UNIVERSIDAD
DE GRANADA

Técnicas de Sistemas Inteligentes
grado en ingeniería informática

PRÁCTICA 1

Técnicas de Búsqueda

Autor
Fernando Vallecillos Ruiz

DNI
77558520J

E-Mail
nandovallec@correo.ugr.es

Grupo de prácticas
TSI 1 Lunes 17:30-19:30

Rama
Computación y Sistemas Inteligentes

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2019-2020

Índice

1. Introducción

Se realiza un práctica compuesta por la creación de uno o más agentes para resolver un problema de *pathfinding* en el entorno de juego GVGAI (*BoulderDash*). El agente deberá calcular un camino desde su posición inicial a la salida con diferentes restricciones o sobrevivir durante un periodo determinado de tiempo. Todas las acciones tienen fuertes restricciones de tiempo. Se describen los objetivos en los diferentes niveles:

- Encontrar camino de salida
- Encontrar camino de salida recogiendo 10 gemas
- Sobrevivir 2000 ticks ser atrapado (1 enemigo)
- Sobrevivir 2000 ticks ser atrapado (2 enemigos)
- Encontrar camino de salida recogiendo 10 gemas sin ser atrapado (1 enemigo)

Se utiliza el algoritmo A* para el problema de *pathfinding* con diferentes heurísticas y variaciones para mejorar su eficacia en el juego y en tiempo. Se describen a continuación las técnicas usadas de forma gradual.

2. Comportamiento Deliberativo

2.1. Simple

Este problema se puede definir como un simple *pathfinding*. Para aumentar eficiencia temporal, se crea una matriz con identificadores del mapa. Se crea una clase simple para dado 1 punto de entrada y orientación, devuelva el camino óptimo. Se describe brevemente el algoritmo A*.

Todos los nodos constan de una posición en el mapa y los valores g y h característicos del algoritmo. La heurística por ahora, se mantiene como la distancia *Manhattan* y además se tiene en cuenta el costo del giro en g. La función *Expandir* se realiza de forma trivial creando 4 nodos alrededor del actual e insertándolos si están en rango y no han sido explorados anteriormente. Se puede resaltar que el algoritmo establece la posición y orientación inicial en una función diferente. Esta división se realiza con la intención de que en futuros niveles, se pueda reutilizar el mapa para encontrar caminos entre diferentes posiciones. Además, para aumentar la eficiencia, se puede permitir que la función se encargue de insertar las acciones

a realizar. Sabiendo la cantidad de veces que se llama la función, el evitar devolver el camino cada vez, no solo disminuye el tiempo de *overhead*, sino que también reduce la memoria utilizada por el programa.

Algorithm 1: FindPathTo

```
1  FindPathTo (Row, Col, Dir)
       input  : Fila Row, Columna Col, Index Ind
       output: Orientación final DirF
2      Now ← Node(IniRow, IniCol)
3      Close ← Close + Now
4      Expand(Now)
5      while Now.Row ≠ Row || Now.Col ≠ Col do
6          if Open is Empty then
7              return -1
8          Now ← Open.Pop()
9          Close ← Close + Now
10         Expand(Now)
11     Result[Ind] ← Path
12     return Direction[Row][Col]
```

2.2. Compuesto

En este nivel se tiene un total de 11 objetivos (10 gemas y 1 salida). Nuestro entorno nos permite saber las posiciones exactas de todas las gemas y la salida. Por ello y con lo anteriormente explicado, se puede obtener el camino óptimo entre cualquier posición (gema o salida). Sin embargo se necesita resolver 2 problemas: qué gemas utilizar y en que orden visitarlas.

Se trata de una generalización del problema del viajante de comercio (NP-Completo). Por lo que se necesitaba buscar algún tipo de heurística para evitar $\binom{n}{10} \cdot 10!$ combinaciones. Primero se realizó una búsqueda de la gema más próxima. Daba un resultado aceptable pero no óptimo. También se prueba con la búsqueda del vecino más próximo simultanea (ambos extremos) pero tampoco daba resultados satisfactorios. Dado que esta búsqueda se realiza en el constructor, se cuenta una mayor cantidad de tiempo para realizarla. Gracias a los experimentos anteriores, se puede apreciar que hay una gran cantidad de soluciones que se pueden descartar de manera rápida. Se realiza entonces una búsqueda *Branch and Bound*. Este tipo de algoritmo se adapta de forma perfecta a las necesidades de nuestro problema.

