



# **Tecnología de la Programación I**

## **Presentación de la Práctica 1**

(Basado en la práctica de Alberto Núñez y Miguel V. Espada)

Ana M. González de Miguel (ISIA, UCM)

# Índice

1. Entrega de las Prácticas
2. Normas de Corrección
3. Control de Copias
4. Introducción
5. Descripción de la Práctica
  - 5.1. Objetos del Juego
  - 5.2. Bucle del Juego
  - 5.3. Colocación Inicial de las Naves
  - 5.4. Parámetros de la Aplicación
6. Implementación
7. Anexo a la Práctica 1

## 1. Entrega de las Prácticas

- ✓ Las prácticas tienen una **fecha límite** de entrega.
- ✓ Las prácticas se envían al profesor a través del **campus virtual**.
- ✓ Se envía un único fichero comprimido con formato **ZIP**, llamado *PracticaN*, siendo *N* el número de práctica. El ZIP debe contener un fichero *alumnos.txt* con el nombre de los componentes del grupo, el directorio *src*, el directorio *doc* y, opcionalmente, los ficheros de información del proyecto de Eclipse.
- ✓ Si fuera necesario modificar el tipo de entrega, el profesor avisará con tiempo suficiente.

## 2. Normas de Corrección

- ✓ La práctica que no funcione se considerará como *no entregada*. Los requisitos de funcionalidad están descritos en el enunciado de la práctica.
- ✓ La **defensa** de la práctica se realiza a partir del primer día de laboratorio después del final del plazo de entrega.
- ✓ En la defensa deberán estar presentes *los dos miembros del grupo*, que contestarán a las preguntas que el profesor estime conveniente hacerles sobre la implementación de la misma. El profesor de prácticas puede decidir no corregir o penalizar en los términos que estime oportunos a los alumnos que no sigan las indicaciones del formato para la entrega.

### 3. Control de Copias

- ✓ Durante el curso se realizará **control de copias** de todas las prácticas comparando las entregas de todos los grupos de TP.
- ✓ 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.
- ✓ En caso de detección de copia se informará al **Comité de Actuación ante Copias** que citará al alumno infractor y considerará si es necesario sancionar.

## 4. Introducción

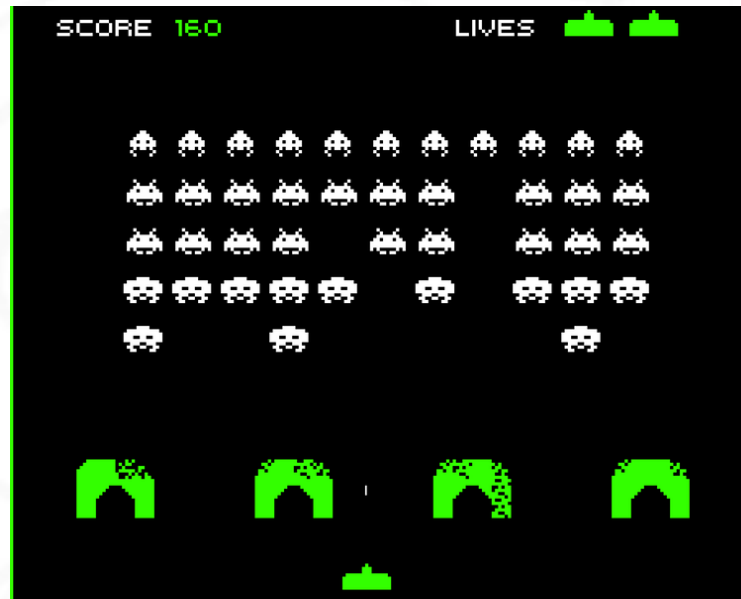
- ✓ Esta práctica pretende ser un **inicio** a la orientación a objetos y a Java.
- ✓ Otros **objetivos** son: uso de arrays y enumerados, manipulación de cadenas con la clase *String* y, entrada y salida por consola.
- ✓ Fecha de entrega: **17 de Octubre** de 2019 a las 9:00.
- ✓ Durante el cuatrimestre se va a desarrollar progresivamente nuestra propia versión del juego **space invaders**. Empezaremos con una versión muy sencilla e iremos aumentando su complejidad. En la primera versión se utilizará la interfaz consola.



