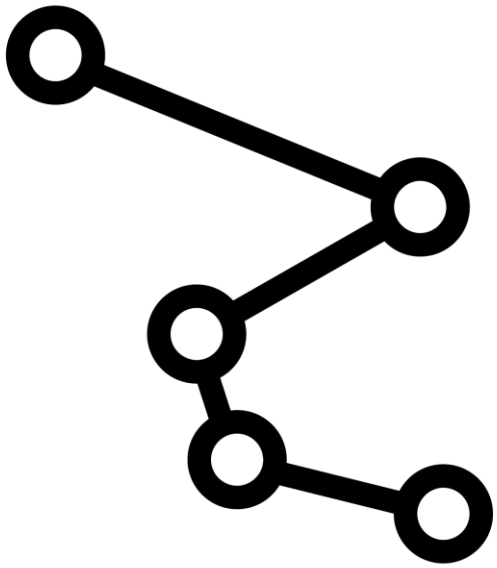


Práctica IA

STRIPS, BREADTH FIRST SEARCH, A*

Sergio Roselló | Inteligencia artificial | 05 de julio de 2017

Tabla de contenido

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 override List<NodeSearch> Expandir(){
    List<Estado> estadosDerivados = Estado.Expandir();
    //Eliminamos bucles simples
    List<NodeSearch> nodosExpandidos = new List<NodeSearch>();
    foreach (var estado in estadosDerivados){
        if (Padre != null){
            if (!Padre.Estado.Equals(estado)){
                if(BusquedaNoInformada.MapNodeStatus[(int) estado.Position.x, (int) estado.Position.y] == 0){
                    nodosExpandidos.Add(new NodeSearch(estado, this));
                    BusquedaNoInformada.MapNodeStatus[(int)estado.Position.x, (int)estado.Position.y] = 1;
                }
            }
        }
        else{
            nodosExpandidos.Add(new NodeSearch(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 override List<NodePathFinding> Expandir(){
    List<Estado> estadosDerivados = Estado.Expandir();
    List<NodePathFinding> nodosExpandidos = new List<NodePathFinding>();
    foreach (var estado in estadosDerivados){
        if (Padre != null){
            NodePathFinding node = new NodePathFinding(estado, this);
            node._gCost = Padre._gCost + 1;
            node._hCost = Distance.EuclideanDistance(estado.Position, PathFinding.final.Estado.Position);
            nodosExpandidos.Add(node);
        }else{
            NodePathFinding node = new NodePathFinding(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. Es importante remarcar que es un algoritmo recursivo.

```
public List<Operator> Search(State currentState, List<string> goals) {
    List<Operator> plan = new List<Operator>();

    while (!currentState.Contains(goals)){
        foreach(string property in goals){
            if(!currentState.Contains(property)){
                List<Operator> operatorsThatProduceProperty = getOperatorsWithProperty(property);
                if (operatorsThatProduceProperty.Count == 0)
                    return null;

                foreach (Operator oprtr in operatorsThatProduceProperty){
                    List<Operator> primePlan = Search(currentState, oprtr.getPreconditionList());
                    if (primePlan == null)
                        return null;
                    foreach (Operator op in primePlan){
                        op.Apply(currentState);
                        plan.Add(op);
                    }
                    oprtr.Apply(currentState);
                    plan.Add(oprtr);
                }
            }
        }
    }
    return plan;
}
```

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 3 métodos y un constructor.

Constructor

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

Buscar

Recibe dos Vector2, la posición inicial y la posición final. 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.

Reset

Elimina los nodos restantes (si los hay) de la lista de abiertos y setea las variables inicial y final a null. Le llama para evitar errores al pasar de una meta a otra.

NodoPathFinding

Hereda de Node, tiene 1 método y un constructor.

Constructor

Llama al constructor de Node que 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. Realiza los cálculos necesarios para A*. Devuelve la lista de nodos expandidos.

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

Hereda de BusquedaNoInformada. Solamente tiene el método Buscar. Hemos optado por esta solución porque viene bien a la hora de añadir nuevos algoritmos de búsqueda como DFS. Simplemente cambias el método Buscar.

Buscar

Recibe dos Vector2, la posición inicial y la posición final. Mientras existan nodos a expandir en la lista de abiertos, los expande y vuelve a añadirlos a la lista de abiertos. Si se ha llegado a una solución, obtiene el camino recorrido para llegar a ese Nodo, lo mete en una lista, le da la vuelta y lo devuelve. Se le da la vuelta porque estamos obteniendo la lista desde la meta hasta el inicio, por tanto, para averiguar el camino que debe hacer nuestro agente, del inicio a la meta, debemos devolver la lista al revés.

NodeSearch

Hereda de Node. Tiene 1 método y un constructor.

Constructor

Llama a el constructor del padre, que recibe un estado y su padre. Luego, 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.

MENTES

Breath first mind

Tiene 1 método y un constructor.

Constructor (privado)

Hemos aplicado el patrón Singleton porque solo debería existir una instancia de este objeto a la vez. Inicializa una nueva búsqueda en amplitud.

