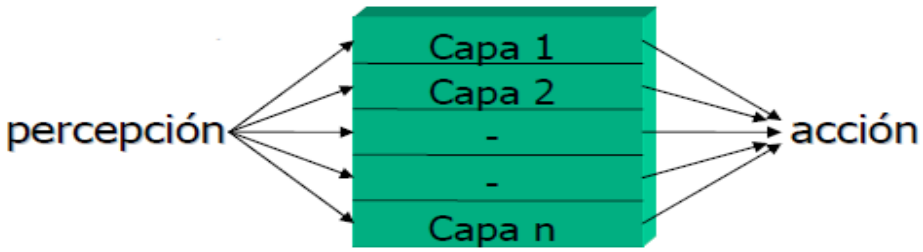


Figura 4.1 Esquema de Arquitectura horizontal

4.2.2 Arquitectura vertical

En esta arquitectura basada en capas únicamente la capa base está conectada con los sensores y/o los actuadores. Cada capa realiza una función para la que es necesaria que se haya realizado la función de la capa anterior.

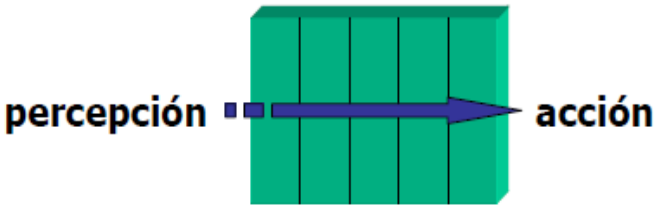


Figura 4.2 Esquema de Arquitectura vertical

4.3 Tipos de agentes atendiendo al tipo de conocimiento empleado

Según este punto de vista podemos dividir a los agentes en 3 tipos:

- Agentes reactivos
- Agentes deliberativos
- Agentes híbridos

4.3.1 Agentes reactivos

En este tipo de arquitectura el agente se construye siguiendo un modelo estímulo/respuesta. El agente actúa ante los cambios en el mundo sin seguir una planificación concreta, sólo realizando la acción que más conveniente le parece en cada momento. Este tipo de agentes no tiene una representación simbólica del mundo ni emplean un razonamiento simbólico complejo, por lo que suelen ser bastante eficientes para tareas que no exigen una planificación o cuya planificación sería demasiado compleja.

Como inconvenientes de esta arquitectura podemos citar que necesita un conocimiento completo del mundo, lo cual no es siempre posible y que un pequeño desconocimiento puede tener muy malas consecuencias para la actuación del agente.

4.3.2 Agentes deliberativos

En este tipo de arquitectura el agente se construye empleando un modelo búsqueda/planificación. El agente tiene un estado inicial, un conjunto de planes y un estado final (que podríamos tomar como el objetivo a cumplir). Dado todo esto el agente planifica las acciones a realizar para conseguir alcanzar el objetivo.

Este tipo de arquitecturas presenta una mayor rigidez y complejidad, siendo así menos eficientes en tiempo. Frente a los agentes reactivos proponen un modelo mucho menos drástico en el cambio de actitud, ya que siguen un plan preestablecido por ellos mismos.

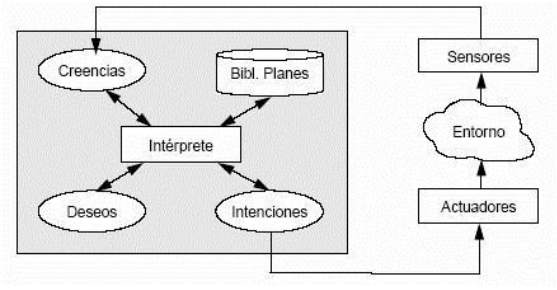


Figura 4.3 Esquema de Arquitectura deliberativa

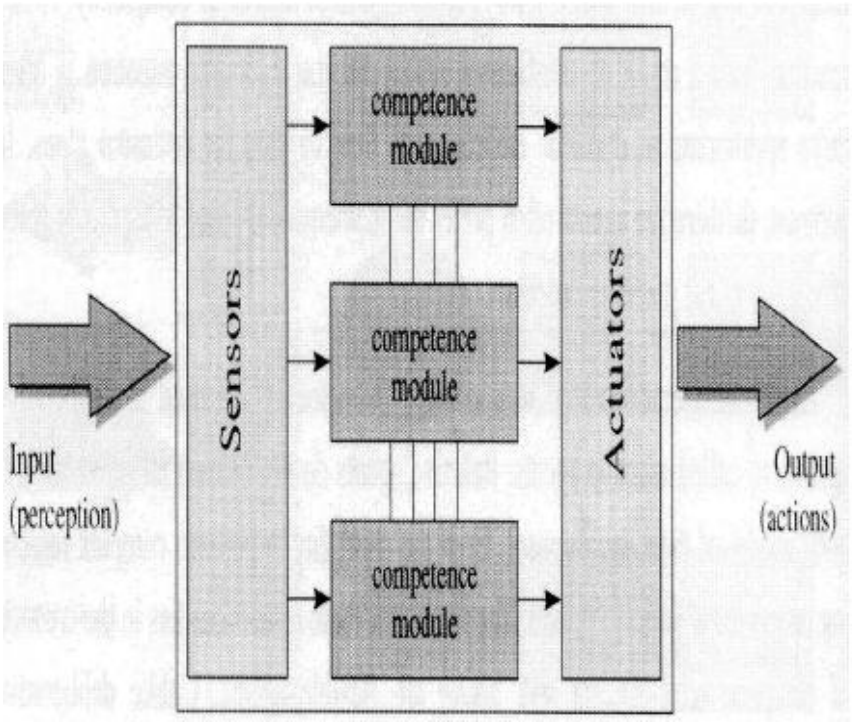


Figura 4.4 Esquema de Arquitectura reactiva

4.3.3 Arquitecturas híbridas

Con estas arquitecturas se pretende que el agente combine aspectos reactivos con deliberativos. Las acciones que no necesitan deliberación las procesan los módulos reactivos mientras que los deliberativos determinan que acciones deben realizarse para satisfacer objetivos locales y cooperativos más complejos.

