

SISTEMAS INTELIGENTES

Práctica 1. Búsqueda heurística

Nombre: Francisco Javier Rico Pérez

Grupo: Martes - 15:00 / 17:00

Correo: fjr8@alu.ua.es

28 de octubre del 2018

Índice

1. Introducción.
2. Explicación del pseudocódigo implementado.
 - a. Posibles casos de suceso.
3. Explicación de la heurística.
4. Explicación y traza de un problema pequeño y similar.
5. Pruebas de los distintos casos del problema.

1. Introducción

Esta primera práctica ha consistido en implementar el algoritmo A* para buscar el camino de menor coste entre dos puntos diferenciados como un caballero y un dragón. El plano sobre el cual hemos trabajado consistía en un mapa de celdas hexagonales, las cuales podían constar de una serie de obstáculos (piedras, bloques, hierba y agua) los cuales varían el coste del camino.

2. Explicación del pseudocódigo implementado

```
ListaInterior = vacio;
ListaFrontera = nodoInicio;
resultado = -1;
mientras ListaFrontera no sea vacia && resultado == -1
    n = obtenerNodoMenorF(ListaInterior);
    si n esMeta
        listaInterior.add(n)
        coste_total = n.f;
        reconstruirCamino(listaInterior);
        resultado = 0;
        encontrado = true;
    sino
        listaFrontera.remove(n);
        listaInterior.add(n);
        hijos = getHijos(n, listaInterior, destino);

        Para m hasta hijos no sea vacio
            si(existeEnFrontera(m, listaFrontera) es falso)
                listaFrontera.add(m);
            sino
                si m.g < enFrontera.g
                    listaFrontera.remove(enFrontera);
                    listaFrontera.add(m);
                fin-si
            fin-si
        fin-para
    fin-si
fin-mientras
```

Este es el pseudocódigo de la implementación del algoritmo A*, consiste básicamente en que tras haber inicializado la listaFrontera con el nodo de la posición inicial (el caballero), obtenemos de dicha lista el nodo con menor F, lo borramos de listaFrontera y lo añadimos en listaInterior una vez hecho comprobamos si el nodo actual en el que estamos (reflejado en el pseudocódigo como 'n') es el nodo meta, el nodo cuyas coordenadas sean las mismas que la coordenada destino (el dragón), sino es el nodo meta sacamos sus hijos y de cada hijo (reflejado como 'm') comprobamos que no está en listaFrontera lo añadimos, si se da el caso de que esta en

listaFrontera comprobamos si la G del nuevo hijo es mejor que la anterior y si lo es, borramos el anterior y añadimos el nuevo, por otro lado si el nodo 'n' es el nodo meta reconstruimos el camino y termina el algoritmo.

Hay que hacer unas pocas explicaciones de los siguientes métodos:

obtenerNodoMenorF

```
public Nodo obtenerNodoMenorF(ArrayList<Nodo> ns){
    Nodo n = null;
    if(ns.size() == 1){
        n = ns.get(0);
    }
    else{
        int pos = 0;
        for(int i = 0; i < ns.size(); i++){
            if(ns.get(pos).getF() > ns.get(i).getF()){
                pos = i;
            }
        }
        n = ns.get(pos);
    }
    return n;
}
```

Básicamente devuelve el nodo de listaFrontera que tenga menor G y si existen dos nodos con la misma G devuelve el nodo que esté antes en el ArrayList.

esMeta

```
public boolean esMeta(Nodo n, Coordenada meta){
    return n.getCoordenadaActual().getX() == meta.getX()
        && n.getCoordenadaActual().getY() == meta.getY();
}
```

Este método sencillamente devuelve true si el nodo que estamos evaluando es el nodo meta es decir el dragón y falso en el caso de que no lo sea.

