Grado en Ingeniería Informática

Inteligencia Artificial

Curso 2019/2020

Universidad de Jaén

Guión 2(b)
Búsquedas

1. MouseRun

1.1. Introducción

MouseRun es un *programming game* escrito en Java donde se ha de dotar de inteligencia a un ratón en un laberinto, que busca quesos y debe recoger antes que sus competidores.

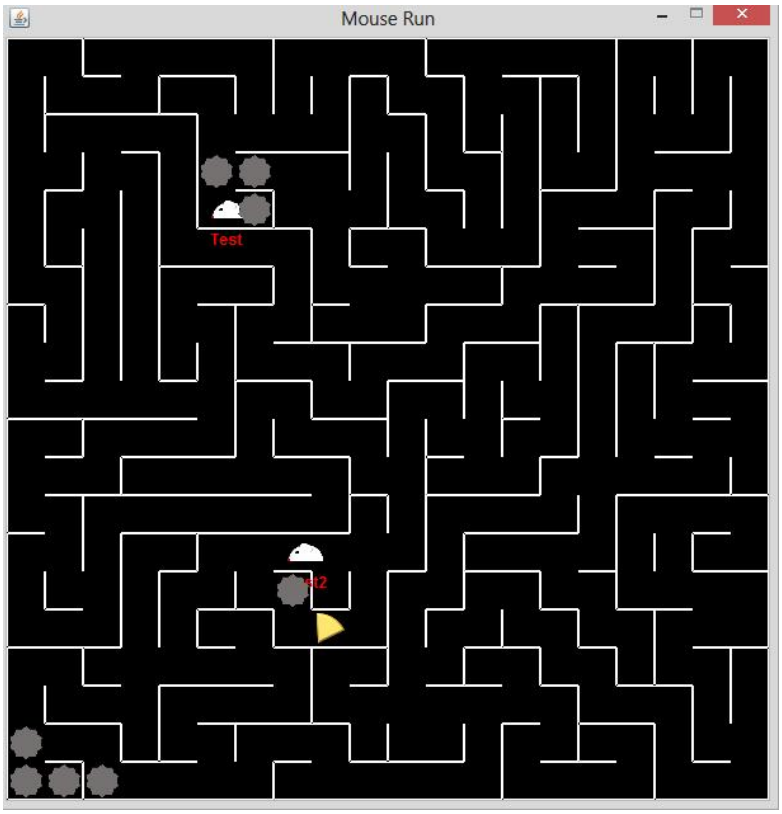

Ilustración 1. Ejemplo de ejecución de Mouse Run

En esta segunda práctica, nos vamos a familiarizar con el entorno en el que se desarrolla el juego. Para ello, vamos a construir un agente reactivo basado en modelos, bajo la forma de un ratón dentro de un laberinto.

El ratón tendrá el propósito de encontrar y coger el trozo de queso existente en el mapa a partir de los estímulos que recibe. El juego permite incorporar varios ratones en el laberinto, por lo que la agilidad, eficiencia y estrategia utilizadas serán cruciales para ser el mejor ratón de todos.

En esta práctica, se pide al alumno la programación libre de cualquier estrategia útil y con sentido que permita alcanzar el queso de la manera más rápida posible. Para ello, podrá usar las estructuras de datos que considere oportunas así como toda la información que un ratón capta en cada momento.