4.4 Conclusión y propuesta

Una vez vistas a grandes rasgos las arquitecturas más comunes de agentes, para esta práctica hemos decidió usar un **agente reactivo y horizontal**, además de usar una arquitectura por **subsunción**.

5. Justificación del modelo teórico

En esta sección trataremos de justificar la elección del tipo de agente realizada en la anterior, aportando razones que a nuestro entender son suficientes para entenderla.

5.1 Aspectos horizontales del agente.

El hecho de que nuestro agente deba ser horizontal es evidente, ya que el juego está divido en fases independientes y para cada una de ellas se usa un módulo distinto del programa. Podríamos dividir las capas en:

- Fase de Movimiento
- Fase de Reacción
- Fase de Ataque con Armas
- Fase de Ataques Físicos
- Fase de fin de turno

Como función mediadora podríamos tomar al programa principal en sí, que elige el módulo que es llamado dependiendo de los parámetros emitidos por el entorno en la llamada.

5.2 Aspectos reactivos del agente

Es en este punto donde más dificultades tuvimos a la hora de elegir la arquitectura, en un principio pensamos en crear una híbrida usando componentes deliberativos para crear una planificación que nos diese buenos resultados en el juego.
Después de un análisis detallado de lo que esto suponía, decidimos descartar el componente deliberativo del agente por las siguientes razones:

- El juego está basado en parte en el azar, por lo tanto aunque creemos un plan éste está sujeto a demasiados factores que no podemos prever.
- La creación de un estado interno para el agente conllevaba algunas dificultades que no se compensaban con el rendimiento que obtendríamos realizando una planificación basada en las acciones anteriores.

- El tiempo que nos llevaría tanto la implementación de éste como las pruebas que deberíamos hacer para afinarlo y volverlo realmente funcional sobrepasaba nuestras expectativas.

Por lo tanto optamos por una arquitectura eminentemente reactiva, en la cual el agente responde con una secuencia de acciones basadas en el estado actual del mundo, intentando siempre cumplir los objetivos (sobrevivir y destruir a los demás jugadores), sin tener en cuenta cómo afectarán esas decisiones a largo plazo al desarrollo de la partida ni las acciones que hemos realizado anteriormente.

5.3 Diseño por subsunción

Este aspecto no fue explicado en el apartado 4 por no alargar excesivamente la exposición teórica de la teoría de agentes. Podemos decir que un diseño por subsunción consiste en incluir una serie de módulos, cada uno de los cuales se ocupa de una tarea concreta dentro del agente. En nuestro caso estas tareas serán las distintas fases y cada módulo corresponderá a una fase.

