Práctica 1: Lemmings

Entrega: Semana del 14 de octubre

Objetivos: Iniciación a la orientación a objetos y a Java; uso de arrays y enumerados; manipulación de cadenas con la clase `String`; entrada y salida por consola.

Preguntas Frecuentes: Como es habitual (y normal) que tengáis dudas, las iremos recopilando en este documento de preguntas frecuentes. Para saber los últimos cambios que se han introducido puedes consultar la historia del documento.

Control de copias

Durante el curso se realizará control de copias de todas las prácticas, comparando las entregas de todos los grupos de TPI. Se considera copia la reproducción total o parcial de código de otros alumnos o cualquier código extraído de Internet o de cualquier otra fuente, salvo aquellas autorizadas explícitamente por el profesor.

1. Descripción de la práctica

1.1 Introducción

Lemmings es un juego clásico, lanzado a comienzos de la década de los noventa, que tuvo una gran influencia en el desarrollo de videojuegos tipo rompecabeza. El juego consiste en guiar a unas criaturas, los lemmings, que caminan automáticamente, a la salida asignándoles distintas tareas, como sacar un paracaídas, cavar o bloquear el camino. Ha habido muchas versiones después de su primer lanzamiento, con entornos 3D

Vista del juego en su versión clásica (Fuente: www.smithsonianmag.com)

En esta práctica desarrollaremos una versión simplificada del juego clásico, si bien más adelante podremos introducir algunas novedades. En el juego original la acción se desarrolla en tiempo real, es decir, los lemmings se mueven de forma continua, independientemente de las acciones que tome el jugador. Sin embargo,

en nuestro caso el juego se desarrollará por turnos, en los que el jugador podrá realizar una acción en cada ciclo del juego, de forma que el juego permanece parado hasta que el jugador indica la acción. Seguidamente, los lemmings se actualizarán para realizar sus movimientos o acciones correspondientes.

Vista del juego en su versión moderna

Si no has jugado, o no conoces el juego, te recomendamos que lo pruebes antes de desarrollar la práctica. Existen varias versiones gratuitas en la web, una de ellas es accesible a través del enlace: https://www.1001juegos.com/juego/html5-lemmings.

Durante el cuatrimestre vamos a ir desarrollando progresivamente nuestra propia versión del juego. Empezaremos en esta práctica con una versión reducida en la que los lemmings solo caminan automáticamente y no podemos cambiar su rol por defecto, caminante, pero pueden alcanzar la salida en caso de que exista. En la práctica 2 incorporaremos más funcionalidad (los distintos roles que pueden realizar los lemmings) pero ya haciendo uso de las capacidades que nos da la POO (herencia y poliformismo).

1.2. Detalles sobre la práctica

En nuestra primera práctica vamos a considerar que el juego consta de un tablero de **10 x 10** casillas (10 columnas por 10 filas). La versión que implementeís debe depender de constantes de tal forma que el tablero cambie de tamaño con el cambio de estas constantes. Internamente en el modelo la casilla de arriba a la izquierda es la (0, 0) y la de abajo a la derecha la (9, 9). No obstante, para hacerlo más amigable al usuario en la vista se mostrarán las filas identificadas con letras, de la `A` a la `J` en nuestro caso, y las columnas se mostrarán con número empezando en el 1 e incrementandose de 1 en en uno, terminando en nuestro caso en el 10. Por lo que, la casilla de arriba a la izquierda es la `A1` y la de abajo a la derecha la `J10`. Cada casilla puede estar ocupada por uno o varios lemmings o por una pared/suelo. Las casillas que no estén ocupadas se considerarán casillas vacías.

Todos los lemmings se mueven automáticamente en la dirección que lleven. Inicialmente se mueven hacia la derecha. Si se topan con una pared o con un lateral invierten su dirección. Si llegan a un precipicio caen, y solo sobreviven a la caída si no es demasiado grande. Cuando llegan a la salida salen en la iteración siguiente y si se salen del tablero mueren. Observar que los lemming inicialmente solo pueden salirse por debajo del tablero, pues los laterales se comportan como paredes solidas.

El jugador gana la partida cuando no queden más lemmings en el tablero y hayan llegado a la salida tantos lemmings como requiere el nivel.

