SISTEMAS INTELIGENTES

Departamento de Ciencia de la Computación e Inteligencia Artificial

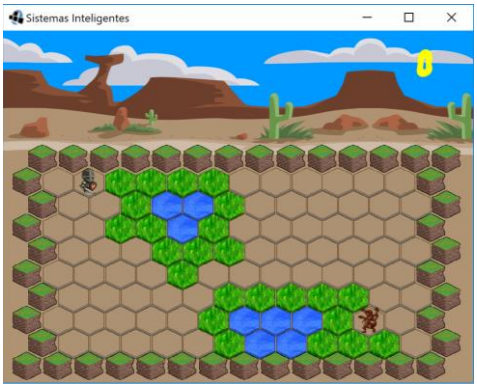
Universidad de Alicante **Curso 2018/2019**

Práctica 1. Búsqueda heurística

Objetivos:

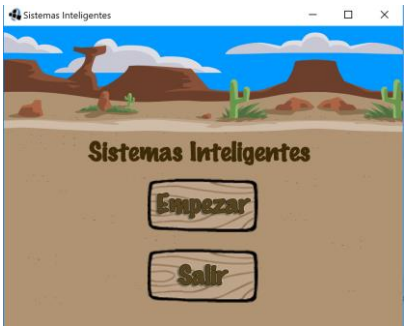
- Comprender el funcionamiento de la búsqueda heurística y en concreto del algoritmo A*.
- Implementar el algoritmo A* y saber cómo seleccionar una heurística apropiada al problema.
- Realizar un análisis cuantitativo respecto al número de nodos explorados con este algoritmo.

En esta primera práctica de la asignatura se debe desarrollar el algoritmo A* para calcular el camino entre dos personajes de un juego. En nuestro juego concreto tenemos un tablero de celdas hexagonales y dos personajes: un caballero y un dragón. Se debe implementar el algoritmo A* para calcular el camino de menor coste por el que el caballero puede llegar hasta el dragón.



Funcionamiento del juego

Al ejecutar el juego aparecerá una pantalla principal, como la mostrada en la siguiente imagen, con el menú principal compuesto únicamente de dos botones. Desde el botón superior iniciamos el juego y con el botón inferior cerramos la aplicación.



Al pulsar sobre Empezar se abre la ventana del juego. En esta ventana encontramos el tablero de juego. Este tablero está formado por celdas hexagonales, por tanto, los personajes tendrán como máximo 6 movimientos posibles. Las celdas que forman el tablero pueden ser de diferente tipo:

- Celdas por donde los personajes pueden cruzar: son las celdas de camino, de agua o de hierba.
- Celdas donde los personajes no pueden acceder: son las celdas de bloque, que se utilizan para marcar los límites del tablero, y las celdas de piedra, que se pueden colocar para evitar el paso por ese lugar.

Las celdas marcadas en amarillo, es el camino detectado por el algoritmo A*.

Cuando accedemos al juego, directamente se calcula el camino entre los dos personajes. En la parte superior derecha de la ventana aparece el coste total de ese camino.

El objetivo del juego es conseguir darle tiempo al dragón para que escape del caballero y, ¿cómo hacemos esto? Intentando alargar el camino hasta un objetivo. Para esto, debemos ir colocando bloques de piedra en el tablero haciendo clic sobre la celda en la que deseamos colocar el bloque. Cada vez que hacemos clic sobre el tablero ocurren varias cosas:

1. se coloca un bloque de piedra en esa celda.
2. se mueven una celda aleatoriamente los personajes.
3. se recalcula el A* para la nueva configuración del tablero.

Debemos tener en cuenta que el coste de pasar por camino, por hierba y por agua no es el mismo. Al caballero le costará más cruzar la hierba que el camino y le costará más cruzar el agua que la hierba.

Si llegamos al coste objetivo, habremos ganado la partida.

¡OJO! Siempre debemos dejar que exista un camino. Si colocando bloques de piedra bloqueamos totalmente que el caballero pueda llegar al dragón, habremos perdido.