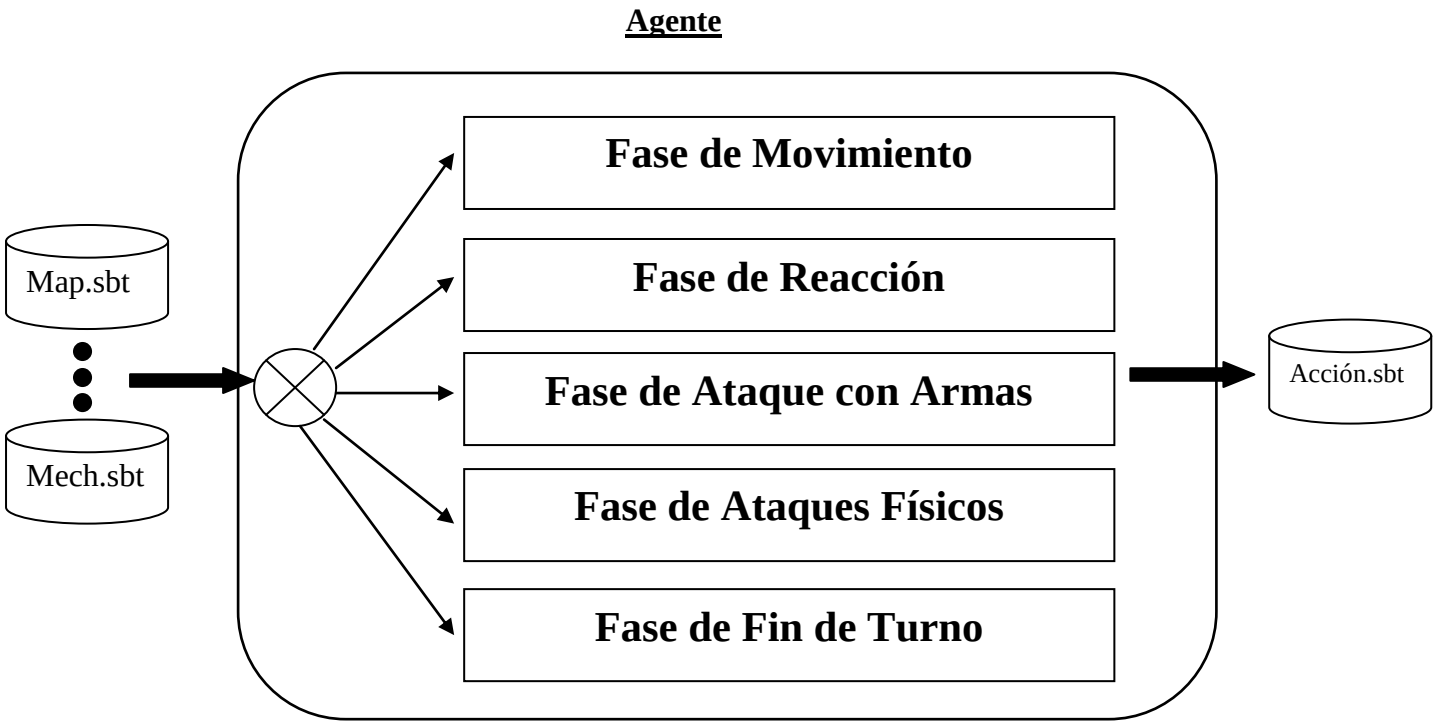


Figura 5.1 Esquema del agente inteligente

6. Descripción de la solución

Una vez explicado el modelo teórico ya tenemos una base con la que describir el sistema. Debido a que se trata de un agente reactivo que se rige principalmente por estados, cuyos estados son las diferentes fases, se explicará por separado cada uno de los posibles estados del agente.

Esta división es posible porque los diferentes estados son prácticamente independientes entre sí, salvo excepciones como el ataque con armas y el ataque físico. Pero aún así se ha diseñado el sistema de manera que aun teniendo alguna relación, la ejecución del sistema para un estado determinado tenga en cuenta, que existen otros estados que comparten información con él. De esto modo la ejecución del sistema para cada uno de los estados será lo más óptimo posible.

6.1 Movimiento

La fase de movimiento quizás sea la fase más compleja e importante del agente. Esto se debe a que debe explorar el mapa buscando cuáles son las posiciones más ventajosas a las que puede acceder el Mech, pero en un tiempo no muy elevado, lo que nos obliga a diseñar heurísticas que aunque no den la mejor solución, sí se acerquen mucho a la óptima.

6.1.1 Heurística de movimiento

Hay diversos factores que influyen en la heurística de movimiento. Estos son principalmente el mech objetivo, el modo de movimiento (andar, correr o saltar), la estrategia a seguir y finalmente la posición a la que nos desplazaremos. Hallaremos la ruta deberá seguir el mech para alcanzar el nodo objetivo (hexágono y orientación) aplicando el algoritmo A*.

6.1.2 Elegir mech objetivo

El primer paso que debemos realizar antes de comenzar a mover el mech, es saber dónde vamos. Necesitamos marcarnos una casilla como objetivo para que el mech pueda moverse hasta ella, el problema es saber qué casilla.

La estrategia del agente es situar al mech que maneja Y, en una posición privilegiada respecto a un mech enemigo K. La elección del mech se realiza usando la proximidad al mech Y y el peso del mech K. El mech K que elegimos es aquél cuyo tonelaje sea máximo y la distancia a Y sea menor o igual que la mínima distancia a todos los mechs más 2, es decir, suponiendo que la distancia mínima a un mech es d, pues aquel mech que esté a distancia menor que d+2 y cuyo tonelaje sea mayor de todos los que estén a dicha distancia.

Para calcular la distancia, se ha realizado una variación de la *distancia Euclídea* entre dos puntos para poder adecuarla al tablero hexagonal, y es la que se usará a lo largo de todo el programa. Dicha variación tiene en cuenta los dos puntos para aplicar algunas variaciones a la fórmula y obtener una aproximación bastante acertada.

El motivo de esta elección, es escoger de aquellos que tengamos más cercanos, el que mayor tonelaje tenga, porque será el que mayor problemas pueda causarnos, e interesa eliminarlo del juego lo antes posible. Además también es importante elegir al más poderoso por si tenemos que huir, ya que nuestro mech es muy débil, y así no perder el tiempo en atacar a otros mechs menos poderosos.

6.1.3 Estrategia de movimiento

Definimos tres estrategias:

- Ataque: Esta estrategia tiene como prioridad acercarse lo más posible al objetivo, a poder ser en una posición adyacente a su espalda.

- Defensa: Usando esta estrategia el mech se colocará a una distancia de seguridad del mech objetivo, buscando la LDV para poder atacarlo. Consiguiendo de esta forma una estrategia ofensiva/defensiva. El mech buscará no darle la espalda nunca al objetivo.

- Huida: Es una estrategia puramente defensiva. Buscará colocarse lo más lejos posible del mech objetivo enemigo, aún en sacrificio de darle la espalda, esto le permitirá alejarse una distancia mayor.

