Practica 1
Búsqueda Heurística

Diseño y Desarrollo de Videojuegos - Ingeniería del Conocimiento IA

Promoción 2023-2024

Ivan Rodriguez Lozano. Grupo 3.3

Índice

Características de implementación generales

En el prefab pathController de ambas escenas(PathFinder y Enemies), hay un script ‘PathFinder.cs’, el cual, tiene un enum gracias al cual puedes elegir que tipo de algoritmo quieres utilizar para la búsqueda del camino. Una vez iniciada la partida, se añade al objeto el script correspondiente al algoritmo elegido.

También hay un script ‘Node.cs’, común a todas las implementaciones de los diferentes algoritmos. Estos nodos tienen una posición un padre y un goal.
Estos nodos se implementan sustituyendo a las celdas para facilitar el trabajo y gestión de las listas dentro de la ejecución de los algoritmos.

El canvas tiene un script ‘CanvasController.cs’, que gestiona los elementos del UI que muestran por pantalla el tiempo que tarda el algoritmo en encontrar el camino, la dirección en la que se mueve el character y los movimientos que le quedan hasta alcanzar goal.
También se encarga de gestionar el inicio de la búsqueda, mediante el botón de la escena llamado ‘Start’, con el que se pone a true o false la variable startSearchin de los algoritmos.

Se ha añadido una cámara virtual del paquete de Unity Cinemachine, que sigue al character mientras se mueve por la escena, para facilitar la visualización de este.

Descripción de A*

El algoritmo utilizado en este script es una implementación del algoritmo A* (A-Star) para encontrar el camino más corto entre un punto de inicio y uno o varios objetivos en un tablero de juego.
A*, es un algoritmo de búsqueda informada que utiliza heurísticas para encontrar el camino óptimo. El algoritmo mantiene dos listas: una lista abierta de nodos que están siendo considerados para la expansión y una lista cerrada de nodos que ya han sido evaluados. A cada paso, el algoritmo selecciona el nodo con el costo total más bajo (fCost), que es la suma del costo acumulado desde el nodo inicial (gCost) y una estimación del costo hasta el objetivo (hCost). Luego expande este nodo y revisa sus vecinos, continuando así hasta que se alcanza el objetivo o no quedan nodos por evaluar.

Características de la implementación A*

El algoritmo de A* se implementa de dos formas distintas, con una búsqueda del camino OffLine en la escena de PathFinding donde tiene que llegar a un goal y una búsqueda OnLine en al escena de Enemies donde tiene que atrapar varios enemigos que se mueven por la escena antes de alcanzar el goal.

Ambos scripts(OnLine y OffLine) heredan de 'AStarPathFinder.cs' que a su vez hereda de 'AbstractPathMind.cs'. En este script se encuentran las variables y métodos comunes a ambas implementaciones, como las listas de nodos abiertos y cerrados, las funciones heurísticas o el método de ordenar la lista de abiertos. También existe un método abstracto llamado 'FindPath()', encargado de implementar el A* y buscar el camino óptimo, que tiene diferentes implementaciones en ambos casos.

A su vez en ambas implementaciones, encontramos como hemos dicho antes el método 'FindPath()' y el metodo 'GetNextMove()', heredado de 'AbstractPathMind.cs', el cual se usa para determinar el siguiente movimiento del character.

OffLine:

En la implementación OffLine de A*, se busca el mejor camino hasta goal una sola vez y antes de que el character comience a moverse. Si no se ha encontrado un camino, la lista de movimientos 'pathMoves()' estará vacía, por lo que el character no se moverá, si se ha encontrado un camino, el character comenzará a moverse en la dirección correspondiente hasta llegar a goal.

OnLine:

En la implementación OnLine, añadimos al algoritmo A* una profundidad (en este caso de 4, pero esta puede cambiarse), ya que que los enemigos no tienen una posición fija si no que se van moviendo por el mapa, buscar el camino completo no es óptimo del todo.
En esta implementación la variable de pathFound se elimina y se añade una variable depth.
Mientras que la lista de abiertos tenga algún nodo y la profundidad sea mayor que 0, se escogen los movimientos óptimos hasta el goal.
Si la lista de abiertos es menor o igual que cero significa que no hemos encontrado un camino durante la expansión de nodos, por lo que se sale del bucle y el character no se mueve.
En esta implementación la búsqueda se realiza cada vez que el character se mueve.

Descripción de Búsqueda no informada en amplitud(BFS)

El algoritmo de búsqueda en amplitud (BFS por sus siglas en inglés, Breadth-First Search) es un algoritmo de búsqueda no informada que se utiliza para recorrer o buscar en un grafo o árbol. A diferencia de A*, BFS no utiliza heurísticas para estimar la distancia al objetivo, sino que se expande uniformemente en todas las direcciones desde el nodo inicial.

El algoritmo mantiene una estructura de datos llamada "cola" (queue), que es una colección de nodos que aún no han sido visitados. Comienza visitando el nodo inicial y agregándolo a la cola. Luego, en cada iteración, el algoritmo extrae un nodo de la cola y examina todos sus nodos vecinos. Si un vecino no ha sido visitado antes, se marca como visitado y se agrega a la cola para su posterior exploración. Este proceso se repite hasta que se encuentre el objetivo o hasta que la cola esté vacía, lo que indica que no hay más nodos por explorar y, por lo tanto, no hay camino hacia el objetivo desde el nodo inicial.

