Grado en Ingeniería Informática

Inteligencia Artificial

Curso 2017/2018

Universidad de Jaén

Guión 1
MouseRun
Búsquedas

1. MouseRun

1.1. Introducción

MouseRun es un *programming game* escrito en Java donde se ha de dotar de inteligencia a un ratón, con el objetivo de alcanzar un trozo de queso en un laberinto antes que sus competidores.

Figura 1. Ejemplo de ejecución de MouseRun

En esta primera 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. 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 trozo de queso y otro que se activa cuando pisa una bomba colocada por un 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 tiene acceso a la posición actual (x, y) donde se encuentra el ratón; 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.
- 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 incluye automáticamente. Si hay algún problema de compilación o el ratón hace cosas ilegales, la aplicación lo descalifica y directamente no entra o bien sale a mitad del juego.
- El ratón no podrá escribir datos en ningún fichero ni debe mostrar información 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 instalado), creamos un nuevo proyecto, bajo la opción *"Java Application"*. No hay que incluir un método *Main* (desactivar dicha opción).

Una vez hecho esto, descargamos desde ILIAS el fichero *MouseRunUJA.zip* con el juego, y lo descomprimimos. Buscaremos la carpeta del nuevo proyecto y allí copiamos tal cual el contenido del fichero descomprimido.

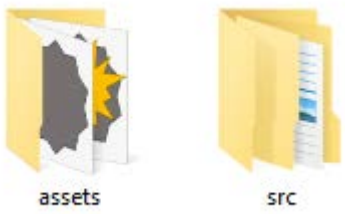

Figura 2. Contenido del archivo MouseRunUJA.zip

Si hemos hecho esto de forma apropiada, al volver a abrir NetBeans y tratar de ejecutar el proyecto, nos pedirá la clase que contiene el método *Main* (debemos indicarle 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"**.

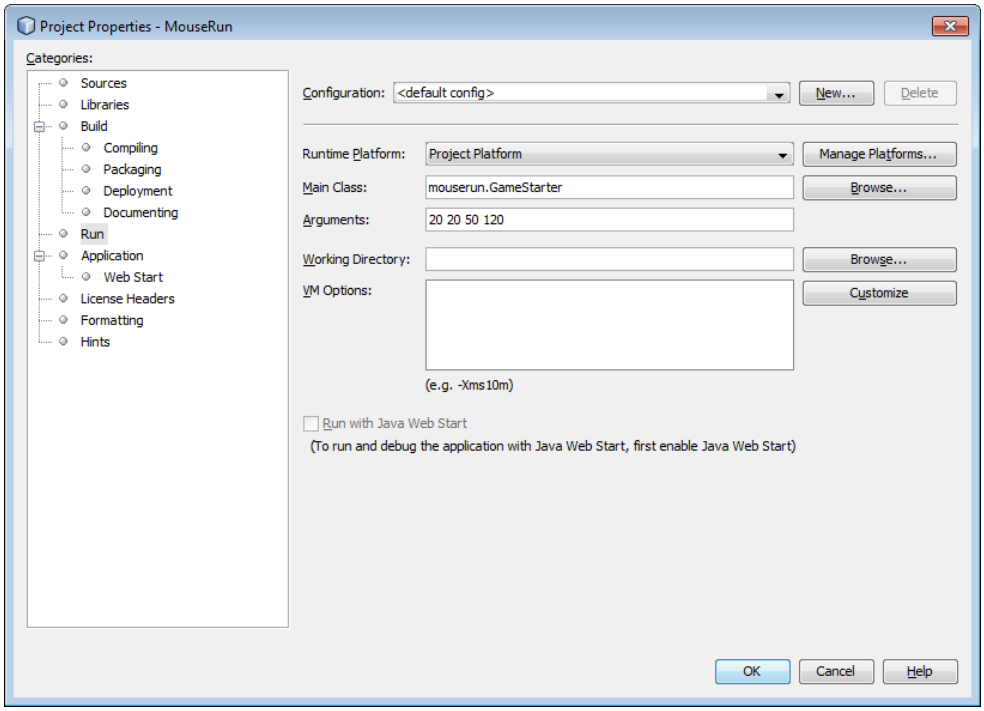

Figura 3. Propiedades del proyecto MouseRun

1.4. Creación de un nuevo ratón

Para crear nuestro ratón, vamos a proceder con los siguientes pasos:

1. Abrir un editor de texto (o con 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 alojará el ratón en el laberinto una vez 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
{

}
```

Figura 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()
     {

     }
}
```

Figura 5. Métodos principales de la clase FooMouse

Hemos creado un constructor para el ratón que le da el nombre *FooMouse* (llamando al constructor de la superclase, *super()*) el cual nos servirá para reconocerlo en el transcurso del juego y 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 ratón (nosotros o un rival) alcanza un queso y se lo come, momento en el cual el juego dispone un nuevo trozo de queso en una nueva posición al azar en el laberinto. Si no se quiere hacer nada, se puede dejar tal cual. 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.

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 comidos, 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.