1.2. Características

- Requiere **Java versión 7 o superior**. Como veremos más adelante, se puede integrar en **Netbeans** (el entorno no es obligatorio, pero sí recomendable por comodidad) de forma muy sencilla, incorporando el contenido del proyecto, que se encuentra disponible en la plataforma ILIAS comprimido como un archivo .zip.
- El tamaño del laberinto es configurable. El tamaño mínimo es de 5x5, y de ahí se puede extender a cualquier tamaño. En nuestro caso, vamos a trabajar con un **tamaño estándar de 20x20**. El tiempo del juego también es configurable, nosotros trabajaremos con partidas de entre **120** y **600 segundos** de duración, según el caso.
- Los laberintos se generan automáticamente por la aplicación, son siempre aleatorios (cambian de una partida a otra), y siempre permiten alcanzar todas las celdas (*grids*). Las celdas son cuadradas y pueden tener pared en las 4 coordenadas cartesianas básicas (N, S, E, O).
- Los ratones implementan principalmente tres métodos: uno para moverse, otro que se activa cuando se recoge un queso y otro que se activa cuando pisa una bomba colocada por un ratón rival (las bombas propias no le afectan). Cada ratón es una clase en Java (por lo que puede implementar o importar cualquier estructura de datos) y siempre, en cualquier momento, tiene acceso a su posición actual (x, y) en el laberinto; a si hay pared en cualquiera de las 4 direcciones, y a la posición actual (x, y) del queso.
- El movimiento del juego se organiza en turnos, en donde se ejecuta la acción programada en cada momento. En cada turno, un ratón puede llevar a cabo solamente una de las siguientes acciones: moverse arriba, abajo, a la izquierda o derecha; o dejar una bomba en una casilla. Si un ratón pisa una bomba ajena, muere y es revivido en una posición aleatoria del tablero. Cada movimiento consume un turno, por lo que poner bombas consume tiempo, y además hay un número limitado de bombas que puede dejar cada jugador.
- El simulador incorpora una clase que se encarga de leer y organizar los ratones disponibles en el juego. Incorporar un nuevo ratón es tan simple como copiar un fichero .java al directorio de ratones y al inicio del juego éste se incluirá automáticamente, siempre y cuando no haya ningún problema de compilación.
- Durante el transcurso del juego, la aplicación podrá descalificar al ratón si éste hace alguna cosa ilegal, como escribir datos (o intentar leerlos) en disco. En ese momento, el ratón saldrá del juego automáticamente.
- Los ratones no deben leer o escribir en disco ni tampoco mostrar información por pantalla. Esto último está permitido por la aplicación, y nos podrá servir en las primeras etapas de programación para nuestras pruebas, pero **los ratones definitivos que se entregarán para la práctica no deben mostrar ningún mensaje por pantalla**.

1.3. Instalación

Mostraremos cómo crear un ratón y hacerlo moverse por el laberinto. Después, ya dependerá de nosotros que se comporte de modo inteligente. Desde **NetBeans** (suponemos que ya tenemos **Java** y **NetBeans** instalados), creamos un nuevo proyecto, bajo la opción *"Java Application"*. No hay que incluir una clase **Main** (desactivar dicha opción).

Una vez hecho esto, descargamos desde ILIAS el fichero **mouseRun.zip** con el juego, y lo descomprimimos. Buscaremos la carpeta del nuevo proyecto y allí copiamos tal cual el contenido del fichero descomprimido, dos carpetas llamadas *"assets"* y *"src"* (ésta última sobreescribirá la carpeta del mismo nombre en la carpeta de nuestro proyecto).

Ilustración 2. Contenido del archivo mouserun.zip

Si hemos hecho este paso de forma correcta, al volver a **NetBeans**, veremos que la carpeta del proyecto se ha actualizado. Si tratamos de ejecutar el proyecto, nos pedirá la clase que contiene el método **main** (debemos indicarle que use la clase **mouserun.GameStarter**), pero la ejecución devolverá un error porque faltan argumentos en el programa. Hay que modificar las propiedades de ejecución para añadirle los 4 argumentos que necesita: *anchura*, *altura*, *númeroDeQuesosPorPartida* y *tiempoMáximoDeJuego*. Los valores que proporcionaremos por defecto serán: **"20 20 50 120"**, es decir, comenzaremos con un laberinto de 20 celdas de alto por 20 de ancho, con 50 quesos (será el número máximo de quesos que aparecerán a lo largo del juego), y la duración del juego será de 120 segundos.