Escogeremos la estrategia de movimiento en función del peso de nuestro mech, el del mech objetivo y el estado en que se encuentre el mech. Los distintos casos son los siguientes:

```
Si enBuenEstado(mechJugador)
        estrategia = elegir_estrategia_por_peso();
en otro caso
        estrategia = HUIR;
```

En la estrategia por peso:

```
Si (pesoJugador >= 80) {//Asalto
   estrategia = ATACAR;
} en otro caso si (pesoJugador >= 60) {//Pesados
   //Atacamos siempre a no ser que nos enfrentemos a un mech de asalto
   si (pesoEnemigo >= 80) {
      estrategia = DEFENDER;
   } en otro caso {
      estrategia = ATACAR;
   }
} en otro caso si (pesoJugador >= 40) {//Medios
   si (pesoEnemigo < pesoJugador + 15)
      estrategia = ATACAR;
   en otro caso
      estrategia = DEFENDER;
} en otro caso si (pesoJugador >= 0) {//Ligeros
   //Huimos del enemigo si pesa 15 ton o + mas que nosotros.
   si (pesoEnemigo < pesoJugador + 15) {
      estrategia = ATACAR;
   } en otro caso {
      estrategia = DEFENDER;
   }
}
```

Por otra parte consideramos que un mech está en buen estado si:

- Temperatura < 13
- Blindaje de cabeza > 2
- Blindaje Torso central > 2
- Blindaje Pierna izquierda > 1
- Blindaje Pierna derecha > 1

6.1.4 Elección de la casilla objetivo

La elección de la casilla objetivo consiste en elegir una casilla que satisfaga determinados requisitos, normalmente de ataque o defensa, de un área determinada.

Estas áreas se basan en el concepto de *"anillo"*. Cada anillo parte de un hexágono del mapa, que podríamos considerar que es el centro de los anillos que vamos a calcular. Los anillos se dividen por niveles, según el grado de proximidad al centro, siendo el nivel 1 el más cercano al mismo.

Definamos de forma más formal lo que es un anillo. Una casilla está en el anillo i-ésimo, si el mínimo número de casillas por las que hay que pasar, contando la final, para ir desde el centro del anillo, hasta la casilla final es i. Entonces todas las casillas que cumplen la definición anterior conforman el anillo de nivel i.

Podemos definir todos los elementos que forma un anillo teniendo en cuenta la definición de adyacencia dada anteriormente, pero aplicándola a todos los elementos de un anillo, y la siguiente ecuación de recurrencia:

$$Anillo[\ -1\ ] = \{\}$$
$$Anillo[\ 0\ ] = \{centro\}$$
$$Anillo[\ i\ ] = \{x \in Adyacentes(\ anillo[\ i - 1\ ]\ ) \wedge x \notin Adyacentes(\ anillo[\ i - 2]\ )\ \}$$

Para ilustrar esta definición podemos observar la figura 5.1. En esta figura tenemos marcado en rojo a que anillo pertenece cada casilla. Como podemos observar las casillas contiguas a la del mech están a nivel 1, esto se debe porque podemos pasar de esa casilla, hasta la del nivel 1. Aunque observamos una excepción, la casilla 0606, que está contigua al

mech pero está en el nivel 2, esto se debe a que si pretendemos ir andando o corriendo desde donde está el mech, antes de llegar a la casilla 0606 debemos pasar como mínimo por la 0605 para llegar hasta ella. Y lo mismo ocurre con las casillas 0707 y 0607.

Figura 6.1: Definición anillos

Esta figura representa los anillos de un mech que pretenda andar o correr, porque las casillas son accesibles si el desnivel entre una casilla y otra no es superior a dos. Pero si pretendiéramos saltar, las casillas son contiguas dependiendo de los puntos de salto del mech. En este caso los únicos cambios serían que la casilla 0606 estaría a nivel 1, y la 0607 y 0707 a nivel 2, con un mech cuyos puntos de salto fueran tres o más.

Una vez que ya sabemos qué es un anillo, es hora de explicar para qué lo utilizamos. Uno de los mayores problemas es que aun calculando la mejor posición donde colocar al mech, no sabemos si será capaz de llegar en un turno a esa posición, y en la mayoría de los casos no lo es. Para saber cuál es la casilla óptima a la que debería moverse, habría que emplear un algoritmo de búsqueda de caminos para cada una de estas casillas y comprobar que el mech puede alcanzar dicha casilla en un turno (explicaremos más adelante qué algoritmo se ha utilizado).

El uso de este algoritmo varias veces emplearía un tiempo excesivo, aunque diese la solución óptima, por lo que hay que buscar una heurística para sólo tener que utilizar este algoritmo una vez. Para calcular esta heurística se usan los anillos, que posiblemente no obtengan la posición óptima, pero obtienen una buena solución.

Tras muchas partidas, con distintos mechs y varios mapas diferentes, se ha estimado que un mech necesita unos 4 puntos de movimiento, andando o corriendo, para moverse a una casilla adyacente y quedarse en la orientación deseada. Es decir, para pasar de un nivel de un anillo a otro se necesitan una media de 4 PM. Con esta estimación (que denominaremos N) obtenemos que un mech situado en el nivel i-ésimo de un anillo, puede moverse en el intervalo de anillos [i – N, i + N], con N = PM / 4, sabiendo casi con certeza que vamos a poder ocupar la posición que queramos dentro de ese anillo y cuya distancia Euclídea sea menor que un coeficiente determinado por los PM.

De esta forma, una vez calculados los niveles que puede alcanzar un mech, sabemos el área en la que se puede mover. Aplicando distintos criterios de selección elegiremos una casilla u otra.

