Práctica: jugador de BattleTech®

Mario J. Barchéin Molina <mariobarchein@gmail.com>

Contenidos

Introducción

Qué es BattleTech®

BattleTech® es un conjunto de juegos (rol, videojuegos, cartas) ambientado en el planeta Tierra en el siglo XXXI en el que los jugadores toman los mandos de gigantescos robots de combate llamados BattleMechs (o simplemente Mechs) e inician un enfrentamiento entre ellos.

En el juego de rol, que es en el que se centra la práctica, los jugadores controlan las acciones de los MechWarriors (pilotos de los BattleMechs) en partidas por turnos. En lo que resta de documento la referencia a BattleTech® se referirá específicamente al juego de rol.

Dispuestos en un mapa de celdas hexagonales, los jugadores deben decidir por turnos las acciones a realizar con el objetivo de derrotar a uno o más mechs rivales. La mayoría de las acciones estarán afectadas por una componente aleatoria que a efectos prácticos viene determinada por una o más tiradas de dados.

Simulador de BattleTech®

El Simulador de BattleTech® permite a dos o más jugadores participar en partidas de forma rápida y sencilla ya que es el simulador el que se encarga de la visualización, aplicar las reglas y resolver las distintas situaciones que se vayan generando a lo largo de la partida. Los jugadores simplemente tendrán que comunicar al simulador lo que quieren que hagan sus BattleMechs a través de una interfaz sencilla, amigable e intuitiva (a través de ficheros de texto en el caso de jugadores automáticos).

Una de las características más destacadas del Simulador de BattleTech® es que cualquier persona con unos mínimos conocimientos de programación puede implementar un jugador automático que interactúe con el simulador. Por lo tanto, es posible desarrollar uno de estos jugadores automáticos y jugar contra él o bien enfrentarlo contra otro u otros jugadores automáticos.

Secuencia de Juego

Una partida de BattleTech® se rige por una serie de turnos. Cada turno representa 10 segundos de tiempo real. En el transcurso de cada turno, todas las unidades del mapa tienen la oportunidad de moverse y disparar sus armas. Un turno consiste en una serie de segmentos más pequeños de tiempo, denominados fases. En cada una de las fases, los jugadores llevarán a cabo un tipo determinado de acción, por ejemplo, el movimiento o el combate.

Los jugadores ejecutan las fases de cada turno en un orden establecido. Las acciones específicas como el movimiento, el efecto de los daños, etcétera, serán explicados más adelante en secciones específicas.

Cada turno incluye las siguientes fases en el orden indicado a continuación:

- Fase de Iniciativa.
- Fase de Movimiento.
- Fase de Reacción.
- Fase de Ataque con Armas.
- Fase de Ataques Físicos.
- Fase de Control de la Temperatura Interna.
- Fase Final.

Fase de Iniciativa

1º.- Cada jugador hace una tirada de 2D6 (ambos dados) sumando el resultado de los dados para determinar su iniciativa. El jugador que obtenga el número más alto ganará la iniciativa para todo el turno presente y actuará en último lugar. Si se produce un empate, se tira de nuevo. El jugador con la siguiente iniciativa más alta será el penúltimo en actuar y así sucesivamente hasta llegar al jugador con una tirada menor que será el primero en actuar.

Fase de Movimiento

2º.- Cada jugador realiza su movimiento según el orden impuesto en la fase de iniciativa. La fase de movimiento termina cuando todas las unidades hayan movido. Todos y cada uno de los jugadores debe realizar un movimiento, aunque dicho movimiento sea simplemente levantarse (o tumbarse) o permanecer inmóvil.

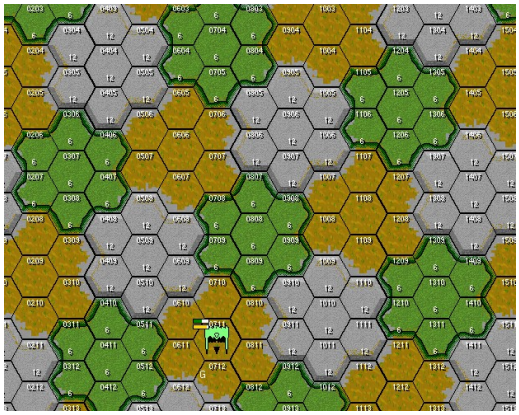


Illustration 1: Fragmento de mapa de juego con un Mech visible

Fase de Reacción

3º.- Cada jugador declara si su unidad pivotará su torso y en qué dirección. Un BattleMech puede girar su torso un lado del hexágono en el que se encuentra (60 grados) a la izquierda o derecha de la dirección en que estén apuntando sus pies. Esta nueva alineación modifica el ángulo de disparo de la parte superior del cuerpo del BattleMech (ver Ángulos de Disparo, en Combate) pero para propósitos de movimiento y localización de impactos, se considera que el BattleMech está encarado en su dirección pre-giro. Esta fase sigue un orden inverso al impuesto en la fase de iniciativa, de manera que el jugador que ganó la iniciativa será el que actúe primero.