En esta práctica solo consideraremos un tipo de lemming, el lemming caminante o, como detallaremos más adelante, al lemming cuyo rol es siempre el de caminante.

En cada ciclo del juego se realizan secuencialmente las siguientes acciones:

1. ***Draw.*** Se pinta el tablero y se muestra la información del juego.
2. ***User command.*** El usuario puede actualizar el juego o ejecutar un comando que no actualiza el juego, como solicitar el listado de comandos disponibles o salir del juego.
3. ***Update***. El juego se actualiza, es decir, todos los lemmings del tablero se actualizan.

1.3. Objetos del juego

En esta sección describimos el tipo de objetos que aparecen en el juego y su comportamiento.

Lemming

Se mueve horizontalmente siguiendo una dirección (izquierda o derecha) o cae si está en el aire, es decir, si no hay ningún objeto sólido en la posición inferior. Consideramos que la fuerza con la que cae el lemming caminante coincide con la altura desde la que cae. Al llegar al suelo tras una caída de 3 o más filas muere (es decir, con una fuerza mayor que 2). Si la fuerza con la que cae es inferior sigue caminando en su dirección. En cada iteración del juego dará un único paso.

Además, los lemmings se consideran elementos *no sólidos* del juego, lo que significa que pueden compartir posición con otros elementos no sólidos (por ejemplo, con otros lemmings).

Pared/suelo Es un elemento pasivo en el tablero, de forma que no hace nada al actualizar el tablero. Son sólidos, lo que quiere decir que no puede compartir posición con ningún otro objeto sólido (ninguna otra pared) y que los lemmings pueden estar de pie encima de ellos.

2. Organización del juego

A continuación, describimos lo que ocurre en cada parte del bucle del juego.

2.1 Draw

En cada ciclo se pintará el estado actual del tablero, así como otra información extra que no se encuentra de forma visual en el tablero: el ciclo actual del juego (inicialmente 0), el número de lemmings que quedan en el tablero, el número de lemmings muertos y el número de lemmings que ya han salido, seguido del número de lemmings que tienen que salir como mínimo para ganar.

Cada lemming en el tablero se muestra mediante un símbolo **'B'** si está caminando hacia la derecha o **' '** si está caminando a la izquierda (sin comillas). La pared se muestra siempre con el símbolo **' '** y la puerta de salida se muestra

con el símbolo ‘ ’. También mostraremos el **prompt** del juego para solicitar al usuario la siguiente acción.

El tablero se pintará por el interfaz consola utilizando caracteres ASCII, como muestra el siguiente ejemplo:

```
Lemmings 1.0

Number of cycles: 0
Lemmings in board: 2
Dead lemmings: 0
Lemmings exit door: 0  2

     1    2    3    4    5    6    7    8    9   10
   -------------------------------------------------
 A                                                B   A
 B                                               B
 C                                                    C
 D              B    B                                D
 E                                             E
 F                                                F
 G                                          G
 H                                                    H
 I   B                                             I
 J                                          J
   -------------------------------------------------
     1    2    3    4    5    6    7    8    9   10

Command >
```

No obstante se ha creado una capa de colores (versión beta) `ConsoleColorsView` que podeís utilizarla para hacer más atractivo el juego. Para utilizar dicha capa sólo es nesario cambiar `ConsoleView` por `ConsoleColorsView` en la línea correspondiente del fichero `Main.java`.

2.2 User actions

En cada turno, tras pintar el tablero, se preguntará al usuario qué quiere hacer, a lo que podrá contestar con uno de los siguientes comandos:

- `help`: Este comando solicita a la aplicación que muestre la ayuda relativa a cómo utilizar los comandos. Se mostrará una línea por cada comando. Cada línea tiene el nombre del comando seguida por ‘:’ y una breve descripción de lo que hace el comando.

```
Command > help

Available commands:
[r]eset: start a new game
```

```
[h]elp: print this help message
[e]xit: end the execution of the game
[n]one | "": skips cycle
```

- `reset`: Este comando permite reiniciar la partida, llevando al juego a la configuración inicial.
- `exit`: Este comando permite salir de la aplicación, mostrando previamente el mensaje *Player leaves game*.
- `none`: El usuario no realiza ninguna acción, se actualiza el juego.