Los criterios para seleccionar una casilla se basan en la estrategia seguida por el mech. En la estrategia de ataque, en el caso de que no podamos acceder a la casilla de su espalda, buscaremos la casilla lo más cercana posible al objetivo.

En la estrategia de defensa tenemos la elección más compleja, pues debemos buscar un equilibrio entre distancia de seguridad y línea de visión con el objetivo, dándole prioridad a la primera, ya que un error en esta podría ser fatal si nos enfrentamos con un mech muy fuerte.

En la estrategia de huida simplemente buscamos la casilla más lejana al objetivo.

6.1.5 Modo de movimiento

Hay tres modos de movimiento: andando, corriendo o saltando. Para las rutas por tierra, andando o corriendo, calculamos la ruta usando el algoritmo de búsqueda de caminos A*. Las diferencias fundamentales entre una ruta andando o corriendo son:

- Un mech no puede correr hacia atrás.
- Las unidades que se mueven hacia atrás no pueden cambiar de elevación.
- Cuando una unidad corre no puede entrar en hexágonos de agua de profundidad 1 o mayor, aunque sí abandonar un hexágono de agua.

  Tenemos esto en cuenta a la hora de aplicar el A*, de forma que este algoritmo tenga dos modos, a pie o corriendo.

En lo referente al salto debemos tener en cuenta que:

- Un mech que intenta levantarse no podrá saltar en esa fase.
- Un mech puede tomar tierra mirando a la dirección que desee.
- Una unidad que disponga de capacidad de salto no puede saltar un desnivel superior a sus puntos de salto.
- Una unidad que tenga N puntos de salto podrá desplazarse al anillo N-ésimo de su área.

Sabiendo esto, un mech que disponga de puntos de salto para alcanzar la posición destino usará este tipo de movimiento.

En el caso de no ser posible el salto se calculará una ruta a pie y andando y finalmente se aplicará aquel tipo de movimiento que suponga un coste menor, siempre que se cumplan las restricciones para este.

Para calcular la ruta de salto se usará la función LDVyC para conocer los hexágonos por los que pasa y poder calcular el desnivel.

6.1.6 Algoritmo de búsqueda de caminos

Como ya hemos comentado antes el algoritmo de búsqueda de caminos que emplearemos será el A*.

Las alternativas podrían ser haber usado un algoritmo Greedy, más eficiente, pero en contraposición no nos aseguraría soluciones de calidad para nuestro problema. También podríamos haber usado otro algoritmo de exploración de grafos, como el de Dijkstra, que aunque si nos proporcionará soluciones de calidad perderíamos en eficiencia.

Por estas razones nos decantamos por el algoritmo A*, que combina una heurística con la búsqueda de anchura en el grafo, y además nos asegura la solución óptima si la heurística no sobreestima el coste.

El pseudo código del algoritmo es el siguiente:

ABIERTOS contiene el nodo inicial
CERRADOS está vacío
while solución no encontrada y *ABIERTOS* no vacío **do**
    Seleccionar el mejor nodo de *ABIERTOS*
    if el nodo seleccionado es un nodo objetivo **then**
        return nodo seleccionado
    else
        expandir dicho nodo
        for all *N* nodo sucesor **do**
            if *N* está en *ABIERTOS* **then**
                insertarlo manteniendo la información del mejor padre
            else if *N* está en *CERRADOS* **then**
                insertarlo manteniendo la información del mejor padre
                y actualizar la información de los descendientes
            else
                Insertarlo como un nodo nuevo en *ABIERTOS*

El algoritmo A* utiliza una función de evaluación f(n) = g(n) + h(n), donde g(n) es el coste real desde la posición inicial hasta la posición que evaluamos y h(n) la heurística que nos proporciona la aproximación al coste hasta el nodo objetivo. Así se combina búsqueda en anchura y profundidad.

La eficiencia del algoritmo está íntimamente ligada a la calidad de la heurística usada. Para una buena heurística obtendremos un tiempo lineal sobre el número de nodos, mientras que si la heurística es mala este será exponencial.

La heurística utilizada es una variación de la distancia Euclídea teniendo en cuenta la forma hexagonal de las posiciones. Se basa simplemente en aplicar el desplazamiento vertical que tienen las columnas pares, añadiendo a la fila 0.5.

Figura 6.2: Distancia en tablero hexagonal

De esta forma la distancia entre la posición con fila 1 y columna 1 y la posición con fila 1 y columna 4 sería la distancia euclidea entra las posiciones (1,1) y (1.5,4).

6.1 Reacción

La fase de reacción es la fase en la que cambiamos el encaramiento de nuestro mech para que en la fase siguiente de ataque tengamos el ángulo correcto para poder atacar al enemigo con todas las armas posibles.

Seguiremos una estrategia ofensiva, ya que lo que siempre pretenderemos es disparar con el mayor número de armas posibles. Debido a que los mechs suelen llevar casi todo el armamento en la parte frontal y que si nos encontramos cuerpo a cuerpo de frente podemos dar dos puñetazos, la mejor posición para atacar a otro mech sería tenerlo de cara.

Lo único que tenemos que realizar es un cambio de encaramiento, así que lo único que hay que saber es hacia qué cara debería estar mirando el mech, y girar a la derecha, izquierda, o no moverse según cuál sea la que más se aproxime a la cara a la que debiera estar.

