Práctica 2

Nivel 0

Para el nivel 0 simplemente se ha seguido el tutorial, y se ha añadido un bucle while, el cual comprueba ,al tomar un nuevo nodo actual de abierto, si este se encuentra ya en Cerrados. Si se encuentra dicho nodo se descarta y se coge uno nuevo de abierto. El proceso se repite hasta que se coge de abiertos un nodo que no esté en cerrados.

Nivel 1

Se ha cogido el código del nivel 0, y la pila (LIFO) se ha sustituido por una cola (queue) FIFO, cambiando las correspondientes llamadas a `.push()` por llamadas a `.push_back()` y las llamadas a `.top()` por llamadas a `.front()`

Nivel 2

Para el nivel 2 se ha añadido al struct `estado` de `jugador.hpp` dos nuevos atributos booleanos, uno denominado bikini y otro denominado zapatillas, ambos inicializados por defecto a false, y un nuevo atributo entero que contendrá el coste de la lista de acciones asociadas a un nodo.

Se ha definido una nueva función `int calcularCosto(...)`, a la cual se le pasan como parámetros una acción, los booleanos z y b por referencia, los cuales corresponden a los valores de bikini y zapatillas del nodo para el que se quiere calcular el costo de dicha acción, el estado antes de realizar la acción y el mapa. Por su parte la función nos calcula el coste de realizar dicha acción, apoyándose en el mapa para calcular dicho costo, y además en caso de pasar por alguna casilla de bikini o zapatillas, si no se tiene ninguna vestimenta, se activa la vestimenta correspondiente.

Hemos definido un nuevo struct `ComparaNodos` el cual contiene el criterio por el que se ordena la cola con prioridad que utilizaremos en la función `pathFinding_CostoUniforme(...)` que explicaremos ahora. El criterio es que si un nodo tiene un coste menor que otro, le precederá en la cola con prioridad.

También se ha modificado el struct `ComparaEstados` para que a la hora de ordenar estados se tenga en cuenta los valores de bikini y zapatillas.

Finalmente se ha creado una función `pathFinding_CostoUniforme(...)`, la cual es igual que las anteriores, cambiando la estructura usada para Abiertos, la cual es ahora una cola con prioridad. Además al expander los nodos se calcula el coste de cada una de las expansiones y se añade al coste de cada uno de los hijos.

También se ha programado el algoritmo A*. Para ello se han creado un struct que define un nuevo tipo de nodos, los cuales tienen los mismo atributos que los usados hasta ahora, pero se han añadido dos atributos enteros, f y h (g no se añadido, pues se corresponde con el parámetro coste, el cual se encuentra en estado).

También se ha definido una heurística, en este caso la distanci Manhtann. Para programarla me he apoyado en la biblioteca `<math.h>` para el cálculo de valores absolutos. Además se ha creado un struct que contiene el criterio de comparación para ordenar la cola de abiertos, que de nuevo es una cola con

prioridad. El criterio es que los que tiene un valor de f mayor van más atrás en la cola, donde f es la suma del coste hasta el nodo y h donde h contiene el valor devuelto por la distancia Manhatann.

Por último se ha tomado el mismo criterio para ordenar el conjunto de cerrados, pues solo depende de los estados y no del tipo de nodo. Y al expandir cada hijo, a parte de añadir el coste de la acción al coste del nodo se calcula la distancia manhatann y se hace la suma para calcular f.

Nivel 3

Para el nivel 3, se ha añadido un atributo de tipo `vecto<bool>` denominado objetivos visitados. Al llamar a la función `pathFindind_CostoUniforme_Nivel3` se insertan en dicho vector tantos `false` como destinos se le haya pasado a la función como parámetro. También se crea localmente un `vector<estado>` que contengan los estados, de esta manera ambos vectores tienen igual longitud y así cada false del vector de bool está asociado a un destino.

Se hace que cada vez que se expande un hijo se comprueba si el estado del hijo coincide con alguno de los estados de nuestro vector de destinos, en cuyo caso el false del vector de bool asociado a dicho destino se pone a true. El criterio de salida del bucle while es que el nodo actual tenga los tres valores del vector de bool a true.

También se ha modificado el `ComparaEstadosCU_Nivel3` de manera que tenga en cuenta los estados visitados.

También se ha programado el nivel 3 con el algoritmo A*. Para ello hemos utilizado el mismo compara estados y la misma condición de salida del bucle while que en el caso de CostoUniforme. También se ha reutilizado la idea del vector booleano. La principal diferencia es que se ha actualizado la eurística, de forma que se calcula la distancia manhatan para todos los objetivos, y se devuelve el menor de ellos. Todo lo demás se ha reutilizado del algoritmo para costo uniforme