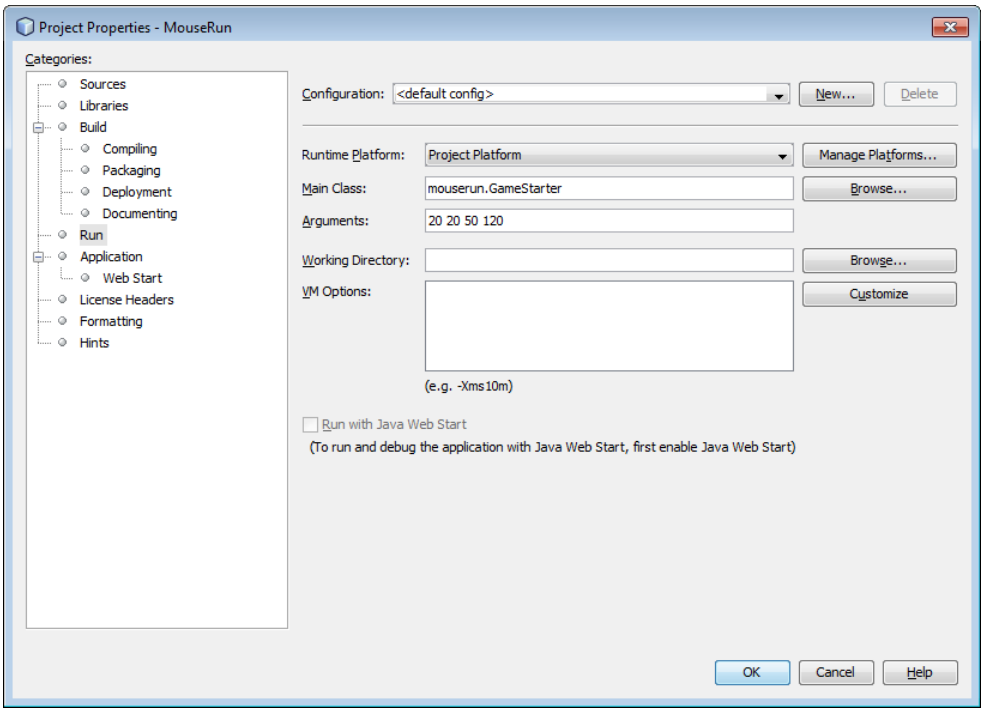

Ilustración 3. Propiedades del proyecto mouserun

1.4. Creación de un nuevo ratón

Para crear nuestro ratón, hemos de seguir los siguientes pasos:

1. Abrir un editor de texto (o bien mediante el mismo NetBeans) y crear un nuevo fichero .java dentro del paquete **mouserun.mouse.** Todos los ratones que implementemos han de colocarse en dicho paquete. Esto asegurará que el juego sabrá dónde encontrar a los ratones antes de situarlos en el laberinto, siempre y cuando hayan compilado sin errores. Añadir código hasta obtener algo así:

```
package mouserun.mouse;

import mouserun.game.Mouse;
import mouserun.game.Grid;
import mouserun.game.Cheese;

public class FooMouse extends Mouse
{

}
```

Ilustración 4. Plantilla inicial para la clase FooMouse

2. Como se puede observar, nuestro nuevo ratón hereda de la clase **Mouse**, una clase *abstracta* que obliga a implementar sus métodos abstractos. Los métodos son los siguientes:

```
package mouserun.mouse;

import mouserun.game.Mouse;
import mouserun.game.Grid;
import mouserun.game.Cheese;

public class FooMouse extends Mouse
{
     public FooMouse()
     {
        super("FooMouse");
     }

     public int move(Grid currentGrid, Cheese cheese)
     {

     }

     public void newCheese()
     {

     }

     public void respawned()
     {

     }
}
```

Ilustración 5. Métodos principales de la clase FooMouse

- **Constructor**. Más adelante, podremos incluir código adicional para inicializar el ratón, pero como mínimo debemos llamar al constructor de la superclase, ***super()***, pasando como argumento el nombre de nuestro ratón (***FooMouse*** en el ejemplo), el cual nos servirá para reconocerlo en el transcurso del juego y poder distinguirlo del resto de ratones presentes en el laberinto.
- ***public int move(Grid currentGrid, Cheese cheese)*** es el método más importante en la clase **Mouse**. Cuando el juego detecta que el ratón está en un **Grid**, éste deberá indicar una decisión sobre qué es lo próximo que va a hacer. **currentGrid** es la celda actual dónde está el ratón y **cheese** es el objeto que representa al queso que tanto anhelamos. Ésta es la única información a la que nos da acceso la aplicación, el resto corre por nuestra cuenta. Con dicha información, más toda la que seamos capaces de obtener, hay que decidir entre una de las cinco acciones distintas que puede devolver el método ***move()***:
  - *return Mouse.UP;*
  - *return Mouse.DOWN;*
  - *return Mouse.LEFT;*
  - *return Mouse.RIGHT;*
  - *return Mouse.BOMB;*
- ***public void newCheese()*** se invoca cuando algún queso (nosotros o un rival) alcanza un queso (y se lo queda), momento en el cual el juego dispone un nuevo queso en una nueva posición al azar en el laberinto. Si no se quiere hacer nada, se puede dejar tal cual está. Este método se utiliza normalmente para reiniciar y/o limpiar información almacenada sobre la partida actual.
- ***public void respawned()*** se llama cuando nuestro ratón pisa una bomba ajena, reapareciendo éste instantes después en otro sitio del laberinto, elegido al azar por el programa. Como el método anterior, se

