Grado en Ingeniería Informática

Inteligencia Artificial

Curso 2022/2023

Universidad de Jaén

Práctica 2:
Exploración y búsqueda

1. Introducción

El objetivo de esta práctica es que el estudiante se familiarice con los agentes que implementan exploración y estrategias de búsqueda no informadas.

Para ello se utilizará el entorno **MouseRun** que implementa un laberinto en el que diferentes agentes se mueven para buscar quesos. En el entorno pueden encontrar dificultades como bombas que provocan que el agente se posicione en otra posición cualquiera del laberinto, alterando el comportamiento del mismo.

La práctica plantea 5 actividades:

- Actividad 1: implementación de un agente explorador (su principal objetivo es explorar el número máximo de casillas del laberinto).
- Actividad 2: análisis del comportamiento del ratón explorador.
- Actividad 3: implementación de un agente con la estrategia de búsqueda no informada con profundidad limitada (BPL).
- Actividad 4. Análisis del comportamiento del agente buscador BPL en un entorno mono-agente y en un entorno multi-agente.
- Actividad 5: Implementación de mejoras del agente BPL en entorno multiobjetivo.

Para la práctica existen dos fechas de entrega:

- 9/3/23: actividades 1 y 2.
- 23/3/23: actividades 3, 4 y 5. También se pueden entregar las actividades 1 y 2 en el caso de que no se hayan podido completar en la fecha anterior.

2. MouseRun

2.1. Introducción

MouseRun es un *programming game* escrito en Java en el que se debe dotar de inteligencia a un ratón (agente) que se mueve en un laberinto (Ilustración 1) buscando quesos con el objetivo de recogerlos antes que sus competidores.

Ilustración 1. Ejemplo de ejecución de Mouse Run

En esta práctica, te vas a familiarizar con el entorno en el que se desarrolla el juego y construirás agentes de resolución de problemas mediante búsqueda (agente basado en objetivos), bajo la forma de un ratón dentro de un laberinto.

El ratón tiene como propósito encontrar (y comerse) el trozo de queso existente en el mapa a partir de la información que percibe. 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, debes plantear y programar diferentes estrategias que permitan explorar el laberinto y buscar para alcanzar el queso de la manera más rápida posible. Para ello, puedes usar las estructuras de datos que consideres oportunas, así como toda la información a la que el ratón tiene acceso en cada momento

2.2. Características del entorno

- Requiere **Java versión 7 o superior**. Como verás 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, disponible en la plataforma PLATEA 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 tu caso, vas a trabajar con diferentes tamaños de tablero, desde **10x10, 20x20 hasta 40x20**. El tiempo del juego también es configurable, vas a experimentar con partidas de entre **150** y **300 segundos** de duración, según el caso.
- Los laberintos se generan automáticamente, 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 cualquier de las 4 coordenadas cartesianas básicas (N, S, E, O).
- Los ratones implementan tres métodos: uno para moverse, otro que se activa cuando se recoge un queso (no tiene por qué haber sido tu ratón) 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 los que se ejecuta la acción programada en cada momento. En cada turno, un ratón puede llevar a cabo solo 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 revive 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. **Intentar moverse a una posición prohibida, porque un muro le impida paso, también consume turnos.**
- 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 te 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.**

2.3. Instalación

A continuación, se muestra el proceso acerca de cómo crear un ratón y hacer que se mueva por el laberinto. Después, ya dependerá de ti que se comporte de modo inteligente.

Desde **NetBeans** (suponiendo que ya tienes **Java** y **NetBeans** instalados), crea un nuevo proyecto, bajo la opción *"Java Application"*. No hay que incluir una clase **Main** (desactivar dicha opción).

Una vez hecho esto, descarga desde PLATEA el fichero **mouseRun.zip** con el juego, y descomprímelo. Busca la carpeta del nuevo proyecto y copia allí tal cual el contenido del fichero descomprimido (Ilustración 2), dos carpetas llamadas *"assets"* y *"src"* (ésta última sobreescribirá la carpeta del mismo nombre en la carpeta de tu proyecto).

Ilustración 2. Contenido del archivo mouserun.zip

Si has hecho este paso de forma correcta, al volver a **NetBeans**, verás que la carpeta del proyecto se ha actualizado.

Si tratas de ejecutar el proyecto, el entorno te pide la clase que contiene el método **main** (debes 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 (Ilustración 3):

- *anchura*,
- *altura*,
- *númeroDeQuesosPorPartida* y
- *tiempoMáximoDeJuego*.

Los valores que se proporcionarán inicialmente serán: **“10 10 20 30”**, es decir, comenzarás con un laberinto de 10 celdas de alto por 10 de ancho, con 10 quesos (es el número máximo de quesos que aparecerán a lo largo del juego) y la duración del juego será de 30 segundos.