Se guardan todas las distancias en una matriz para evitar recalcular. Se ha

implementado de forma recursiva (facilitando su lectura) ya que el nivel de profundidad máximo no es muy alto (no existe demasiado *overhead*). Se realizan diferentes mejoras para evitar en lo posible la creación de variables locales innecesarias, reducir el paso de parámetros y mantener la legibilidad. Dado que es solo una heurística, no tiene en cuenta obstáculos o giros pero la solución final consume entre 10-15ms en total para encontrar el camino "óptimo" con 10 gemas en total. Dado que el número de gemas puede variar, se ha comprobado su eficiencia con mayor número de estas. En los experimentos realizados, el límite se sitúa en alrededor de 23 gemas siguiendo una distribución basada en la original (el límite será menor si las gemas se reparten de forma totalmente uniforme por el mapa). Para un mayor número de gemas, se puede combinar con otra heurística para elegir las gemas mas cercanas al punto de entrada y salida.

Sabiendo el orden de las gemas, se puede utilizar la función $FindPathTo$ para encontrar los 11 caminos necesarios para resolver este nivel de forma rápida y eficaz.

3. Reactivo

3.1. Simple

Para este nivel, se necesita evitar a un enemigo durante una cantidad determinada de tiempo. Se intenta dividir este nivel en una serie de modificaciones al algoritmo anterior. Primero se supondrá un enemigo estacionario en el mapa y un objetivo al que queramos llegar. Se puede añadir una heurística a nuestra función para encontrar un camino que evite no solo al enemigo, si no sus alrededores.

Algorithm 2: MonsterHeuristic

```
1 MonsterHeuristic (Node)
      input  : Nodo N, Limite Limit
      output: Nodo N
2     if AreClose(Node, Monster, Limit) then
3         Dist ← DistManhattan(Node, Monster)
4         if Dist == 0 then
5             Node.h ← Node.h + 100 * Limit
6         else
7             Node.h ← Node.h + 100 * (Limit/Dist)
8     return Direction[Row][Col]
```

Se basa en la idea presentada en clase sobre mapas de calor con una ligera

variación. Los mapas de calor basados en distancia de Manhattan se prueban algo deficientes contra este tipo de enemigos. A diferencia del avatar, ellos no necesitan gastar movimientos en girar. Por ello, a partir de la función $AreClose$, se establece que el límite del mapa de calor en una forma cuadrada. Es decir, se incluye cualquier posición tal que $Node \in (Monster.row \pm limit, Monster.col \pm limit)$. Si el nodo pertenece a la región definida, añadimos un valor heurístico. Este evitará el centro a toda costa y se irá rebajando conforme la distancia aumente. Se elige 100 como una constante lo suficientemente grande para evitar que el avatar pase por esta zona. Se realiza un mapa de calor en vez de bloquear todo para que el avatar, en caso de no tener otra forma de salir (arrinconado), pueda calcular la mejor salida.

El avatar sabe evitar enemigos cuando camina hacia un objetivo. Sin embargo, en el nivel actual no existen objetivos. Por ello se crean "gemas" en las esquinas transitables del mapa. Estos objetivos sirven a nuestro avatar como motivo para moverse evitando los enemigos. Se añade un comportamiento extra en el que, si el enemigo se encuentra cerca de el objetivo, intercambie este por otro asegurando que no haya enemigos cerca. Esto permite que el avatar se quede quieto en uno de los objetivos mientras que no haya enemigos cerca. Este límite es variable dependiendo del mapa y número de enemigos. Se discutirán los rangos escogidos en el siguiente apartado.

Por último, gracias a la rapidez de la función $FindPathTo$, se ha podido realizar una búsqueda hacia el objetivo actual en cada iteración del juego. Esto permite que, a pesar que los enemigos se muevan en el mapa, se encuentren caminos óptimos al objetivo que lo evitan.

3.2. Compuesto

Las modificaciones realizadas en el apartado anterior permiten llegar a este nivel de forma trivial. Se necesita tener en cuenta los dos enemigos tanto en la heurística del camino como en el intercambio de objetivos. Gracias a los mapas de calor, se asegura que evite la zona cercana a cualquiera de los enemigos. Incluso al juntarse y forzar el paso entre ellos, la suma de las zonas de calor provocaran escoger el camino más cercano a la mitad entre estos.

Las variables seleccionadas tanto para la función heurística como para el intercambio de objetivos determinan fundamentalmente el comportamiento del agente. Al aumentar el valor de la función heurística, se describe una zona de calor mayor y por tanto a evitar. Un valor muy pequeño puede provocar que el enemigo atrape al avatar al no haber dejado la suficiente distancia. Un valor muy grande no solo provoca un mayor rodeo (pérdida de eficiencia respecto al camino óptimo), sino que puede ser contraproducente con más de un enemigo. La desviación causada

puede provocar que el agente sea empujado a callejones sin salida más fácilmente. No se aprecia demasiado con un solo enemigo, pero al aumentar a 3 o 4, la suma de zonas de calor puede empujar al agente a realizar movimientos ineficientes y evitar caminos de huida muy seguros. Por ello, para este nivel se ha escogido un valor de 8. Dado que huiremos automáticamente de los enemigos, la única oportunidad que tendrán de ganar será cuando ambos bloqueen la ruta de escape mas obvia. Por ello, se intenta dejar una distancia mínima de 4 casillas entre ambos enemigos al pasar entre ellos o buscar otra ruta.

