

# Práctica 0: Instalación y Puesta en marcha de la plataforma GVGAI

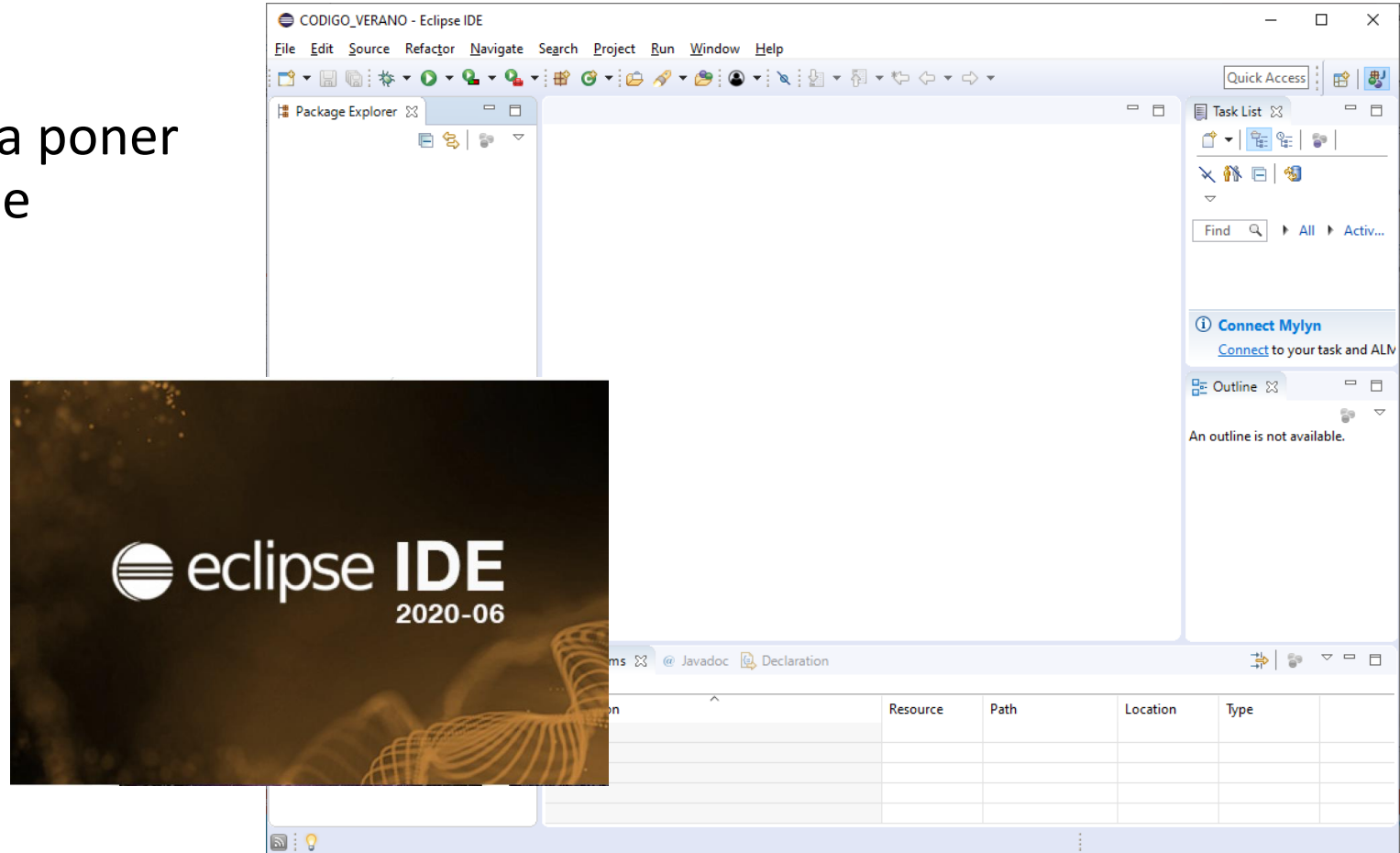
Sistemas Inteligentes. 2020-21

Grado en Ingeniería Informática

Especialidad: Computación

# Entorno de programación

- Libre!!
  - Pero nosotros vamos a poner los ejemplos en Eclipse



# Plataforma GVGAI

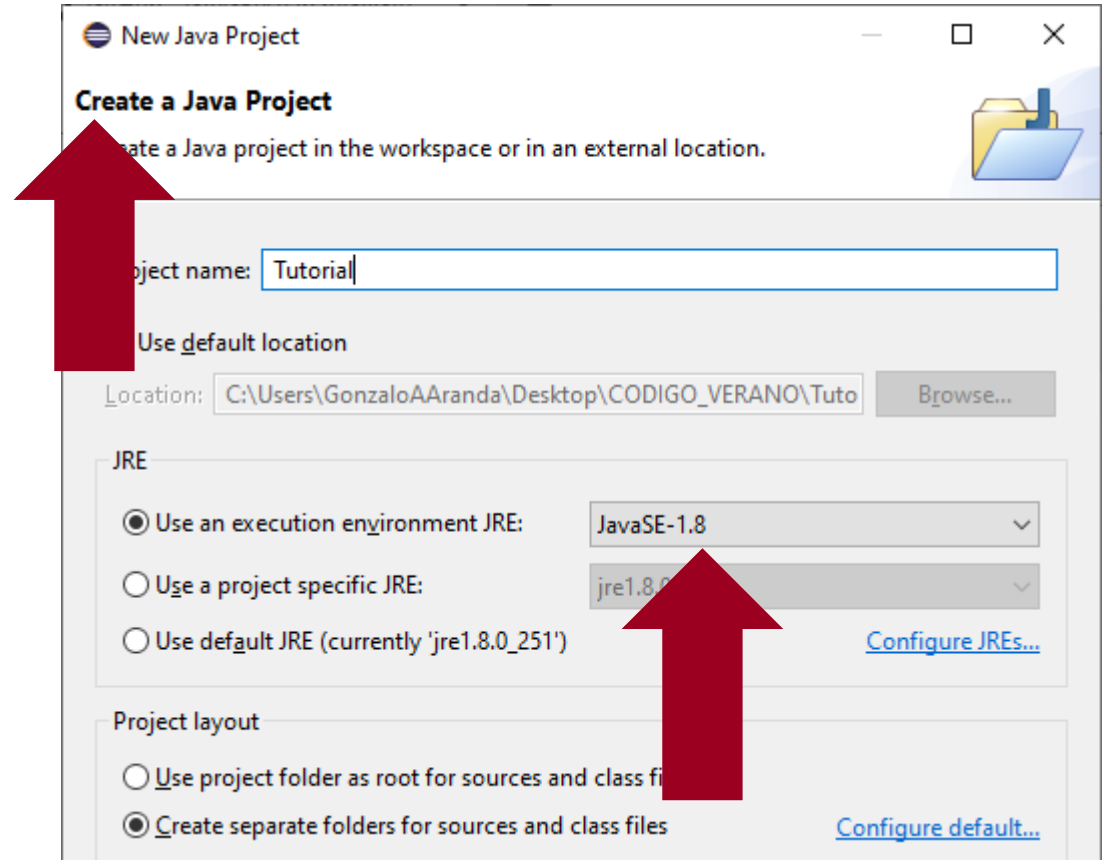
- El código actual se encuentra en la plataforma Moodle
  - Fue diseñada para la competición de algoritmos de Inteligencia Artificial.
  - Actualmente, sólo queda la competición de Aprendizaje Automático



- Descargamos el código en un directorio cualquiera
- Y descomprimos el contenido en una carpeta