Características de la implementación BFS

Para este algoritmo contamos con un script ‘UninformedSearch.cs’ en el que como en el A*, este hereda de ‘AStarPathFinder.cs’ y en él encontramos variables y métodos comunes de este algoritmo para varias implementaciones, por si en un futuro se quiere hacer una implementación OnLine. A diferencia que en el A*, no encontramos ninguna funcion heuristica y en vez de listas se utiliza una cola para almacenar los nodos a evaluar y un HashSet para los nodos visitados. HashSet es una estructura de datos en C# que representa una colección de elementos únicos, donde el orden de los elementos no está definido.

OffLine:
Luego, también tenemos el script de ‘UninformedSearch_OffLine.cs’, el cual es el que se añade al path controller en caso de ser escogido y el encargado de implementar el método ‘FindPath()’ que busca el camino óptimo y ‘GetNextMove()’ que mueve al character.
Buscará el camino óptimo antes de que el character comience a moverse y la ejecución de este en un entorno con enemigos sería ineficiente.

OnLine:

En la implementación Online, se seguirán los mismos pasos para encontrar el camino que en la implementación OffLine, pero se ejecutará el algoritmo en cada ciclo y este contempla a los enemigos.

Resultados

A* OffLine:

- Semilla 2020: Camino no encontrado, en 0ms y 0 movimientos.
- Semilla 2021: Camino encontrado, en 66ms y 7 movimientos.
- Semilla 2022: Camino encontrado, en 780ms y 25 movimientos.
- Semilla 2023: Camino encontrado, en 261ms y 19 movimientos.
- Semilla 2024: Camino encontrado, en 68ms y 9 movimientos.

Con estos resultados, se puede apreciar que el costo de tiempo del algoritmo, va ligado a la cantidad de movimientos que se tienen que realizar, o lo que es lo mismo, la distancia a la que se encuentra el nodo meta.

A* OnLine:

- **Semilla 2020:**
    - Profundidad 10: Camino hasta los enemigos alcanzado en 44 movimientos y 1107ms, camino hasta el goal no alcanzado.

- **Semilla 2021:**
    - **Profundidad 10**: Enemigos eliminados y goal encontrado en 70 movimientos y 1257ms
    - **Profundidad 8**: Enemigos eliminados y goal encontrado en 70 movimientos y 921ms
    - **Profundidad 6**: Enemigos eliminados y goal encontrado en 70 movimientos y 850ms

- **Semilla 2022:**
    - **Profundidad 10**: Enemigos eliminados y goal encontrado en 53 movimientos y 250ms
    - **Profundidad 8**: Enemigos eliminados y goal encontrado en 53 movimientos y 290ms
    - **Profundidad 6**: Enemigos eliminados y goal encontrado en 61 movimientos y 281ms

- **Semilla 2023:**
    - **Profundidad 10**: Enemigos eliminados y goal encontrado en 61 movimientos y 290ms
    - **Profundidad 8**: Enemigos eliminados y goal encontrado en 57 movimientos y 276ms
    - **Profundidad 6**: Enemigos eliminados y goal encontrado en 63 movimientos y 300ms

- **Semilla 2024:**
    - **Profundidad 10**: Enemigos eliminados y goal encontrado en 51 movimientos y 302ms
    - **Profundidad 8**: Enemigos eliminados y goal encontrado en 51 movimientos y 307ms
    - **Profundidad 6**: Enemigos eliminados y goal encontrado en 58 movimientos y 374ms

En esta implementación OnLine se puede observar como no hay casi diferencia de costo en tiempo y movimientos entre una profundidad y otra, debido, en mi opinión, a la simplicidad del entorno dado.

Búsqueda no Informada OffLine:

- Semilla 2020: Camino no encontrado, en 0ms y 0 movimientos.
- Semilla 2021: Camino encontrado, en 8ms y 7 movimientos.
- Semilla 2022: Camino encontrado, en 25ms y 25 movimientos.
- Semilla 2023: Camino encontrado, en 19ms y 19 movimientos.
- Semilla 2024: Camino encontrado, en 9ms y 9 movimientos.

Se puede observar, como el camino se realiza en los mismos movimientos que en el anterior algoritmo OffLine, pero con la diferencia de que este se ejecuta mucho más rápido.

Búsqueda no Informada OnLine:

- Semilla 2020: Camino no encontrado hasta los enemigos en 9302ms y 43 movimientos, camino hasta goal no encontrado.
- Semilla 2021: Camino encontrado, en 8364ms y 44 movimientos.
- Semilla 2022: Camino encontrado, en 2340ms y 61 movimientos.
- Semilla 2023: Camino encontrado, en 2537ms y 43 movimientos.
- Semilla 2024: Camino encontrado, en 2557ms y 47 movimientos.

En este caso el costo de tiempo es muy alto en comparación con el resto de implementaciones, principalmente por la gran cantidad de nodos(todo el tablero) que se tienen que evaluar cada vez que se llama al algoritmo y el enemigo está en una celda no transitable.