getHijos

```

public ArrayList<Nodo> getHijos(Nodo n, ArrayList<Nodo> li, Coordenada finalcoord){
    ArrayList<Nodo> hijos = new ArrayList<Nodo>();
    Coordenada actual = n.getCoordenadaActual();
    Nodo hijo;
    Cubo c1 = null;
    Cubo c2 = null;
    int valorCeldaHijo;
    boolean par = esFilaPar(n);
    boolean inalcanzable1, inalcanzable2;
    Coordenada [] cs = {new Coordenada(-1, -1), new Coordenada(-1, 0),
        new Coordenada(-1, 1), new Coordenada(0, 1), new Coordenada(1, 1),
        new Coordenada(1, 0), new Coordenada(1, -1), new Coordenada(0, -1)};
    if(par){
        Coordenada inalcanzable_Par_1 = new Coordenada(actual.getX() - 1, actual.getY() - 1);
        Coordenada inalcanzable_Par_2 = new Coordenada(actual.getX() - 1, actual.getY() + 1);
        for(int i = 0; i < cs.length; i++){
            Coordenada coordenadaHijo = new Coordenada(cs[i].getX() + actual.getX(), cs[i].getY() + actual.getY());
            inalcanzable1 = esCoordenadaInalcanzable(coordenadaHijo, inalcanzable_Par_1);
            inalcanzable2 = esCoordenadaInalcanzable(coordenadaHijo, inalcanzable_Par_2);
            if(!inalcanzable1 && !inalcanzable2){
                valorCeldaHijo = getValorCeldaHijo(coordenadaHijo);
                if(valorCeldaHijo != 0){
                    if(!existeEnInterior(coordenadaHijo, li)){
                        float heuristica = heuristicaManhattan(coordenadaHijo, finalcoord);
                        //float heuristica = heuristica0();
                        //float heuristica = heuristicaEuclidea(coordenadaHijo, finalcoord);
                        //float heuristica = heuristicaHexagonales(c1, c2);
                        hijo = new Nodo(coordenadaHijo, n.getG() + valorCeldaHijo, heuristica, n);
                        hijos.add(hijo);
                    }
                }
            }
        }
    }
}

```

Este método es el más complejo de la práctica, consiste en devolver un ArrayList con los hijos del nodo que estamos explorando, para ellos necesitamos saber si estamos en una fila par o una fila impar, sabiendo eso podemos ver que coordenadas son inaccesibles debido a la condición de que estamos trabajando con un mapa hexagonal, una vez sabemos eso recorreremos el array de posibles coordenadas comprobamos si la que hemos generado es alguna de esas dos coordenadas y si no lo es calculamos el valor de la celda que estamos evaluando, es decir miramos si es camino, hierva, agua o una piedra y de seguido comprobamos si está en listaInterior y si no lo está lo añadimos al arrayList de posibles hijos (para no hacer una imagen más grande solo he mostrado el proceso de las filas pares, pero el funcionamiento es idéntico para los impares).

reconstruirCamino

```

public void reconstruirCamino(ArrayList<Nodo> li, char c[][]){
    Nodo actual = null;
    Nodo padre = null;
    actual = li.get(li.size() - 1);
    padre = actual.getPadre();
    while(padre != null){
        c[actual.getCoordenadaActual().getY()][actual.getCoordenadaActual().getX()] = 'X';
        actual = padre;
        padre = actual.getPadre();
    }
}

```

Para reconstruir el camino sencillamente he ido apuntando a los padres desde el último nodo insertado en listaInterior. el dragón, hasta llegar al nodo cuyo padre sea valor null es decir el caballero.

2.1 - Posibles casos de suceso

Durante el proceso de ejecución del algoritmo se pueden producir una serie de casos a la hora de evaluar nodos.

CASO 1º

Este es normal de ver cuando estamos evaluando con la heurística con un valor fijo de 0, y es que se da el caso de que en ningún momento vamos a entrar en la condición de comparar el valor de G cuando la G del nodo recién encontrado es menor que la del anterior que ya se encontraba en listaFrontera.

CASO 2º

Es el contrario del anterior, se puede dar el caso de que encontremos un nodo con menor G que tenga las mismas coordenadas que uno que ya está en listaFrontera pero con menor G, si se da este caso sencillamente borramos el de menor G de listaFrontera y añadimos el que hemos encontrado.

