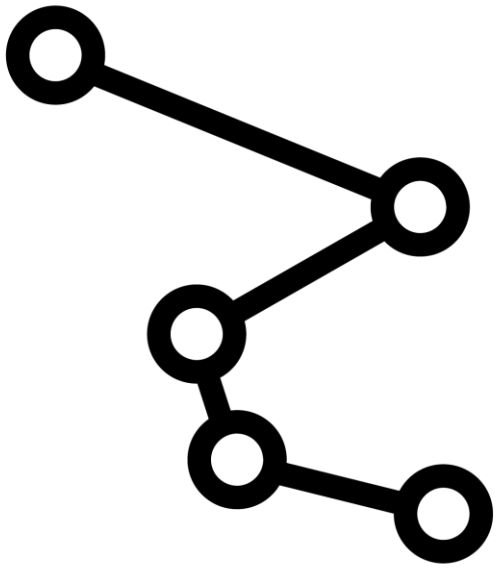

Práctica IA

STRIPS, BREADTH FIRST SEARCH, A*

Sergio Roselló | Inteligencia artificial | 05 de julio de 2017

Tabla de contenido

Búsqueda en amplitud 2

¿En que consiste? 2

¿Cómo funciona? 2

Método expandir 2

A* 2

¿En qué consiste? 2

¿Cómo funciona? 2

método expandir 3

STRIPS 3

¿En que consiste? 3

¿Cómo funciona? 3

Operadores 3

Estado 3

Algoritmo 3

Implementación 3

Clases compartidas entre A* y busqueda en amplitud 4

Estado 4

a* pathfinding 4

PathFinding 4

NodoPF 4

Distance 5

Búsqueda en amplitud 5

BusquedaAmplitud 5

Nodo 5

Mentes 6

Breath first mind 6

A mind* 6

STRIPS 6

Strips ..... ***¡Error! Marcador no definido.***

State 6

Operator 7

GoTo 7

Búsqueda en amplitud

¿EN QUE CONSISTE?

Dado un espacio de búsqueda en forma de árbol, la búsqueda en amplitud consiste en recorrer todos los nodos de una forma que prioriza a los nodos hermanos antes que a los nodos hijos. De esta forma, este algoritmo asegura encontrar una solución al problema si la hay, además, encuentra la solución de camino más corto. Una de las desventajas de este algoritmo de búsqueda es que su complejidad incrementa de forma exponencial.

¿CÓMO FUNCIONA?

Al ver que a medida que el espacio de búsqueda aumentaba, el algoritmo tardaba más y más, acabamos desarrollando una pequeña mejora que cambiaba drásticamente el tiempo de búsqueda. El algoritmo clásico, expande nodos y a lo sumo, comprueba que el nodo expandido no es padre del nodo del que se ha expandido. Esto es un control para prevenir bucles simples en el programa. Nosotros implementamos una matriz que simula el espacio de búsqueda. Cuando un nodo ya ha sido expandido, marcamos la casilla de la matriz perteneciente al nodo con un bit. Esto quiere decir, que el algoritmo no va a buscar en nodos por los que ya ha pasado y esto lo hace mucho más óptimo.

MÉTODO EXPANDIR

```
public virtual List<Nodo> Expandir(){
    List<Estado> estadosDerivados = Estado.Expandir();
    //Eliminamos bucles simples
    List<Nodo> nodosExpandidos = new List<Nodo>();
    foreach (var estado in estadosDerivados){
        if (Padre != null){
            if (!Padre.Estado.Equals(estado)){
                if(mapNodeStatus[(int) estado.Position.x, (int) estado.Position.y] == 0){
                    nodosExpandidos.Add(new Nodo(estado, this));
                    mapNodeStatus[(int)estado.Position.x, (int)estado.Position.y] = 1;
                }
            }
        }
        else{
            nodosExpandidos.Add(new Nodo(estado, this));
        }
    }
    return nodosExpandidos;
}
```

A*

¿EN QUÉ CONSISTE?

