Memoria Practica 2.-Belkan

Niveles 0 y 1

Para estos niveles no tengo nada que comentar significativo ya que el 0 era seguir el tutorial y para el 1 cambiar el tipo de contenedor para abiertos, de stack a queue.

Nivel 2

La diferencia entre el nivel anterior y este era implementar el algoritmo A*, es decir, que lleve en cuenta el costo de las acciones del jugador ordenándolos de menor a mayor para la lista *Abiertos (priority queue)*. Lo más problemático que me encontré fue como adaptar los nuevos nodos y donde implantar las variables bikini y zapatillas.

Para ello creé una nueva clase llamada *supernodo* la cual albergaba el operador > para que se ordenaran en función del costo en la lista de *Abiertos.*

Mi siguiente reto a completar fue como hacer que los hijos heredasen el gasto del padre y aumentaran este en función del movimiento por el que fueron creados. Lo primero fue fácil porque los hijos eran copias del padre por lo que implementé una funcion *calcularGasto* la cual incrementaba el gasto en funcion de la casilla que se situara por argumento y el tipo de movimiento. Quedando tal que *hijoTurnLeft.calcularGasto(mapaResultado[fil][col], actTurnLeft).*

Al principio pensé en dejar las variables para bikini y zapatillas en mi clase SuperNodo pero eso me generaba problemas en el struct *ComparaEstados* los cuales no iban a diferenciar si tenían estos extras. Pues tras varios intentos de como acceder a esas variables o crear un nuevo struct que comparase supernodos en vez de estados, opté por el comentario del profesor de que lo pusiéramos en Estado, lo cual lo agilizó todo con dos ligeros cambios que implicaba acceder desde la clase super nodo a estado para comparar esos atributos según la casilla que estuviese para la función *calcularGato.*

Probando con los ejemplos, me encontré con un error común a otros compañeros de que en el quinto (debía rodear el bosque para coger zapatillas y volver hasta adentrarse de nuevo en este) solo cogía las zapatillas e iba por el bosque, acortando la batería. La solución fue, lo más extraña posible, cambiar en compara estados la comparación entre booleanos, de != a <. Lo cual lo arregló.

Nivel 3

Para este nivel pensé en un principio iterar sobre el list destinos. Lo implementé, y aunque no era problema de esto, no encontraba ningún plan.

Pensé en implementar un contador como condición de parada del while, incrementando éste al final si la casilla actual era un destino.

Con eso no tenia en cuenta en la secuencia si he recorrido el objetivo, además que no reconocía si era la secuencia de objetivos óptima y se basaría en mejor camino objetivo por objetivo y no para los tres. Por recomendación del profesor, incorporé a los nodos cuantos objetivos ha recorrido. De nuevo era otro contador y actualicé ComparaEstados además de modificar super nodo para devolver si uno > otro si uno tiene menos objetivos que otro.

Con esto hice que el while comprobara que el nodo actual no hubiese cogido los tres objetivos, pero ello no funciona. Ni si quiera entraba en el while. Problema mío seguramente.

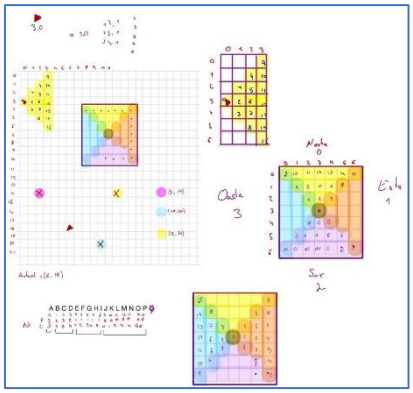
De nuevo, estaba en el problema que no tenia en cuenta **que** objetivo había visitado y cual no. A partir de aquí no lo pensé demasiado bien y di demasiadas vueltas a una solución. Los errores que fui cometiendo los escribo a continuación:

1. Cambié de estrategia a un array de booleanos en estado haciendo los respectivos cambios en el resto de programa, el cual tampoco me funcionó (problema mío de dejarme algo por el camino) ya que me hacia un bucle infinito.
2. Implementé un set en super nodo para los visitados y comparar en el while con su size. También, cambié el for de comprobar los destinos con un find. Además de comprobar que el actual no sea esa casilla y si no la tiene.
3. El set no me funcionaba por comparaestado, si hiciera set tenía que meter en hpp el comparaestados y me daba errores de undefined reference por el orden de escribirlos.
4. Intenté con vector y hacer find yo mismo.
5. Con vector funcionaba, he cambiado el > de super nodo, pero no funcionaba bien. El recorrido no es óptimo, pero si lo quitaba me hacía bucle infinito.
6. Cambié el ultimo for para hacer que se almacene los destinos según su posición en el vector para compararlos uno por uno en el comparaestados y no con size.
7. Creé otro struct para comparar solo filas y columnas para así no hacer recursividad ya que en comaparaestados tenía otro comparaestados para ver la diferencia entre los objetivos.
8. Cambié el compara estados porque me daba fallos para el nivel dos, culpa del vector estaba vacío pues daba segmentation fault.
9. El fallo a esta altura era que no igualaba bien los destinos objetivo con actual.
10. Lo arreglé, pero no llegaba a ningún camino.
11. Pensé: El operador > de supernodo solo tiene en cuenta el gasto y no los objetivos.
    a. Si le añado que compare nvisitados tampoco termina pero si coge nodos con más objetivos pero con más costo.

Me di cuenta de toda la vuelta que estaba dando cuando la solución ya la pasó por mis manos. Implementé de nuevo el array de booleanos en estado para saber, solo por índice en éste, cual había encontrado. Actualicé Comparaestados para que comparase cada uno de los booleanos que representan los objetivos.

Nivel 4

Mi primera meta fue como encontrar una función que me cambiara entre el vector de terreno a un triangulo en una matriz. Opté por la opción de fuerza bruta y hacer para cada opción algo diferente. Siendo común para cada una que el triangulo tenia una altura de cuatro, o cuatro niveles como lo pensé yo y asignando un intervalo para cada nivel.



La foto de la derecha es el croquis que me hice para intentar lograr alguna función.

A esto lo llamé *pintamapa*. Y funciona de perlas.

Para este nivel opté por reutilizar el algoritmo A* para un solo objetivo. Pero mi primer problema con esto es que solo me recorría un objetivo y nunca pasaba al siguiente.

Otro problema que me encontré a la par fue que si me encontraba con un aldeano, teniendo un apartado que me midiera que tenia un aldeano delante y meter un actIDLE en plan, después de irse mi jugador no llegaba al objetivo y se quedaba parado. Mostrando que no encontraba camino.

El problema residia en como detectaba que tenía un aldeano delante. Cambié a ver sensor.superficie para detectar los objetos móviles y quedarme quieto hasta que se fueran.

El problema de que solo alcanzara un objetivo aparentemente se solucionó. A partir de aquí conseguía tres pero de nuevo se quedaba parado.

Cambié que los objetivos solo se rellenasen cuando estuviese vacío contenedor de éstos.

En sí, ahora funcionaba sin problema.

Para que fuera más optimo, implementé que cogiera el camino mas cercano. Creando funciones que me median las distancias entre estados y me ordenasen un vector.

Ahora si iba a por los objetivos mas cercanos. Conseguí varios objetivos más en total (39).

Recordé que había casillas para la recarga de batería, pues implementé que si después de conseguir algún objetivo tuviese poca batería, como en ese momento tiene que rehacer plan, meto la casilla batería (si la encontrara en el mapa) como primer objetivo a tener en cuenta en pathFinding.

Además, hasta que no estuviera cargado a mas de 2000 puntos (a veces no se me recargaba más que eso) no rehacía plan al siguiente objetivo.

Probando valores para decidir cuando recargarse y hasta cuanto, he llegado a 71 objetivos totales (ejemplo 1).

Intenté que se recargara si pasa cerca de una casilla de recarga y hay mas distancia al objetivo que hasta ella. Pero con este cambio se quedaba bloqueado para siempre en la casilla de recarga, sin recargarse al completo.

Por falta de tiempo, no lo implementé y opté por entregar la práctica así.

Cierto es que hasta que no llegué al nivel 3, mas de mediados de nivel 3, no recibí satisfacción por esta práctica. Pero ver como se mueve mi jugador por todos los objetivos que encuentra en el nivel 4 es muy reconfortante y me incita a seguir mejorándolo. Ya que el resto de niveles consigue unos objetivos paupérrimos.

Puede que después de los exámenes ordinarios intente mejorarlo, pero ahora no puedo.

Gracias, Profesor, por dar una semana extra a la entrega y haber podido disfrutar estos dos últimos niveles.