CASO 3º

Puede darse el caso de que uno de los nodos que estamos evaluando ya se encuentre en insertado anteriormente en listaInterior, si se da este caso sencillamente no tomamos en cuenta a este nodo y pasamos a evaluar el siguiente.

CASO 4º

Este es un caso muy concreto de trabajar con un mapa hexagonal, a la hora de seleccionar los nodos que vamos a evaluar, tenemos que tener en cuenta si la fila en la que nos encontramos es par o impar, dado que realmente tenemos acceso a ocho nodos pero al trabajar con hexágonos sólo podemos tener seis hijos, tal y como queda reflejado en la siguiente imagen.

A -> Nodo accesible
N -> Nodo no accesible

Nodo Actual con fila IMPAR

A -1,-1	A 0,-1	N 1,-1
A -1,0	Nodo Actual	A 1,0
A -1,1	A 0,1	N 1,1

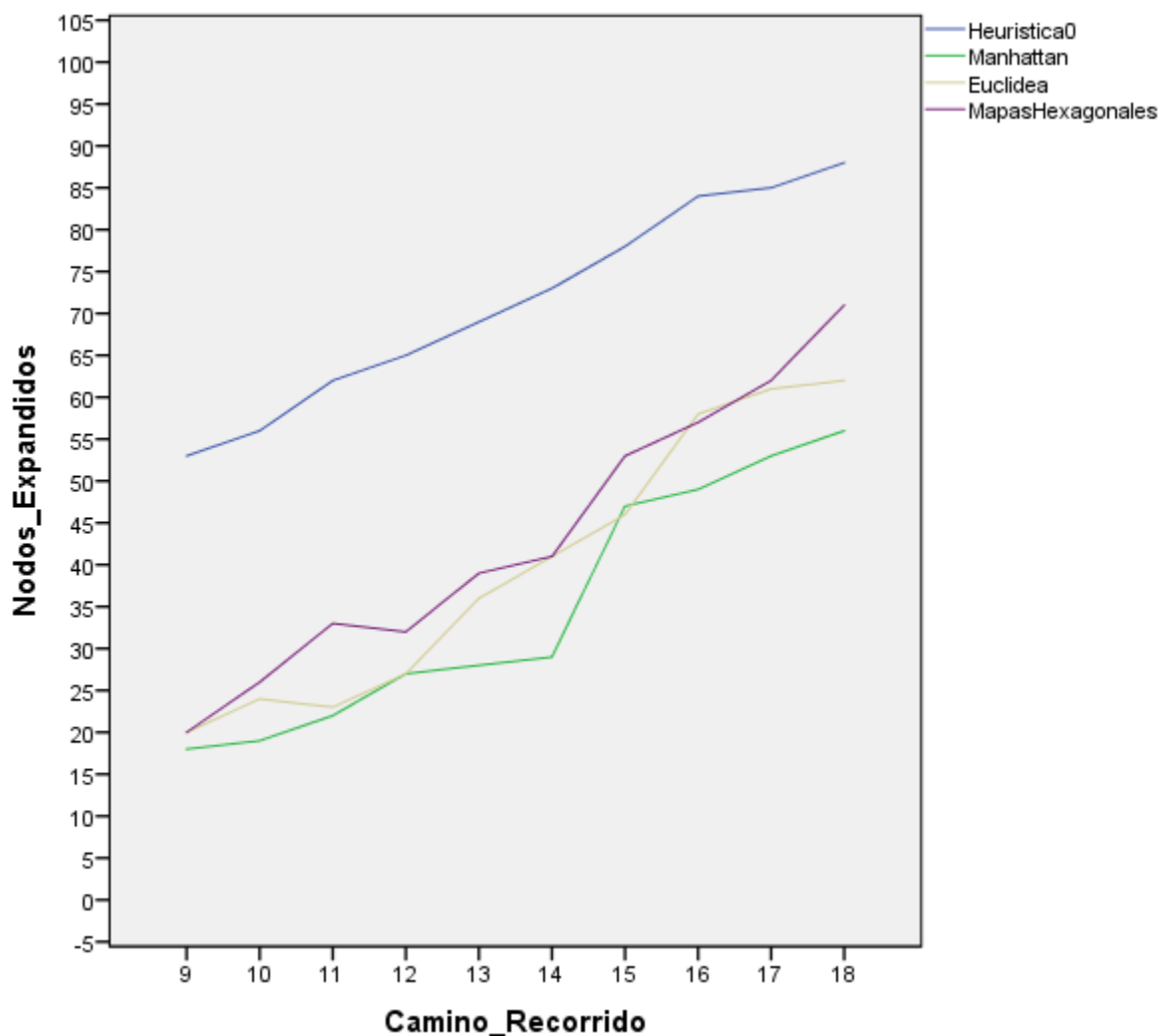
Nodo Actual con fila PAR

N -1,-1	A 0,-1	A 1,-1
A -1,0	Nodo Actual	A 1,0
N -1,1	A 0,1	A 1,1


3. Explicación de la heurística.