Observaciones sobre los comandos:

- La aplicación debe permitir comandos escritos en minúsculas, mayúsculas o mezcla de ambas.
- La aplicación debe permitir el uso de la primera letra del comando (o la indicada entre corchetes, si esa letra ya se utiliza) en lugar del comando completo `[R]eset`, `[H]elp`, `[E]xit`, `[N]one`.
- Si el comando es vacío se identifica como `none` y se avanza al siguiente ciclo de juego.
- Si el comando está mal escrito, no existe, o no se puede ejecutar, la aplicación mostrará un mensaje de error.
- En el caso de que el usuario ejecute un comando que no cambia el estado del juego, o un comando erróneo, el tablero no se debe repintar.

2.3 Update

En cada ciclo se produce la actualización de cada lemming, que da lugar a sus movimientos (y más adelante posiblemente a otras acciones).
El juego finalizará cuando no queden más lemmings en el tablero o cuando el usuario ejecute el comando `exit`.

Cuando el juego termine por no haber lemmings en el tablero se debe mostrar el mensaje **'Player looses'** o el mensaje **'Player wins'**, en función de si han salido suficientes lemmings. Consulta la clase `Messages.java` donde ya están definidas estas constantes de tipo `String`.

Parámetros de la aplicación

En esta primera versión de la práctica el juego los tableros se almacenaran de manera ***ad-hoc***, colocando lemmings y paredes en distintas posiciones en métodos dedicados a ello: `initGame0()`, `initGame1()`…

El programa debe aceptar un parámetro opcional por línea de comandos, llamado **level**. En el ejemplo de la imagen se cagará el nivel `1`. En caso de que no exista el argumento se debería cargar dicho nivel. El `initGame0()` es el mismo

pero sin el lemming de la posicion `D3`, el resto de niveles los podeís crear para realizar otras pruebas, como en las que los lemmings ganen, o que algún lemming se quede bloqueado y nunca termine, etc. Más adelante os daremos varios mundos.

Opciones de ejecución

3. Implementación

La implementación propuesta para la primera práctica no es la mejor, ya que no hace uso de **herencia** y **polimorfismo**, dos herramientas básicas de la programación orientada a objetos. Más adelante, veremos formas de mejorarla mediante el uso de las herramientas que nos brinda la programación orientada a objetos.

Para implementar la primera versión tendremos que copiar y pegar código y esto casi siempre es una mala práctica de programación. La duplicación de código implica que va a ser poco mantenible y dificíl de *testear*. Hay un principio de programación muy conocido llamado **DRY (Don't Repeat Yourself)** (**No te repitas**, en castellano). Según este principio, ninguna información debería estar duplicada, ya que la duplicación incrementa la dificultad de los cambios y evolución posterior, puede perjudicar la claridad y dar pie a posibles inconsistencias.

En la siguiente práctica veremos cómo refactorizar el código para evitar repeticiones.

Para lanzar la aplicación se ejecutará la clase `tp1.Main`, por lo que se aconseja que todas las clases desarrolladas en la práctica estén en el paquete `tp1` (o subpaquetes suyos).

Objetos del juego

Para representar cada uno de los tipos de elementos que pueden aparecer en el tablero, a los que llamaremos objetos del juego, necesitarás, al menos, las siguientes clases:

- `Lemming`: clase que representa a un lemming. Tiene como atributos su posición `(columna, fila)`, un booleano que indica si está vivo o no, la dirección de su movimiento, la fuerza de caida, … Además, los lemming tienen un atributo de tipo `WalkerRole` y otro de tipo `Game` que explicamos a continuación.

  El atributo de tipo `WalkerRole` representa el rol del lemming en ese momento. Actualmente el único rol implementado será el de ***caminante***. El lemming delega muchas de sus tareas en este atributo. Esto nos permitirá en la práctica 2 cambiar el rol del lemming por otro distinto (y por lo

tanto su comportamiento y cómo se muestra en el tablero) en tiempo de ejecución.

El atributo de tipo `Game` (ver más adelante) permitirá al lemming interactuar con su entorno para, por ejemplo, saber si la posición a la que se pretende mover está ocupada. Esto tiene la desventaja de que, en principio, un lemming tiene acceso a todos los métodos públicos de `Game`, no solo aquellos pensados para la interacción del lemming con su entorno. Esto lo resolveremos en la práctica 2 mediante el uso de ***interfaces***.

