Inteligencia Artificial - Grado en Ingeniería Informática – 2019/2020

Actividad nº1 Ejercicio de búsqueda en un grafo.

Ejercicio de búsqueda heurística. Buscar un ejemplo de la vida real y modelizarlo para poder aplicarle cualquier algoritmo/s de búsqueda heurística de los explicados en clase.

Estudiante: Borja López Pineda

Profesor: Luis Ignacio López Gómez

Índice

Resumen

El ejemplo que he escogido trata de la búsqueda de recursos minerales en estructuras cavernosas naturales.

Los datos provienen de un mundo generado el videojuego Minecraft y la implementación del algoritmo se ha realizado en Java.

El objetivo de la búsqueda es encontrar diamantes en el mundo y obtener una ruta que un jugador pueda seguir para llegar hasta ellos.

Adjunto una comprimido DEMO.zip que contiene los ficheros necesarios para ejecutar el algoritmo, incluyendo el mundo. Un archivo CaveFinder.zip con un proyecto comprimido de NetBeans con el código.

Introducción

Para el modelado he tomado un mapa del videojuego 3D Minecraft y he definido cada bloque como un nodo. Un nodo está conectado con los cuatro nodos adyacentes, norte, sur, este y oeste. A demás se contempla que los nodos estén en 3 estados: aire, piedra y diamante.

Minecraft es un videojuego 3D cuyos mundos se generan proceduralmente a partir de ruido Perlin. En estos mundos aparecen minerales de forma aleatoria en el terreno dependiendo de la alternativa. Los diamantes son el recurso más valioso y solo aparece en alturas muy bajas. Estos mundos incluyen redes de cuevas, algunas son visibles desde la superficie.

La altura se mide contando el número de bloques hasta el límite inferior del mundo. Los diamantes se generan por debajo de la capa 16. El nivel del mar se sitúa en la capa 63 y el terreno llano no suele superar la altura 80. El límite superior es de 256 bloques.

Experimento

Reglas de la simulación

No se permite atravesar bloques sólidos ni romper bloques, solo está permitido moverse a las posiciones a las que un jugador real podría acceder.

Para mantener estas condiciones se han impuesto las siguientes reglas que rigen la simulación:

- La posición actual siempre será aire, en caso de ser piedra la posición actual ascenderá hasta que deje de ser piedra.
- Si la posición inferior a la actual es aire, la posición actual bajará hasta que sea piedra.
- Se habrá encontrado la solución cuando la posición interior sea diamante.
- Los movimientos posibles son 4, avanzar en las 4 direcciones cardinales en las que se conectan los bloques. En caso de que la posición resultante no cumpla las dos primeras reglas, será necesario modificarla antes de sumarla a la lista de movimientos posibles.
- La heurística está determinada por nuestro conocimiento sobre la generación de los diamantes. A mayor es la profundidad, más probable es encontrarlos. La función heurística es la altura del nodo, cuando menor sea el valor, mejor.

El punto de partida es siempre el mismo para todos los algoritmos.

Pseudocódigo

Pseudocódigo algoritmo de búsqueda Primero el mejor.

```
Función PrimeroElMejor(grafo, inicio) : nodo
   abiertos <- conjuntoVacio()
   cerrados <- conjuntoVacio()

   actual <- inicio
   actual.anterior <- nada

   Mientras No actual.esSolucion() Haz
         adyacentes <- grafo.explorar(actual)
         ParaCada adyacente En adyacentes
               Si No (cerrados.contiene(adyacente) O
                abiertos.contiene(adyacente)) Haz
                     adyacente.anterior <- actual
                     abiertos.agregar(adyacente)
               FinSi
         FinParaCada

         actual <- nada
         ParaCada abierto En abiertos Haz
               Si actual = nada O abierto.heuristica > actual.heuristica Haz
                     actual <- abierto
               FinSi
         FinParaCada
         cerrados.agregar(actual)
         abiertos.eliminar(actual)
   FinMientras

   Devolver actual
FinFunción
```

Esta función devuelve un nodo solución. Para obtener el camino optimizado que ha obtenido el algoritmo, hay que invertir la lista enlazada que se obtiene al seguir el atributo anterior del nodo devuelto.

g.explorar(n) devuelve los nodos conectados con el nodo n en el grafo g.