- ✓ El **objetivo del juego** consiste en controlar una nave para destruir – disparando un misil – las naves alienígenas que aparecen en el espacio.
- ✓ Actualmente existen multitud de clones y versiones modernas del juego que han introducido novedades, como tipos de armas, escudos y superpoderes.
- ✓ Para esta práctica, utilizaremos tanto elementos del juego clásico como algunas de las novedades introducidas en las versiones más modernas.
- ✓ En el juego original la acción se desarrolla en tiempo real, es decir, los enemigos actúan de forma continua independientemente de las acciones que tome el jugador.
- ✓ En nuestro caso el juego se desarrollará por **turnos**.

- ✓ El jugador podrá realizar una **acción** en cada ciclo del juego que permanecerá parado hasta que el jugador indique la acción. Seguidamente, las naves se actualizarán para realizar sus movimientos o acciones correspondientes.





## 5. Descripción de la Práctica

- ✓ En nuestra primera práctica vamos a considerar que el juego consta de un **tablero** de  $8 \times 9$  casillas (8 filas por 9 columnas).
- ✓ La casilla de arriba a la izquierda es la (0, 0) y la de abajo a la derecha la (7, 8). Cada casilla puede estar ocupada por una nave alienígena, por *UCM-ship* (la nave que controla el usuario), un ovni, el misil de *UCM-ship* o un proyectil lanzado por una nave alienígena.
- ✓ Las casillas que no estén ocupadas se considerarán casillas vacías. Para distinguir los disparos de ambas naves, de aquí en adelante utilizaremos el término **misil** para hacer referencia al disparo realizado por *UCM-ship* y **proyectil** al disparo realizado por las naves alienígenas.

- ✓ Las naves alienígenas se moverán en grupo. Es decir, todas realizarán el mismo movimiento – o permanecerán paradas – en el mismo ciclo.
- ✓ Inicialmente, las naves se desplazarán hacia la izquierda con la velocidad relativa al nivel de dificultad seleccionado.
- ✓ Cuando una de las naves llegue al borde del tablero, todas las naves se desplazarán una casilla hacia abajo. En el siguiente ciclo, las naves comenzarán a moverse horizontalmente en dirección al borde opuesto, tal y como ocurre con el juego original.

- ✓ Si una nave alienígena logra llegar a la fila en la que se encuentra *UCM-ship* o *UCM-ship* se queda sin puntos de daño, el jugador habrá perdido la partida.
- ✓ Sin embargo, el jugador ganará cuando haya destruido todas las naves alienígenas, es decir, si durante el *Update*, no quedan naves alienígenas en el tablero.
- ✓ En esta práctica sólo consideraremos dos tipos de naves alienígenas: Nave *común* y Nave *destructora*.

- ✓ En cada **ciclo** del juego se realizarán secuencialmente las siguientes acciones:
1. **Draw.** Se pinta el tablero y se muestra la información del juego.
  2. **User command.** El usuario puede realizar una acción, por ejemplo: moverse lateralmente o realizar un disparo. El usuario puede no hacer nada en un ciclo y dejar pasar el tiempo.
  3. **Computer action.** El ordenador puede decidir si una nave destructora realiza un disparo o si aparece un ovni (ver más adelante) en la primera fila del tablero.
  4. **Update.** Se actualizan los objetos que están en el tablero.



## 5.1. Objetos del Juego

✓ Los objetos del juego son:

- **Nave UCM-ship:**
  - Comportamiento: Se mueve horizontalmente y realiza disparos. Inicialmente se coloca en la casilla (7,4).
  - Resistencia: 3 puntos de daño.
  - Disparo: Lanza un misil. El comportamiento de este disparo tiene como restricción que no puede haber más de un misil – simultáneamente – en el tablero. El misil avanzará verticalmente – en la misma columna – de forma que hasta que no haya alcanzado una nave alienígena, un proyectil o haya llegado al final del tablero, no se podrá disparar uno nuevo. Si el jugador realiza el disparo, el misil tendrá – en el momento de realizar la acción – la misma coordenada que *UCM-ship*. Sin embargo, su trayectoria se actualizará en la acción *update* del mismo ciclo.
  - Daño. El comportamiento estándar del disparo ocasiona un punto de daño en la nave impactada.



- **Nave común:**
  - Comportamiento: Se desplaza horizontalmente. Cuando una nave alienígena llega al borde del tablero, todas las naves se desplazarán una casilla hacia abajo y su movimiento lateral se realizará hacia el borde opuesto, tal y como ocurre con el juego original. El desplazamiento vertical se realizará en el ciclo siguiente al haber alcanzado el borde del tablero, independientemente de la velocidad establecida por la dificultad del juego.
  - Resistencia: 2 puntos de daño.
  - Disparo: Esta nave no realiza disparos.
  - Puntos: 5 puntos al ser destruida.