Fase de Ataque con Armas

4º.- Cada jugador declara los disparos que efectuará su unidad siguiendo el orden impuesto en la fase de iniciativa. Cada jugador declarará los ataques que planea realizar empleando las armas de su unidad, especificando las armas que abrirán fuego, así como el objetivo u objetivos de dichos disparos. Si un arma utiliza cargas de munición especial, como por ejemplo la munición de racimo LB-X, el tipo específico de munición deberá también ser declarado en este momento.

La declaración de ataques se va sucediendo hasta que se hayan declarado todos los ataques. El jugador que ganó la iniciativa declara el último ataque.

5º.- El disparo del armamento se va resolviendo unidad por unidad. Todo el combate con armamento se considera que tiene lugar simultáneamente, por lo que no importa el orden en que se resuelva, aunque para que a los jugadores les sea más fácil controlar y determinar las armas que han sido disparadas, primero deben resolverse todos los disparos con armamento de una unidad antes de empezar con otra.

6º.- Los daños se anotarán a medida que se resuelven los ataques, pero el daño no surte efecto hasta después de resolverse todos los ataques con armas. Es al fin de esta fase cuando todos los daños surten efecto inmediatamente y cuando los jugadores deberán realizar los chequeos de pilotaje requeridos debido a los efectos de los ataques con armas. Hay que destacar que los daños sufridos por una unidad durante la Fase de Ataque con Armas surten efecto antes del inicio de la Fase de Ataques Físicos del mismo turno.

Fase de Ataques Físicos

7º.- Cada jugador declara los ataques físicos que efectuará su unidad. El jugador que controla la unidad declarará los ataques que planea realizar empleando los ataques o las armas de su unidad, especificando los ataques y las armas que abrirán fuego, así como el objetivo u objetivos de dichos ataques.

La declaración de ataques se va sucediendo hasta que se hayan declarado todos los ataques. El jugador que ganó la iniciativa declara el último ataque.

8º.- El ataque se va resolviendo unidad por unidad. Todo el combate se considera que tiene lugar simultáneamente, por lo que no importa el orden en que se resuelva, aunque para que a los jugadores les sea más fácil controlar y determinar los ataques que han sido realizados, primero deben resolverse todos los ataques de una unidad antes de empezar con otra.

9º.- Los daños se anotarán a medida que se resuelven los ataques, pero el daño no surte efecto hasta después de resolverse todos los ataques físicos. Es al fin de esta fase cuando todos los daños surten efecto inmediatamente y cuando los jugadores deberán realizar los chequeos de pilotaje requeridos debido a los efectos de los ataques físicos.