En esta práctica hemos trabajado con un total de cuatro heurísticas (la $h = 0$, la distancia Manhattan, la distancia Euclídea y por último las distancias adaptadas a mapas hexagonales).

A continuación veremos una serie de gráficos los cuales muestran una recopilación de datos de cada una de las heurísticas y sus resultados en función del camino recorrido y la cantidad de nodos expandidos.



Se observa que con la distancia Manhattan como valor de la heurística es con la que menos nodos se expanden pero que la con la Euclídea cuando el



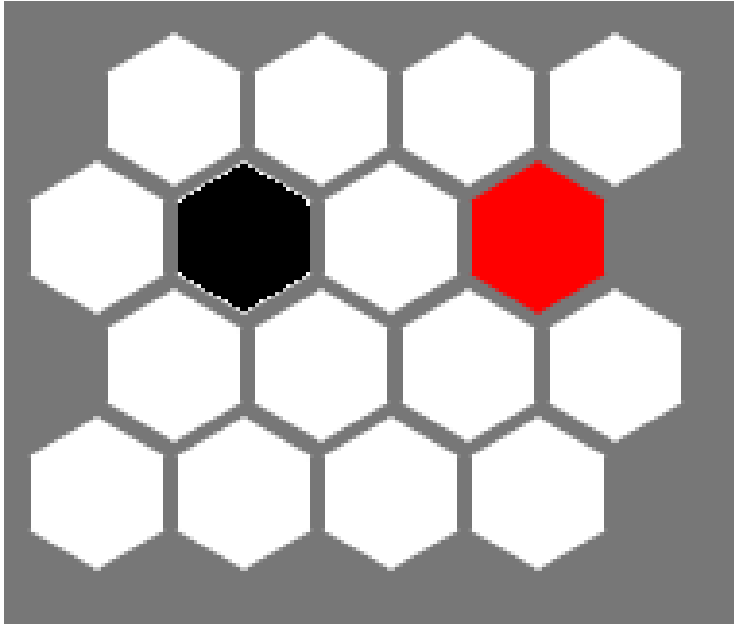
camino tiene como valor mínimo 12, que la Euclídea tiene una media menor, esto puede deberse a que con esta distancia para cada mínimo abarcan un rango de nodos bastante amplio. Podemos observar en el gráfico para mapas hexagonales que se exploran bastantes nodos.

Para este problema en concreto, la mejor opción para calcular el camino es la distancia adaptada a mapas hexagonales, ya que el valor de heurística es la misma alrededor de los nodos hasta llegar al destino.