- **Nave destructora:**
  - Comportamiento: El movimiento es el mismo que la nave *común*.
  - Resistencia: 1 punto de daño.
  - Disparo: Deja caer un proyectil. El movimiento del proyectil es vertical – en la misma columna – en el sentido hacia donde se encuentra *UCM-ship*.
  - Daño: 1 punto de daño.
  - Puntos: 10 puntos al ser destruida.
- **Ovni**
  - Comportamiento: Se desplaza – horizontalmente – una casilla por cada ciclo, hasta alcanzar el borde del tablero o ser destruido por un misil de *UCM-ship*. Si es alcanzado por un disparo de *UCM-ship*, proporcionará un superpoder – llamado *shockWave* – al jugador. Inicialmente aparece en la casilla (0,8), es decir, en la esquina superior derecha.

El *shockWave* no se acumula, es decir, si el jugador ya dispone de uno y alcanza a un ovni, tendrá un *shockWave* activo, no dos. Una vez recorridas las casillas de la fila donde aparece, el ovni se elimina del tablero.

- Resistencia: 1 punto de daño.
- Disparo: Esta nave no realiza disparos.
- Puntos: 25 puntos al ser destruido.

## 5.2. Bucle del Juego

- ✓ **Draw.** En cada ciclo se pintará el estado actual del tablero, así como los puntos de daño (*life*) que le quedan al jugador, el ciclo de juego en el que nos encontramos (inicialmente 0), los puntos acumulados, el número de naves alienígenas que quedan actualmente en pantalla y si disponemos – o no – de un *shockWave* para poder utilizarlo.
- ✓ Al lado de cada nave en el tablero aparece la vida, o puntos de daño, que les queda (entre corchetes). El misil del usuario se representa con los caracteres 'oo' (sin comillas) y los proyectiles de las naves alienígenas con '.' (sin comillas). También mostraremos el prompt del juego para pedir al usuario la siguiente acción.

- ✓ El tablero se pintará por el interfaz consola utilizando caracteres ASCII, como muestra el ejemplo del enunciado (páginas 4 y 5).
- ✓ **User Command.** Se le preguntará al usuario qué es lo que quiere hacer, a lo que podrá contestar una de las siguientes alternativas:
  - **move <left|right><1|2>:** Este comando se utiliza para desplazar a *UCM-ship*. Acepta dos argumentos, que indican la dirección en la que realizará el desplazamiento: izquierda (left) o derecha (right) y el número de casillas que se desplaza (uno o dos).
  - **shoot:** La nave *UCM-ship* realiza un disparo (si es posible).
  - **shockwave:** La nave *UCM-ship* lanza una onda que causa un punto de daño a todas las naves alienígenas del tablero. Esta acción tiene lugar en el mismo ciclo que se ejecuta el comando.
  - **reset:** Este comando permite reiniciar la partida, llevando al juego a la configuración inicial.

- **list:** Mostrará el nombre de las naves actuales con su resistencia y su daño. En esta versión:

```
Command > list
```

```
[R]egular ship: Points: 5 - Harm: 0 - Shield: 2
```

```
[D]estroyer ship: Points: 10 - Harm: 1 - Shield: 1
```

```
[O]vni: Points: 25 - Harm: 0 - Shield: 1
```

```
^__^: Harm: 1 - Shield: 3
```

- **none:** El usuario no realiza ninguna acción.
- **exit:** Este comando permite salir de la aplicación, mostrando previamente el mensaje “Game Over”.
- **help:** Este comando solicita a la aplicación que muestre la ayuda sobre 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

move <left|right><1|2>: Moves UCM-Ship to the indicated direction.

shoot: UCM-Ship launches a missile.

shockWave: UCM-Ship releases a shock wave.

list: Prints the list of available ships.

reset: Starts a new game.

help: Prints this help message.

exit: Terminates the program.

[none]: Skips one cycle.

## ✓ Observaciones sobre los comandos:

- La aplicación debe permitir comandos escritos en minúscula y mayúscula o mezcla de ambos.
- 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 [M]ove, [S]hoot, shock[W]ave, [N]one, [L]ist, [R]eset, [H]elp, [E]xit.

