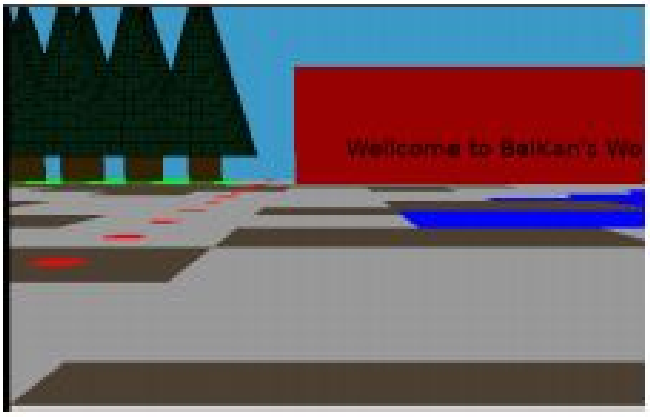
INTELIGENCIA ARTIFICIAL

E.T.S. de Ingenierías Informática y de Telecomunicación

Práctica 2



Agentes Reactivos/Deliberativos

los extraños mundos de BelKan

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL
UNIVERSIDAD DE GRANADA
Curso 2019-2020

Alumno: Raúl Rodríguez Pérez
Grupo teoría: C
Grupo prácticas: C1
S.O: Linux

1. Nivel 1b: Búsqueda en Anchura

Para la implementación de este algoritmo solo me fue necesario visualizar un par de veces el video donde nuestro profesor de prácticas nos explicaba en qué consistía dicho algoritmo, y tras esto, partiendo del algoritmo previo a este (búsqueda en profundidad) me dí cuenta de que solo tendríamos que cambiar el orden en el que inspeccionamos los nodos, es decir, pasar de inspeccionar mediante una relación padre e hijo, a inspeccionar por niveles, tal y como se explicaban en los videos comentados anteriormente. Para ello bastaba solo con intercambiar la estructura usada para almacenar los nodos, sustituyendo la pila por una cola.

2. Nivel 1c: Búsqueda de costo uniforme

Para la implementación de este algoritmo tuve que estudiar más afondo las características de dicha búsqueda. Para ello, a parte de los videos y apuntes de la asignatura busqué más información que me ayudara tanto a entender la búsqueda, como algunas pistas de como implementarlo. Un video que me sirvió mucho de ayuda fue este:
https://es.coursera.org/lecture/resolucion-busqueda/algoritmo-de-costo-uniforme-ucs-W4vmS

En dicho video se explica de manera muy sencilla, el procedimiento que sigue el algoritmo con un ejemplo práctico. Una vez tuve las ideas claras, sabía que debía modificar el orden de inspección de los nodos, para ordenarlos de menor a mayor coste. Para ello, fijándome en como habíamos realizado las dos implementaciones anteriores, cree un nuevo nodo “nodo_unif”, el cual sería el encargado de, entre otras funciones, almacenar el costo que suponía llegar hasta dicho nodo. Como no podía comparar un nodo con otro, implementé un functor “comparaCoste” el cual tenía como propósito comparar dos nodos según su coste. Una vez implementado esto, la estructura elegida para el orden de inspección de los nodos fue una cola de prioridad en donde se ordenaban los nodos de menor a mayor coste.

Tras esto, mientras no encuentres el nodo destino y tengas nodos en los que buscar, se calcula el costo de desplazarse al siguiente nodo (empleo una función que me calcula el coste dependiendo del tipo de casilla que sea), una vez calculado dichos costes, expandimos (tal y como muestra el video del enlace que dejé), a los hijos del nodo que menor coste tenga. Así para llegar al nodo ‘n’, el coste va a ser la suma del nodo y del nodo ‘n-1’, para cada uno de los nodos.

Quiero destacar que implementé la función de que llevara bikini o zapatillas creando dos atributos inicializados previamente a false, en los que sí pasas por una de dichas casillas se ponen a true. Esto implica que en mi función donde calculo el coste “costeAvanzarNodo”, tengo en cuenta que sí tengo el bikini y paso por el agua, el coste disminuye, y lo mismo con las zapatillas y el bosque.

3. Nivel 2: Agente reactivo/deliberativo

Este nivel era el que realmente suponía un desafío para nosotros los estudiantes, partiendo de un mapa inexplorado, y teniendo que conseguir el máximo número de objetivos posibles antes de acabar el tiempo para pensar el plan, la batería o los instantes de la simulación.

Como podíamos realizar dicho nivel con cualquier algoritmo de búsqueda que nosotros vieramos oportuno, yo decidí utilizar el algoritmo de búsqueda de costo uniforme, aprovechando así la implementación que había realizado en el anterior nivel. Implementé la forma de que a medida que nos desplazamos por el mapa, fuéramos guardando en 'mapaResultado' todas las casillas que íbamos observando gracias al uso de los sensores que poseía nuestro jugador. Destacar que me sirvió de mucha ayuda el tutorial de la prac2 en prado para realizar esta parte.

Ya solo con esta parte el número de objetivos que encontrábamos para mapas pequeños era buena, pero en mapas grandes daba malos resultados. Por lo que, para mejorar los resultados, implementé lo necesario para la recarga de la batería. Así pues, cuando el jugador detectaba la casilla de la batería, almacenaba su posición (fil,col) en variables. En el momento en el que la batería llegase a estar por debajo de 1000, se replanificaría el siguiente objetivo, poniendo como destino las coordenadas de la casilla para recargar. Una vez allí, el jugador recargaba 1500 más de batería y continuaba con su labor. Esto hizo que los resultados en mapas pequeños mejoraran significativamente, llegando a obtener 96 objetivos en el mapa30. Aun así, en los mapas grandes al ser difícil encontrarse con dicha casilla por la magnitud del mapa continuaba siendo difícil obtener buenos resultados. Destacar también que no realicé muchas pruebas con la cantidad de batería a recargar y el momento para recargar, pero al mirar las últimas ejecuciones que hice me di cuenta que tal vez reducir la cantidad que recarga (por ejemplo pasar a unos 700 en vez de 1500), y reducir también en qué momento recargar (pasar de cuando esté por debajo de 1000 a cuando esté por debajo de 800) podría darme mejores resultados de los ya obtenidos, pero como ya tenia buenos resultados no probé a hacerlo.

Por otro lado, para evitar el tema de las colisiones, he implementado una condición en la que sí encontramos un muro, un precipicio o un aldeano, el jugador realice la acción "actIDDLE", es decir, se mantenga quieto y recalcule así una nueva ruta para evitar chocarse con estos obstáculos. Para solucionar el problema de los mapas grandes o con grandes cantidades de casillas costosas como el mapa de islas, no he conseguido encontrar la forma para realizar buenos resultados. En mi código tengo una condición que sí la siguiente casilla a la que va el jugador sea agua o bosque, pare y recalcule la ruta. Esta condición está comentada puesto que aunque en algunos mapas grandes me de mejores resultados con ella, no creía que fuera una buena opción o una forma correcta de resolver dicho error.

En conclusión, la implementación que he realizado para es nivel 2 logra con éxito realizar una buena cantidad de objetivos en mapas pequeños o sin muchas aglomeraciones de casillas con alto coste, pero en mapas grandes o con muchas casillas de alto coste, obtenemos bajos resultados.