Para esto buscamos el encaramiento óptimo para atacar a nuestro objetivo, y una vez calculado este, giramos a la izquierda o derecha buscando que nuestra orientación sea lo más parecida posible a la óptima.

Figura 6.3: Encaramiento del jugador rojo girando el torso a la derecha

6.2 Ataque con armas

La fase de ataque con armas es junto con el movimiento una de las más importantes del juego debido a que si nos cubrimos o buscamos una posición ventajosa para atacar al enemigo y luego no disparáramos bien sería inútil. Además esta fase junto con el ataque físico es la única forma de la que podemos eliminar a un Mech enemigo.

En el ataque con armas se pueden dar dos situaciones.

- Se dispare a un enemigo
- No se dispare a un enemigo

6.2.1 Se dispara a un enemigo:

Se deben cumplir ciertas condiciones para que el jugador inteligente dispare a un mech enemigo. El enemigo tiene que estar dentro del radio de ataque de algún arma disponible, que el disparo de dicha arma no sobrepase el calor máximo que puede aguantar, que exista visión entre él y el objetivo (Usando el programa LDVyC.exe), que el arma a disparar si es de munición aún posea, etc…

6.2.1.1 Elección de mech enemigo:

Para elegir un mech objetivo nos fijaremos en la distancia hexagonal (anteriormente descrita). El objetivo será el mech más cercano a nuestro jugador inteligente. Además se tendrá en cuenta el blindaje en caso de empates en distancia de la siguiente forma. Si dos mechs están a la misma distancia pero uno tiene menos blindaje (será un mech más débil o estará más dañado) centraremos el fuego en éste con el objetivo de intentar eliminarlo antes, pensando que si se encuentra en peores condiciones habrá más posibilidades de acabar con él en esta fase de disparo.

6.2.1.2 Elección de las armas para el ataque:

El problema del ataque con armas es similar al ya conocido problema de la mochila 0/1, busca seleccionar las armas sin que el calor generado supere el umbral (esto último sería la capacidad de la mochila).

Para la resolución hemos adoptado una estrategia sencilla que no nos asegura la solución óptima pero que es muy rápida y eficiente y en muchas ocasiones nos va a proporcionar soluciones lo suficientemente buenas para nuestro jugador.

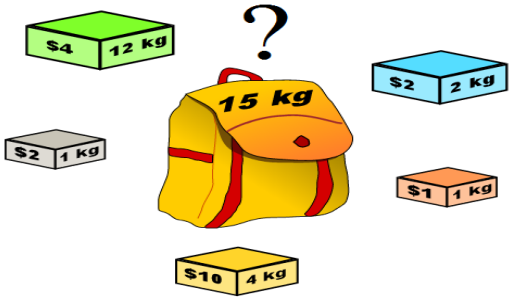
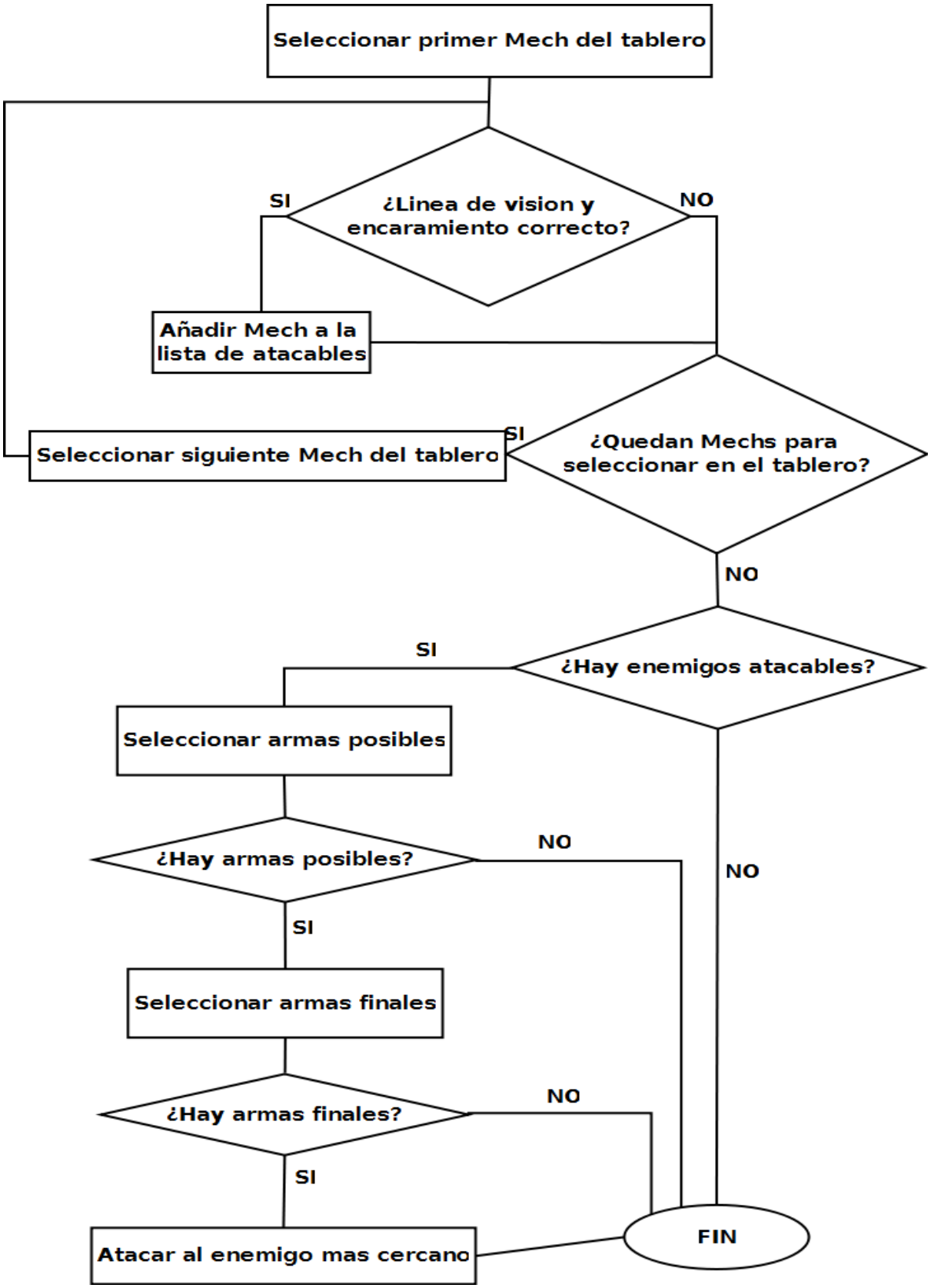