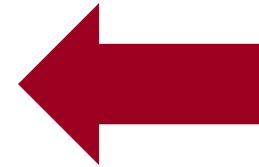
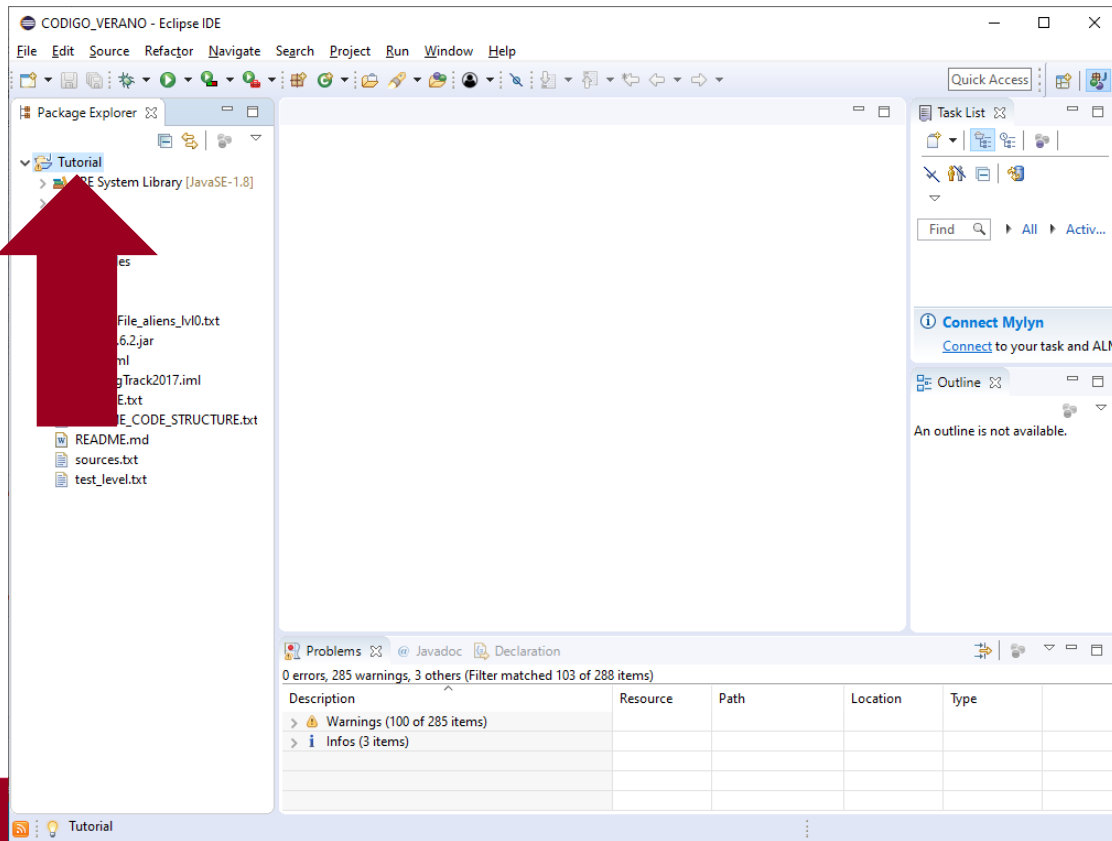
# Creación del proyecto GVGAI

- Se crea un proyecto Java
- Se elije la version 1.8 de Java



# Creación del proyecto GVGAI

- Se seleccionan TODOS los ficheros de la carpeta descomprimida
- Se copian en el directorio principal del Proyecto
  - Se sobrescribe Todo.