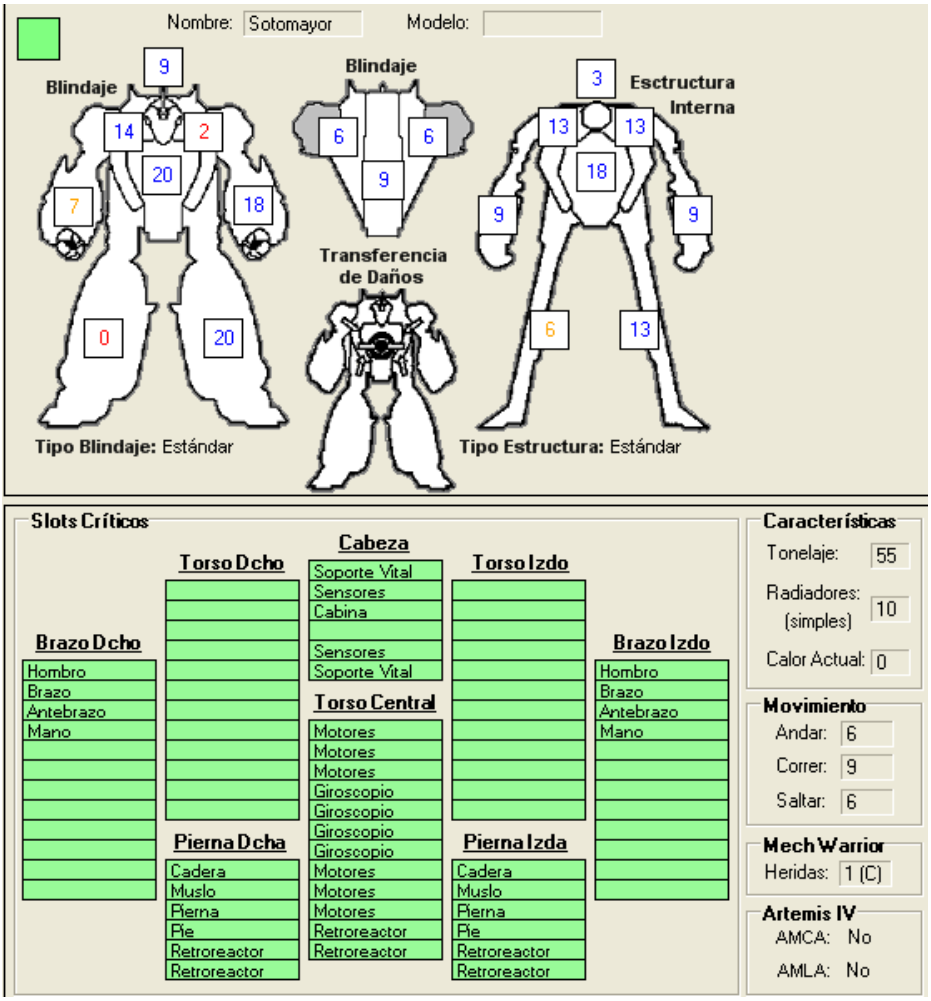


Illustration 2: Hoja de control de un Mech

Fase de Control de Temperatura Interna

10º.- Se ajustan los módulos de temperatura interna de los BattleMechs, reflejando así cualquier cantidad de calor acumulada o perdida durante el turno. Todo daño temporal o permanente causado por un exceso de temperatura interna se resuelve en este momento.

Fase Final

11º.- Los jugadores cuyos MechWarriors hayan perdido el conocimiento en un turno anterior, realizan una tirada de dados para averiguar si el piloto recupera el conocimiento durante este turno.

12.- En caso de que el MechWarrior esté consciente cada jugador podrá decidir preparar para la expulsión tantos módulos de munición como desee si considera que ya no le serán útiles en el resto de la partida. La munición que es preparada para ser expulsada será expulsada al principio de la Fase Final del siguiente turno de manera que no podrá volver a usarse en el juego.

13º.- Se ejecutan las acciones restantes del turno, como por ejemplo determinar si se propagan los incendios a otros hexágonos. Las reglas específicas para tales acciones determinarán si éstas deben surtir efecto durante la Fase Final de turno.

14º.- La secuencia de los pasos 1º al 13º se repite hasta que uno de los jugadores consiga la victoria. Normalmente, el jugador con la última unidad superviviente sobre el tablero ganará la partida. Si varias unidades se destruyen entre sí simultáneamente, los jugadores que controlan dichas unidades quedarán empatados en la primera posición.

Nota: La explicación detallada de las reglas aplicables para cada una de las fases se pueden encontrar en el documento “Manual de Reglas de BattleTech.doc”, proporcionado junto al material para la realización de la práctica

Objetivo

El objetivo de la práctica es la realización de un jugador automático capaz de jugar (bien) a BattleTech® usando el simulador. Este jugador automático debe recoger toda la lógica necesaria para ser capaz de jugar con distintos tipos de mechs, mapas variados, uno o más rivales, etc.

Modelo teórico

Agente inteligente

Un agente inteligente, es una entidad capaz de percibir su entorno, procesar tales percepciones y responder o actuar en su entorno de manera racional, es decir, de manera correcta y tendiendo a maximizar un resultado esperado. Es capaz de percibir su medioambiente con la ayuda de sensores y actuar en ese medio utilizando actuadores (elementos que reaccionan a un estímulo realizando una acción).

En este contexto la racionalidad es la característica que posee una elección de ser correcta, más específicamente, de tender a maximizar un resultado esperado. Este concepto de racionalidad es más general y por ello más adecuado que inteligencia (la cual sugiere entendimiento) para describir el comportamiento de los agentes inteligentes. Por este motivo es mayor el consenso en llamarlos agentes racionales.

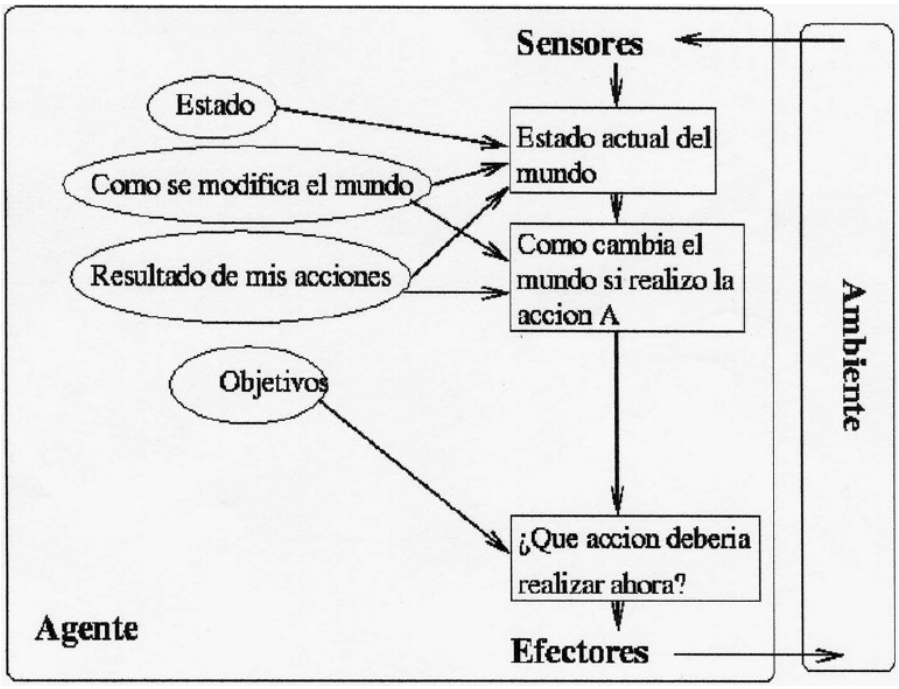
Un agente inteligente puede ser una entidad física o virtual. Si bien el término agente racional se refiere a agentes artificiales en el campo de la Inteligencia Artificial, también puede considerarse agentes racionales a los animales incluido el hombre.