GetNextMove

Recibe la posición actual y la posición final. Depende del estado de la búsqueda devuelve la dirección a la que tiene que ir el agente o genera un nuevo plan para poder continuar con la búsqueda. Entra en este caso cuando en medio de una ejecución STRIPS, el agente llega a una meta parcial y tiene que cambiar a la siguiente meta. En caso extremo, cuando no quedan nodos en la lista de resultados, devuelve Move.MoveDirection.None.

A* mind

Tiene 1 método y un constructor.

Constructor (privado)

Hemos aplicado el patrón Singleton porque solo debería existir una instancia de este objeto a la vez. Inicializa una nueva búsqueda heurística A*.

GetNextMove

Recibe la posición actual y la posición final. Depende del estado de la búsqueda devuelve la dirección a la que tiene que ir el agente o genera un nuevo plan para poder continuar con la búsqueda. Entra en este caso cuando en medio de una ejecución STRIPS, el agente llega a una meta parcial y tiene que cambiar a la siguiente meta. En caso extremo, cuando no quedan nodos en la lista de resultados, devuelve Move.MoveDirection.None.

STRIPS

State

Tiene dos constructores y dos métodos.

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 todas las propiedades pasadas por parámetro. Si no las contiene, devuelve False.

Contains (sobrecarga)

Comprueba que el estado actual contenga la propiedad pasada por parámetro.

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.

Strips

Hereda de MonoBehaviour por tanto puede hacer uso de Start. Consta de 4 metodos.

Start

Se llama a Start una vez, al iniciar el entorno. Se encarga de Iniciar el algoritmo.

Init

Setea todo lo que necesita STRIPS para funcionar, incluyendo la generación de todos los operadores y búsqueda de la meta en el entorno de Unity.

Search

Es el core del programa. Se encarga de encontrar una solución al problema mediante llamadas recursivas. Mientras no se cumplan todas las propiedades (goals) del estado actual, para cada propiedad que no se cumple, si el estado actual no contiene la propiedad, busca todos los operadores que produzcan tal propiedad. Si no hay, no se puede solucionar el problema. Si sí que hay operadores que producen esa propiedad, llama de manera recursiva a Search con el estado actual y las precondiciones del operador actual como goals. Aplica al estado actual el operador y añade el operador a la lista que contiene los operadores para cada plan que si que ha funcionado. Finalmente, cuando se cumplan todas las precondiciones del estado actual, devuelve el plan.

GetOperatorsWithProperty

Devuelve una lista de operadores que cumplen cierta propiedad.

OTRAS CLASES

iMind

Nuestro GetNextMove no necesita recibir el mapa, pero si la posición final.

Move

Hemos añadido una cabezera a la clase Move para que pida como dependencia el script Strips

Hemos añadido un campo (None) en el enum MoveDirection para que espere cuando llega al final de la lista de movimientos posibles.

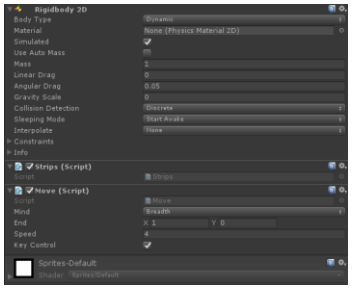
Hemos creado un enum MindType para poder cambiar en el editor de forma sencilla entre mentes. (tipos de algoritmos de búsqueda)

Hemos hecho uso del método OnEnable proporcionado por Unity para ayudar a la configuración del script en el editor.

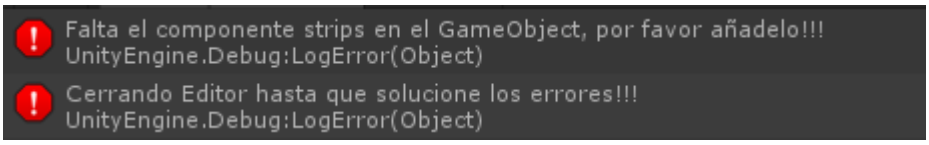
Hemos cambiado los Debug.Log de los estados que habilitan el movimiento para que impriman también la posición en la que están.

COMO PREPARAR EL EDITOR

IF NOT



THEN



RESULTADOS OBTENIDOS

Breadth first search

Cuando no habíamos optimizado el algoritmo, este tardaba relativamente poco en tablas de juego pequeñas, pero a medida que aumentabas el tamaño de la tabla, el tiempo que tardaba el algoritmo en encontrar una solución se prolongaba exponencialmente. Cuando implementamos la matriz, el tiempo de búsqueda se redujo muchísimo tanto en tablas pequeñas como en tablas grades.

A*

Teniendo en cuenta el tipo de algoritmo de búsqueda, (heurístico) creemos que tanto el tiempo de búsqueda como el tiempo de búsqueda es óptimo.

STRIPS

Este algoritmo encuentra una solución al problema, no necesariamente la mejor ni la más óptima, pero si existe una solución, la encuentra. No podemos saber si el algoritmo tarda mucho o poco porque no tenemos nada con lo que compararlo, pero el tiempo que tarda nunca nos ha hecho sospechar, por tanto, entra dentro de nuestras expectativas de tiempo.