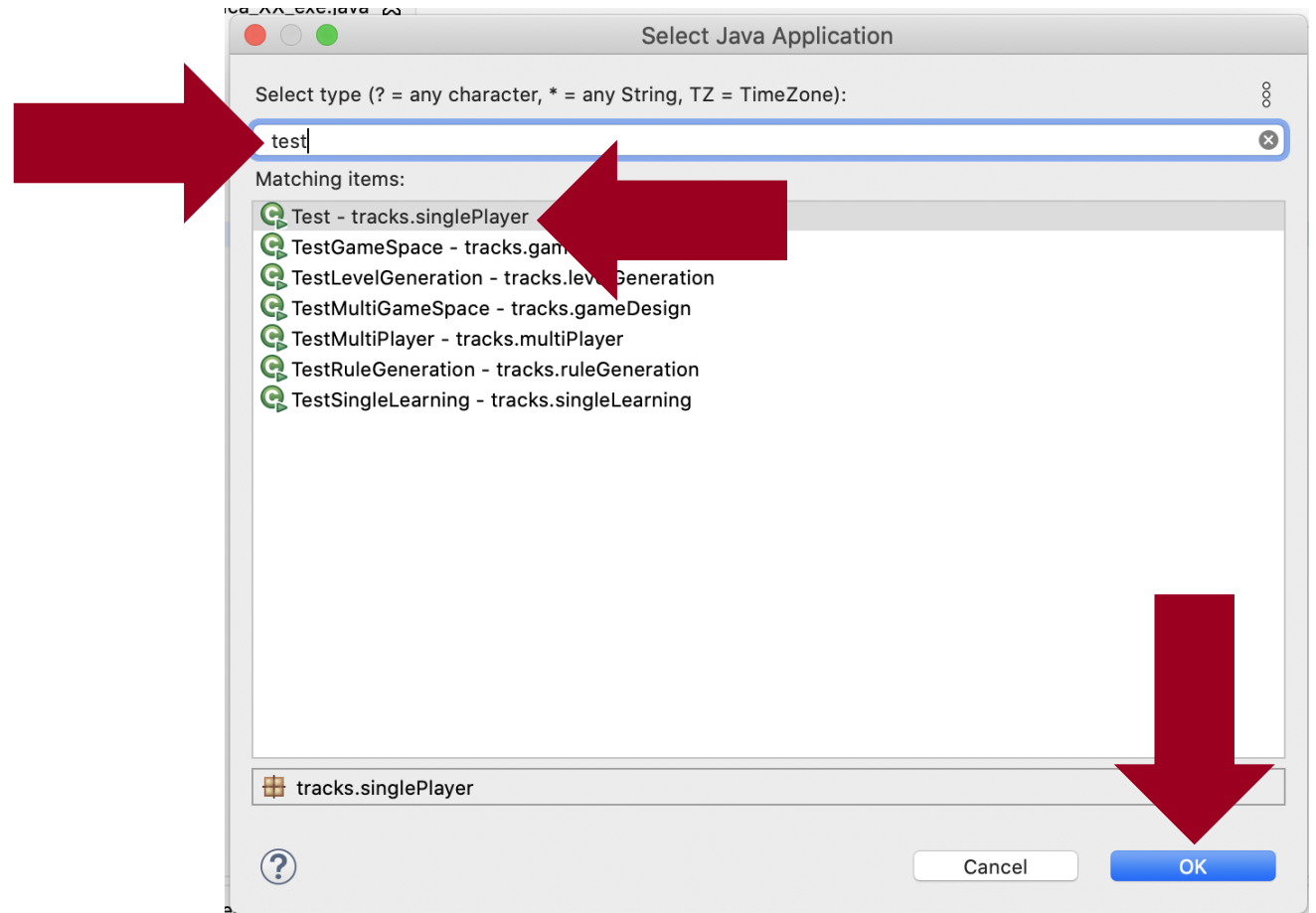
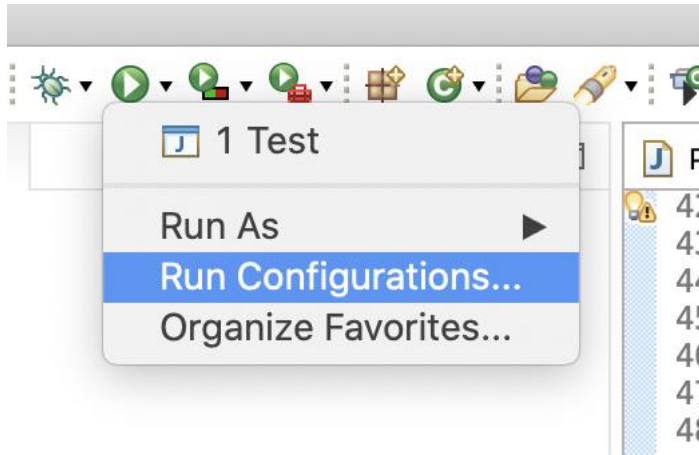
.idea	30/05/2019 13:58	Carpeta de archivos	
clients	30/05/2019 13:58	Carpeta de archivos	
doc	30/05/2019 13:58	Carpeta de archivos	
examples	30/05/2019 13:58	Carpeta de archivos	
logs	30/05/2019 13:58	Carpeta de archivos	
sprites	30/05/2019 13:58	Carpeta de archivos	
src	30/05/2019 13:58	Carpeta de archivos	
.gitignore	30/05/2019 13:58	Documento de te...	2 KB
actionsFile_alien_lvl0.txt	30/05/2019 13:58	Documento de te...	11 KB
gson-2.6.2.jar	30/05/2019 13:58	Executable Jar File	225 KB
gvgai.iml	30/05/2019 13:58	Archivo IML	1 KB
learningTrack2017.iml	30/05/2019 13:58	Archivo IML	1 KB
LICENSE.txt	30/05/2019 13:58	Documento de te...	2 KB
README.md	30/05/2019 13:58	Archivo MD	3 KB
README_CODE_STRUCTURE.txt	30/05/2019 13:58	Documento de te...	5 KB
sources.txt	30/05/2019 13:58	Documento de te...	0 KB
test_level.txt	30/05/2019 13:58	Documento de te...	2 KB

Ya está instalado!