Proyecto NetBeans

Para desarrollar el juego se ha utilizado libGdx. LibGdx es un framework de desarrollo de videojuegos open-source basado en Java. Esta librería permite desarrollar aplicaciones para Android, iOS, HTML5 o PC con el mismo código base. Para crear un proyecto se debe descargar la librería (gdx-setup.jar) y al ejecutar este fichero aparece una ventana como la mostrada a continuación.

¡OJO! No debéis crear el proyecto con libGdx, el proyecto ya os lo damos creado. Esto es meramente explicativo para entender la estructura del proyecto NetBeans.

En esta ventana, además de darle un nombre al proyecto se debe indicar para qué plataformas se desea desarrollar y qué extensiones (como por ejemplo librerías de físicas) se desean utilizar. Al pulsar sobre Generate, crea automáticamente un proyecto. Este proyecto creado es el que debemos importar a NetBeans, Eclipse u otro IDE que queramos utilizar.

Una de las características del proyecto generado por libGDX es que está basado en Gradle, un sistema de gestión para automatizar el proceso de construcción (build) de un proyecto con la ventaja de que gestiona la descarga y configuración automática de dependencias u otras librerías.

Para abrir el proyecto desde NetBeans

NetBeans soporta este tipo de proyectos, pero no por defecto, para ello se debe instalar el plugin *Gradle Support*.

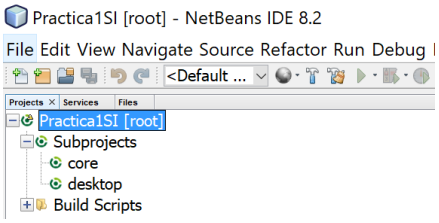
En NetBeans, seleccionamos el menú *Tools->Plugins*.

En la ventana que aparece, seleccionamos la pestaña *Available Plugins* y buscamos el plugin *Gradle Support*. Pulsamos sobre *Install*. Cuando termine de instalar el plugin, ya podremos abrir el proyecto NetBeans.

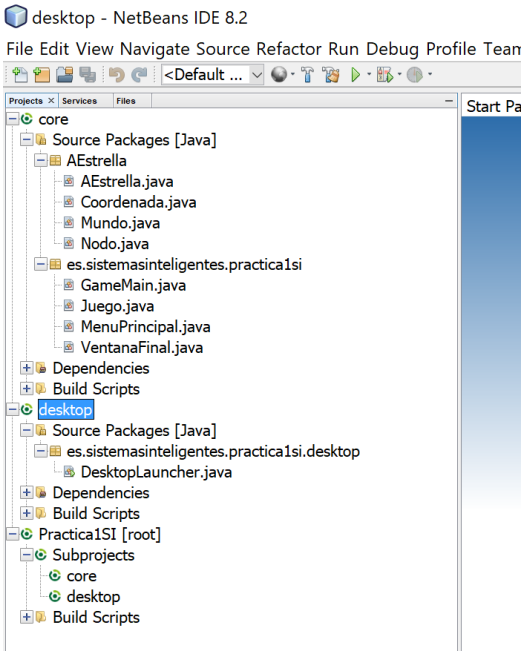
Junto al enunciado disponéis del proyecto NetBeans de la práctica. En este proyecto está desarrollado todo el entorno del juego a falta de implementar el algoritmo A*.

Si hemos instalado el plugin Gradle Support, en NetBeans debemos seleccionar la opción Abrir proyecto y seleccionar la carpeta donde este se encuentra.

Al abrir el proyecto, en la ventana Projects donde se muestra la estructura de ficheros, aparece Practica1SI que es el nombre del proyecto general y, bajo él, aparece Subprojects que, si lo desplegamos, veremos dos subproyectos: core y desktop. Cuando creamos un proyecto con libGdx, se crea un subproyecto general (core) y un subproyecto por cada tipo de dispositivo que hayamos elegido, en nuestro caso, como solo creamos la aplicación para escritorio, solo tenemos el subproyecto desktop, si hubiéramos seleccionado Android o iOS, tendríamos un subproyecto para cada uno de estos dispositivos.



Para abrir los subproyectos solo tenemos que hacer doble clic sobre ellos.