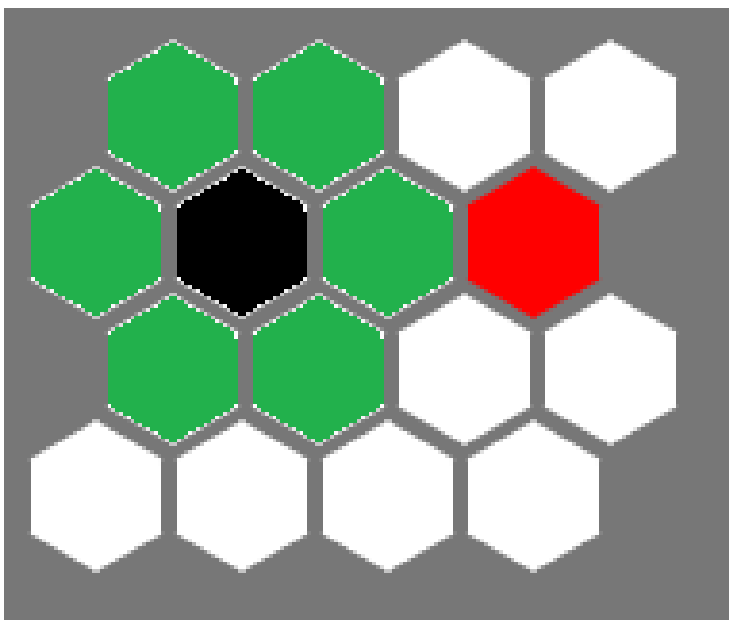
Lo que queda claro es que cuanto más se aleja h de h^* es que la solución es menor óptima, por otro lado ocurre lo contrario cuando más se acerca mejor es la solución.

4. Explicación y traza de un problema pequeño y similar.

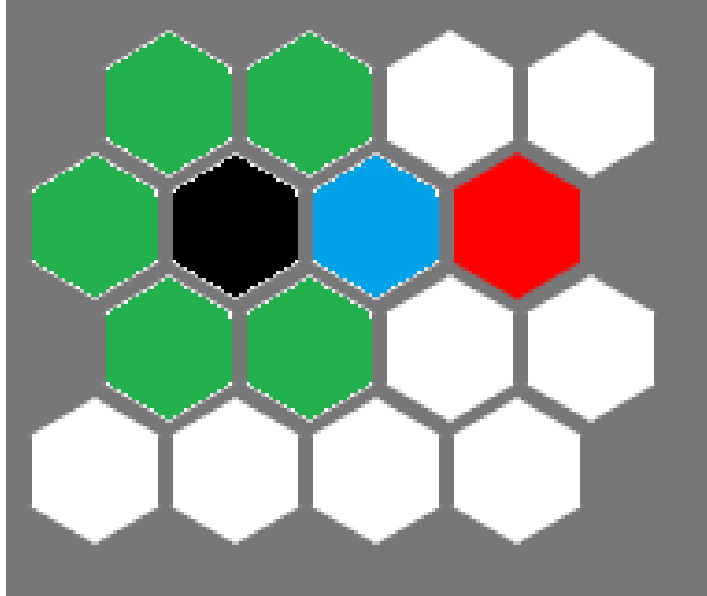
Vamos a hacer un ejemplo con un mapa hexagonal de 4x4



Primero tenemos que comprobar si el nodo en el que estamos, el de la casilla negra es el nodo de la meta, como no lo es, entonces lo añadimos a `listaInterior` y lo borramos de `listaFrontera` y sacariamos a sus hijos tal y como se ve en la siguiente imagen.



Una vez hemos obtenido los hijos, vamos obteniendo de listaFrontera los nodos con mejor G y sus hijos hasta que llegamos al nodo que genera como hijo el nodo meta, tal y como se muestra en la siguiente imagen.