- 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.
- 
- ✓ **Computer Action.** En esta primera práctica el ordenador tendrá un comportamiento pseudoaleatorio.
  - ✓ Uno de los parámetros de entrada será el nivel. Definiremos tres niveles: EASY, HARD y INSANE (ver Tabla 1.1).
  - ✓ Cada nivel determinará:

- El número de naves comunes que aparecen al inicio de la partida.
- El número de naves destructoras que aparecen al inicio de la partida.
- La frecuencia de disparo de las naves destructoras. La frecuencia determina la probabilidad de que una nave deje caer un proyectil. Así pues, si la frecuencia es 0.2, una nave tendrá un 20% de probabilidad de lanzar un proyectil en un ciclo, aunque al tratarse de una probabilidad pueden salir más espaciados – o menos – según la aleatoriedad. Es importante remarcar que no puede haber en pantalla dos proyectiles lanzados por la misma nave.
- La velocidad de las naves alienígenas. Este parámetro determina el número de ciclos que deben transcurrir para que las naves se desplacen una casilla.
- La probabilidad de que aparezca un ovni. De forma similar, este parámetro indica la probabilidad de que el programa genere un ovni en la partida. Cuando esto ocurre, el ovni se situará en la casilla (0, 8) y avanzará una casilla por ciclo, independientemente de la dificultad establecida. De igual manera que ocurre con los disparos, sólo puede haber un ovni en la pantalla.

- ✓ Para controlar el comportamiento aleatorio del juego y poder repetir varias veces la misma ejecución utilizaremos una semilla, o como se conoce en inglés, **seed**.
- ✓ Este valor opcional proporciona un control del comportamiento del programa lo que nos permitirá repetir exactamente una misma ejecución.
- ✓ En cada ciclo, el programa deberá decidir si cada nave *destructora* lanza un proyectil (si es posible) o si el *ovni* sale por pantalla (si no hay uno actualmente en el tablero).
- ✓ Sería interesante que probaras otros valores de número de naves y velocidad para encontrar aquellas combinaciones que sean más divertidas y jugables.

- ✓ **Update.** Las actualizaciones que ocurren en cada ciclo son, por el siguiente orden indicado:
- Avance en la trayectoria del **misil** lanzado por *UCM-ship*. El misil se desplazará verticalmente – desde la última fila hasta la primera – una casilla en cada ciclo. Si no existe un misil en el momento de realizar *Update*, esta acción será ignorada.
    - Si el misil lanzado por *UCM-ship* alcanza un proyectil de una nave alienígena, ambos se eliminan del tablero.
    - Si un misil de *UCM-ship* alcanza una nave alienígena, se decrementa – a la nave impactada – un punto de daño.
  - Avance en la trayectoria de los **proyectiles** existentes en pantalla. Cada proyectil se desplazará verticalmente – desde la nave que lanza el proyectil hacia la última fila – una casilla en cada ciclo.
    - Si un proyectil de una nave alienígena alcanza a *UCM-ship*, se decrementa – a *UCM-ship* – un punto de daño.



- Si *UCM-ship* llega a 0 puntos de daño, se utilizará el siguiente String para representar que la nave está destruida: '!xx!?' (sin las comillas).
  - Si una nave alienígena llega a 0 puntos de daño, desaparece del tablero.
  - Actualizar la posición de las naves alienígenas.
  - Si está en pantalla, se actualiza la trayectoria del ovni.
- ✓ El juego finalizará si durante la acción *Update* ocurre uno de los escenarios siguientes: i) todas las naves alienígenas son destruidas; ii) una de las naves alienígenas alcanza la fila donde se encuentra *UCM-ship*, o iii) la resistencia de *UCM-ship* llega a 0 puntos de daño. Cuando el juego termine se debe mostrar quién ha sido el ganador: **“Player wins”** o **“Aliens win”**.



### 5.3. Colocación Inicial de las Naves

- ✓ En el nivel EASY, las naves se colocarán en filas de 4, comenzando por la fila 1 y columna 3. Las naves destructoras están centradas con respecto a las naves comunes, como en el ejemplo del enunciado (página 8).
- ✓ En el nivel HARD, debe aparecer otra fila de naves comunes en la fila 2 del tablero, de forma que las naves destructoras aparecen en la fila 3.
- ✓ En el nivel INSANE, aparecen dos filas de naves comunes – en filas 1, 2 – de forma que las cuatro naves destructoras aparecen en la fila 3.

## 5.4. Parámetros de la Aplicación

- ✓ El programa debe aceptar un parámetro obligatorio y uno opcional por línea de comandos (ver Figura 3). En este caso, el parámetro del nivel es obligatorio, mientras que la semilla es opcional.
  - El primero, llamado **level** es el nivel de juego.
  - El segundo, llamado **seed**, contiene el valor de la semilla usada para el comportamiento pseudoaleatorio del juego.

