Pathfinding & FSM

David Parra Ausina
3º HND in Videogame Programming
ESAT

David Parra Ausina
ESAT

Introducción

En esta práctica se nos encomendó la investigación y posterior desarrollo de una aplicación que utilizase un algoritmo de búsqueda de caminos del tipo A* en un mapa 2D, y a su vez integrarlo con nuestro sistema de agentes implementado en la práctica anterior, la cual contiene una mente y un cuerpo, totalmente independientes el uno del otro, y aplicando una lógica mediante una máquina de estados finita.

El juego se desarrolla en una fortaleza, donde podremos encontrar tres tipos de entidades: los soldados, los prisioneros y los guardas. Cada uno tendrá una finalidad específica que explicaremos más adelante.

La finalidad de la aplicación es que los soldados, situados fuera de la fortaleza, entren dentro de la fortaleza y rescaten a los prisioneros abriendo las puertas de la fortaleza, los cuales se encuentran trabajando en ella. Sin embargo, los guardas intentarán a toda costa cerrar las puertas y dar la alarma, para que los otros guardas estén alerta y no dejen escapar a ningún prisionero.

Búsqueda de caminos

Como indica su nombre, la búsqueda de caminos es una técnica, mayormente utilizada en videojuegos, para encontrar un camino óptimo de un punto a otro de un mapa esquivando los posibles obstáculos que hayan en el mapa.

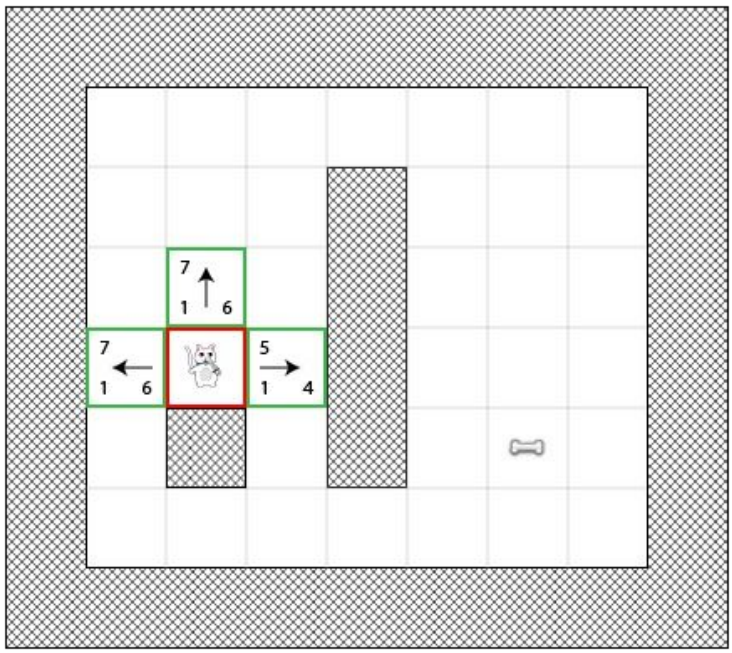
Hay muchos tipos de algoritmos de búsqueda de caminos, pero en esta ocasión nos centraremos en el algoritmo de búsqueda de caminos del tipo A* (A-star).

Algoritmo A*

Este algoritmo trata de buscar el camino mas optimo dependiendo del coste de cada movimiento, es decir, cuanto menos cueste moverse de un punto a otro y cuanta menor sea la distancia de ese nuevo punto al objetivo final, será el punto más óptimo para avanzar.

Para entender bien este algoritmo, hay que entender sus variables y su funcionamiento. Principalmente el algoritmo divide el mapa en nodos (el tamaño del nodo dependerá del nivel de detalle que quieras obtener, cuanto menor sea el tamaño del nodo, mayor detalle, pero eso también implica mayor coste computacional).

Se van calculando los nodos alrededor del punto inicial del camino, 4 u 8 respectivamente (dependiendo si permites movimiento diagonal), teniendo en cuenta los puntos no transitables del mapa (*ver mapa de costes - Pag 5*).



