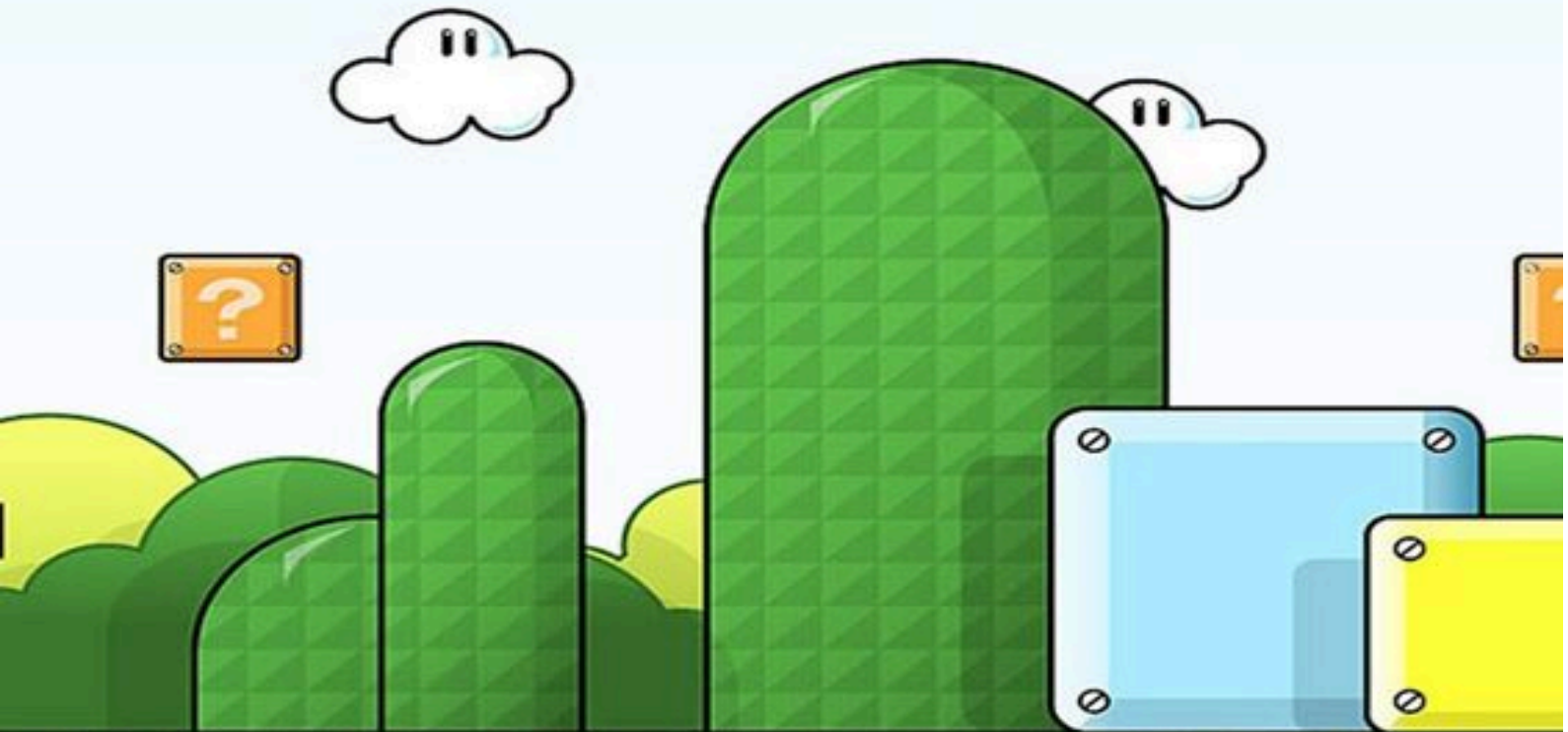
Práctica 1
Desarrollo de Juegos con Inteligencia Artificial.

GD&DV 2024/2025 Grupo A

Fecha: 11-11-2024

Autores: Julio Hornos Rodriguez, Alba Poza Vela,

Purificación Méndez Peñalver



Índice

Contexto de la práctica

En esta práctica se busca guiar a un agente a través de un laberinto para que: En un inicio, llegue a la salida; luego, que recoja los tesoros y se dirija a la salida y, por último, que atrape a los enemigos que se mueven por el laberinto además de realizar las dos acciones anteriores.

Para hacerlo se necesitaban incluir dos scripts que definen el movimiento del agente:

- Un script que gestiona el agente inteligente, implementando la interfaz *INavigationAgent*, encargándose de seleccionar los objetivos y utilizar el algoritmo de navegación para encontrar los caminos óptimos.
- Un script que define el algoritmo de navegación, implementando la interfaz *INavigationAlgorithm*, qué deberá ser capaz de calcular la ruta óptima entre dos puntos del espacio de estados.

Para ello se utilizaron las clases WorldInfo y CellInfo, que indican cómo se conforma la escena , y que sirven de guía para mover al agente por el escenario.

Desarrollo de la Práctica

Búsqueda del camino de salida

El problema planteado consiste en que desde cualquier posición en la escena, el agente debe ser capaz de encontrar la meta de forma óptima.

Para ello, se ha decidido implementar un algoritmo A* que recibe la posición inicial del agente y devuelve el camino óptimo que debe seguir para llegar al objetivo, es decir, a la salida.