El agente inteligente desarrollado es del tipo “racional”, por lo que cumple las siguientes características:

1. Es reactivo, de manera que responde a los cambios del entorno porporcionados por el simulador
2. es proactivo, ya que toma sus decisiones con el doble objetivo de sobrevivir y destruir a sus contrincantes
3. Es autónomo y actúa sin la intervención del usuario dentro de los límites que proporciona el entorno de simulación.

La corrección y adecuación de cada una de las acciones que lleva a cabo se evalúan entre todas las posibles y se opta por la que proporciona, a priori, mayor ventaja de éxito.

Según su arquitectura, se trata de un agente híbrido basado en objetivos (“metas”). El agente analiza su entorno, en base a los dos objetivos anteriormente descritos (sobrevivir y destruir a los enemigos) evalúa cuál podría ser el resultado de la “modificación del mundo” según el hipotético resultado de sus acciones y con toda esa información toma una decisión para cuál debe ser la accion a llevar a cabo en la fase correspondiente.



Ambiente

El ambiente en el que se desenvuelve el agente inteligente tiene estas características:

1. Accesible. Todos los parámetros relevantes son conocidos y detectados.
2. No determinista. Al haber una componente de azar en el resultado de algunas de las acciones (tiradas de dados para resolver ataques y algunos movimientos), el siguiente estado del ambiente tras realizar una acción no queda totalmente determinado.
3. Episódico. El esquema de turnos y fases en las que se divide la interacción de cada jugador determinan este tipo de ambiente.
4. Estático. El ambiente no varía mientras que el agente está tomando la decisión de cuál va a ser la siguiente acción a realizar.
5. Discreto. Hay un número limitado y claramente definido de percepciones y acciones realizables.

Descripción de la solución

Representación del tablero de juego

El tablero de juego está formado por una retícula hexagonal, en la que cada casilla (hexágono) tiene una serie de propiedades caracterísitcas (tipo de terreno, elevación, objetos y/o accidentes, etc) que limitan el movimiento y las posibilidades de disparo e impacto de las armas y ataques físicos de los Mechs.

Por otra parte, la posición de los Mechs en el tablero viene dada tanto por la casilla en la que se encuentran como por el "encaramiento" hacia uno de los 6 lados del hexágono en el que se encuentran. Este encaramiento afecta tanto a las acciones de movimiento (ya que se consumen puntos extra de movimiento para cambiar el encaramiento dentro de un mismo hexágono), como a las de ataque, ya que puede limitar o imposibilitar el uso de determinadas armas o ataques físicos.

Para modelar esta información de forma que sea fácilmente tratable por los diferentes algoritmos que se utilizan en diversas partes del programa (más adelante explicadas), se ha optado por la representación en forma de grafo de los hexágonos del tablero y los caminos que los unen. En total, se definen tres grafos diferentes, cada uno de ellos utilizado con diferentes fines:

Grafo de distancias simples

Este grafo no dirigido asigna como vértices los hexágonos del tablero, y como arcos las conexiones entre hexágonos adyacentes.

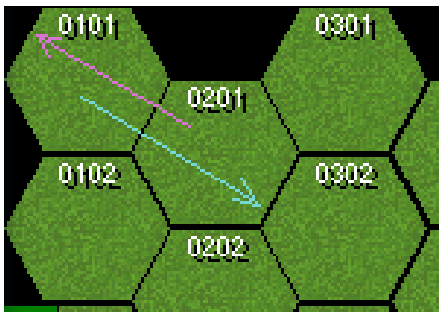
El "peso" asignado a cada arco es de "1" sin importar los tipos de terreno (hexágonos) que conectan, de manera que, por ejemplo, recorrer una línea recta de 4 hexágonos tiene un coste de "4".

Este grafo se utiliza para calcular distancias de salto, radios de acción con respecto a una casilla dada, encaramientos óptimos hacia un hexágono arbitrario y ángulos de disparo permitidos.

Grafo de distancias mediante movimiento "Andar"

Se trata de un grafo dirigido en el que cada pareja (encaramiento, hexágono) se asigna a uno de los vértices del mismo. De esta manera, cada hexágono tendrá asociados 6 vértices en el grafo, uno por cada encaramiento posible.