puede dejar en blanco al principio. Más adelante los reutilizaremos para recalibrar o reiniciar la estrategia de búsqueda.

Cada ratón, contiene además un par de atributos especiales:

- ***private long steps*** indica el número de pasos que da el ratón a través del laberinto. No debemos modificar su valor, ya que es la aplicación la que se encarga de gestionarlo de forma automática. Nos servirá, al término de cada partida, para evaluar el nivel exploratorio de nuestro ratón, siempre y cuando éste haya sobrevivido todo ese tiempo y no haya sido descalificado.
- ***private long exploredGrids*** almacenará el número de celdas únicas visitadas por nuestro ratón en una partida. Al contrario que el atributo anterior, **éste sí que debemos manejarlo nosotros**, e incrementarlo correctamente cada vez que el ratón visite una nueva celda. Podemos acceder a dicho atributo mediante los métodos ***getExploredGrids()*** (nos devuelve el valor actual) e ***incExploredGrids()*** (incrementa en 1 el número de celdas visitadas). Al término de la partida, y si el ratón sigue vivo para entonces, el juego nos mostrará una serie de estadísticas, como el número de quesos recogidos, el número de pasos dados y, si nos hemos encargado de actualizar convenientemente el atributo, el número de celdas únicas visitadas. Conociendo el tamaño del laberinto, nos servirá para estimar la bondad de nuestra estrategia exploratoria.

2. Objetivos

2.1. Ejercicio 1. Implementación de un ratón explorador

Los alumnos deberán implementar un agente inteligente, representado como un ratón, e incluirlo dentro de la aplicación **MouseRun**, para evaluar su comportamiento dentro del laberinto. En este primer ejercicio, no es tan importante el factor competitivo, como es el hecho de que el ratón deba alcanzar los quesos antes que sus competidores, al tiempo que les impida avanzar por medio de obstáculos (bombas), como el factor exploratorio, mediante el cual **se debe procurar que el ratón cubra el mayor espacio posible del laberinto mientras se acerca a su objetivo**, que es el queso.

Para ello, tal y como se ha explicado en el paso anterior, los alumnos implementarán un ratón como una subclase de la clase **Mouse**. Dicha subclase incluirá un constructor (mediante el cual dar un nombre a nuestro ratón, entre otras cosas) junto con los métodos (abstractos en la clase **Mouse**) ***move()***, ***newCheese()*** y ***respawned()***. Estos métodos van a definir el comportamiento de nuestro ratón en el laberinto.

Se proporciona el código de **TRES** ratones, **TestMouse.java, ProblematicMouse.java**, y **MXXA04.java** como ejemplo y punto de partida para definir qué puede (o no debe) hacer nuestro ratón.

Haciendo uso de las librerías que Java nos proporciona, podemos definir e implementar tantos atributos y métodos auxiliares como consideremos necesarios, pero **siempre dentro de nuestra clase**. Es decir, podemos revisar el código del resto de la aplicación para aprender cómo funciona ésta, pero bajo ningún concepto debemos modificarlo, ya que éste ha de ser siempre el mismo para todos los ratones que compitan en el juego.

Por este motivo, el único código que los alumnos deben entregar, y el único que se tendrá en cuenta en la corrección de la práctica, será el relativo a nuestro ratón explorador, contenido en un único archivo .java.