A cada nodo obtenido se le calculará una serie de variables que más adelante servirán para conseguir el camino mas optimo:

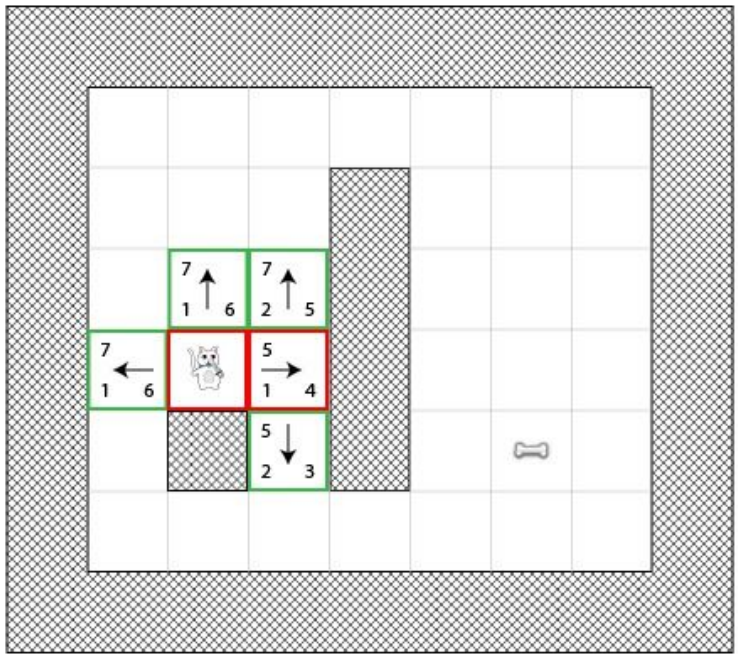
- H: La distancia desde este nodo hasta el final del camino
- G: El coste desde el punto inicial del camino hasta este nodo.
- F: El sumatorio de los dos anteriores.
- Padre: El nodo desde el cual ha venido.

Todos los nodos calculados se irán guardando en una lista que posteriormente utilizaremos para sacar los posibles nodos más óptimos (pueden no ser los definitivos), a esta lista la llamaremos “Lista abierta”.

Una vez hemos completado el primer cálculo tendremos que avanzar a un siguiente nodo para seguir calculando. Para ello buscaremos en la lista abierta el nodo que tenga la menor F, lo que nos dará el mejor posible nodo teniendo en cuenta su distancia al punto final del camino y su coste actual.

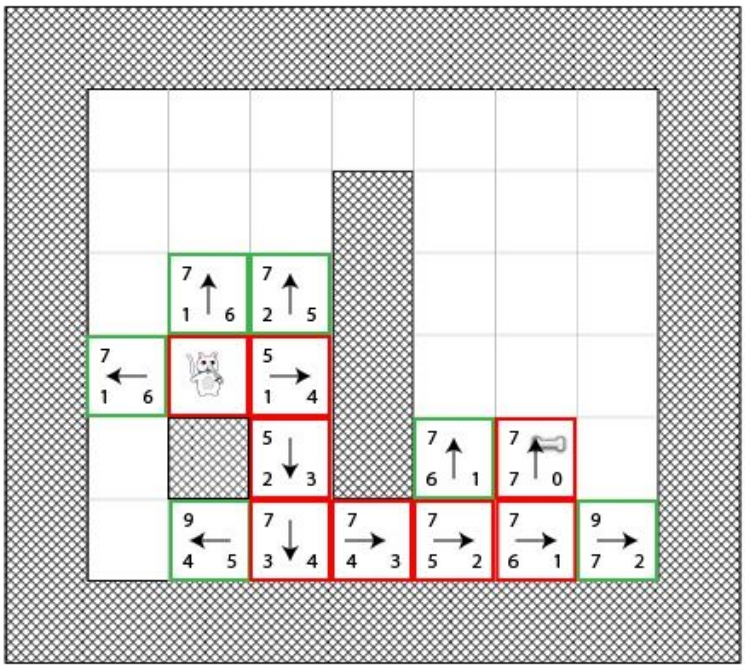
Cuando tengamos el nodo, lo borraremos de nuestra lista abierta y lo trasladaremos a la que será nuestra lista de posibles nodos definitivos, a la cual la llamaremos “Lista cerrada”.