Pseudocódigo algoritmo de búsqueda Alfa estrella.

```
Función AlfaEstrella(grafo, inicio) : posición
      abiertos <- conjuntoVacio()
      cerrados <- conjuntoVacio()

      actual <- inicio
      actual.anterior <- nada
      actual.distancia <- 0

      Mientras No actual.esSolucion() Haz
            adyacentes <- grafo.explorar(actual)
            ParaCada adyacente En adyacentes
                  Si No (cerrados.contiene(adyacente) O
                         abiertos.contiene(adyacente)) Haz
                        adyacente.anterior <- actual
                        adyacente.distancia <- actual.distancia +
                                                grafo.coste(actual,
adyacente)
                        abiertos.agregar(adyacente)
                  FinSi
            FinParaCada

            actual <- nada
            ParaCada abierto En abiertos Haz
                  Si actual = nada O abierto.heuristica + abierto.distancia >
                                    actual.heuristica + actual.distancia
Haz
                        actual <- abierto
                  FinSi
            FinParaCada
            cerrados.agregar(actual)
            abiertos.eliminar(actual)
      FinMientras

      Devolver actual
FinFunción
```

g.coste(n1, n2) devuelve el coste de desplazarse desde el nodo n1 al nodo n2, suponiendo que están directamente conectados, en el grafo g.

Implementación

La implementación se ha realizado en el lenguaje Java, ya que cuenta con una librería que permite utilizar los ficheros guardados del videojuego. Dado el gran número de nodos, se han introducido una optimización en los algoritmos que permite mejorar el rendimiento sacrificando la exhaustividad de estos. Esta modificación consiste en agregar un límite para la lista de nodos abiertos y cerrados, de forma que, si este límite es superado, los nodos cuya función heurística sea superior a un umbral serán eliminados. Se perderán nodos poco atractivos a primera vista que podrían permitir acceder a una mejor solución y se olvidan algunos nodos que ya fueron visitados, aunque sean los que tienen menos probabilidad de volver a aparecer.

Actualmente la estructura permite agregar más algoritmos por distintos que sean.

La clase que adapta el mundo a un grafo e interactúa con el algoritmo se llama CaveFinder.
El requisito para que una clase sea usada como algoritmo de búsqueda es que implemente la interfaz AlgoritmoBusqueda.
Para reunir características y evitar repeticiones, la clase AlgoritmoBusquedaGeneral es una clase abstracta que implementa AlgoritmoBusqueda parcialmente y maneja el nodo actual, es una necesidad que tienen todos los algoritmos y puede implementarse siempre así.
La clase AlgoritmoBusquedaEnGrafo es una clase abstracta que hereda de AlgoritmoBusquedaGeneral y contiene la parte común entre Primero el mejor y Alfa estrella.

Los métodos que proporciona la interfaz AlgoritmoBusqueda son:
void explorar(Nodo nuevo) : Introduce el nodo nuevo en la lista de nodos que el algoritmo tiene visibles.
Nodo getSiguiente() : Fuerza al algoritmo a decidirse por sustituir su nodo actual por un nodo agregado mediante la función existente, a de más, devuelve ese nodo.
Nodo getActual() : Devuelve el nodo actual en el que se encuentra.

Resultado

El resultado obtenido es el esperado. Es posible visualizar el camino seguido por el algoritmo gracias a un complemento para el juego que desplaza al jugador por una lista de coordenadas generadas por el algoritmo al visitar nodos.

El algoritmo de búsqueda Primero el mejor explora la superficie hasta adentrarse en la cueva cercana. La cueva se bifurca y el algoritmo decide ir en la dirección que permite minimizar la función heurística antes (aumentar la profundidad), pero llega aun punto muerto donde no es posible continuar. Deshace sus movimientos visitando los nodos que ignoró y encuentra el otro camino. Una vez allí cae por la galería y continúa explorándola hasta llegar al diamante.

El algoritmo de búsqueda Alfa estrella toma en consideración la distancia real recorrida a de más de la función heurística. Esto hace que se arrepienta a menudo y regrese a explorar posiciones antes de continuar con el camino que, se supone, debería seguir. Tan exhaustivo llega a ser este algoritmo, que busca en otras cuevas cercanas antes de seguir con la primera cueva. Al final, consigue encontrar el diamante.