Por último, el valor de la distancia con la que se cambia de objetivo. Este límite también es calculado de la misma forma que el límite heurístico (cuadrado). De nuevo, un valor muy bajo permite que el enemigo se acerque demasiado antes de empezar a moverse. Un valor muy grande provoca cambiar de objetivo cuando el enemigo no esté viniendo (ineficiente y mayor probabilidad de ser empujado a callejones). Por ello se aplica la idea anterior de 4 casillas de distancia antes de cambiar de objetivo y escapar.

4. Deliberativo-Reactivo

En esta última sección se juntan las diferentes partes descritas anteriormente. La parte deliberativa calculará el orden de las gemas y el camino hacia el portal que pase por 10 de ellas. Existe 1 enemigo en el mapa el cual se evitará utilizando la heurística con mapas de calor. Además, se adapta el cambio de objetivos por cercanía de enemigo a una mejor heurística. En esta primero se intercambian objetivos con la siguiente gema que se fuera a buscar. De esta forma se mantiene cerca si el enemigo decide irse y la desviación será mínima. Puede ser que el enemigo se encuentre cerca de las próximas dos gemas. Por ello también se permite el intercambio con la gema dos posiciones por delante en el orden establecido y se implementan medidas para bloquear ciclos inmediatos.

Las variables de límite en la heurística en A* y en el intercambio de objetivos se han reducido considerablemente debido a la reducción de número de enemigos a 1. Para la heurística, se reduce el límite a 2. Dado que es solo un enemigo y siempre mantendrá 2 o más espacios, es suficiente para que en caso de que decida perseguir al avatar, este pueda huir a la misma velocidad. El límite de distancia para el intercambio de objetivos también se reduce a 2 por el mismo motivo.

Estas modificaciones plantean situaciones que han de ser tenidas en cuenta. Primeramente, la desviación resultado de esquivar un enemigo, puede hacer que el avatar se sitúe sobre una gema. Si esta gema estaba en el plan de recogida, solo se necesita saltarla cuando se vaya a planear su camino. Si la gema no estaba en el plan de recogida, se pasa al siguiente objetivo ignorando el actual y creando un

nuevo camino.

También se puede dar el caso donde el enemigo está cerca del objetivo y no es posible cambiarlo (última gema o portal). En este caso la única posibilidad es esperar hasta que el enemigo se aleje del objetivo para poder alcanzarlo de forma segura. Por ello, se crea un objetivo virtual la misma fila con una distancia de seguridad (si no es posible encontrar una celda transitable y segura en la misma fila, se van recorriendo filas hasta encontrar una). Gracias a este objetivo virtual, se permanece en una celda cercana a nuestro objetivo hasta que el enemigo se mueva lo suficiente. Se decide una distancia mínima de 7 casillas. Se elige para asegurar que el enemigo no pueda bloquear el objetivo real y virtual al mismo tiempo, y dar tiempo de maniobrar (esquivarlo y llegar al objetivo) en el caso de que fuese directo a por el avatar.

Se implementa una última medida para la generalización del agente. A pesar de que *FindPathTo* sea un método muy rápido en el mapa actual, su consumo de tiempo es exponencial al tamaño del mapa y número de enemigos. Esto puede provocar que en mapas de (aproximadamente) doble o triple de tamaño, provoque *timeouts*. Dada la restricción temporal tan estricta y posibilidad de encontrar caminos no óptimos, se ha decidido modificar el patrón de búsqueda del algoritmo A*. Se aprovecha la heurística la cual proporciona caminos óptimos o muy cercanos para evitar la actualización de nodos en la lista de abiertos. Esto aumenta el tamaño de mapas permitidos de forma exponencial. Además, la pérdida de pasos es menor 5 % (en casos complejos con obstáculos, en simples no existe diferencia). Se incluye la versión con actualización de la lista de nodos abiertos por si el lector desea comparar resultados.

Con esto se termina la construcción del agente. Como se ha podido ver se cubren gran variedad de casos y situaciones particulares para dar la mayor eficiencia posible en cualquier momento. Los parámetros discutidos permiten modificar el agente de forma rápida y sencilla cambiando radicalmente su comportamiento y resultados en diferentes situaciones. Un pequeño estudio puede conseguir los mejores valores para estas variables dependiendo del tamaño de mapa, número de gemas, número de monstruos y el riesgo tomado (a menores límites, mayor rapidez). Se ha descrito los diferentes comportamientos del agente y los parámetros usados para hacerlo lo más general posible en 500 mapas generados aleatoriamente para evitar sobreajuste.