Subproyecto Desktop

Dentro del subproyecto desktop solo encontramos la clase DesktopLauncher.java. Es la clase que lanza la aplicación para escritorio. Es aquí donde se lee el fichero del mundo (tablero). Si se le pasa un nombre de fichero por parámetro, leerá este fichero, si no se le pasa ningún parámetro cargará un mundo por defecto. Después de cargar el mundo, lanza la aplicación pasándole el mundo leído.

Subproyecto Core

En este subproyecto es donde se incluye el código general de toda la aplicación. Se compone de dos packages. En uno de ellos se encuentran todas las clases de visualización y gestión del juego, en el otro se encuentran las clases utilizadas para el A*.

GameMain.java: Es la clase principal que se lanza al ejecutar la aplicación. Esta clase lee el mundo que le llega por parámetro y lanza la ventana con el menú principal.

MenuPrincipal.java: Dibuja el menú inicial del juego con dos botones, para empezar a jugar o para salir.

VentanaFinal.java: Muestra la pantalla final, es la que indica si se ha ganado o perdido.

Juego.java: Es la clase que contiene toda la lógica del juego. Desde aquí se dibuja el tablero, los personajes, se llama al A* y se dibuja su resultado. Gestiona los clics de ratón para modificar el mundo y recalcular el resultado.

Por otro lado, dentro del paquete AEstrella, encontramos las clases:

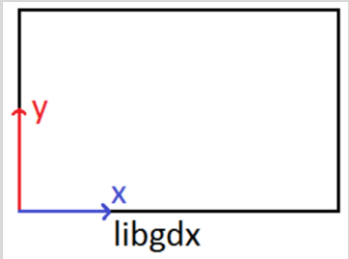
Coordenada.java: Esta clase es de apoyo para el mundo y el AEstrella. Es una clase compuesta de dos valores enteros para almacenar la coordenada x y la coordenada y, de una celda o de alguno de los personajes.

Mundo.java: Esta clase guarda la estructura del mundo. Contiene su tamaño (x, y), las coordenadas del caballero y del dragón y cada una de las celdas del tablero en una matriz de caracteres.

Tamaño del mundo

El tablero se compone de 10 filas y 14 columnas. En la clase mundo, tamayo_x guarda las columnas y tamanyo_y guarda las filas. Es decir, para seguir la nomenclatura de libGdx, el mundo se crea como:

```
mundo = new char[tamanyo_y][tamanyo_x];
```



AEstrella.java: En esta clase debéis implementar el algoritmo A*. En un apartado posterior veremos los detalles de esta clase.

Ejecución del proyecto

La clase que lanza la aplicación para el ordenador se encuentra en el subproyecto Desktop (DesktopLauncher.java), por lo tanto, para ejecutar el proyecto se debe realizar desde este subproyecto. Es posible hacerlo pulsando con el botón derecho sobre el subproyecto Desktop y seleccionar la opción Run del menú que aparece o bien, seleccionando el subproyecto Desktop y pulsar la opción *Run project (dektop)* del menú superior *Run* (o pulsando F6).

Definición del mundo

La especificación del mundo se realiza en un fichero de texto que se pasa como parámetro al programa.

Este fichero de texto tendrá 10 filas y 14 columnas y se utilizarán unos caracteres definidos:

- ‘b’: celda de bloque.
- ‘c’: celda de camino.
- ‘h’: celda de hierba.
- ‘a’: celda de agua.
- ‘p’: celda con piedra.
- ‘k’: posición inicial del caballero.
- ‘d’: posición inicial del dragón.