El tercer ratón MXXA04 os presenta las estructuras de datos más adecuadas para el manejo de exploración del laberinto y algunos métodos útiles:

- ***private Grid lastGrid:*** *Permite almacenar la última casilla visitada.*
- ***private HashMap<Pair<Integer, Integer>, Grid> celdasVisitadas:*** Esta primera estructura es una tabla hash, utilizando como clave las coordenadas y un Grid como valor. Aquí se almacena toda la información de la exploración del ratón.
- **private Stack<Grid> pilaMovimientos:** Una pila que almacena los movimientos realizados. Se almacena información del camino recorrido por el ratón.
- **public boolean testGrid(int direction, Grid currentGrid):** Método para verificar que no volvemos a la casilla justamente anterior. Sirve para evitar ciclos.
- **public boolean visitada(Grid casilla):** Nos devuelve si una casilla ha sido visitada. Hace uso de la estructura *celdasVisitadas*, para la comprobación.
- **public boolean actualArriba(Grid actual, Grid anterior)**
- **public boolean actualAbajo(Grid actual, Grid anterior)**
- **public boolean actualDerecha(Grid actual, Grid anterior)**
- **public boolean actuallzquierda(Grid actual, Grid anterior).** Estos cuatro métodos permiten identificar la posición relativa de una celda respecto a otra.

2.2. Análisis del comportamiento del agente

Una vez que hayamos puesto en marcha nuestro primer ratón, llevaremos a cabo varias ejecuciones (partidas) sobre diferentes laberintos. Se propone un mínimo de **5 partidas**, con los siguientes parámetros:

- **Ancho = Alto = 20 celdas, Tiempo = 60 segundos**.
- **Ancho = Alto = 20 celdas, Tiempo = 120 segundos**.
- **Ancho = 40 celdas, Alto = 20 celdas, Tiempo = 120 segundos**.
- **Ancho = 40 celdas, Alto = 20 celdas, Tiempo = 240 segundos**.

De esta forma, podremos comparar y analizar el comportamiento de nuestro ratón sobre el tablero de juego. Estudiaremos si es capaz de orientarse

adecuadamente por el laberinto, sin dar vueltas en círculos, aproximarse lo más posible al queso, etc., todo ello sin ser descalificado.

Anotando los valores que nos muestra el juego al término de la simulación (entre ellos, el número de quesos recogidos, número de pasos y número de celdas visitadas), construiremos una tabla como la que se muestra a continuación:

Ejecución	Quesos	Pasos	Casillas exploradas	Ratio de exploración
1				
2				
3				
4				
5				
…				

Tabla 1. Análisis de la exploración

- En la columna **Quesos** apuntaremos los quesos que nuestro ratón ha logrado recoger.
- Los valores de las columnas **Pasos** y **Casillas exploradas** (este último, si las controlamos correctamente) nos los facilita el juego al término de una partida.
- El valor de la columna **Ratio de exploración** se obtiene como el cociente entre **Casillas exploradas** y las dimensiones (alto x ancho) del laberinto.

3. Búsquedas

Partiendo del ratón implementado para la primera parte de la práctica, se incorporará un mecanismo de búsqueda que permita tomar decisiones precisas a la hora de determinar el mejor camino posible hacia el ratón. Obviamente, los algoritmos de búsqueda requerirán información acerca del laberinto explorado, por lo que el éxito de las búsquedas vendrá principalmente determinado por la capacidad exploradora del primer ratón.

Ejercicio 3.1. Búsqueda no informada

Los alumnos incorporarán el algoritmo de **Búsqueda Primero en Profundidad**, y el ratón se llamará igual que el explorador, pero añadiendo –DFS antes del .java. Como sabemos, el algoritmo consiste en ir expandiendo todos los nodos

por los que va pasando, de manera recurrente, explorando un camino en particular. Cuando no quedan más nodos por visitar, nos permite volver atrás (*backtracking*) y continuar la búsqueda por cada uno de los hermanos del último nodo procesado. Para ello, podrá usar las estructuras de datos que considere oportunas así como toda la información que el juego facilita al ratón en cada momento.

El tamaño del laberinto no variará con respecto al ejercicio 1 (20x20), pero en este caso **se ampliará la duración de todas las partidas hasta el doble de tiempo:**

1) Partida 1: **20 x 20 300 segundos**.
1) Partida 2: **20 x 40 600 segundos**.