Figura 6.4: Problema de la mochila 0/1

Hemos usado un algoritmo voraz de Greedy en el que tendremos un conjunto de candidatos que serán las armas disponibles del mech. Aquellas que no estén operativas debido a un daño no contarán.

Seguidamente se comprobará que el arma que se tiene elegida tiene suficiente alcance para el mech objetivo. Si es así se comprueba si está en una localización correcta (el ángulo que forma con el mech objetivo es válido).

Si supera dichos filtros sabremos que el arma es prometedora y finalmente miraremos si es factible hacer uso de ella. Se mirará si el calor generado no supera un umbral que ponga en peligro a nuestro mech. En caso de hacer uso de un arma que no sea de energía y emplee munición se comprobara si le queda, en caso negativo el arma no se añadirá al conjunto solución. Si se cumplen todos los requisitos anteriores el arma podrá ser disparada y se incluirá en el conjunto solución.


Seleccionar primer Mech del tablero
¿Linea de vision y encaramiento correcto?
SI
NO
Añadir Mech a la lista de atacables
¿Quedan Mechs para seleccionar en el tablero?
SI
Seleccionar siguiente Mech del tablero
NO
¿Hay enemigos atacables?
SI
Seleccionar armas posibles
¿Hay armas posibles?
NO
SI
Seleccionar armas finales
¿Hay armas finales?
NO
SI
Atacar al enemigo mas cercano
NO
FIN

Como umbral para la temperatura de las armas lo que vamos a hacer es ver cuanto calor puede disipar nuestro mech y disparar en función a esto. Es decir cuanto más calor pueda disipar un mech más armas usaremos. Esto lo haremos así para evitar que el calor se acumule en el mech y éste se vea perjudicado por la temperatura.

6.2.2 No se dispara a un enemigo:

Si las condiciones expuestas en el apartado anterior no se cumplieran pondremos que el número de armas a disparar será cero y el hexágono objetivo el 0000.

6.3 Ataque físico

El ataque físico será la otra forma que tendremos de infligir daño al mech enemigo. En un mismo turno se puede realizar un ataque con el puño izquierdo, uno con el derecho y una patada.

A continuación se expondrán las condiciones necesarias para poder efectuar un ataque físico:

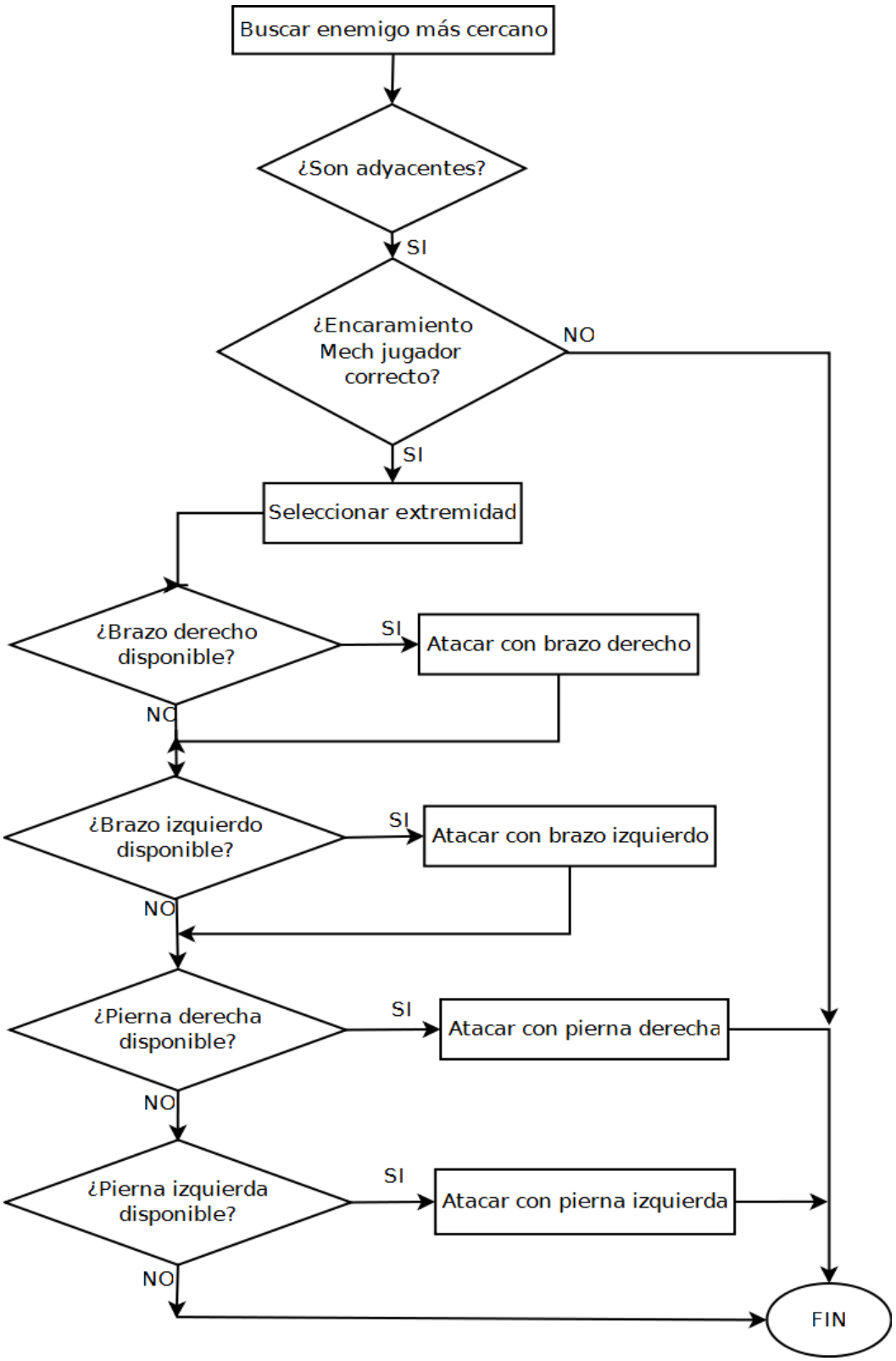
- Ambos mechs se encontrarán en casillas adyacentes.
- El encaramiento del torso o las piernas nos permitirán atacar al mech enemigo
- La diferencia de nivel entre la casilla del mech jugador y el enemigo no podrá ser superior a uno.

Si se cumplen estas condiciones se procede a comprobar con que extremidades atacar. Lo primero que haremos será comprobar si el mech posee dichas extremidades y si no ha disparado en el turno anterior con ella.

Tras esto se comprueba el ángulo entre el mech jugador y el enemigo y en base a esto veremos con qué extremidad pegar. Si se encuentra a la izquierda podremos pegar un puñetazo izquierdo y una patada izquierda. Si está a la derecha un puñetazo derecho y una patada derecha. Si esta justo enfrente podremos pegar un puñetazo derecho uno izquierdo y una patada (preferentemente derecha). Todo esto teniendo en cuenta las condiciones necesarias citadas anteriormente.

Finalmente si existe un desnivel se tendrá en cuenta también. No se pegará una patada si el mech enemigo está un nivel por encima e igualmente no se pegarán puñetazos si el enemigo se encuentra un nivel por debajo.

Si no hay enemigos adyacentes o enfrentados para pegarse no se hará nada.



6.4 Fin de turno

En la fase de fin de turno, hay dos acciones distintas que podemos hacer: expulsar munición y apagar/encender radiadores.

Con este agente no realizaremos ninguna de estas acciones en esta fase, luego no haremos nada. El motivo de esta decisión es que nuestra estrategia de ataque es ofensiva, y no queremos perder munición que luego pueda ser aprovechable para atacar. Y tampoco apagaremos los radiadores porque no hemos encontrado ninguna situación, aplicable al juego, en el que sea beneficioso apagarlos.

7. Conclusión

Con la realización de esta práctica hemos aprendido como se desarrolla un agente inteligente. El enfoque usado ha sido pragmático pero siempre basándonos en cierta base teórica. Hemos aprendido también como abstraer el problema para alcanzar una solución válida y eficiente. También hemos practicado un poco de programación de inteligencia artificial usando algoritmos que posteriormente se estudian en dicha asignatura (como las heurísticas empleadas en el movimiento). También hemos aprendido a diseñar un agente inteligente con sus respectivas capas.

La elección de C++ como lenguaje se debe principalmente a que es un lenguaje orientado a objetos que nos ha facilitado la programación de nuestra abstracción del agente inteligente, gracias a que está estructurado en módulos y con las clases y objetos ha sido fácilmente implementable. Otra de las razones por las que escogimos este lenguaje es debido a que hemos hecho bastante uso de él anteriormente y nos hemos visto más capacitados que con otros que planteamos anteriormente como Java, C, Python o C# en los que tendríamos que haber aprendido algunos conceptos para llevar a cabo la resolución del problema aunque nos hubieran dado de ventaja el uso dinámico de punteros en Java por ejemplo o el no tipado de python que hace muy flexible el código.

8. Bibliografía

http://www.virtual.unal.edu.co/cursos/ingenieria/2001394/docs_curso/capitulo1/leccion1.3.html

http://derekkopy.blogspot.com.es/2010/05/final-3d-renders-mech-concept.html
http://www.battletech.es/

Apuntes de la asignatura.

http://es.wikipedia.org/wiki/Battletech

http://es.wikipedia.org/wiki/A*