Los arcos de este grafo son cada uno de los caminos posibles entre dos vértices (encaramiento, hexágono) adyacentes que permiten un movimiento de tipo "Andar" según las reglas de movimiento descritas en el manual de juego y el coste asociado a cada arco se corresponde con el coste de movimiento calculado para desplazarse entre ambos vértices. Si el movimiento no es posible (cambio de elevación no permitido), no se crea el arco entre los dos nodos.



Por ejemplo, los dos hexágonos adyacentes 0101 y 0201 con un mismo nivel de elevación y un terreno de tipo “norma” (ver figura) tendrán como vértices en el grafo:

(1,0101), (2,0101),…,(6,0101) y (1,0201),(2,0201),...,(6,0201)

y como arcos (3,0101)→(3,0201) y (6,0201)→(6,0101), ambos con un coste o peso de “1”, los cuales indican que desde la posición (3,0101) se puede “avanzar” gastando 1 punto de movimiento a la posición (3,0201) y que desde (6,0201) se puede “avanzar” a (6,0101). Además, habrá arcos adicionales para indicar el coste de cada rotación (cambio de encaramiento mediante movimiento “Izquierda” o “Derecha”) dentro de un mismo hexágono, con lo que encontraremos los arcos (1,0101)→(2,0101), (2,0101)→(3,0101), etc., y viceversa, cada uno de ellos con un coste de “1”.

Este grafo es especialmente útil para computar cuales son las distancias más cortas, incluyendo costes de rotación en cada giro del camino para realizar movimientos de tipo “Andar”. El cálculo de caminos óptimos para este grafo se realizan siempre utilizando el algoritmo A*

Grafo de distancias mediante movimiento “Correr”

Es un grafo muy similar al de movimiento “Andar”, con la diferencia de que se utilizan las reglas para el movimiento “Correr”, a la hora de calcular qué arcos conectan cada pareja (encaraminto, hexágono) y cuál es el coste asociado a cada uno de ellos. Al igual que en el anterior, ek cálculo de caminos óptimos para este grafo se realizan siempre utilizando el algoritmo A*

Fase de movimiento

Esta es la primera de las fases del turno en la que el agente tiene que tomar una decisión, ya que la fase de iniciativa es resuelta previamente y de forma completa por el simulador del juego. En esta fase, el agente puede tomar dos caminos de decisión diferentes, dependiendo de si tiene armas operativas o no.

En cualquier caso, si el Mech ha caido al suelo, se intenta levantar antes de continuar con el siguiente movimiento.

Movimiento para Mech con armamento operativo

1. El agente calcula cual puede ser el movimiento del enemigo, para ello, dependiendo de la posición del enemigo y de sus puntos de movimiento, se obtiene una “nube” de posibles posiciones de destino para el enemigo. Para obtener esta nube se utiliza el “grafo de movimiento Andar” y se calculan todos los recorridos que puede hacer el enemigo con sus puntos de movimiento.

2. Una vez se ha calculado la nube de destinos posibles a los que puede moverse el enemigo, se asume que éste intentará acercarse a nuestra posición, por lo que se fija como casilla de destino aquella de la nube que quede más cerca de la posición del jugador mediante el movimiento “Andar” y que permita línea de visión con el enemigo. Para ello se analizan las distancias a cada una de las posiciones de la nube y se queda con la más corta que tiene LDV y además encaramiento hacia la dirección por la que vendrá el enemigo para poder realizar un ataque potente con las armas montadas en el Mech. Si no es posible la línea de visión en ninguna posición alcanzable, directamente se recorre el camino más corto. Si no es posible el movimiento mediante “Andar”, se intenta el movimiento mediante “Saltar” (ver más adelante).
3. Se ejecuta el movimiento seleccionado

Movimiento para Mech sin armamento

En este caso, se busca una confrontación “cuerpo a cuerpo” con el enemigo. Para ello:

1. Se obtienen todas las posiciones viables para ataque físico (cumpliendo las restricciones de desnivel que se indican en el reglamento) que rodean de forma inmediata al enemigo y que dejan al agente directamente encarado hacia el enemigo.
2. Se calcula el coste total en unidades de movimiento para llegar andando y saltando a dicha posición.
3. Si el control de calor lo permite, se realiza el movimiento mediante el comando “Saltar”
4. En caso de que no se haya podido “Saltar”, el mech se aproxima a la posición mediante “Andar”

Saltar

En cualquiera de los dos tipos de movimiento, es posible que el mech decida “Saltar” en vez de “Andar” para llegar a su objetivo. El cálculo de camino óptimo para saltar se realiza mediante una aproximación *greedy*, de manera que se hace el salto hacia la posición más cercana con respecto al objetivo utilizando el grafo de movimientos simples. Una de las motivaciones para usar este tipo de aproximación en vez de una búsqueda A* entre diferentes caminos de salto es una violación o mala implementación de las reglas de salto en el simulador de Battletech. Haciendo pruebas, se ha detectado que el simulador marca saltos como “no posibles” debido a obstáculos en el camino en algunas situaciones de forma errónea.

