Universidad
Rey Juan Carlos

Práctica 1

Desarrollo de juegos con Inteligencia Artificial - Grado en Diseño y Desarrollo de Videojuegos

Grupo 8

Jose Mª Soriano Villalba
Juan Coronado Gómez

ETSII
Escuela Técnica Superior de Ingeniería Informática

Universidad
Rey Juan Carlos

ÍNDICE

Universidad
Rey Juan Carlos

Resolución ejercicio 1

Para la implementación del algoritmo A*, se han creado dos clases nuevas: AStarMind y Nodo.

Clase Nodo

Para poder desarrollar el árbol de búsqueda del algoritmo, se ha creado esta clase nodo, que recoge toda la información necesaria para cada "paso" que pueda dar el personaje:

- Fila
- Columna
- Coste acumulado
- Heurística
- Función *f*
- Si es meta
- Datos acerca de su casilla

Para crear cada uno de estos nodos, se declara un constructor que recibe una serie de parámetros y después calcula la función *f* de cada uno de ellos.

Clase AStarMind

En esta clase se desarrolla como tal el algoritmo de búsqueda. Se comienza comprobando si ya hay un camino calculado; en caso afirmativo, se ejecuta. La primera vez que entra en la función *GetNextMove* no hay ninguna planificación hecha, por lo que se continúa a la función *AlgoritmoBusqueda*.
Esta clase tiene una serie de atributos necesarios para la correcta ejecución del algoritmo:

- Lista abierta de nodos. Esta lista se utiliza para añadir los nodos que se obtienen mediante la función expandir, y así explorarlos poco a poco.
- Lista cerrada de nodos. Esta lista se utiliza para tener un registro de todos los nodos que se visitan a lo largo del algoritmo, evitando de esta forma que se produzcan ciclos.
- Coste acumulado. Mediante esta variable, se inicializa el coste del primer nodo a 0.
- Camino, lista de movimientos. Esta lista es la solución obtenida a través del algoritmo de búsqueda, y se va recorriendo para poder obtener los movimientos del personaje hasta la meta.

- Nodo actual. Esta variable se utiliza para facilitar la comprobación de los ciclos.

El cuerpo de la función *AlgoritmoBusqueda,* primero construye un nodo con la información que se pasa como parámetro a la función, que pertenece a la casilla inicial (se calcula su heurística mediante sus filas y columnas y las de la meta a través de una función auxiliar) y se añade a la lista abierta. Después, se entra en un bucle que se ejecuta de manera indefinida hasta que se encuentra un nodo meta:

1. Mediante el método *WalkableNeighbours* de la clase CellInfo, se lleva a cabo la función expandir.
2. A través de los datos obtenidos, se construyen los nodos correspondientes mediante un bucle.
3. Antes de añadir dichos nodos a la lista abierta, se comprueba si han sido explorados anteriormente recorriendo la lista cerrada.
4. Después, se ordena la lista abierta mediante el método *Sort*. Esta función requiere un comparador como parámetro, que ha sido implementado en una clase independiente, en un método llamado ***CompareNodesByF***. Este método se encarga de comparar la función *f* de dos nodos y devolver 1 o -1, dependiendo de cual sea mayor. En caso de empate, se consulta la heurística de los nodos.
5. Una vez ordenada la lista, se escoge el primer nodo y se comprueba si es meta (mediante una función auxiliar, que comprueba la fila y la columna y los compara con la meta). En caso negativo, se vuelve a repetir todo el proceso. Si resulta ser la meta, se abandona el bucle y se calcula el camino que debe realizar el personaje para llegar a esa casilla.

El método *CalcularCamino* recibe como parámetro el nodo meta, y explora el nodo padre de manera sucesiva hasta que se topa con un nodo cuyo padre sea nulo. Mediante la comparación del valor de la fila y la columna de un nodo con los valores de su padre, se deduce si el movimiento debe ser derecha, izquierda, arriba o abajo. Todos estos movimientos se almacenan en la lista de movimientos, que se recorrerá de final a inicio a la hora de realizarse.

Universidad
Rey Juan Carlos

Resolución ejercicio 2

Para la resolución de este segundo ejercicio, se ha modificado ligeramente el código de la clase BoardInfo y se ha creado otra clase nueva, OnLineEnemies.

Clase BoardInfo

Las únicas modificaciones que se han realizado en esta clase se han producido en la función encargada de generar los enemigos, *LayoutEnemiesAtRandom*. En el cuerpo de la función, cada vez que se crea un enemigo, se añade que el objeto se incluya en la lista de enemigos, declarada como atributo.

```
Enemies.Add(enemy);
```

Además, tras haber terminado de crear todos los enemigos, se pasa dicha lista al controlador del movimiento, para que se pueda construir un algoritmo que conozca la posición de los enemigos.

```
GameObject.FindGameObjectWithTag("GameController").GetComponent<OnLineEnemies>().AddEnemies(Enemies);
```

Clase OnLineEnemies

Esta clase se ha elaborado mediante una copia y modificación de la ya explicada AStarMind. Lo primero que se añade es el atributo objetos, una lista de GameObjects que recibe la lista de enemigos mediante la función *AddEnemies*.
Después, en la función *GetNextMove*, antes de llevar a cabo la planificación para la búsqueda de la salida, se comprueba cuántos enemigos hay activos, para primero eliminarlos a ellos. El algoritmo que ejecuta esta búsqueda es el método *BusquedaEnemigos*.

1. Primero, comprueba qué enemigo está más cerca, mediante el cálculo de la heurística de ambos utilizando la posición de dichos enemigos.
2. Una vez seleccionado el objetivo, se crea el nodo inicial de la misma forma que en el algoritmo A*.
3. Después, se expande el nodo inicial y se construyen los nodos hijos.
4. Tras ello, se añaden a la lista abierta y esta se ordena, escogiendo el nodo más prometedor, es decir, el que queda en primer lugar. Dicho nodo es al que se va a ir en el siguiente movimiento,
5. Una vez escogido el nodo más prometedor, mediante la comparación de las filas y las columnas, se elige la dirección del próximo movimiento.

Este algoritmo, búsqueda por ascenso de colinas, se ejecuta las veces que sea necesario para acabar con todos los enemigos de la escena. Después, el personaje inicia su búsqueda de la salida mediante A*.

Resultados obtenidos

En cuanto a los resultados obtenidos al final de esta práctica, destaca la consecución de un programa en Unity, en el que mediante el código y los algoritmos explicados anteriormente, se ha conseguido tener mapa, con un personaje inicial, dos enemigos y una meta, dónde el personaje se mueve en búsqueda de eliminar al enemigo más cercano y una vez eliminados ambos enemigos, se dirige a la meta, dónde termina la ejecución del programa alcanzando dicho nodo.