Comprobar que funciona

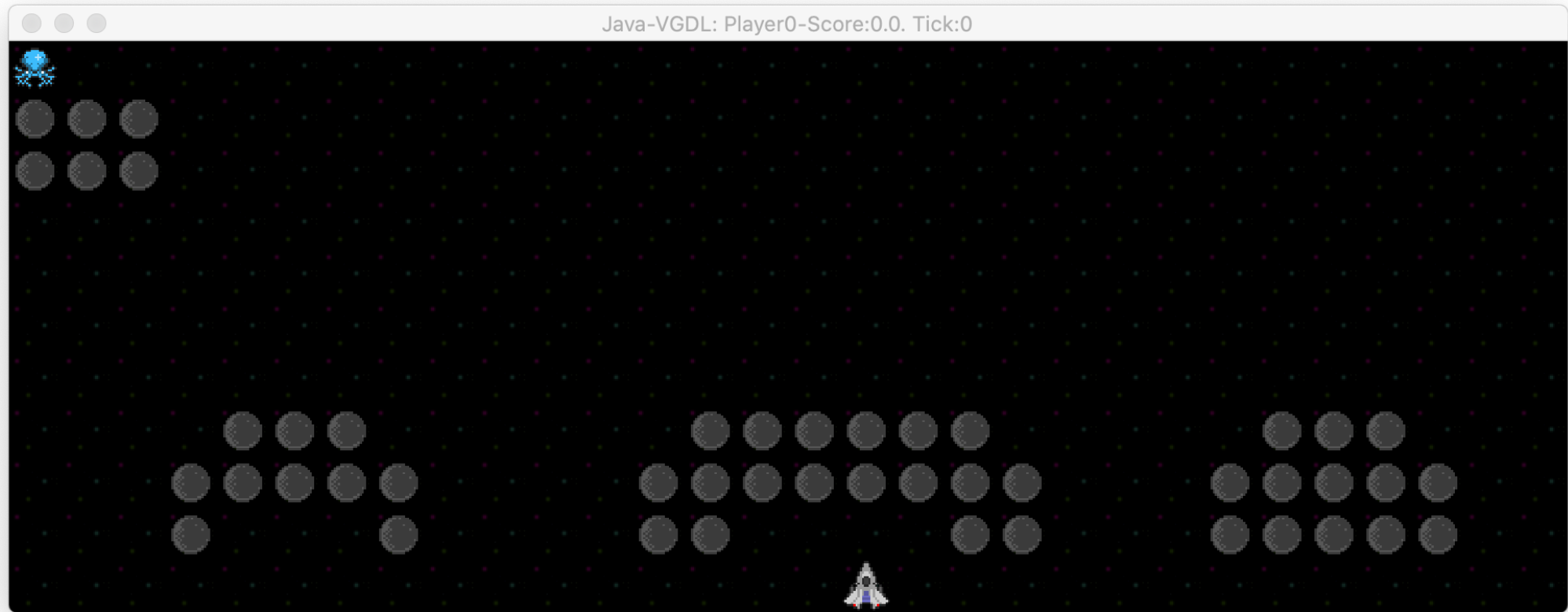
# Ejecución

- Para la ejecución de la plataforma:
  - Vamos a Run Configurations
  - Elegimos como clase main:  
`tracks.singlePlayer.Test`



# Ejecución

- Ya deberíamos de tener funcionando la plataforma.








# Plataforma GVGA

## • Opción 2

- También podemos descargar 2 ficheros de la plataforma Moodle de la UHU
  1. Fichero JAR
  2. un fichero de recursos
  3. Y una clase main

Plataforma GVGA

-  The General Video Game AI Competition - 2018
-  SI2021 GVGA (fuentes)
-  GVGA UHU (jar)
-  recursos
-  Practica XX exe