- `Wall`: clase que representa la pared o el suelo en el tablero y que en esta versión de la práctica va a tener poca o ninguna funcionalidad. Tiene un atributo para su posición. Como no interactúa con su entorno no necesita el atributo de tipo `Game`.
- `ExitDoor`: clase que representa la puerta de salida. Si un lemming está en una puerta de salida (es decir, comparte posición con una puerta) al actualizarse entrará en la puerta y desaparecerá del tablero. Esto hará que se considere que ha salido correctamente del mundo. Al igual que los otros elementos, tiene un atributo para su posición. Como no interactúa con su entorno no necesita el atributo de tipo `Game`.

Rol `WalkerRole`

Como hemos indicado antes esta clase será la responsable de ejecutar el rol sobre el lemming y también la responsable de devolver el icono del lemming. En un futuro podrá ser responsable de más tareas y además nos permitirá, con la misma idea, implementar otros roles. Para realizar dichas tareas solo es necesario implementar los siguientes métodos que reciben el lemming sobre el que se aplicará:

```
public void play( Lemming lemming ) {...}
public String getIcon( Lemming lemming ) {...}
```

El atributo de tipo `WalkerRole` que tendrá el lemming representa el rol del lemming, en este caso el único valor posible para dicho rol es el de ***caminante***. Como por defecto el lemming ya es un caminante lo único que será necesario en la implementación del `WalkerRole` es llamar al método del lemming en el que has implementado la tarea de caminar.

Update

Todos los objetos del juego implementarán un método

```
public void update() {...}
```

en el que se implementará la actualización del objeto en función de su estado y del estado del juego. Esa actualización puede, por lo tanto, modificar el estado del objeto pero también el de su entorno, gracias a la referencia `game` que hemos dicho que íbamos a mantener (lo que de hecho ocurrirá en la práctica 2).

La actualización de una pared/suelo es trivial (no hacer nada). Los lemmings, sin embargo, deberán:

- Comprobar que están vivos
- Delegar en el `WalkerRole` que llamará al método correspondiente de caminar del lemming, el cual realizará las siguientes tareas:
  - Si están cayendo gestionar la caída. En particular, morir si alcanzan el suelo tras una caída demasiado grande. Se considera que alcanza el suelo en el momento que se estrella con este. Por lo cual si está cayendo aparecerá justo encima del suelo para en la siguiente iteración aterrizar, que consistirá en morir o dar un paso.
  - Si no están cayendo pero están en el aire tendrán que caer.
  - Si no se dan ninguna de las situaciones anterirores dar un paso normal. El paso consistirá en avanzar o cambiar de dirección.

Para saber si están cayendo o si mueren en una caída también es necesario delegar en su acción. Ten en cuenta que más adelante se pueden dar situaciones en las que estar en el aire no implique caer (escalador) o caer demasiado no implica morir (paracaídas).

Contenedor y gestor de objetos de juego

En el tablero hay una única puerta de salida, pero puede haber múltiples lemmings y múltiples suelos/paredes. Por ello, necesitamos clases que representen *contenedores* de estos objetos del juego.

Una posibilidad es tener una clase `GameObjectContainer` que además de la puerta, contenga ***arrays*** (incompletos, es decir, suficientemente grandes y junto con un contador), uno para los lemmings y otro para las paredes.

Esta clase tendrá también métodos para su gestión. En particular, tendrá métodos

```
public void add(Lemming lemming) {...}
public void add(Wall wall) {...}
```

y

```
public void add(ExitDoor exitDoor) {...}
```

Fíjate que esos métodos están sobrecargados, es decir, se llaman igual pero el compilador los distingue por el tipo de su parámetro.

Además, el contenedor será responsable de llevar las peticiones del juego a cada uno de sus objetos.

El modelo: la clase `Game`

La clase `Game` encapsula la lógica del juego. Habrá una única instancia de `Game` en el programa. Contiene una instancia de `GameObjectContainer`, entre otras

instancias de objetos. También mantiene el contador de turnos, el número de lemmings en el tablero, …

En cuanto a sus métodos, tiene, entre otros, el método `update` que actualiza el estado de todos los elementos del juego. Su implementación consistirá esencialmente en incrementar el turno del juego e invocar al método `update` del `GameObjectContainer`, que a su vez invocará al método `update` de cada uno de los objetos del juego.