Ejercicio 3.2. *Greedy* del primer mejor

Usando de nuevo como referencia el explorador del primer ejercicio, desarrollaremos un tercer ratón (igual nombre que el anterior, pero poner GRE, en vez de DFS) que implemente el **algoritmo Greedy del primer mejor**. Para la función de evaluación usaremos como medida heurística la **distancia de Manhattan** entre la posición que representa la casilla en cada nodo y la posición absoluta del queso. La distancia de Manhattan se define como:

$$|x_i - x_g| + |y_i - y_g|$$

donde (x_i, y_i) es la posición de la casilla en la que se encuentra el enano en ese momento, y (x_g, y_g) corresponde a la posición del cofre del tesoro.

Se repetirán las mismas condiciones de juego que en el apartado 3.1, dimensiones del laberinto, **20 filas y 20 columnas**, y dos versiones:

1) **300 segundos**, y
2) **600 segundos** de tiempo de juego.

NOTA: Estos algoritmos tienen distintas puntuaciones. Revisar en la sección correspondiente.

Análisis del comportamiento de los agentes

Análogamente, cuando hallamos implementado los ratones buscadores, generaremos una tabla como la siguiente, para evaluar el comportamiento de los ratones ante diferentes escenarios (distintos laberintos, tipo de búsqueda implementada, comportamiento frente a los posibles rivales, etc.):

Laberinto	Ejecución	Tipo	Quesos	Pasos	Casillas exploradas	Ratio de exploración	Ratio de repetición
20x20	1	DFS					
		GRE					
	2	DFS					
		GRE					
	3	DFS					
		GRE					
20x40	…	…					

Tabla 2. Análisis de las búsquedas

- Las condiciones iniciales de Tiempo, Ancho y Alto ya han sido indicadas previamente.
- Además, para cada laberinto y cada algoritmo de búsqueda **se deben de realizar tres ejecuciones**.
- En la columna **Quesos** apuntaremos los quesos que cada ratón ha logrado recoger.
- Los valores de las columnas **Pasos** y **Casillas exploradas** (recordad que este último hemos de calcularlo manualmente) nos los facilita el juego.
- El valor de la columna **Ratio de exploración** se obtiene como el cociente entre **Casillas exploradas** y las dimensiones de nuestro laberinto. Mediante este parámetro estimaremos el grado en el que cada ratón es capaz de explorar el laberinto en su mayor parte durante el tiempo que dure la partida.
- El valor de la columna **Ratio de repetición** lo podemos obtener como el cociente entre **Casillas exploradas** y **Pasos**, y nos permitirá estimar el número medio de veces que un ratón pasa por una misma celda.

Ojo, el juego nos proporcionará estas estadísticas siempre y cuando nuestros ratones continúen en la partida cuando ésta finalice (es decir, que no hayan sido descalificados previamente), por lo que, a la hora de rellenar la tabla, debemos asegurarnos de que nuestros ratones funcionan adecuadamente.

4. Consideraciones importantes

- Asegurad de generalizar el ratón para todo tipo de escenarios. Por ejemplo, se podría optar por aprender el mapa conforme se explora, pero podemos cambiar drásticamente de posición si pisamos una bomba ajena. Esto puede desencadenar en bucles infinitos, o que el ratón se salga de los límites del laberinto, y es lo peor que nos puede pasar.
- También hay que tener en cuenta que el tamaño y estructura del laberinto, así como la duración del juego, pueden variar de una partida a otra y que esta información no es conocida en ningún momento por los ratones. La configuración y distribución de las casillas se genera aleatoriamente en cada nueva partida, y el tamaño (ancho y alto) se define a partir de los parámetros de entrada. **Nuestro ratón ha de ser capaz de moverse en laberintos de cualquier tipo y tamaño**.
- Puedes ser descalificado. El juego no perdona y se asegura una competición justa con respecto al resto de ratones. Cada ratón debe tomar una decisión de acción bajo un límite de tiempo (alrededor de 100 milisegundos por defecto). Durante este tiempo, el ratón no debe producir ninguna excepción de ejecución. El juego descalifica y elimina a aquellos ratones que causen problemas de forma continuada durante la ejecución.
- Los ratones **no podrán escribir datos en ningún fichero ni mostrar información por pantalla**. Podemos usar estos canales para la depuración, durante la realización de la práctica, pero debemos asegurarnos de que las versiones definitivas de nuestros ratones no incumplan esta norma.