1

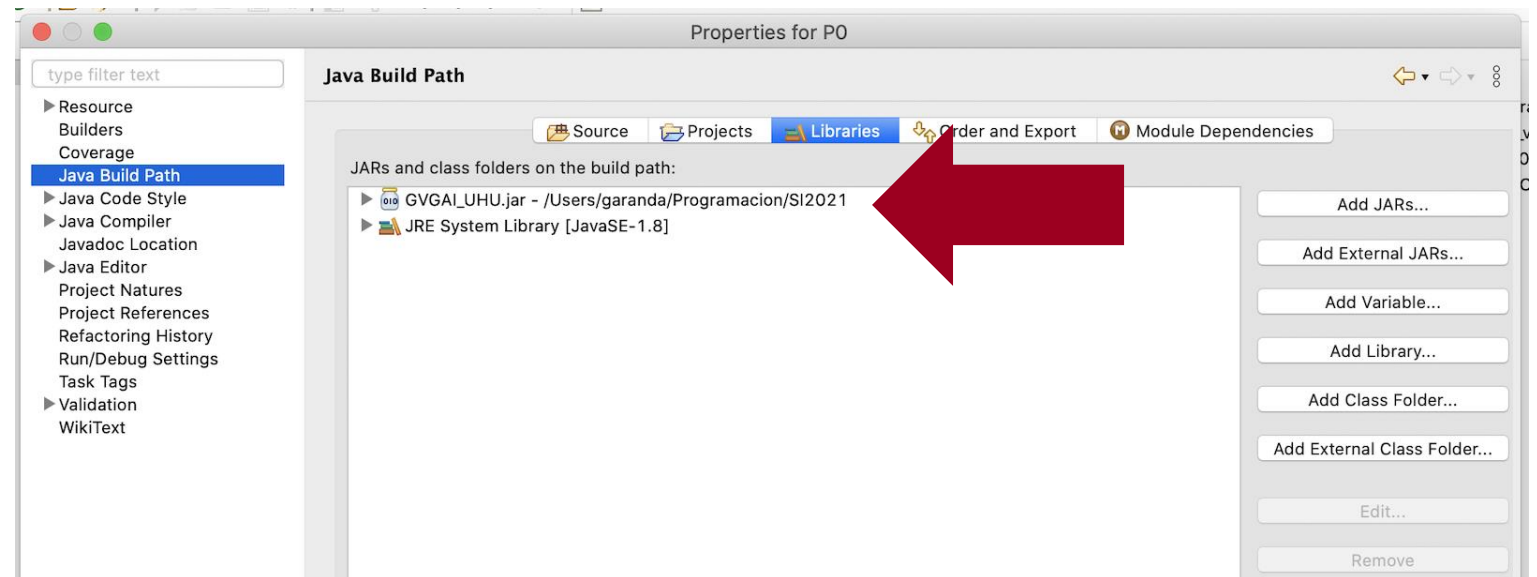
2

3

# Plataforma GVGAI

## Opción 2

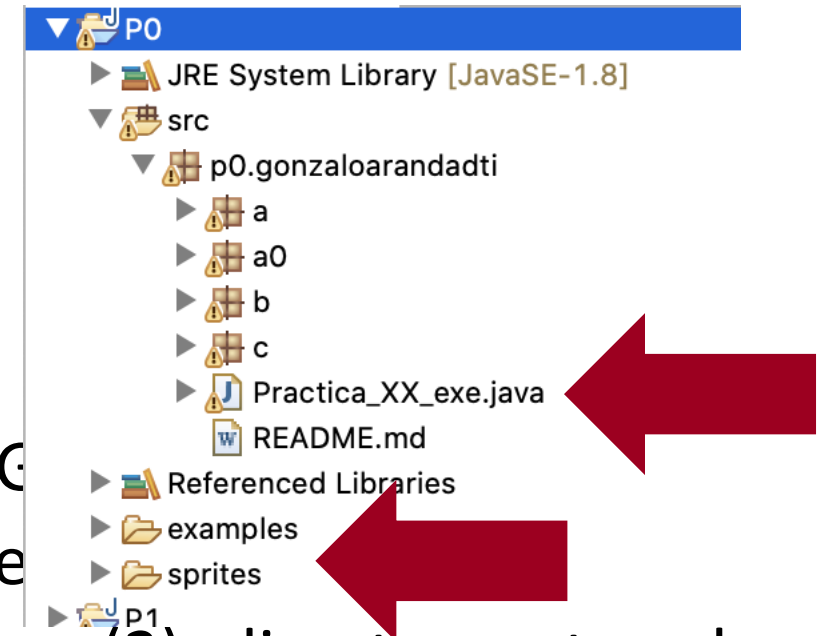
- Creamos un proyecto nuevo (diferente del GVGAI)
  1. Añadimos la librería jar (1) como referencia