A partir de este nodo, volveremos a calcular sus nodos adyacentes y repetiremos el procedimiento anterior.



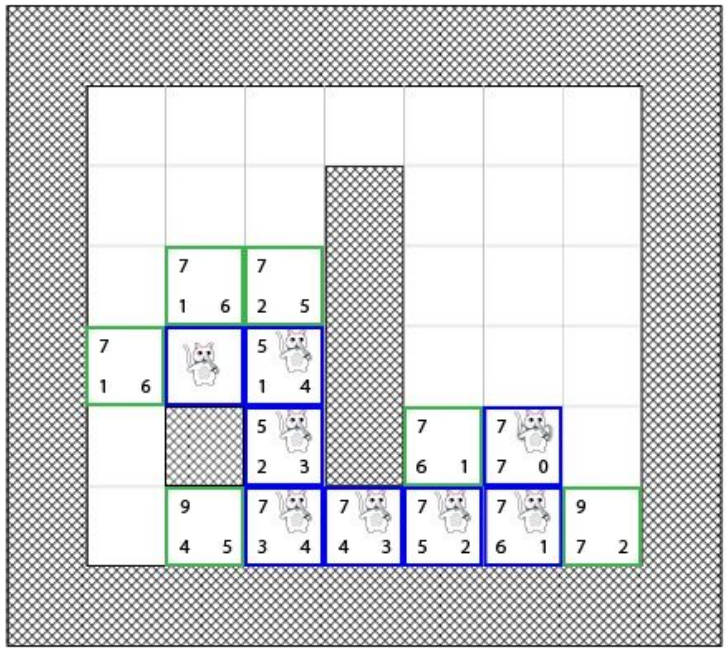
Hay que tener en cuenta los nodos que ya hemos calculado anteriormente para no repetir cálculos innecesarios, por lo que para cada nuevo nodo, miraremos si ya existe en la lista abierta o en la lista cerrada.

Sabiendo todo esto, repetiremos el proceso anterior hasta llegar al punto final del camino.



Una vez tenemos todos los posibles nodos calculados en nuestra lista cerrada, procederemos a sacar los nodos que son los más óptimos. Para ello recorreremos la lista cerrada desde el nodo final y accediendo a su nodo padre, y este a su vez a su nodo padre, hasta llegar al punto inicial del camino.

El resultado será el camino mas optimo de todo el cálculo A*.