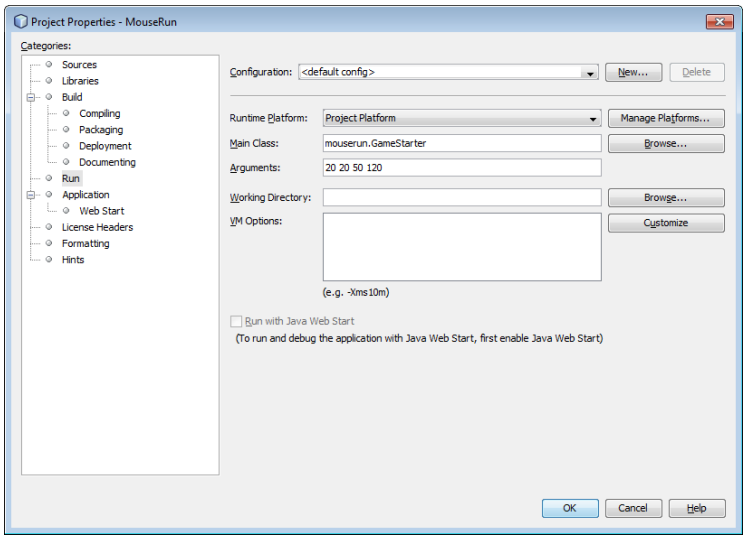

Ilustración 3. Propiedades del proyecto mouserun

2.4. Creación de un nuevo ratón

Para crear tu ratón, debes 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 implementes deben ubicarse 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 (Ilustración 4):

```
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 puedes observar, tu nuevo ratón hereda de la clase **Mouse**, una clase *abstracta* que obliga a implementar sus métodos abstractos. Los métodos son los siguientes (Ilustración 5):

```
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, puedes incluir código adicional para inicializar el ratón, pero como mínimo debes llamar al constructor de la superclase, ***super()***, pasando como argumento el nombre de tu ratón (***FooMouse*** en el ejemplo), el cual te 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. Ésta es la única información a la que te da acceso la aplicación, el resto corre por tu cuenta. Con esta información, más toda la que seas capaz 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 cada vez que algún ratón (tu agente o un rival, si existen distintos ratones en el laberinto porque se está ejecutando en modo multiagente) alcanza un queso (y se lo come), 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 invoca automáticamente cada vez que tu 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 lo puedes utilizar para recalibrar o reiniciar la estrategia de búsqueda.

Cada ratón contiene además dos propiedades especiales:

- ***private long steps*** indica el número de pasos que da el ratón a través del laberinto. No debes modificar su valor, ya que es la aplicación la que se encarga de gestionarlo de forma automática. Te servirá, al final de cada partida, para evaluar el nivel exploratorio del ratón, siempre y cuando éste haya sobrevivido todo ese tiempo y no haya sido descalificado.
- ***private long exploredGrids*** almacena el número de celdas únicas visitadas por tu ratón en una partida. Al contrario que el atributo anterior, **éste debes manejarlo tú, incrementándolo correctamente cada vez que el ratón visite una nueva celda**. Puedes acceder a dicho atributo mediante los métodos ***getExploredGrids()*** (devuelve el valor actual) e ***incExploredGrids()*** (incrementa en 1 el número de celdas visitadas).

Al final de la partida, y si el ratón sigue vivo para entonces, el juego muestra una serie de estadísticas, como el número de quesos recogidos, el número de pasos dados y, si te has encargado de actualizar convenientemente el atributo ***exploredGrids***, el número de celdas únicas visitadas. Conociendo el tamaño del laberinto, te servirá para estimar la bondad de la estrategia exploratoria.

3. Actividades relativas al entorno y la exploración

3.1. Actividad 1. Implementación de un ratón explorador

Los estudiantes deben 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 lo más importante es el factor exploratorio: **se debe intentar que el ratón recorra el mayor espacio posible del laberinto mientras se acerca a su objetivo**, que es el queso. Nota: el objetivo en esta actividad no es alcanzar los quesos antes que el resto de ratones, sino explorar el mayor espacio posible del laberinto.

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

En el diseño del ratón, hay que tener en cuenta que **el ancho y alto del laberinto puede variar en cada nueva ejecución** y que el ratón no puede tener ninguna información sobre los parámetros del entorno.

Se proporciona el código de tres ratones, **TestMouse.java, ProblematicMouse.java**, y **MXXA04A.java** como ejemplos y punto de partida para definir qué puede (o no debe) hacer tu ratón.