Por ejemplo, supongamos que un Mech tiene 4 puntos de salto y está en la posición 0517 del siguiente mapa:

Si desea saltar hacia la posición 0717, según las reglas podría hacerlo tomando el camino 0616-0717 (imposible debido a restricción de altura) ó 0617-0717 (posible). En este caso, el simulador devuelve que el salto *no es posible*, ya que no considera el camino alternativo. Esto hace que la búsqueda mediante A* no funcione, puesto que no se puede codificar esta regla y por ello se utiliza una estrategia *greedy* para el salto, que, aunque no alcanza en todos los casos un resultado óptimo, suele producir una muy buena aproximación.

Consideraciones acerca de la implementación de la búsqueda mediante A*

Para aquellas búsquedas de caminos más cortos en cualquiera de los grafos expuestos, se utiliza el algoritmo A*. A* garantiza encontrar el camino óptimo considerando el coste real del camino desde el nodo x y, en cada paso, utiliza como evaluación del coste

$$f(x) = g(x) + h(x)$$

Siendo g(x) el coste real para alcanzar el nodo desde x (igual que en el algoritmo de Dijkstra) y h(x) el coste aproximado (heurística) para alcanzar el nodo destino desde x. Para que esto funcione:

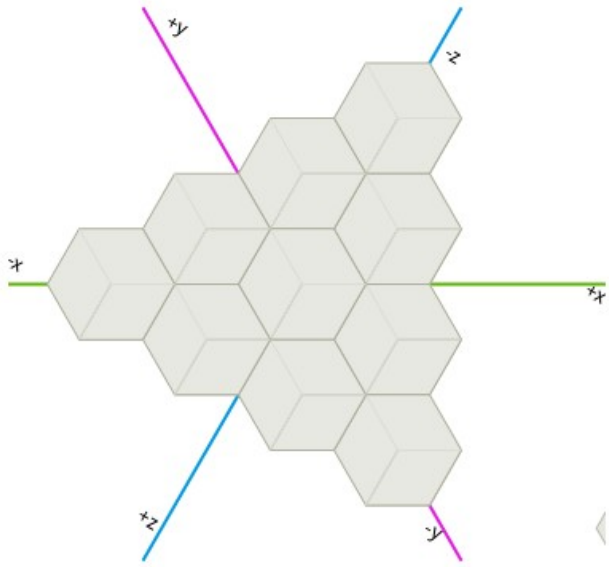
1. cada nodo del grafo debe tener un número finito de sucesores
2. el coste de cada arco del grafo debe ser mayor o igual que cero
3. la función heurística h(x) nunca debe sobreestimar el coste real

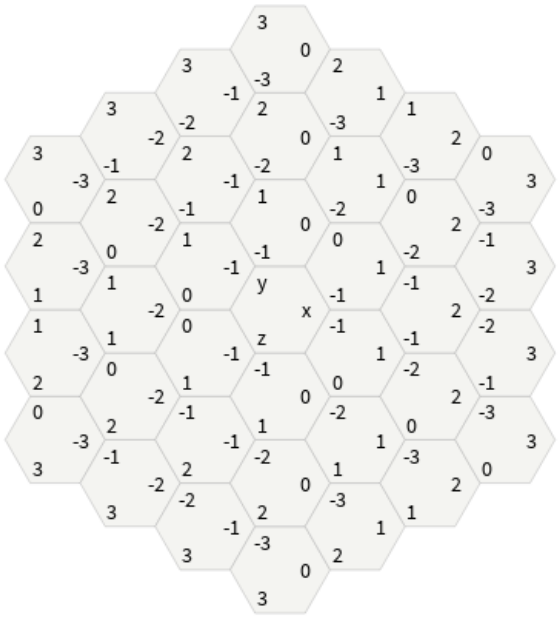
   $$g(x) >= h(x)$$

A* utiliza una cola con prioridad para expandir los nodos y la introducción de la heurística hace que primero se expandan los nodos “prometedores” según la heurística. El algoritmo de Dijkstra es un caso particular de A*, en el que h(x) = 0 para todos los nodos.

En el problema de la práctica, se utiliza la “distancia de Manhattan” entre hexágonos como heurística. Para poder calcular esta distancia de forma correcta y dada la naturaleza hexagonal del tablero de juego, no es válido utilizar la suma de las diferencias en valor absoluto de las coordenadas, ya que por ejemplo, para dos hexágonos adyacentes <0201> y <0302> que se encuentran a una distancia real de “1”, este cálculo daría una distancia de “2”, lo cual sería una sobreestimación del coste real y haría que A* no funcionara correctamtente.

Para obtener la “distancia de Manhattan” de forma correcta entre dos hexágonos cualesquiera del tablero, lo primero necesario es hacer un cambio del sistema de coordenadas, de manera que se pueda manipular de forma sencilla la naturaleza hexagonal del tablero y las seis posibles direcciones en las que pueden aparecer los hexágonos adyacentes. Para ello se utiliza un sistema de coordenadas “cúbicas” que asigna tres valores de posición (x,y,z) a cada casilla.