La búsqueda heurística, a diferencia de métodos de búsqueda no informada como BFS (breadth first search) o DFS (Depth first search) dispone de una función de evaluación, que debe medir la distancia hasta el objetivo. Esto significa que el algoritmo va a expandir los nodos más cercanos al objetivo.

¿CÓMO FUNCIONA?

El algoritmo A* mide las distancias desde el objetivo a todos los demás nodos. También mide las distancias del nodo actual a los demás nodos. De esta forma, sumando ambas distancias, podemos averiguar que nodo es el más cercano. Esta suma está representada mediante: $f^*(n) = g(n) + h^*(n)$

donde f*(n) es la distancia del nodo actual al objetivo, g(n) es la distancia del nodo actual al nodo n y h*(n) es la distancia del nodo n al objetivo. A medida que se expanden nodos, se insertan en una lista abierta los nodos expandidos ordenados según su f*. El algoritmo expande uno a uno los nodos de la lista abierta hasta que da con la solución. Este método de búsqueda te asegura una solución, si la hay.

MÉTODO EXPANDIR

```
public virtual List<NodoPF> Expandir(){
    List<Estado> estadosDerivados = Estado.Expandir();
    List<NodoPF> nodosExpandidos = new List<NodoPF>();
    foreach (var estado in estadosDerivados){
        if (Padre != null){
            NodoPF node = new NodoPF(estado, this);
            node.gCost = Padre.gCost + 1;
            node.hCost = Distance.EuclideanDistance(estado.Position, PathFinding.final.Estado.Position);
            nodosExpandidos.Add(node);
        }else{
            NodoPF node = new NodoPF(estado, this);
            node.gCost += 1;
            node.hCost = Distance.EuclideanDistance(estado.Position, PathFinding.final.Estado.Position);
            nodosExpandidos.Add(node);
        }
    }
    return nodosExpandidos;
}
```

STRIPS

¿EN QUE CONSISTE?

STRIPS (Stanford Research Institute Problem Solver) consiste en un inicio y una meta. Ambas descritas con una serie de propiedades. STRIPS llega a la meta aplicando operadores (conjunto de precondiciones, adiciones y eliminaciones) al estado actual modificando así su estado (adiciones, eliminaciones) para llegar al estado meta tras varias iteraciones.

¿CÓMO FUNCIONA?

Operadores

Se encargan de cambiar el estado en el que se encuentra el algoritmo en cada iteración.

En este caso en concreto, solo necesitamos un operador, GoTo, que añade un objeto a la lista de añadidos. No añade nada a la lista de eliminados porque en ningún caso el operador se verá en situación de eliminar un objeto recogido.

Estado

Representación del estado en el que se encuentra el algoritmo en cada iteración.

Algoritmo

Este se encarga de ejecutar el algoritmo de STRIPS para encontrar la solución al problema, si la hay.

Implementación

Detalles de código, como funciona y su estructura.

CLASES COMPARTIDAS ENTRE A* Y BUSQUEDA EN AMPLITUD

Estado

Tiene un constructor y 5 métodos.

Constructor

Se le pasa la posición actual, mapa y acción. Las asigna a sus variables.

EsMeta

Comprueba si la casilla en la que se encuentra el estado es meta o no.

IsPositionInMap

Comprueba si la posición que se le pasa por parámetros es parte jugable del mapa

Equals

Devuelve True si la posición de los dos estados es la misma.

CanMoveToDirection

Comprueba si se puede acceder a la dirección pasada por parámetros.

Expandir

Devuelve una lista de todos los estados a los que se puede llegar con un movimiento desde ese estado.

A* PATHFINDING

PathFinding

Tiene 2 métodos y un constructor.

Constructor

Crea una lista de nodos llamada abiertos. En esta lista, se guardarán los nodos por expandir.

Buscar

Recibe dos Estados, el inicial y el final. Inicializa ambos y añade el inicial a la lista de abiertos. Mientras existan nodos en la lista de abiertos, expandir el primero de la lista y eliminarlo de la misma. Para cada nodo expandido, añadir a la lista de abiertos todos sus hijos. Ordenar la lista de abiertos según la función heurística f*. Cuando se llega a la meta, devuelve una lista de nodos ordenados de forma reversa.