Haciendo uso de las librerías que Java proporciona, puedes definir e implementar tantos atributos y métodos auxiliares como consideres necesarios, pero **siempre dentro de tu clase**. Es decir, puedes revisar el código del resto de la aplicación para aprender cómo funciona ésta, pero bajo ningún concepto debes 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 estudiantes deben entregar (y el único que se tendrá en cuenta en la corrección de la práctica) será el relativo a tu ratón explorador, contenido en un **único archivo .java**.

El ratón **MXXA04A** 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 estructura es una tabla hash, que utiliza como clave las coordenadas y un Grid como valor. Aquí se puede almacenar toda la información de la exploración del ratón.
- **private Stack<Grid> pilaMovimientos.** Pila que almacena los movimientos realizados. Guarda información del camino recorrido por el ratón.
- **public boolean testGrid(int direction, Grid currentGrid).** Método para verificar que no vuelves a la casilla justamente anterior. Sirve para evitar ciclos.
- **public boolean visitada(Grid casilla).** 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 actualIzquierda(Grid actual, Grid anterior).**
  Estos cuatro métodos permiten identificar la posición relativa de una celda respecto a otra.

3.2. Actividad 2. Análisis del comportamiento del agente explorador

Cuando se haya puesto en marcha tu primer ratón, realiza varias ejecuciones (partidas) sobre diferentes laberintos. Se propone un mínimo de **3 partidas con cada uno de los siguientes parámetros**:

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

De esta forma, puedes comparar y analizar el comportamiento de tu ratón sobre el tablero de juego. Estudia 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 muestra el juego al término de la simulación (entre ellos, el número de quesos recogidos, número de pasos y número de celdas visitadas), construye una tabla como la que se muestra a continuación (Tabla 1):

Laberinto	Ejecución	Quesos	Pasos	Casillas exploradas	Ratio de exploración
20x20 (60 seg.)	**1**				
	2				
20x20 (120 seg.)	**3**				
…					

Tabla 1. Análisis de la exploración

- En la columna **Quesos** indica los quesos que tu ratón ha logrado recoger.
- Los valores de las columnas **Pasos** y **Casillas exploradas** (este último, debes programarlo tú) 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. Mediante este parámetro se estima el grado en el que cada ratón es capaz de explorar el laberinto durante el tiempo que dure la partida.
- El valor de la columna **Ratio de repetición** se obtiene como el cociente entre **Casillas exploradas** y **Pasos** y permite estimar el número medio de veces que un ratón pasa por una misma celda.

Nota: el juego proporciona estas estadísticas siempre y cuando los ratones correspondientes 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, debes asegurarte de que tus ratones funcionan adecuadamente.

4. Estrategias de búsqueda

Dispones de un agente con una capacidad de exploración alta, desarrollado en los apartados anteriores, y en esta sección avanzarás con un agente cuyo objetivo fundamental no sea explorar sino buscar quesos. En particular, en esta sección se va a diseñar un algoritmo de búsqueda *no informado*. Por tanto, no debe utilizar ninguna información para estimar la distancia al objetivo.

4.1. Actividad 3. Implementación de un agente que represente la estrategia de búsqueda no informada con profundidad limitada (BPL)

Los estudiantes incorporarán el algoritmo de **Búsqueda en Profundidad Limitada (BPL**), estudiado en la parte teórica de la asignatura, y la clase con el ratón se llamará igual que la del explorador, pero añadiendo el sufijo “bpl” antes de la extensión .java. En la implementación se podrán usar las estructuras de datos necesarias así como toda la información que el juego facilita al ratón en cada momento.

Nota: No se puede utilizar ninguna función heurística puesto que es una búsqueda no informada. Es decir, no se debe utilizar información sobre la posición del queso para calcular una estimación de la distancia de una posición cualquiera a la posición del objetivo. Sí se puede utilizar la posición del queso como test objetivo.

4.2. Actividad 4. Análisis del comportamiento del agente buscador en un entorno mono-agente y en un entorno multiagente.

El análisis del agente buscador (con búsqueda no informada con profundidad limitada) se realizará en dos pasos:

1. Se ejecutará el agente en un entorno mono-agente, es decir, controlando que en la carpeta de agentes solo esté el código del agente bpl, y considerando diferentes límites de profundidad. En estas condiciones, se ejecutarán 3 partidas, con dos posibles límites, **x1** y **x2**, con cada una de estas dos configuraciones:

    a. Configuración 1: **20 x 20 y 150 segundos.**

    b. Configuración 2: **40 x 20 y 300 segundos**.