1.5. 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 trozos de queso 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 trozo de 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 dos ratones, *TestMouse.java* y *ProblematicMouse.java*, como 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, contenido en un único archivo .java.

1.6. Ejercicio 2. Análisis del comportamiento del agente

Una vez que nos hayamos familiarizado con el entorno y 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 = 120 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 para cada uno de los ratones presentes en el laberinto (entre ellos, el número de quesos, 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				

- En la columna **Quesos** apuntaremos los trozos de queso que nuestro ratón ha logrado comer.

- 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** se obtiene como el cociente entre **Casillas exploradas** y **Ancho x Alto**.

Ojo, el juego nos proporcionará estas estadísticas sólo si nuestro ratón continúa en la partida cuando ésta finalice (es decir, que no haya sido descalificado previamente), por lo que a la hora de rellenar la tabla, debemos asegurarnos de que nuestro ratón funciona adecuadamente.

1.7. Ejercicio 3. Búsquedas no informadas

Partiendo del ratón implementado para el ejercicio anterior se incorporará un mecanismo de búsqueda que permita tomar decisiones precisas a la hora de determinar el mejor camino posible hacia el queso. 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 ratón (ejercicio anterior).

Los alumnos incorporarán el algoritmo de **Búsqueda Primero en Profundidad**. 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 la partida hasta 600 segundos**, para dar margen a que los ratones puedan explorar el laberinto y realizar la búsqueda, que será más precisa conforme avance el tiempo.

1.8. Consideraciones importantes sobre las búsquedas

El árbol de estados tendrá un aspecto similar al que se describe a continuación. Consideramos un trozo del laberinto como el del ejemplo, en la figura 6, donde las zonas sombreadas son casillas aún no exploradas por nuestro ratón:

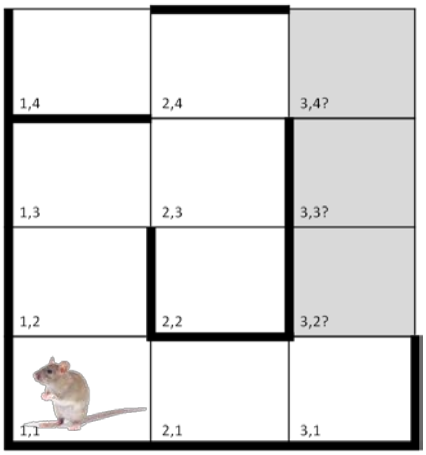

Figura 6. Porción de laberinto

El árbol resultante de este laberinto sería el siguiente:

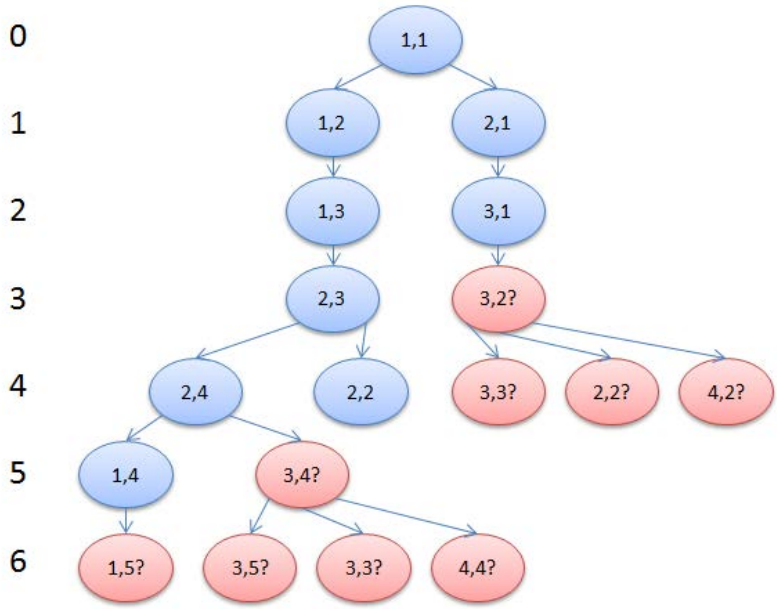

Figura 7. Expansión del árbol de movimientos para la figura 6

Se puede observar que:

- Cada nodo expandirá los posibles caminos que tiene como hijos en el árbol.
- Una vez visitado el hijo, se podrá eliminar como descendiente el nodo padre del que viene. Es decir, si te mueves a la (2,1) desde la (1,1), la casilla (1,1) no será considerada descendiente de (2,1). Esto es una simplificación importante que reduce considerablemente el número de nodos a expandir. No está contemplada en los algoritmos originales, pero parece lógico hacerlo.
- En el caso de visitar un nodo que representa una casilla sin explorar (casillas (3,2) y (3,4) en el ejemplo, figura 3), se supondrá que ésta tiene abiertas todas las direcciones (aunque realmente no sea verdad). Las casillas sin explorar siempre tendrán tres descendientes, equivalentes a las tres direcciones diferentes de la que venimos cuando visitamos dicha

casilla. Esto es una aproximación heurística muy optimista y la realidad, conforme vayamos avanzando, nos dirá que no está todo abierto y habrá muros que impidan el camino. Por este motivo, es muy importante haber explorado antes el laberinto lo más rápida y completamente posible.