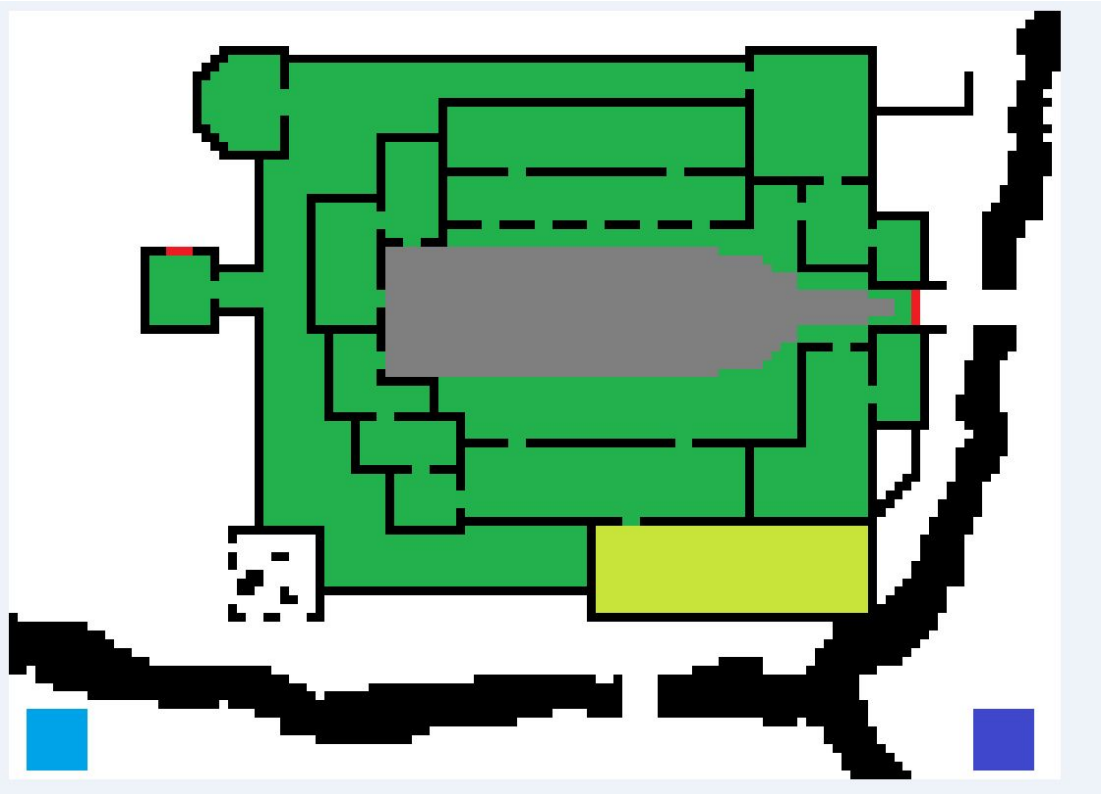
Agentes

Como hemos comentado anteriormente, los agentes se componen de la mente y el cuerpo, los cuales son independientes el uno del otro, pero se complementan dentro de las máquinas de estados para realizar una función lógica específica de cada comportamiento.

Cuerpo
El cuerpo se compone de todas las cosas relacionadas con el movimiento, sin ningún tipo de lógica, y será la máquina de estados la que le diga cómo y a dónde moverse.

Como en esta práctica hemos integrado la búsqueda de caminos, lo tenemos que integrar junto con el movimiento rectilíneo uniforme implementado en prácticas anteriores. En estos casos tendremos la lista de puntos a los que moverse y iremos recorriendo los puntos uno a uno hasta llegar al punto final, indicando la dirección y velocidad a la que se tiene que mover.

Únicamente pediremos que el algoritmo A* nos calcule un camino cuando la distancia sea demasiado grande o los puntos a moverse no estén dentro de la misma zona (solo para la zona de trabajo y la zona de descanso)



Caché de caminos

Como hemos comentado anteriormente, el algoritmo de búsqueda de caminos A* es considerablemente costoso, sobretodo para caminos largos con obstáculos. Por ello es necesario optimizar de algún modo este cálculo.

En nuestro caso elegimos implementar un sistema de cacheado de caminos, el cual funciona de manera muy similar a un cacheado simple de datos de un sistema operativo.

Todos los caminos que se han podido calcular se guardarán temporalmente en una lista con una puntuación inicial, la cual irá decreciendo cada vez que se pida un camino diferente al guardado, hasta que no tenga ninguna puntuación y se procese su eliminación. Sin embargo, si alguna vez se pide un camino que ya estaba guardado, se le incrementará la puntuación, aumentando así su durabilidad en la caché.

Con esto nos evitamos tener que recalcular caminos similares, además de poder precalcular los caminos que nosotros determinemos como costosos y que se van a utilizar repetidamente.