    Los resultados se reflejarán en una tabla de este tipo:

Laberinto	Ejecución	Tipo	Quesos	Pasos	Casillas exploradas	Ratio de exploración	Ratio de repetición
20x20	1	BPL = x1					
	2	BPL = x1					

	3	BPL = x1					
	4	BPL = x2					
	5	BPL = x2					
	6	BPL = x2					
20x40	…	…					

Tabla 2. Análisis de la exploración

2. Se ejecutará el agente en un entorno multi-agente, es decir, poniendo en la carpeta de agentes los agentes de prueba y una copia del agente de búsqueda no informada que se ha implementado, con otro nombre. Se volverán a ejecutar el mismo número de partidas de cada una de las dos configuraciones anteriores. Los resultados se mostrarán en una tabla como la anterior (Tabla 2), indicando que se trata del entorno multiagente.

Se debe analizar, al menos:

- El comportamiento del agente de búsqueda bpl en entorno mono-agente.
- El comportamiento del agente de búsqueda bpl en entorno multi-agente.
- Las dificultades -si existen- a las que se enfrenta el agente bpl en entorno multiagente y propuestas de solución en su caso.

4.3. Actividad 5: Implementación de mejoras del agente bpl en entorno multiobjetivo.

A la vista de los resultados recogidos en las tablas anteriores, proponer y realizar los cambios necesarios en la implementación del agente de búsqueda no informada bpl para evitar posibles problemas en entornos multiobjetivo.

Reflejar los resultados de la ejecución de este nuevo agente en una tabla como las anteriores (ver Tabla 2). Discutir los resultados obtenidos.

5. Consideraciones importantes

- Es necesario generalizar el comportamiento del agente para que el funcionamiento sea adecuado en todo tipo de escenarios. Por ejemplo, se podría optar por aprender el mapa conforme se explora, pero puedes cambiar drásticamente de posición si pisas una bomba ajena. Esto puede desembocar en bucles infinitos, o que el ratón trate de salir de los límites del laberinto, y es lo peor que te puede pasar.
- 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. **Tu ratón debe ser capaz de moverse en laberintos de cualquier tipo y tamaño**.

- El ratón puede 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**. Puedes usar estos canales para la depuración, durante la realización de la práctica, pero debes asegurartes de que las versiones definitivas de tus ratones no incumplan esta norma.

En la práctica se valorará:
- 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.

6. Entrega y evaluación

- La puntuación máxima de la práctica 2 será de **4 puntos**, distribuidos de la siguiente forma:
    - Entrega 1:
        - Actividad 1: 0,75 puntos
        - Actividad 2: 0,25 puntos
    - Entrega 2:
        - Actividad 3: 1,25 puntos
        - Actividad 4: 1 punto
        - Actividad 5: 0,75 puntos
- La entrega de las actividades 1 y 2 se llevará a cabo a través de la actividad correspondiente en PLATEA con fecha límite el 9/3/23.
- La entrega de las actividades 3, 4 y 5 se realizará a través de la tarea asociada en PLATEA con fecha límite el 23/3/23.
- La defensa de ambas partes de la práctica se realizará en la sesión de prácticas siguiente a la entrega. En ella, los dos miembros de cada pareja de prácticas 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 se ha visto). Es el texto que aparecerá escrito bajo tu ratón una vez dé comienzo el juego, y que te 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

implementarás y, por ende, del fichero .java que hay que entregar: ha de ser único por pareja de prácticas, y no se puede repetir.

- **No se debe confundir el identificador del ratón con el nombre del ratón.**
- El identificador del ratón tendrá el siguiente formato: **M23NXX.[java|zip|rar], M23NXXbpl.[java|zip|rar], y M23NXXextra.[java|zip|rar]**, donde:
  - **N** será igual al nombre del grupo:
    - A (grupo 1, 8:30)
    - B (grupo 2, 10:30)
    - C (grupo 3, 12:30)
    - D (grupo 4, 15:30)
    - E (grupo 5, 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, añade el sufijo
    - **"bpl"** para el ratón que implementa la estrategia de búsqueda no informada con profundidad limitada.
    - **"extra"** para el ratón que implementa la estrategia de búsqueda no informada modificado para adaptarse a un entorno multiagente.
    - el ratón explorador no necesita sufijo.
- **IMPORTANTE: Cada grupo es responsable del identificador numérico asignado.** El profesor no va a llevar la cuenta del identificador de los estudiantes. 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., te recomendamos que, desde el primer momento, llames a tus archivos .java (y a la clase que contienen) 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, pero sí puedes usar pseudocódigo o esquemas. Rellenar las tablas y razonar brevemente los resultados. Extensión recomendada: 4-6 páginas.
- Cada 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. Se incluirá también un índice.

El incumplimiento de las normas anteriores repercutirá negativamente en la calificación de la práctica.