Práctica 1

Desarrollo de agentes basado en técnicas de búsqueda heurística dentro del entorno GVGAI

Asignatura: TSI
Curso académico: 2024/2025
Grupo de Prácticas: 3 (Martes)
Email: mariacribilles@correo.ugr.es

María Cribillés Pérez
DNI: 75573410X

1. Tablas con resultados

Algoritmo	Mapa	Runtime	Tamaño ruta	Nodos expandidos	Abiertos/ Cerrados
Labyrinth Dual (id 59)					
Dijkstra	Pequeño	2 ms	68	157	
	Mediano	3 ms	60	337	
	Grande	17 ms	184	4202	
A*	Pequeño	3 ms	68	156	1/156
	Mediano	4 ms	60	244	9/244
	Grande	31 ms	184	1628	44/1628
RTA*	Pequeño	1 ms	242	242	
	Mediano	2 ms	350	350	
	Grande	2 ms	1014	1014	
LRTA*	Pequeño	1 ms	1330	1330	
	Mediano	1 ms	1858	1858	
	Grande	2 ms	932	932	

Cuadro 1: Tabla de resultados

2. Preguntas

1. **Imaginemos que la representación del estado en nuestro juego solo incluyese la posición de la casilla en que nos encontramos. Por ejemplo, no se incluiría la capa como parte del estado. ¿Qué problemas y/o ventajas podría tener esta representación?**

   Si solo incluyésemos en el estado la posición del avatar, el espacio de búsqueda sería mucho más pequeño ya que dos estados solo serían diferentes si y solo sí están en diferentes casillas. Por tanto, hay el mismo número de estados que de casillas. Esto haría que la búsqueda fuese mucho más rápido porque se expande mucho menos.

   Sin embargo, dos nodos que tengan la misma posición y diferente capa se van a considerar que tienen el mismo estado y esto puede hacer que se descarten rutas válidas o al revés, que se generen soluciones incorrectas. Por ejemplo, si para llegar al portal se debe pasar obligatoriamente por un muro azul y tenemos por un camino más largo una capa azul, si solo considerásemos en el estado la posición, no iría por la ruta larga, si no por la corta y llegaríamos a que no encuentra la solución porque no puede pasar sin capa azul.

   Además de la posición y de si tiene actualmente capa roja o capa azul,

en el estado hay que incluir una lista de capas restantes de ambos colores. Así, también consideramos diferentes un nodo que ya ha cogido varias capas pero tiene solo ahora la roja al nodo que es la primera capa roja que coge. Esto también nos puede evitar complicaciones, como por ejemplo un posible encierro al haber cogido una capa que no debía.

2. **¿RTA* y LRTA* son capaces de alcanzar el portal en todos los mapas? Si la respuesta es que no, ¿a qué se puede deber esto?**

   No siempre se va a encontrar solución. Puede haber situaciones en las que el avatar se quede encerrado. Al ser en tiempo real, no piensan la ruta entera antes de actuar, por lo que a diferencia de Dijkstra y A* que sabrían que no hay solución o que por una determinada zona no se puede pasar para no quedarse encerrado. Sin embargo, RTA* y LRTA* actuarían y podrían entrar en zonas sin salida. Un caso sencillo es si hay una zona que no ha explorado, va a querer ir porque la zona ya explorada tiene más heurística por la actualización del propio algoritmo. Si la zona que no ha explorado (que es la que elige por tener menos heurística) obligatoriamente pasa por una capa roja y necesita la azul para volver para la meta y ya no hay más, el avatar se quedará encerrado sin poder regresar ya que tiene una capa diferente a la necesaria.

3. **Imaginemos que debemos implementar búsqueda en profundidad como algoritmo de búsqueda para resolver este u otro problema. La forma más evidente de hacerlo es partir de una implementación de búsqueda en anchura y sustituir por una pila la cola que se emplee para almacenar los nodos abiertos. ¿Qué otra alternativa tendríamos a esta estrategia naive, y qué pros y contras tendría dicha alternativa?**

   Una alternativa a la búsqueda en profundidad (DFS) es implementar una búsqueda en profundidad de forma recursiva. Las ventajas son las siguientes: Las ventajas son varias. Por una parte, el código es más simple y claro al no gestionar nosotros la pila. Hay menos cantidad de porque no se utiliza una pila de nodos abiertos. Por último, es buena para grafos no muy profundos. Sobre todo es útil en árboles porque nos aseguramos que no hay ciclos.

   Por otro lado, tiene algunos inconvenientes. Puede existir un riesgo de desbordamiento de la pila por alcanzar el límite de llamadas recursivas. Si los grafos son muy profundos o con ciclos, puede superarse el límite de llamadas recursivas o ser infinito. Además, tiene un menor control sobre el proceso ya que es más difícil implementar variaciones o añadir lógica adicional. Por último, es poco adecuada para problemas donde se

requiere un manejo detallado del recorrido o se trabaja con estructuras de grafo muy grandes. Podría llegar a consumir mucha memoria.