Mente

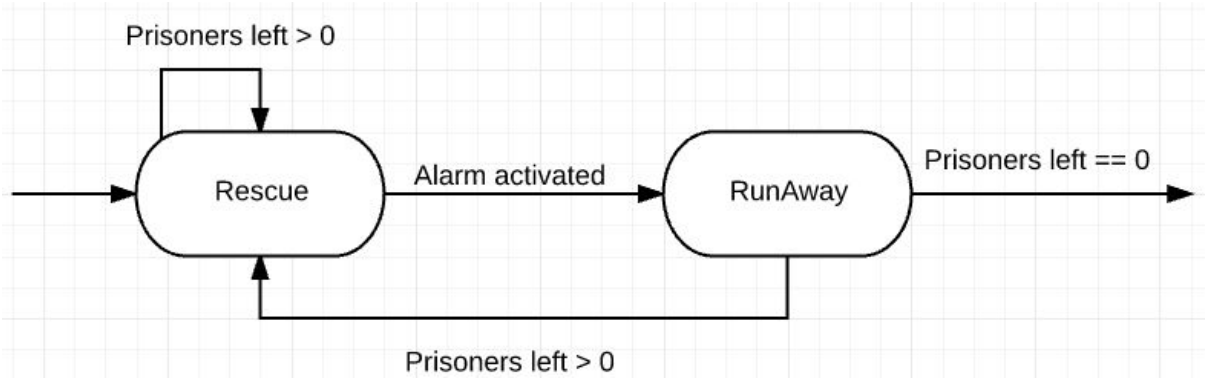
En nuestra aplicación no interactiva podemos encontrar cuatro tipos diferentes de comportamientos

- **Fortaleza**

  Se encarga de todas las cosas contenidas en la fortaleza, tales como las cajas y las puertas, además de relevar a los prisioneros sin energía por otros que estén descansando.

  También se encarga de informar a todos los agentes que están dentro de la fortaleza cuando se activa la alarma.

- **Soldado**

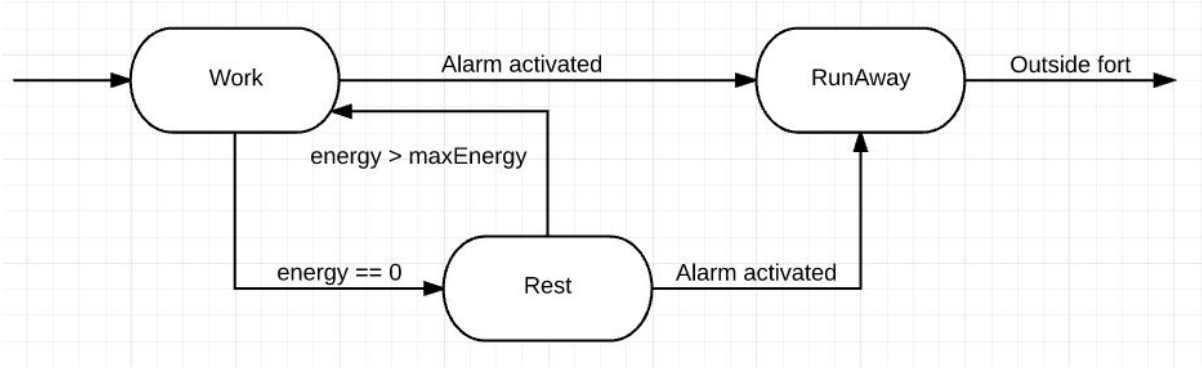


  El soldado tiene la finalidad de rescatar a todos los prisioneros de la fortaleza, pero cuando la alarma se activa huye de la fortaleza hacia la base y vuelve a la fortaleza si aún quedan prisioneros por rescatar.

  En el estado Rescue, el soldado buscará las puertas de la fortaleza para abrirlas y que así los prisioneros escapen, hasta que se dé la alarma o hasta que no quede ningún prisionero en la fortaleza.

  En el estado RunAway, el soldado buscará la puerta que tenga más cerca y saldrá corriendo hacia su base más cercana, y si queda algún prisionero por rescatar volverán a rescatarlos.