A continuación se muestra un ejemplo de mundo:

```
b b b b b b b b b b b b b b
b c k h h h h c c c c c c b
b c c h a a h c c c c c c b
b c c h h a h c c c c c c b
b c c c h h h c c c c c c b
b c c p c h p c p p c c c b
b c c c c c h h h h h c c b
b c c p c c h a a a h d c b
b c c c c c h a a h h h c b
b b b b b b b b b b b b b b
```

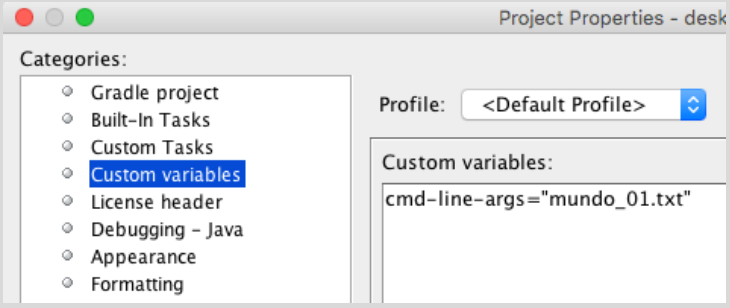
Como resultado obtendremos el siguiente tablero:

Paso del fichero mundo por parámetro

Cuando ejecutamos la práctica, por defecto se ejecuta cargando el mundo "mundo_defecto.txt". Este mundo se encuentra en la carpeta */core/assets*.

Si queremos cargar otros mundos, debemos incluir los archivos txt que definen el mundo en la carpeta anterior y podemos pasar por parámetro el nombre del mundo que queremos cargar.

Ese paso de parámetro se realiza desde el menú *Run >> Set Project Configuration >> Customize*, en el apartado *Custom variables*.



El algoritmo A*

Este algoritmo es uno de los más utilizados para encontrar un camino o ruta entre dos puntos, con la característica que, si se cumplen unas condiciones, el camino encontrado será el camino de menor coste entre los dos puntos y además, si existe, siempre encontrará ese camino (algoritmo completo).

El algoritmo A* ha sido, y continúa siendo, un algoritmo utilizado en diferentes campos por su sencillez y agilidad. Por ejemplo, en robótica móvil se utiliza para controlar la ruta que debe seguir un robot. En el siguiente video podéis ver un ejemplo del algoritmo A* en funcionamiento para controlar un vehículo autónomo.

https://www.youtube.com/watch?v=qXZt-B7iUyw

Por otro lado, un problema clásico dentro del mundo de los videojuegos es dotar a los personajes controlados por la máquina de la capacidad de moverse por el entorno de una manera cada vez más realista, bien para perseguir al jugador o para competir por un objetivo común.

El juego *Age of Empires* es un juego de estrategia en tiempo real que utiliza el algoritmo A* para mover las tropas. Se basa en un grid cuadrado de 256x256 posiciones.

Otro ejemplo del uso de este algoritmo en juegos, es el videojuego *Civilization V*. Este juego, a diferencia del anterior, se basa, como en nuestro caso, en celdas hexagonales.

Además, es un algoritmo también utilizado en juegos en primera persona, como por ejemplo *Counter-Strike* o también *War of Warcraft* para mover a los personajes controlados por el ordenador.

En nuestro caso concreto vamos a utilizar el algoritmo A* para que, en el juego descrito anteriormente, el caballero encuentre el camino de coste mínimo para llegar hasta el dragón.

A* es un algoritmo de búsqueda heurística, por lo tanto utiliza una función heurística, que nos dará un valor para cada celda. La función heurística es una estimación optimista de cómo de lejos está el objetivo. Es decir, es una estimación optimista de lo que le costará al caballero llegar hasta la celda en la que se encuentra el dragón.

```
h(x,y) ~<= distancia al objetivo
```

A continuación, se muestra el pseudocódigo del algoritmo A*.

Pseudocódigo

```
Alg A*
      listaInterior = vacio
      listaFrontera = inicio

      mientras listaFrontera no esté vacía

            n = obtener nodo de listaFrontera con menor f(n) = g(n) + h(n)

            si n es meta devolver
                  reconstruir camino desde la meta al inicio siguiendo los
                  punteros
                  Salir
            sino
                  listaFrontera.del(n)
                  listaInterior.add(n)

                  para cada hijo m de n que no esté en lista interior
                        g'(m) = n·g + c(n, m) //g del nodo a explorar m

                        si m no está en listaFrontera
                              almacenar la f, g y h del nodo en (m.f, m.g, m.h)
                              m.padre = n
                              listaFrontera.add(m)
                        sino si g'(m) es mejor que m.g //Verificamos si el nuevo
                                                       camino es mejor
                              m.padre = n
                              recalcular f y g del nodo m
                        fsi
                  fpara
      fmientras
      Error, no se encuentra solución
falg
```

¡OJO! La implementación del algoritmo A* se debe realizar en la clase AEstrella.java. Es posible utilizar métodos y clases adicionales, pero siempre dentro del fichero AEstrella.java.

