Esther García Gallego.
2ºB Ingeniería Informática.

MEMORIA DE LA PRÁCTICA 2

Nivel 1: Búsqueda en anchura.

Para el nivel 1 utilicé la misma estructura que el nivel 0 (búsqueda en profundidad) pero cambié el contenedor de una pila (`stack`) por una cola (`queue`).

```
bool ComportamientoJugador::pathFinding_Anchura(const estado &origen, const estado &destino, list<Action> &plan) {
    //Borro la lista
    cout << "Calculando plan\n";
    plan.clear();
    set<estado,ComparaEstados> Cerrados; // Lista de Cerrados
    queue<nodo> Abiertos;                // Cola de Abiertos
```

Como consecuencia, hubo que cambiar la forma de obtener el siguiente valor de la cola de nodos abiertos puesto que las colas no poseen el método `top()` sino el `front()`.

```
// Tomo el siguiente valor de la Abiertos
if (!Abiertos.empty()){
    current = Abiertos.front();
}
```

Para optimizar los tiempos, impuse la condición de que, para añadir un nuevo nodo dentro del `while`, no puede estar en la lista de nodos “cerrados”.

```
while (!Abiertos.empty() and (current.st.fila!=destino.fila or current.st.columna != destino.columna)){

    Abiertos.pop();

    if(Cerrados.find(current.st) == Cerrados.end()){    // optimización
        Cerrados.insert(current.st);
```

Nivel 2: Búsqueda de coste uniforme.

El nivel 2 es un poco más complejo. Primeramente, he utilizado una cola con prioridad (`priority_queue`) para ordenar los nodos en función de su coste. Así pues, he tenido que añadir un atributo de tipo `int` llamado coste al `struct` de los nodos. Este coste depende de si se va a girar o a seguir recto y del tipo de casilla en la que se va a realizar el movimiento, para determinarlo he creado dos funciones: `getCoste` y `getCosteGiro`.

```
struct nodo{
    estado st;
    int coste = 1;
    list<Action> secuencia;
};
```

```
// Devuelve el coste de avanzar en una casilla en el mapa
int ComportamientoJugador::getCoste(estado &hijo){
    int coste = 1;
    unsigned char casilla = mapaResultado[hijo.fila][hijo.columna];

    if(casilla == 'A' && hijo.bikini == true)
        coste = 10;
    else if (casilla == 'A' && hijo.bikini == false)
        coste = 200;

    if(casilla == 'B' && hijo.zapatillas == true)
        coste = 15;
    else if (casilla == 'B' && hijo.zapatillas == false)
        coste = 100;

    if(casilla == 'T')
        coste = 2;

    return coste;
}
```

Como en este nivel debemos tener en cuenta las casillas de bikini y zapatillas, he añadido dos booleanos a la estructura que definía el estado de los nodos. Posteriormente, tuve que introducir dos condicionales dentro del bucle que alteraran el estado de dichos atributos si el jugador se encontraba en alguna casilla “K” o “D”, respectivamente.

```
struct estado {
  int fila;
  int columna;
  int orientacion;
  bool bikini, zapatillas;
};
```

```
// Siguiente valor en cola
if (!cola.empty()){
    current = cola.top();

    // Comprobar si tenemos bikini o zapatillas
    if(casilla == 'K' && current.st.bikini == false){
        //cout <<"\t CASILLA BIKINI: ";
        current.st.bikini = true;
        current.st.zapatillas = false;
        //cout << current.st.bikini;
    } else if (casilla == 'D' && current.st.zapatillas == false){
        //cout <<"\t CASILLA ZAPATILLAS: ";
        current.st.bikini = false;
        current.st.zapatillas = true;
        //cout << current.st.zapatillas;
    }
}
```

Por último, para optimizar el código, he incluido la misma condición que en el nivel anterior.