Ambos algoritmos logran hacer tratable el problema de buscar diamantes, los 4 minutos de media que tardan en ejecutarse no se comparan con el tiempo que supondría una búsqueda exhaustiva de todos los bloques. En comparativa, la búsqueda Alfa estrella requiere más iteraciones, pero obtiene un camino más optimizado que Primero el mejor.

Para ser exactos, Primero el mejor es capaz de encontrar un buen camino en un tiempo razonable. Pero Alfa estrella encuentra un camino bastante mejor a costa de dedicar un tiempo un algo superior. Alfa estrella localiza un acceso a la cueva que estaba escondido en el bosque y le permitió llegar a las profundidades antes que si siguiese por la entrada obvia.

Visualización de los resultados

He agregado una función al programa para que escriba en un fichero los nodos que visita y, una vez termina, escribe en otro el camino final obtenido. En los siguientes videos se muestra al jugador teletransportándose a esas posiciones.

El juego no está pensado para este tipo de movimiento y sufre de ciertos errores gráficos incómodos, a pesar, es posible distinguir la trayectoria que se intenta mostrar.

La duración de los videos en los que se muestra la lista de nodos recorridos es siempre de un minuto y los videos con el camino final de 10 segundos (quitando principio y fin), es por esto que el número de desplazamientos por unidad de tiempo no se mantiene constante.

Primero el mejor: En este video puede verse al jugador moviéndose por los nodos que ha visitado el algoritmo Primero el mejor.

Camino de Primero el mejor: Aquí el jugador sigue la lista enlazada de nodos que representa el camino final obtenido por Primero el mejor, invertido.

Resultado de Primero el mejor: Aquí recorro manualmente le camino conseguido por Primero el mejor.

Alfa estrella: En este video puede verse al jugador moviéndose por los nodos que ha visitado el algoritmo Alfa estrella.

Camino de Alfa estrella: Aquí el jugador sigue la lista enlazada de nodos que representa el camino final obtenido por Alfa estrella, invertido.

Resultado de Alfa estrella: Aquí recorro manualmente le camino conseguido por Alfa estrella. Puede apreciarse la superioridad de este camino, frente al obtenido por Primero el mejor.

Anexo 1.- Otros algoritmos

Gracias a la modularidad del diseño adoptado, ha sido fácil introducir más algoritmos de búsqueda informada, aunque los resultados no son buenos.

HillClimb y Simullated Annealing caen en un estado sin salida, tienen la tendencia de salirse del camino obvio e intentar explorar exterior cayendo siempre en mesetas de las que no pueden salir.

El mundo no está hecho para esos algoritmos. Creo que se debe a que en la gran mayoría de casos las opciones que se le plantean son exactamente iguales. Si hubiese una diferencia en la altura entre todos los bloques, por pequeña que fuera, estos algoritmos podrían seguir el camino.

Anexo 2.- Carga de mundos

Minecraft divide el mundo en bloques de 16x16 y sus 256 bloques de altura, esta agrupación se llama chunk. Los chunks son la unidad mínima en la que el mundo puede ser cargado o descargado. Los chunks se agrupan en regiones de 32 x 32 chunks. Las regiones son la unidad mínima que se guarda en el fichero. El índice de chunks y bloques es siempre positivos, pero las regiones se indexan con números positivos y negativos para poder centrar el mundo en la región 0 0, chunk 0 0, bloque 0 0.

Enklume es una librería para Java que permite cargar estos mundos. La utilizo únicamente para interpretar el fichero de guardado y he creado algunas funciones de ayuda para poder referirme a las coordenadas de forma absoluta, en lugar de como se guardan en el fichero del mundo.

Anexo3.- Visualización de los resultados

Actualmente Minecraft no cuenta con ninguna herramienta de automatización, así que he recurrido a un complemento para servidores CraftBukkit creado por la comunidad que permite al jugador definir puntos de control y mover al jugador entre ellos de forma automática.

El programa genera archivos de texto en formato YAML que pueden ser leidos por este complemento.