Como resumen, una búsqueda recursiva de DFS es una buena alternativa para ciertos tipos de problemas simples y claros. Sin embargo, para problemas complejos puede no ser tan buena elección.

4. **¿De qué modo LRTA*(k) se diferencia de LRTA*? ¿En qué consiste el algoritmo y cuáles son sus pros y contras en relación a algoritmos previos (como LRTA*)?**

   LRTA*(k) es una extensión basada en LRTA* que es capaz de actualizar las estimaciones heurísticas de hasta k estados, no necesariamente distintos, manteniendo la admisibilidad de la heurística.

   La única diferencia es que LRTA* actualiza la heurística de un único estado por iteración, mientras que LRTA*(k) actualiza la estimación heurística de hasta k estados, no necesariamente distintos, por iteración siguiendo una estrategia de propagación acotada. LRTA*(k) es un caso particular de LRTA*(k) cuando k=1.

   Las ventajas de LRTA*(k) son:

   a) Mejores soluciones iniciales: en la primera ejecución, LRTA(k)* suele encontrar trayectorias más cortas que LRTA*, ya que aprovecha mejor la información disponible. Si el agente revisita un estado, la heurística ya está más ajustada, evitando decisiones subóptimas.

   b) Convergencia más rápida: al propagar actualizaciones a más estados, las estimaciones heurísticas se acercan antes a sus valores reales. Esto reduce el número de ensayos necesarios para alcanzar soluciones óptimas, tanto en pasos como en tiempo de CPU.

   c) Mayor estabilidad en las soluciones: las trayectorias generadas son más consistentes, con menos fluctuaciones entre ejecuciones. Esto es un efecto secundario de la mejora en la calidad de las soluciones.

   Las desventajas son:

   a) Coste computaciones adicional: el actualizar hasta k estados por iteración requiere más tiempo de planificación que LRTA*. Sin embargo, este coste está acotado (por el parámetro k) y puede ajustarse.

   b) Elección del valor de k: en el paper dicen que un k pequeño, como 2 o 3, ofrece ya mejoras con un coste moderador. Con un k más grande mejora aun más los resultados pero aumenta el tiempo de planificación.

Como conclusión, tenemos que LRTA(k)* mejora a LRTA* al propagar actualizaciones heurísticas de manera más eficiente a cambio de un ligero aumento en el tiempo de cómputo.

5. **¿Bajo qué condiciones podemos estar seguros de que no vamos a entrar nunca dentro de este if? ¿Es siempre necesario implementar esa parte del algoritmo?**

```
if cerrados.contains(sucesor)
  and mejorCaminoA(sucesor): # menor g(n)
  • cerrados.remove(sucesor)
  • abiertos.add(sucesor) # actualizar g(n)
```

Nunca va a entrar en ese bloque siempre que se cumpla que A* nunca expande un mismo estado más de una vez con un coste menor, es decir, cuando usamos una heurística admisible y consistente (monótona). Es decir, si la heurística es monótona, está garantizado que cualquier cerrado repetido que salga nunca va a ser mejor que el que se metió primero. Vamos a desarrollar las definiciones:

Una heurística es consistente si, para cualquier nodo n y para cualquier sucesor n' alcanzado desde n mediante una acción de coste $c(n, n')$, se cumple:

$$h(n) \leq h(n') + c(n, n')$$

Una heurística es admisible siempre que el costo estimado sea menor o igual que el costo mínimo de alcanzar la solución: $H(n) < H^*(n)$ donde $H(n)$ es el costo estimado y $H^*(n)$ es el costo verdadero para llegar a la solución.

Cuando es monótona, la estimación del coste desde un nodo hasta la meta nunca es mayor que el coste para ir a un sucesor más la estimación desde ese sucesor hasta la meta. En nuestro algoritmo A*, esto implica que una vez expandimos un nodo (es decir, lo sacamos de abiertos y lo metemos en cerrados), ya lo hicimos con el mejor coste posible, por lo tanto, nunca aparecerá un camino mejor hacia ese mismo nodo después.

Una implicación es: las heurísticas consistentes son admisibles, pero no todas las heurísticas admisibles con consistentes.

Nosotros trabajamos con la distancia Manhattan como heurística que es consistente porque como mucho en un movimiento va a cambiar en uno, es decir, si te mueves hacia la meta disminuye en uno y si te mueves en otra dirección se mantiene o sube uno, por lo cual es menor o igual que el coste de un movimiento que es siempre 1. Por tanto, por la implicación anterior sabemos que es admisible. Además, el que una heurística sea admisible y consistente implica que la heurística es monótona.