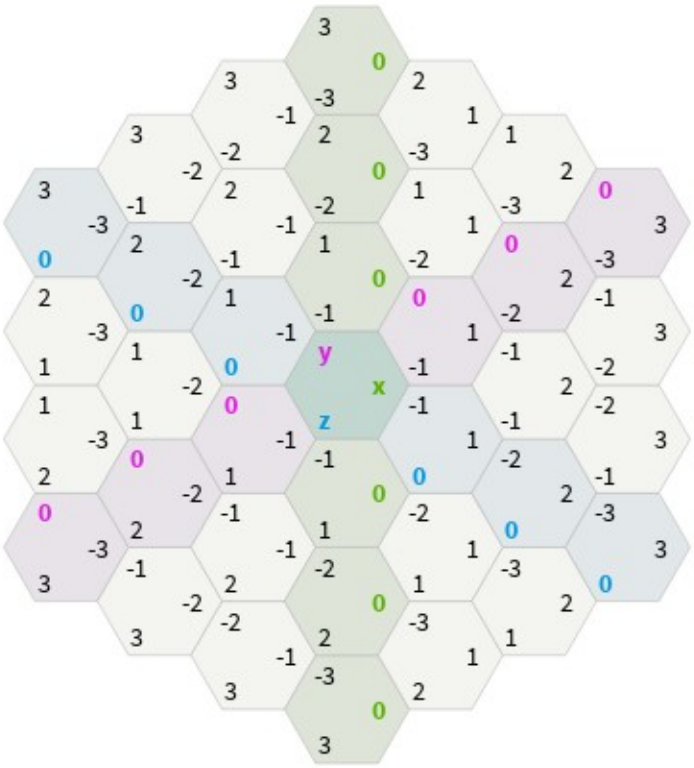


En este sistema de coordenadas cúbicas:

1) cada uno de los tres valores se corresponde con una dirección lineal en la rejilla

2) cada una de las direcciones x, y o z es una combinación de las otras dos, de manera que moverse en una dirección concreta de las seis posibles implica sumar 1 a una coordenada y restar 1 a otra de ellas. Por ejemplo, ir hacia “arriba” significa moverse en dirección +y -z, o dicho de otro modo, sumar 1 a y y restar 1 a z

En cualquier casilla se cumple que x + y + z = 0.



Para convertir las coordenadas de tipo “desplazamiento” proporcionadas por el entorno de simulación a coordenadas cúbicas, se aplican estas reglas:

x = col

z = row - (col - (col&1)) / 2

y = -x-z

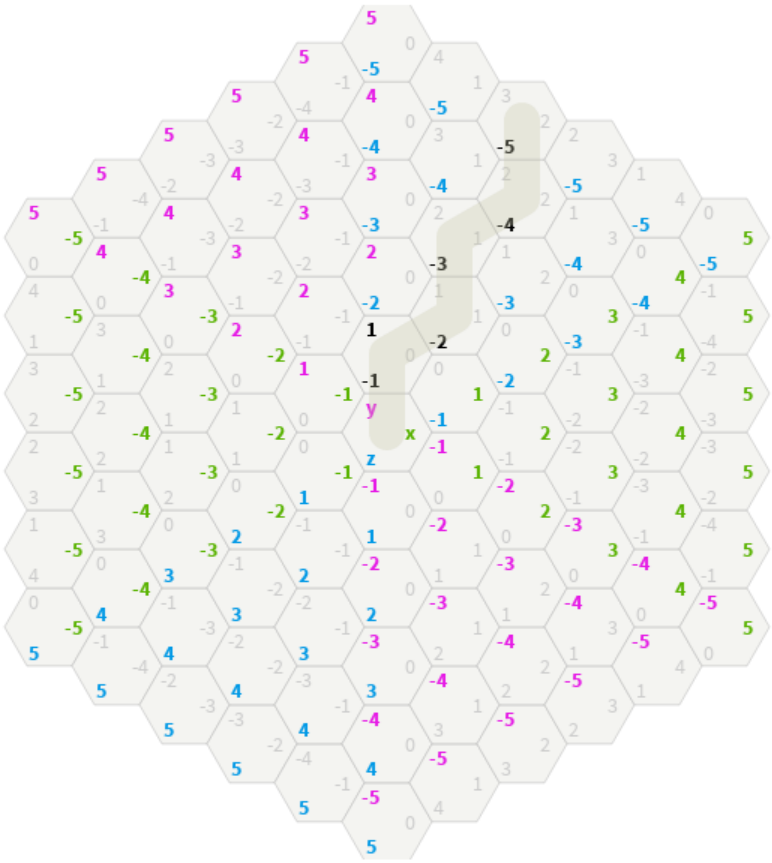
donde “col” es el nº de columna y “row” el nº de fila.