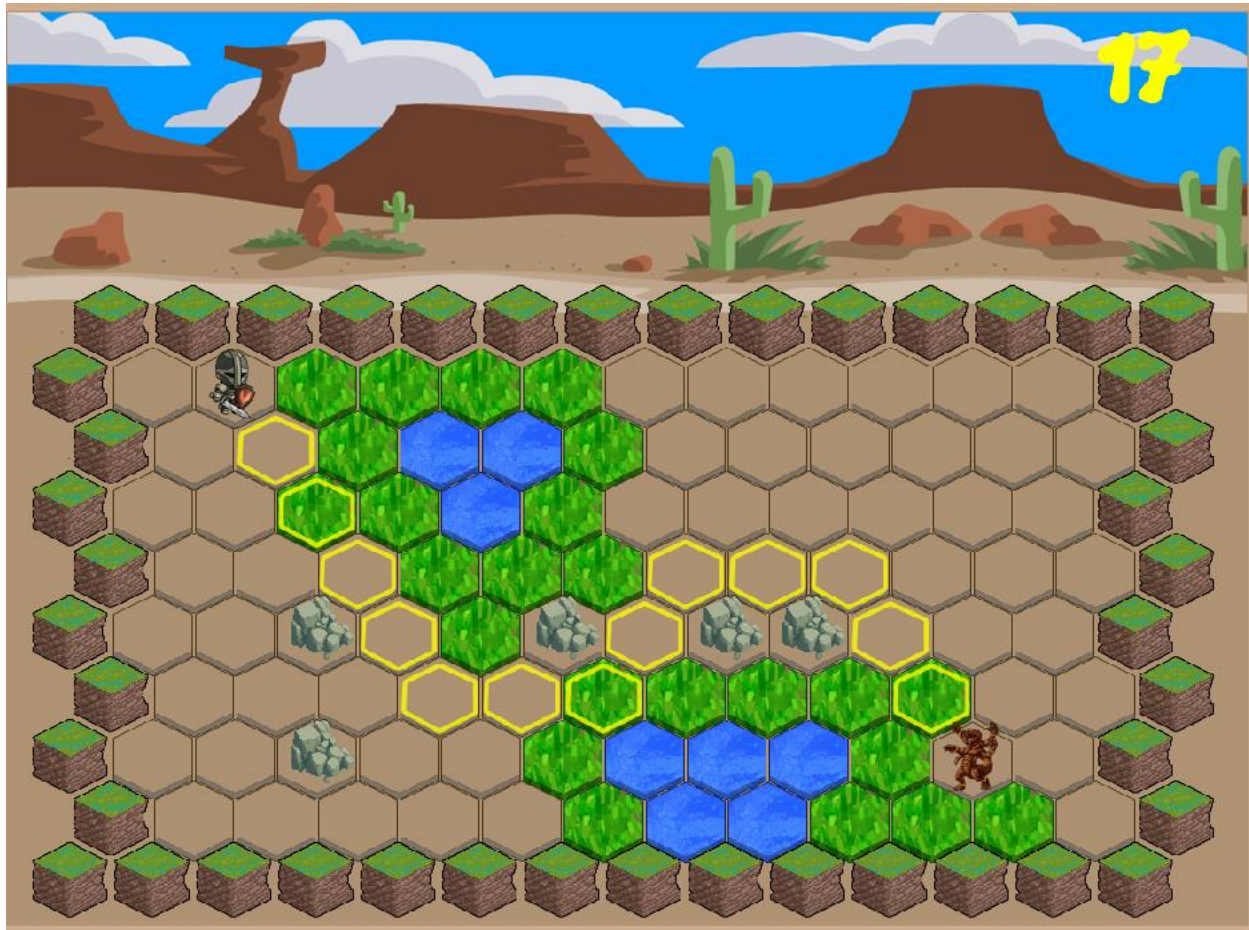
Este nodo generará el nodo meta como hijo y lo añadirá a listaFrontera y cuando estemos buscando en listaFrontera y lo seleccione como nodo con mejor G, se comprobará que es la meta y devolverá el camino a partir de ese nodo.