# Plataforma GVGAI

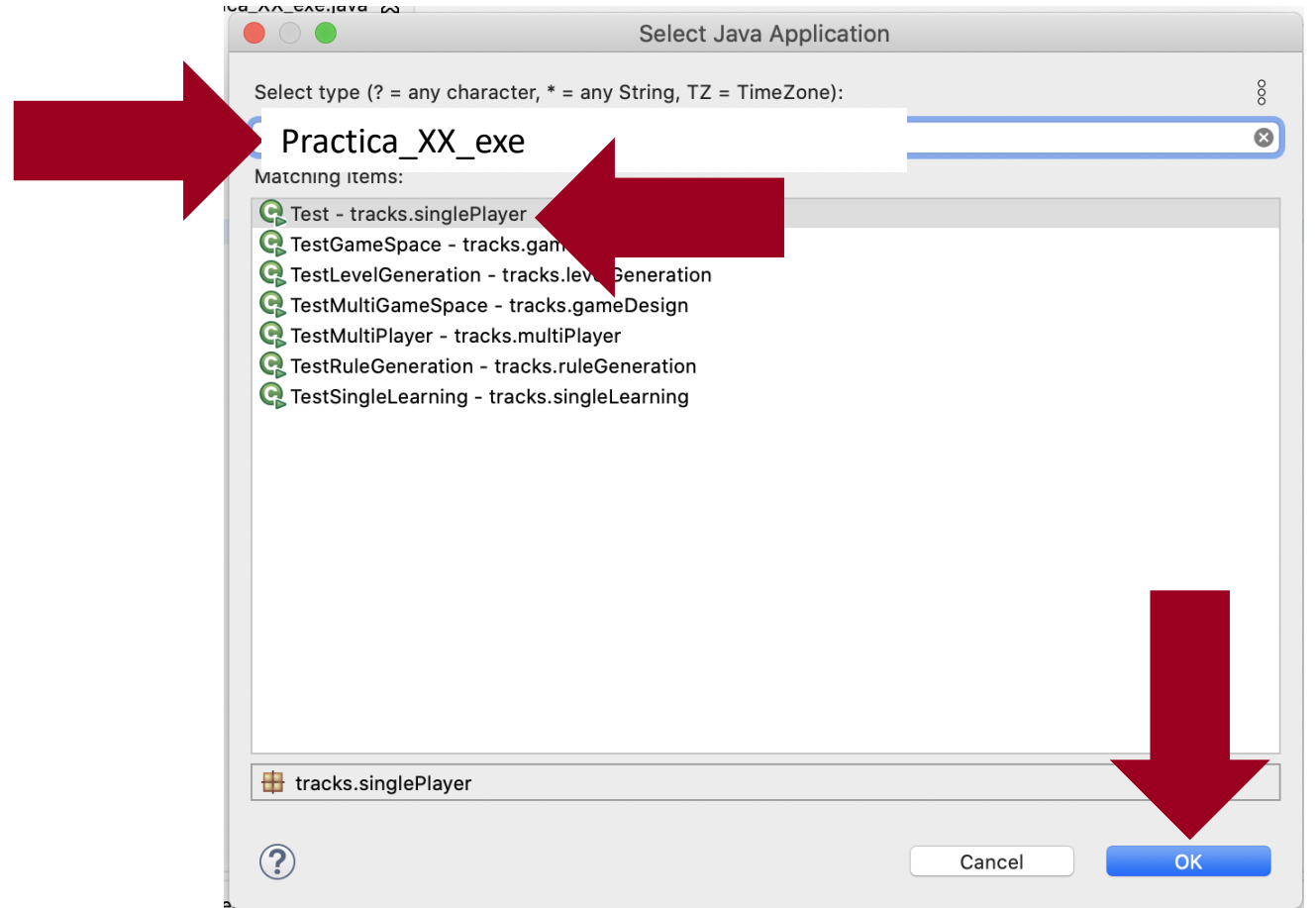
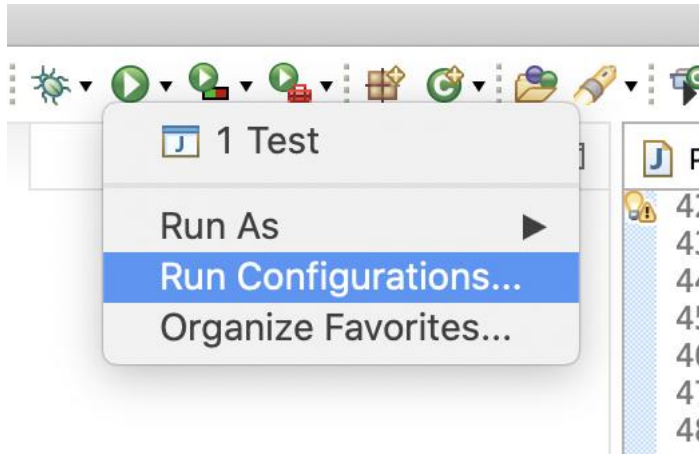
## Opción 2

- Creamos un proyecto nuevo (diferente del G
  - 1. Añadimos la librería jar (1) como referenciada
  - 2. Añadimos los dos directorios de recursos (2), directamente sobre el raíz del proyecto
  - 3. Añadimos la clase del “Practica\_XX\_exe.java” al directorio (paquete) que deseemos
  - 4. Corregimos los fallos de directorios.