Una vez que se tienen estas coordenadas, calcular la distancia de Manhattan es sencillo aplicando la siguiente fórmula:

$d(a,b) = \max (\text{abs}(a_x - b_x), \text{abs}(a_y - b_y), \text{abs}(a_z - b_z))$

Por ejemplo, aquí se puede ver un camino de distancia 5 desde (0,0,0) hasta (2,3,-5) en el que

$d(a,b) = \max (\text{abs}(-2), \text{abs}(5), \text{abs}(-3)) = \max(2,3,5) = 5$



Fase de reacción

En esta fase, se permite hacer una rotación opcional del torso hacia la izquierda o derecha. El agente calcula la dirección en la que se encuentra el enemigo haciendo un cálculo del camino más corto entre su propia posición y la del enemigo utilizando el grafo de distancias simples y una búsqueda

mediante A* y con esta información gira su torso hacia la dirección del hipotético “primer paso” de este camino directo.

De esta manera, el agente hace el mayor esfuerzo posible para obtener un encaramiento frontal del torso con respecto a su enemigo.

Fase de ataque con armamento

Para evitar un sobrecalentamiento excesivo que lleve a la desconexión del Mech o al riesgo de producir daños críticos por recalentamiento, se fija un umbral de calor máximo de acuerdo a la temperatura actual del Mech, los daños en la estructura que pueden generar calor adicional (motores) y el nº de radiadores operativos, de manera que tras la acción de disparo de armas y la posterior refrigeración en la “Fase de calor”, el calor total quede por debajo del umbral que haría necesaria una tirada para calcular si se producen daños por calor (explosiones de armas).

Para cálculo de las armas que se pueden disparar, el agente:

1. Determina si hay línea de visión con el enemigo. Si no la hay, se omite el ataque con armamento.
2. Obtiene un listado de armas operativas y con munición. Se excluyen de la lista las armas que están montadas en una extremidad que no casa con la posibilidad de disparo a la posición enemiga debido a restricciones del ángulo de tiro
3. Para cada arma en el listado obtenido en el paso anterior, se calcula su tirada modificada para impacto según las reglas del manual. Si la tirada supera el valor de 12, se descarta de la lista.
4. Para las armas que han quedado en el listado, se eligen aquellas que cumplen las restricciones de calor y que además son mejores en cuanto a la probabilidad de impacto porque su tirada modificada es menor. Para ello se utiliza un simple “algoritmo de la mochila” discreto en el que el beneficio es la tirada modificada (cuanto menor es esta, mayor es el beneficio) y el coste es la cantidad de calor generada por el disparo del arma.
5. Si la ejecución del algoritmo de la mochila ha dejado armas en la lista final, se disparan. En caso de que no haya armas disponibles (el calor resultante dejaría al Mech en zona de peligro), se omite el disparo.

Fase de ataque físico

Si el enemigo está en una posición adyacente, se realiza un ataque físico con todas las extremidades que no han sido utilizadas para el ataque con armamento y teniendo en cuenta las restricciones impuestas por el reglamento en cuanto a ataques permitidos según elevaciones relativas entre atacante y objetivo y ángulos de disparo.

Este ataque se realiza siempre que sea posible, ya que nunca produce un perjuicio para el atacante en forma de puntos de calor.

Fase de control de temperatura interna

Esta fase se resuelve de forma directa y sin intervención del agente.

Fase final

Se cede el control del juego y se pasa a la siguiente ronda.

Detalles de la implementación

- El agente está programado utilizando el lenguaje Python 3.4
- Se utiliza la bilioteca NetworkX para implementar grafos (dirigidos y no dirigidos) y algoritmos de búsqueda.
- El programa está diseñado para funcionar en un sistema Windows XP SP 2 con el Framework .NET 3.5 que tenga instalado Python 3.4
- El fichero requirements-windows.txt incorpora los requisitos para el virtualenv Python. Para crear este virtualenv, en la carpeta donde se haya descomprimido el proyecto, ejecutar desde un intérprete de comandos
    - python -m venv venv-win
    - venv-win\Scripts\Activate.bat
    - pip install -r requirements-windows.txt
- El ejecutable para el agente es el fichero “player.bat”. Este fichero se encarga de activar el entorno virtual (que debe estar previamente creado) y lanzar el script Python que realiza todo el cómputo de la siguiente acción a realizar

Bibliografía

- Russell, S.J.; Norvig, P. Inteligencia artificial: un enfoque moderno (2ª edición). Mexico. Prentice-Hall, 2004.
- Nils J. Nilsson. Inteligencia artificial: una nueva síntesis. Madrid. McGraw-Hill, 2000.
- Patel, Amit; Hexagonal Grids. Red Blob Games, 2015

  http://www.redblobgames.com/grids/hexagons
- Python Software Foundation; Python 3.4.3 Documentation, Python Software Foundation, 2015

  https://docs.python.org/3/
- NetworkX developer team; NetworkX Documentation 1.10; NetworkX developer team, 2014

  https://networkx.github.io/documentation.html
- Bills, Randall N.; Nelson, Jim; Classic Battletech Introductory Rulebook (version 2.0); Chicago. FanPro LLC. 2006