En esta clase disponéis de las siguientes variables:

Mundo: Este objeto contiene toda la descripción del entorno. La matriz del mundo (tablero) y las coordenadas del caballero y del dragón.

Camino: En esta matriz, que se inicializa toda a '.', debéis indicar las celdas que forman parte del camino con una 'X'.

Camino_expandido: En esta matriz, que se inicializa toda a -1, debéis indicar el orden en el que se expanden las celdas.

```
//Mundo sobre el que se debe calcular A*
Mundo mundo;

//Camino
public char camino[][];

//Casillas expandidas
int camino_expandido[][];

//Número de nodos expandidos
int expandidos;

//Coste del camino
float coste_total;
```

Expandidos: Es un entero donde debéis almacenar el número de nodos expandidos.

Coste_Total: es un número donde debéis indicar el coste total del camino calculado por el A*. **Es importante que le deis valor a esta variable porque se utiliza para calcular el final del juego.**

En esta clase tenéis ya creado un método:

```
public int CalcularAEstrella()
```

En este método es donde se debe implementar el algoritmo A*. Ya que es este método el llamado desde Juego.java. Podréis crear métodos auxiliares que necesitéis y nuevas clases, pero siempre dentro de AEstrella.java.

Es importante que sigáis estas instrucciones y al final del A* le deis a las diferentes variables los valores indicados y llaméis a los métodos mostrarCamino() y mostrarCaminoExpandido() para mostrar por pantalla.

En concreto, al final del A*, las variables deben contener:

- La variable **camino**, como se muestra en la siguiente imagen, debe contener una X en las celdas que forman parte del camino calculado por el A*.
- La variable **camino_expandido**, como se muestra en la siguiente imagen, debe contener el orden en el que se han ido expandiendo los nodos.
- La variable **expandidos**, debe contener el número total de nodos expandidos.
- La variable **coste_total**, debe contener el coste total del camino calculado por el A*.

```
Camino
 . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
 . . X . . . . . . . . . . . .
. . X . . . . . . . . . . . .
 . . X X X X X X X X . . . . .
. . . . . . . . . . X . . . .
 . . . . . . . . . . X . . . .
. . . . . . . . . . X . . .
 . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
Camino explorado
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 28  0 30 38 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 16  1 27 34 46 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 19  2 20 32 41 47 -1 -1 -1 -1 -1 -1 -1 -1
 -1 11  3 15 21 31 37 40 42 48 -1 -1 -1 -1 -1
-1 14  4 -1 26 33 -1 43 -1 -1 49 -1 -1 -1 -1
 -1  7  5  9 13 23 36 45 -1 -1 50 -1 -1 -1 -1
-1  8  6 -1 10 12 22 35 44 -1 -1 51 -1 -1 -1
 -1 18 17 25 24 29 39 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

La función heurística

Como se ha comentado anteriormente, la función heurística es una estimación optimista de cómo de lejos está el objetivo al que se desea llegar. Esta función lo que hace es asociar a cada celda un valor que evalúa lo prometedora que es esa celda para llegar al objetivo, normalmente es una estimación de lo próximo que se encuentra el objetivo.
El algoritmo A* irá explorando los nodos en función de lo prometedores que sean, es decir, irá seleccionando el nodo que menor coste ($g+h$) tenga.

Según la función heurística que implementemos el algoritmo A* puede obtener resultados diferentes. Si la función heurística **es admisible**, el algoritmo A* siempre obtendrá el camino óptimo, si este existe. Aunque dependiendo de lo 'buena' que sea esa función heurística, el algoritmo será más o menos eficiente y explorará más o menos nodos. Si la función heurística **no es admisible**, puede ser que el algoritmo A* encuentre un camino, pero que este no sea el óptimo.

Una heurística es admisible si nunca sobrestima el coste de alcanzar el objetivo. Es decir, si el valor que da para cada celda es siempre menor o igual que el coste mínimo para alcanzar el objetivo.

$$h(n) \leq h^*(n)$$

Siendo $h^*(n)$ el coste mínimo real de la celda *n* al objetivo.

Por todo esto, la función heurística es una de las piezas clave dentro del algoritmo A*. Según el tipo de problema al que nos enfrentemos, la heurística óptima para ese problema será diferente. Si no se conoce cuál es la heurística óptima para el problema al cual nos enfrentamos, probaremos diferentes heurísticas y las analizaremos viendo si el camino que obtiene es el óptimo (el de menor coste) y el número de nodos explorados.

Así se podría probar:

- Una primera heurística $\boldsymbol{h = 0}$. Es decir, que la función heurística evaluara con 0 cada celda. Con la cual obtendríamos resultados de una búsqueda con coste uniforme.

- **Distancia Manhattan**. Esta distancia es la suma de las diferencias absolutas de las coordenadas de la celda origen y de la celda destino. Es decir, la distancia entre la celda *1* y la celda *2*, sería:
$$|x_2 - x_1| + |y_2 - y_1|$$

- **Distancia Euclídea**. Define la línea recta entre dos puntos. Se deduce a partir del Teorema de Pitágoras y se calcularía la distancia entre la celda *1* y la celda *2* como:
$$\sqrt{(\boldsymbol{x_2 - x_1})^2 + (\boldsymbol{y_2 - y_1})^2}$$

- **Distancias adaptadas a mapas hexagonales**. Estas distancias están adaptadas a mapas con casillas hexagonales. Es posible, que para calcular estas distancias sea necesario un cambio de sistema de coordenadas, ya que, en muchas ocasiones, en mapas hexagonales se trabaja con coordenadas cúbicas.

Al probar estas heurísticas, veremos que algunas de ellas serán admisibles (siempre obtendrán el camino óptimo) y puede ser que otras no sean admisibles (puede ser que el camino que encuentren no sea óptimo).

Documentación de la práctica

La documentación es la parte más importante de la práctica (60%). Como mínimo debe contener:

- **Pseudocódigo del algoritmo A*** que se ha implementado. Explicando con ejemplos cada uno de los **posibles casos** que nos podemos encontrar a la hora de evaluar nuevos nodos.

- Explicación de cuál es la **mejor heurística** para este problema y **qué ocurre cuando *h* se acerca o aleja de *h**.
    - Se deben **analizar diferentes heurísticas** y ver cómo repercuten en el número de nodos explorados.

- Explicación y **traza de un problema pequeño** donde se observe el funcionamiento del algoritmo A*. Este problema debe ser similar al entorno en el que nos encontramos en la práctica, es decir con tablero hexagonal.

- Se debe **realizar un conjunto de pruebas bien diseñado** que abarque todas las casuísticas del problema.

Entrega de la práctica

La fecha límite de entrega de la práctica es el **28 de octubre de 2018** a las 23:55h. La entrega se realizará a través de Moodle. Además, la entrega constará de un fichero .zip que contendrá:

- El fichero AEstrella.java donde el A* debe llamar a la mejor heurística implementada. La práctica se corregirá con esta heurística.

- La documentación de la práctica en un fichero .pdf.

¡¡¡AVISO IMPORTANTE!!!

No cumplir cualquiera de las normas de formato/entrega anteriores puede suponer un suspenso en la práctica.

Recordad que **las prácticas son INDIVIDUALES** y NO se pueden hacer en parejas o grupos.

Cualquier código copiado supondrá un suspenso de la práctica para todas las personas implicadas en la copia y, como indica el Reglamento para la Evaluación de Aprendizajes de la Universidad de Alicante (BOUA 9/12/2015) y el documento de Actuación ante copia en pruebas de evaluación de la EPS, **se informará a la dirección de la Escuela Politécnica Superior para la toma de medidas oportunas**.