5. Aspectos que se valorarán en la nota final

- La explicación o descripción de las estrategias seguidas por el ratón en su exploración.
- La idea o lógica de las estrategias utilizadas.
- La efectividad de las estrategias aplicadas.
- La limpieza del código entregado, la documentación interna del código, la documentación externa de cada ratón en el informe y el uso correcto de convenciones de Java.
- La inexistencia de bucles infinitos, movimientos repetitivos o descalificaciones por el juego.
- Evitar copiar y usar códigos de otros compañeros porque se pasará un test de “plagio”.

6. Entrega y evaluación

- La práctica se realizará por parejas (recomendado). La puntuación máxima de la práctica será de **4 puntos**, de los cuales esta primera etapa de exploración sumará hasta **1.5 puntos**. En la parte de búsqueda se puntuará como **DFS (1.5 puntos) y GREEDY (1 punto).** La entrega **DE LA PRÁCTICA COMPLETA** se llevará a cabo a través de la actividad correspondiente en ILIAS. La defensa de la práctica se llevará a cabo el día siguiente de la entrega durante la sesión de prácticas. En ella, los alumnos deberán demostrar su conocimiento sobre el trabajo realizado.
- Cada trabajo práctico debe ir comprimido en un archivo .zip, incluyendo documentación y código fuente (un único archivo .java por ratón), según lo especificado a continuación:
    - El nombre de un ratón es libre y se define mediante el método ***super()*** del constructor (como hemos visto). Es el texto que aparecerá escrito bajo nuestro ratón una vez dé comienzo el juego, y que nos ayudará a distinguirlo de los otros.
    - Sin embargo, el identificador del ratón no es libre para evitar conflictos y solapamientos en las competiciones y pruebas de efectividad. Éste condiciona el nombre de la clase que implementaremos y, por ende, del fichero .java que hay que entregar: ha de ser único por pareja de prácticas, y no se puede repetir.
    - **No confundir el identificador del ratón con el nombre del ratón.**
    - El identificador del ratón tendrán el siguiente formato: **M20NXXa.[java|zip|rar], o M20NXX-DFS.[java|zip|rar]**, donde:
        - **N** será igual al nombre del grupo:
            - A (grupo 1, 8:30)
            - B (grupo 2, 10:30)
            - C (grupo 3, 15:30)
            - D (grupo 4, 17:30)
        - **XX** será el identificador numérico de cada pareja (01, 02, 03...). **Se asignará en la sesión de explicación de la práctica**.
        - Por último, añadiremos el sufijo **"a"** para aclarar que se trata del ratón explorador.
    - **IMPORTANTE: Cada pareja es responsable del identificador numérico asignado.** El profesor no va a llevar la cuenta del identificador de los alumnos. Esto quiere decir que:
        - El número asignado se debe conservar, memorizar, apuntar, etc., para poder entregar la práctica en perfectas condiciones. No se admiten olvidos.
        - Si se entregan dos prácticas con el mismo identificador, quedan anuladas para su corrección posterior.
    - **CONSEJO:** Para evitar olvidos, confusiones, errores posteriores en la compilación y ejecución del código, etc. os recomendamos que, desde el primer momento, llaméis a vuestro archivo .java (y a la clase que contiene) con el identificador asignado.

- Además del código, se entregará un documento en formato PDF describiendo el desarrollo de la práctica. Se comentará, en lenguaje natural, la estrategia seguida en cada uno de los métodos implementados. No debe incluir código fuente. Rellenar las tablas y razonar brevemente los resultados. Extensión recomendada: 4-6 páginas.
- El documento comenzará, además, con una portada con, al menos, la siguiente información: nombre de los autores, curso académico, grupo de prácticas, nombre del profesor de prácticas, identificadores y nombres de los ratones.
- **El no cumplimiento de las anteriores normas repercutirá negativamente en la calificación de la práctica.**