La estrategia a seguir dependerá de dónde se halle el queso:

- Si se encuentra en una casilla ya explorada, lo más sensato es recurrir al algoritmo de búsqueda para ir por el camino óptimo.
- Si no está en una casilla ya explorada, habría que llegar a aquella casilla ya explorada más cercana a la posición del queso y comenzar a partir de ahí una nueva fase de exploración de laberinto dirigida.

En definitiva, cada ratón ejecutará tres fases:

1) Fase de exploración dirigida (corresponde al ejercicio 1).
2) Fase de búsqueda del camino mínimo hacia el queso, que será mejor conforme el laberinto sea mejor conocido en el tiempo.
3) Fase de molestia a los demás ratones, donde de forma libre, los ratones deberían molestar al resto situando bombas en sitios clave del laberinto.

Las fases 1 y 3 harán que los ratones sean más competitivos, puesto que la fase 2 (bien implementada) debe proporcionar un comportamiento similar en todos los ratones.

1.9. Ejercicio 4. Análisis del proceso de búsqueda

Al igual que antes, para evaluar el comportamiento del ratón ante diferentes escenarios (distintos laberintos, presencia o no de rivales, etc.), efectuaremos un mínimo de 5 ejecuciones (partidas) diferentes, y con los resultados obtenidos rellenaremos la siguiente tabla:

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

- Las condiciones iniciales de **Tiempo, Ancho y Alto** del laberinto han de ser las mismas para todas las ejecuciones, y se corresponden con los

parámetros por defecto con los que trabajaremos en el anterior ejercicio (tiempo = 600 segundos, ancho = alto = 20 celdas).

- En la columna **Quesos** apuntaremos los trozos de queso que nuestro ratón ha logrado comer, igual que en el ejercicio 2.
- 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 **Ancho x Alto**. Mediante este parámetro estimaremos el grado en el que nuestro 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 **Pasos** y **Casillas exploradas**, y nos permitirá estimar el número medio de veces que un ratón pasa por una misma celda.

Recordad que el juego nos proporcionará estas estadísticas sólo si nuestro ratón sigue vivo cuando la partida finalice (es decir, que no haya sido descalificado).

2. Consideraciones importantes

- Aseguraos de generalizar 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 se proporciona de primeras para 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 nunca perdona y se asegura una competición justa con el resto de ratones. Cada ratón debe tomar una decisión de acción bajo un límite de tiempo (alrededor de 100 microsegundos por defecto). Durante este tiempo, el ratón no debe producir ninguna excepción de ejecución. El juego descalifica y elimina cualquier ratón que cause problemas continuos durante la ejecución.
- Los ratones entregados **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 los ratones no incumplan esta norma.

3. Aspectos que se valorarán en la nota final

- **La explicación o descripción de las estrategias seguidas por el ratón en las tres fases.**
- **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 del ratón en el informe y el uso correcto de convenciones de Java.**
- **La inexistencia de bucles infinitos, movimientos repetitivos o descalificaciones por el juego.**
- **Que el ratón encuentre SIEMPRE el camino mínimo hacia el queso, si éste se encuentra en una zona del laberinto ya explorada y donde no hay un hueco sin explorar durante el camino. Se observará en la parte final de la partida.**

4. Entrega y evaluación

- La práctica se realizará por parejas (recomendado). La puntuación máxima de la práctica será de **4 puntos** (**Ejercicio 1: 1.5 puntos. Ejercicio 2: 0.5 puntos, Ejercicio 3: 1.5 puntos. Ejercicio 4: 0.5 puntos**). La entrega se llevará a cabo a través de la actividad correspondiente en ILIAS. El plazo de entrega termina el **12 de marzo de 2018 a las 23:59**. La defensa de la práctica se llevará a cabo el día **13 de marzo de 2018** 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 elige mediante el método super() del constructor. 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 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.**
  - Los identificadores de ratón tendrán el siguiente formato: **M18NXX[a|b]**.[java|zip|rar], donde:
    - **N** será igual a
      - A (grupo 1, 10:30)
      - B (grupo 2, 12:30)

        - C (grupo 3, 15:30)
        - D (grupo 4, 17:30)
        - E (grupo 5, 12:30)
    - **XX** será el identificador numérico de cada pareja (01, 02, 03...). Se asignará en la sesión de explicación de la práctica.
    - Por último, el sufijo **"a"** se refiere al ratón explorador, y el sufijo **"b"**, al ratón que implementa la búsqueda.
  - **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.
- Además del código, se entregará un documento en formato PDF describiendo el desarrollo de la práctica. Para el primer y tercer ejercicios se comentará, en lenguaje natural, la estrategia seguida en cada uno de los métodos implementados. No debe incluir código fuente. Para el segundo y cuarto ejercicios, rellenar las tablas y razonar brevemente los resultados. Extensión recomendada: 3 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, identificador y nombre del ratón.
- **El no cumplimiento de las anteriores normas repercutirá negativamente en la calificación de la práctica.**