- **Prisionero**

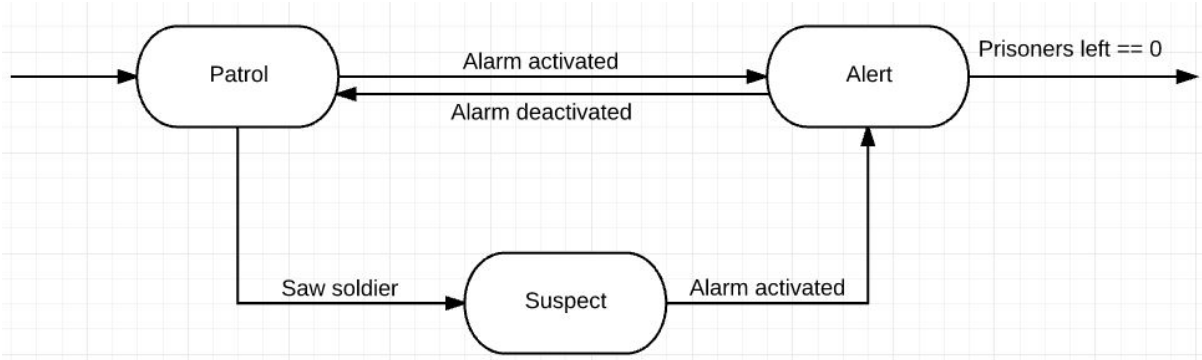


El prisionero estará trabajando o descansando, según la energía que tenga, indefinidamente hasta que suene la alarma, entonces correrá hacia la salida más cercana y intentará salir, si no puede, buscará la siguiente salida y intentara salir. Si la alarma se acaba y no han conseguido salir, volverán a su estado anterior.

En el estado Work, el prisionero trabajará moviendo unas cajas de un lado a otro dentro de la zona de trabajo, hasta que se queden sin energía y volver a la zona de descanso.
En el estado Rest, el prisionero se moverá aleatoriamente por la zona de descanso mientras recupera energía, hasta que se le dé la orden de ponerse a trabajar.

En el estado RunAway, el prisionero irá a la puerta más cercana para intentar salir de la fortaleza, si no lo consigue irá a la siguiente puerta, y así sucesivamente hasta que encuentre una salida o la alarma se desactive, en tal caso volverá al estado previo a que la alarma se activase.

- **Guarda**



El guarda se dedicará a patrullar por la fortaleza comprobando que las puertas estén cerradas, si ven a un soldado cerca lo perseguirán. Cuando ven una puerta abierta dan la alarma y todos los guardas estarán en alerta, buscando a soldados o prisioneros cercanos para perseguirlos e impedir que salgan de la fortaleza hasta que la fortaleza se desactive.

En el estado Patrol, como su nombre indica, el guarda se pondrá a patrullar por unos puntos específicos de la fortaleza de manera aleatoria, de vez en cuando comprobará que las puertas están cerradas, y si no es así dan la alarma. Si ven a un soldado cerca, pasan al estado Suspect.

En el estado Suspect, el guarda se pondrá a perseguir al soldado que ha visto indefinidamente hasta que el soldado salga fuera de la fortaleza. Si el guarda ve que el soldado abre una puerta, la intentará cerrar y dará la alarma.

En el estado Alert, el guarda buscará a los prisioneros o soldados que tenga más cerca y los perseguirá hasta que se hayan salido de la fortaleza, o, en el caso de los prisioneros, se haya desactivado la alarma. Cuando el guarda persigue a un prisionero intentará impedir que salga de la fortaleza cerrando la puerta antes de que se escape.

Conclusión

Al implementar esta aplicación lo primero que nos damos cuenta es en el gran coste que tiene un algoritmo de búsqueda de caminos si se usa de manera constante y sin ningún tipo de optimización. Por ello es muy importante usar lo menos posible dicho algoritmo con comprobaciones simples, como por ejemplo si sabes que no van a haber obstáculos entre un punto y otro, o si la distancia entre los puntos es muy pequeña, etc…

Además de esto se puede hacer un sistema de cacheado de caminos (explicado más arriba), y un sistema de peticiones de camino con programación multihilo, de esa manera se reduce el coste considerablemente.

El resultado es una aplicación con 30 agentes, con sus comportamientos específicos y con unos tipos de movimiento calculados en tiempo real, basados en la zona y en la distancia.

Bibliografía

Pathfinding & A* algorithm:

- https://www.raywenderlich.com/4946/introduction-to-a-pathfinding
- http://www.redblobgames.com/pathfinding/a-star/introduction.html
- https://en.wikipedia.org/wiki/A*_search_algorithm