Clases para el control y la visualización

- `Controller`: clase para controlar la ejecución del juego. Coordina la vista y el modelo. Para preguntar al usario qué quiere hacer utilizará el método de la vista `getPrompt` y actualizara la partida de acuerdo a lo que éste indique. La clase `Controller` necesita, al menos, dos atributos privados:

  ```
  private Game game;
  private GameView view;
  ```

  La clase `Controller` implementa el método público `public void run()` que controla el bucle principal del juego. Concretamente, mientras la partida no esté finalizada, solicita órdenes al usuario y las ejecuta.

- `GameView`: recibe el `game` y tiene un método `showGame` que sirve para pintar el juego como veíamos anteriormente. Además contiene algunos otros métodos para mostrar otros mensajes.

Otras clases

- `Direction`: enumerado para representar los distintos movimientos que pueden hacer los objetos del juego.
- `Position`: clase que deberás hacer que sea inmutable para representar una posición del tablero, es decir, para encapsular una columna y una fila.
- `Main`: Es la clase que contiene el método `main` de la aplicación. En este caso, el método `main` crea una nueva partida (objeto de la clase `Game`), crea una vista objeto de la clase `GameView`, crea un controlador (objeto de la clase `Controller`) con dicha partida, e invoca al método `run` del controlador.

Observaciones a la implementación

Durante la ejecución de la aplicación solo se creará un objeto de la clase `Controller`. Lo mismo ocurre para la clase `Game`, que representa la partida en curso y solo puede haber una activa.

Junto con la práctica, os proporcionaremos una plantilla con partes del código.

El resto de información concreta para implementar la práctica será explicada por el profesor durante las distintas clases de teoría y laboratorio. En esas clases

se indicará qué aspectos de la implementación se consideran obligatorios para poder aceptar la práctica como correcta y qué aspectos se dejan a la voluntad de los alumnos.

4. Entrega de la práctica

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual, no más tarde de la **fecha y hora indicada en la tarea del campus virtual**.

El fichero debe tener, al menos, el siguiente contenido [1]:

- Directorio `src` con el código de todas las clases de la práctica.
- Fichero `alumnos.txt` donde se indicará el nombre de los componentes del grupo.

Recuerda que no se deben incluir los `.class`.

> **Nota**: Recuerda que puedes utilizar la opción `File > Export` para ayudarte a generar el .zip.

5. Pruebas

Junto con las instrucciones de la práctica, tendrás una carpeta con trazas del programa. Encontrarás varios ficheros con la siguiente nomenclatura:

- `00_1-play_input.txt`: es la entrada `1` del mapa `00` para probar cuestiones del funcionamiento (`play`).
- `00_1-play_expected.txt`: es la salida esperada para la entrada anterior.
- `01_1-command_input.txt`: es la entrada `1` del mapa `01` centrada en probar los `comandos`.
- `01_1-command_expected.txt`: es la salida esperada para la entrada anterior.

En Eclipse, para usar un fichero de entrada y volcar la salida en un fichero de salida, debes configurar la redirección en la pestaña `Common` de la ventana `Run Configurations`, tal y como se muestra en la siguiente figura. Lo más cómodo es crear, al menos, una `Run Configuration` para cada caso de prueba.

Redirección entrada y salida

Hay multitud de programas gratuitos para comparar visualmente ficheros, por ejemplo Eclipse ya tiene integrada una herramienta para comparar archivos que puedes lanzar al seleccionar dos archivos, pulsar con el botón derecho y en el menú emergente seleccionar `Compare With > Each other`.

Cómo comparar dos archivos en Eclipse

Aparecerá una nueva ventana donde se mostrarán marcadas claramente las diferencias entre los archivos.

[1] Puedes incluir también opcionalmente los ficheros de información del proyecto de Eclipse

Por supuesto, nuestra salida puede tener algún error, así que si detectas alguna inconsistencia por favor comunícanoslo para que lo corrijamos.

Durante la corrección de prácticas usaremos otros ficheros de prueba para asegurarnos de que vuestras prácticas se generalizan correctamente, así que asegúrate de probar no solo los casos que te damos, sino también otras posibles ejecuciones.