Se ha decidido usar el algoritmo A* ya que se trata de un algoritmo de búsqueda con complejidad no muy alta y que permite obtener el camino óptimo.

A* utiliza una función de evaluación que equilibra el coste acumulado desde el inicio con una estimación del coste restante y busca, como se ha mencionado antes, el camino óptimo entre un nodo inicial y un nodo objetivo en un grafo o red.

Clase Cell Node

Para evaluar el camino se ha creado la clase intermedia Cell Node que servirá como clase para guardar la información necesaria sobre los nodos a evaluar:

- La información de la celda en cuestión. (CellInfo)
- El CellNode padre del que viene.
- El valor total que se usa para ordenar los nodos según la prioridad.
- El valor de la heurística.
- El valor acumulado.

Además, para poder utilizar los métodos orderby y short se ha tenido que definir un método comparador para la clase Cell Node para que así se pudiera filtrar y ordenar correctamente.

A partir de la clase Cell Node podemos implementar el algoritmo A*.

Se usa una cola de prioridad para los nodos evaluados y una cola de visitados que guardará todos los nodos por los que el algoritmo ha pasado para no volver a pasarlos.

El nodo inicial se añadirá directamente a la lista de visitados y a partir de ahí se empezarán a evaluar los vecinos que rodean al nodo actual.

A continuación, se calcularán las heurísticas.

Usamos la distancia de manhattan por el movimiento del personaje dado que es la que más representa el movimiento siendo el movimiento únicamente en los ejes x e y.

Seguido, se añadirán a la cola de prioridad los nodos en orden de sus valores y si es un nodo walkable.

Esta iteración se repetirá hasta encontrar el nodo objetivo (en este caso la meta) y se recuperará el camino a seguir accediendo al nodo padre de todos los Cell Nodes, para luego invertirlo porque si no llegaría en sentido contrario.

Recogida de tesoros

El problema que se plantea en este apartado es que desde cualquier posición en la escena, el agente debe ser capaz de recoger los cofres y posteriormente alcanzar la meta, según el camino óptimo.

Para ello, se ha decidido reutilizar el algoritmo A* junto con una búsqueda por subobjetivos para encontrar el camino óptimo desde la posición inicial al primer subjetivo, desde un subjetivo a otro y desde el último subjetivo a la salida. Esta decisión es debido a que al tener varios cofres resulta más eficiente y sencillo hacer una búsqueda por subobjetivos donde cada cofre y finalmente la meta representan cada uno un objetivo distinto.

Dado que el funcionamiento base del agente para la selección del objetivo actual era obteniéndose del array de objetivos, lo que se ha decidido hacer es introducir las celdas en las que se encuentran los cofres en el array de objetivos, y reordenar estos en base a la distancia con la posición en la que se encuentra el agente.

Además, se ha decidido implementar una reordenación de los subobjetivos cada vez que se alcanza un cofre, de forma que el agente se dirija al cofre más cercano desde esta nueva posición en la que se encuentra.

Finalmente, una vez que se han alcanzado todos los subobjetivos (los cofres), se pasa a la siguiente fase en la que el objetivo es la salida, para ello se añade la salida a la lista de objetivos como ocurría en el apartado anterior.

Durante la realización del siguiente apartado, se evidenció la aparición de unos cofres ‘fantasma’. Es decir, aquellos cofres que se recogían durante la persecución de los enemigos y que aparecían dentro del array de objetivos aunque ya no estuvieran visibles en pantalla.

Para solucionar esto, se decidió comprobar la información de las celdas en la que se encuentran los cofres de la escena, de forma que la condición para introducirlas como objetivos es que el tipo de la celda sea ‘Treasure’, dejando fuera aquellas celdas ahora definidas como ‘Floor’ (suelo).

Atrapar a los enemigos

El problema que se plantea en este apartado consiste en que, desde cualquier posición en la escena, el agente debe ser capaz de atrapar a los zombies, recoger los cofres y por último alcanzar la meta.

Se ha decidido utilizar el mismo algoritmo usado para los cofres (A* con subobjetivos), pero, esta vez, declarando un horizonte límite en el que se evalúa la heurística de las casillas ante el horizonte dado que al ser un objetivo en movimiento no tiene sentido calcular el camino entero para solo coger el primer paso (que es el que cogemos).

Dependiendo de la distribución de los muros el agente puede llegar a mesetas (caminos sin salida) y no llegar a encontrar el objetivo (los zombies), por ello se ha optado por poner un horizonte amplio para que no suponga a un problema para la búsqueda del agente.

Además para llevar el conteo y situación de los zombies se ha declarado un array de enemigos donde se evalúa que todos los elementos internos son zombies (no han sido pillados) ya que cuando el agente pilla a un zombie el elemento pasará de ser tipo zombie a tipo floor y cuando busquemos los zombies restantes se omitirán debido a que serán tipo floor.

Una vez terminada la búsqueda de los zombies, nuestro agente fija como objetivo los cofres y procederá a generar el path óptimo de todos los cofres y una vez recogidos todos, generará el path para llegar a la meta.

Conclusión

En la práctica se han reafirmado los contenidos aplicados a nivel teórico obtenidos en clase y a partir de cada uno de los apartados se ha conseguido aplicar estos conocimientos a nivel práctico mediante un motor de desarrollo como es Unity.