## 6. Implementación

- ✓ La implementación propuesta para la primera práctica no es la mejor; ya que la vamos a hacer sin utilizar **herencia** y **polimorfismo**, dos herramientas básicas de la programación orientada a objetos.
- ✓ Para implementar la primera versión tendremos que copiar y pegar mucho código y esto casi siempre es una mala práctica de programación. La duplicación de código implica que va a ser poco mantenible y *testeable*. Hay un principio de programación muy conocido que se conoce como **DRY (Don't Repeat Yourself)** o **No te repitas** en castellano.
- ✓ A lo largo de las siguientes prácticas veremos cómo refactorizar el código para evitar repeticiones.

## 6.1. Detalles de Implementación

- ✓ Para lanzar la aplicación se ejecutará la clase *tp.p1.Main*, por lo que se aconseja que todas las clases desarrolladas en la práctica estén en el paquete *tp.p1* (o subpaquetes suyos).
- ✓ Para implementar la práctica necesitarás, al menos, las siguientes clases:
  - **UCMShip, RegularShip, DestroyerShip, Ovni**: Estas cuatro clases encapsulan el comportamiento de las naves del juego. Tienen atributos privados, como su posición (fila, columna), su vida, etc...
  - **RegularShipList, DestroyerShipList, BombList**: Contienen arrays de los respectivos elementos enemigos del juego, así como métodos auxiliares para su gestión.

- **Game:** Encapsula la lógica del juego. Tiene, entre otros, el método *update* que actualiza el estado de todos los elementos del juego. Contiene una instancia de *RegularShipList*, una *DestroyerShipList* y una *BombList*, entre otras instancias de objetos. También mantiene el contador de ciclos y la puntuación del jugador. Entre otros, tiene un atributo privado *Random rand* para generar los valores aleatorios.

También es posible que alguna de estas clases necesite un atributo en el que almacenen el juego, eso es, una instancia de la clase *Game* (que será la única en el programa) y que, como veremos, contiene la lógica del juego. De este modo, estas clases podrán usar los métodos de la clase *Game* para consultar si pueden hacer o no una determinada acción.

- **Level:** Es un clase enumerada con la que se encapsulan los tres niveles de juego.

- **GamePrinter:** Recibe el *game* y tiene un método *toString* que sirve para pintar el juego como veíamos anteriormente. Recomendamos el uso de las siguientes variables locales al método:

```
int cellSize = 7;  
String space = ;  
String vDelimiter = "|";  
String hDelimiter = "-";
```

- **Controller:** Clase para controlar la ejecución del juego, preguntando al usuario qué quiere hacer y actualizando la partida de acuerdo a lo que éste indique. La clase *Controller* necesita al menos dos atributos privados:

```
private Game game;  
private Scanner in;
```



El objeto *in* sirve para leer de la consola las órdenes del usuario. 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.

- **Main:** Es la clase que contiene el método *main* de la aplicación. En este caso el método *main* lee los valores de los parámetros de la aplicación (1, quizá 2), crea una nueva partida (objeto de la clase *Game*), crea un controlador (objeto de la clase *Controller*) con dicha partida, e invoca al método *run* del controlador.

## 6.2. Observaciones a la Implementación

- ✓ Durante la ejecución de la aplicación sólo se creará un objeto de la clase *Controller*. Lo mismo ocurre con la clase *Game* (que representa la partida en curso y solo puede haber una activa).
- ✓ En el **Anexo a la Práctica 1** se proporciona parte de la implementación de la clase *MyStringUtils*. Dicha clase proporciona 2 métodos para formatear strings y facilitar el “pretty-print” del tablero. Su uso no es obligatorio.
- ✓ En el anexo encontrarás unas trazas de la ejecución; la salida de la práctica debe coincidir con los ejemplos.

## 7. Anexo a la Práctica 1

```
package tp.pr1.util;

public class MyStringUtils {

    public static String repeat(String elmnt, int length) {
        String result = "";
        for (int i = 0; i < length; i++) {
            result += elmnt;
        }
        return result;
    }

    public static String centre(String text, int len) {
        String out = String.format("%"+len+"s%s%" + len + "s", "", text, "");
        float mid = (out.length())/2;
        float start = mid - (len/2);
        float end = start + len;
        return out.substring((int)start, (int)end);
    }

} // MyStringUtils
```