Defensa de la práctica 2:
Los extraños mundos de Belkan.

Nivel 1:
El objetivo de este nivel es crear y llevar a cabo un plan en el que el agente alcance la casilla objetivo desde una posición inicial en el menor número de acciones. He logrado la creación de dicho plan implementando un algoritmo de búsqueda en anchura.
Este algoritmo dispone de un conjunto ordenado de nodos cerrados que son las casillas del mapa sin visitar y a diferencia del algoritmo de búsqueda en profundidad, disponemos de una cola de nodos abiertos que son los nodos que vamos visitando conforme avanzamos. Mientras no lleguemos al nodo objetivo, casilla objetivo, vamos generando los hijos de la casilla actual, un descendiente que gire a la derecha, un descendiente que gire a la izquierda y otro que avance. En este último, comprobamos si delante hay un obstáculo que impida el avance del agente y cada hijo se añade a la cola de Abiertos. Al final del While generamos el siguiente nodo y actualizamos el nodo current. Una vez terminada la búsqueda, el algoritmo se quedara con el plan más corto y el agente llegará al objetivo.

Nivel 2:
El objetivo de este nivel es crear y llevar a cabo un plan en el que el agente alcance la casilla objetivo desde una posición inicial teniendo en cuenta el menor consumo posible de batería. En este nivel he implementado un algoritmo de búsqueda de coste uniforme.

Para este nivel he necesitado varias funciones auxiliares para implementar el algoritmo:

- -Un struct comparaCoste para que en la priority_queue de nodos Abiertos del algoritmo me ordene los nodos según su coste uniforme.
- -Una función CalcularCosteCasilla que calcule el coste de cada casilla según su tipo, además si dispone de un bikini o unas zapatillas reduce el coste dependiendo si la casilla es ‘A’ o ‘B’.
- -Una función CalcularCosteUniforme que calcule el coste uniforme de la casilla que viene calculado por la suma de su coste de casilla y el coste uniforme de su nodo padre.
- -En el struct comparaEstados también añadí una comparación de si el agente tiene bikini o tiene zapatillas.
- -En el struct estado añadí costeUniforme, costeCasilla, zapatillas y bikini de atributos, cada uno inicializado, ya que me resultaba mas fácil la implementación de todo.

Hablando del algoritmo, tiene el mismo conjunto de nodos cerrados que el anterior algoritmo pero con la diferencia que la lista de nodos abiertos es una priority_queue donde va ordenando los nodos según su coste uniforme. Primero comprobamos si en la casilla inicial hay o no zapatillas o bikini, el agente solo puede tener uno de los dos. Después calculamos el coste de la casilla inicial, lo añadimos a Abiertos y mientras no lleguemos al nodo objetivo, generamos los hijos de la casilla actual, un descendiente que gire a la derecha, un descendiente que gire a la izquierda y otro que avance y al mismo tiempo calculamos sus costes según su casilla y sus costes uniformes. Una vez generados se añaden a Abiertos. Al final del While generamos el siguiente nodo y actualizamos el nodo current y volvemos a comprobar si el siguiente nodo tiene bikini o zapatillas. Finalmente, el algoritmo se quedará con el camino que consuma menos energía, interpretando si es necesario, que el agente se desvie si dispone de unas zapatillas o bikini cercanos a él y que el camino final sea el menos costoso.