EsMeta

Comprueba si el estado actual es un estado meta.

NodoPF

Tiene 2 métodos y un constructor.

Constructor

Recibe un estado y su padre y se los asigna.

Expandir

Calcula los estados derivados de ese nodo y los añade a la lista de expandidos. Devuelve la lista de nodos expandidos.

ToString

Override del método ToString por defecto.

Distance

Tiene dos métodos.

ManhattanDistance

Recibe dos Vector2 y calcula la distancia de manhattan entre ambos.

EuclideanDistance

Recibe dos Vector2 y calcula la distancia euclídea entre ambos.

BÚSQUEDA EN AMPLITUD

BusquedaAmplitud

Tiene 2 métodos y un constructor.

Constructor

Crea una lista de nodos llamada abiertos. En esta lista, se guardarán los nodos por expandir.

Buscar

Recibe el estado inicial y mientras la lista de abiertos no se quede vacía, la va vaciando y comprobando si el nodo que ha sacado es meta. Si es meta, devuelve el nodo actual, sino, continua expandiendo el nodo que ha sacado y metiendo sus nodos hijos en la lista de abiertos.

EsMeta

Comprueba si el estado actual es un estado meta.

Nodo

Tiene 2 métodos y un constructor.

Constructor

Recibe un estado y su padre y se los asigna.

Expandir

Calcula los estados derivados de ese nodo y los añade a la lista de expandidos si es la primera vez que han sido expandidos. Devuelve la lista de nodos expandidos.

ToString

Override del método ToString por defecto.

MENTES

Breath first mind

Tiene 1 método y un constructor.

Constructor

Inicializa una nueva búsqueda en amplitud.

GetNextMove

Recibe la posición actual y el mapa. Inicializa la matriz que simula el espacio de búsqueda si no estaba ya inicializada y devuelve la siguiente acción.

A* mind

Tiene 1 método y un constructor.

Constructor

Inicializa una nueva búsqueda A*.

GetNextMove

Si no existen resultados que devolver, los busca. Devuelve la dirección del primer nodo de la lista de resultados. Elimina el primer nodo de la lista de resultados.

STRIPS

State

Tiene dos constructores y un método.

Constructor sin parámetros

Inicializa la lista de propiedades. Sirve para crear estados vacíos.

Constructor con un parámetro

Inicializa la lista de propiedades a la pasada por parámetros. Sirve para crear un estado de otro ya existente, para tener un registro de los estados por los que ha ido pasando STRIPS.

Contains

Comprueba que el estado actual contiene las propiedades pasadas por parámetro. Si no las contiene, devuelve False.

Operator

Tiene un constructor y 8 métodos.

Constructor

Inicializa todas las listas que van a ser necesarias para el objeto Operador, incluyendo su posición, lista de añadidos, lista de precondiciones y lista de eliminados.

Produces

Comprueba si el operador contiene la propiedad pasada por parámetros en la lista de adiciones. Si la contiene, devuelve True.

Apply

Aplica un operador al mundo haciendo una copia del estado pasado por parámetros y alterando sus listas de añadidos y eliminados.

Delete

Elimina las propiedades del estado (que forman el mundo de ese estado) contenidas en la lista de eliminaciones del operador.

Add

Añade las propiedades del estado (que forman el mundo de ese estado) contenidas en la lista de eliminaciones del operador.

IsAplicable

Comprueba si el operador es aplicable sobre un estado recorriendo su lista de propiedades. Si el operador no es aplicable, devuelve false.

Get{Add, Precondition, Elimination}List

Getters, devuelven la lista solicitada.

GoTo

Clase hija de Operator, por tanto, tiene acceso a sus métodos y variables.

Constructor

Recibe un entero que simula el objeto recogido en el tablero de juego, una posición y una lista de precondiciones. Asigna estos tres parámetros a sus variables.