5. Pruebas de los distintos casos del problema.

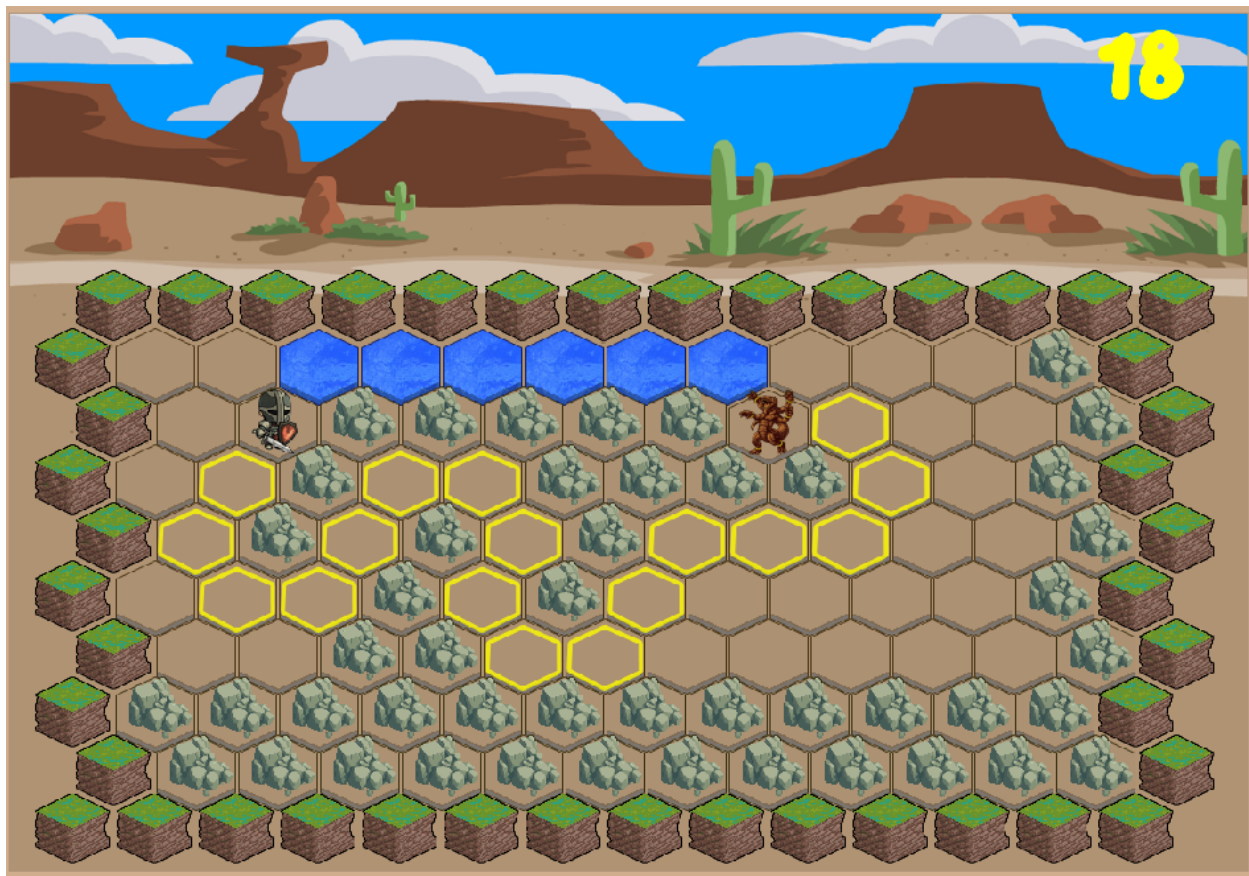
Las pruebas van a ser implementadas sobre el mapa por defecto que se nos entrega como parte de los archivos de la práctica, y además será probado con la distancia adaptadas a mapas hexagonales ya que es la más adecuada para este tipo de problemas:

1 - Mapa sin modificaciones:



Como podemos observar gráficamente el se realiza el camino normal sin ningún tipo de problema.

2 - Mapa con solo 2 opciones de camino



Como se observa en este mapa concretamente sólo hay 2 opciones de camino uno de todo agua y otro de camino, dado a la forma de recoger los nodos de la práctica que he implementado va por el camino normal.

3 - Mapa en el que no se puede encontrar un camino

```
:desktop:classes UP-TO-DATE
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 14
    at AEstrella.Mundo.CargarMundo(Mundo.java:84)
    at es.sistemasinteligentes.practicalsi.desktop.DesktopLauncher.main(DesktopLauncher.java:28)
:desktop:run

Deprecated Gradle features were used in this build, making it incompatible with Gradle 5.0.
See https://docs.gradle.org/4.6/userguide/command\_line\_interface.html#sec:command\_line\_warnings

BUILD SUCCESSFUL in 4s
4 actionable tasks: 1 executed, 3 up-to-date
```

Como se observa en la imagen si generamos un mapa en el cual haya piedras que no permiten crear un camino el programa saltará con una excepción.

—