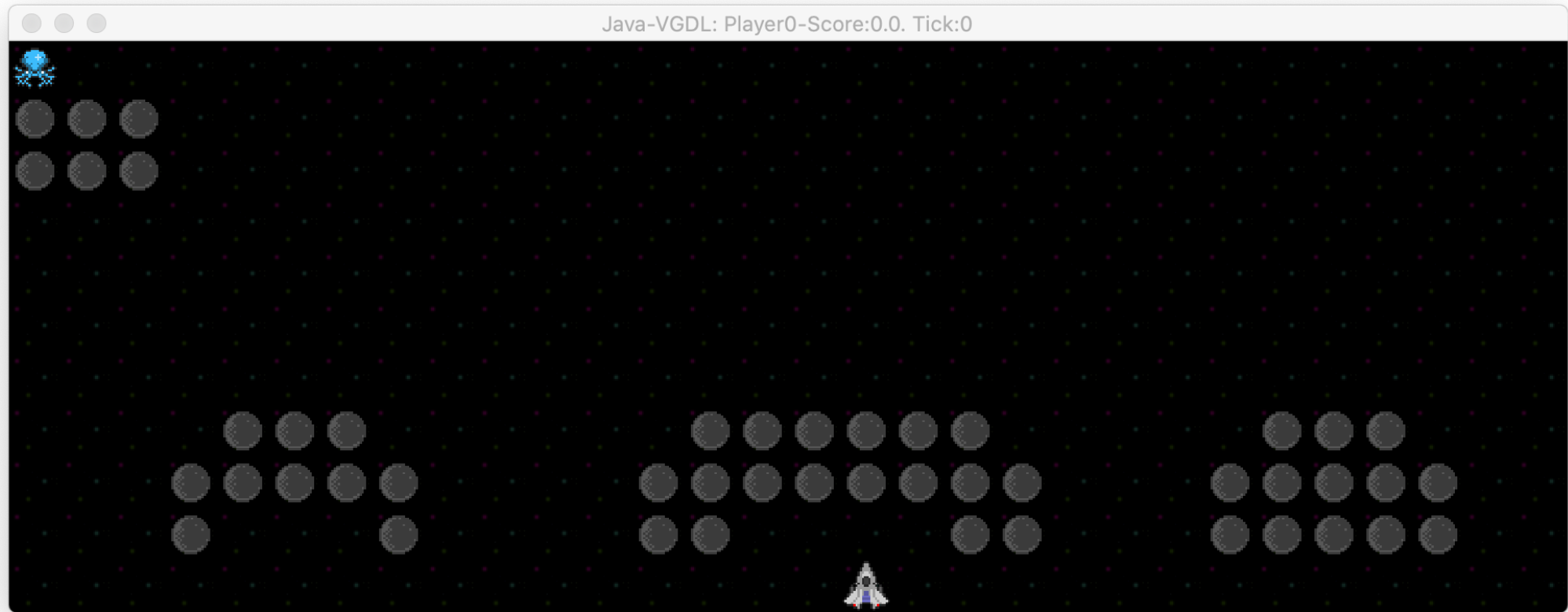
# Ejecución

- Para la ejecución de la plataforma:
  - Vamos a Run Configurations
  - Elegimos como clase main: Practica\_XX\_exe



# Ejecución

- Ya deberíamos de tener funcionando la plataforma.



El jugador básico

# El jugador básico: Esquema

- Para la creación del jugador Básico, tendremos que
  1. extender la clase abstracta: `core.player.AbstractPlayer`
  2. Constructor
  3. Actuador

```
package p0.gonzaloarandadti.a0;

import core.game.StateObservation;
import core.player.AbstractPlayer;
import ontology.Types;
import tools.ElapsedCpuTimer;

public class Practica_00_vacio extends AbstractPlayer {

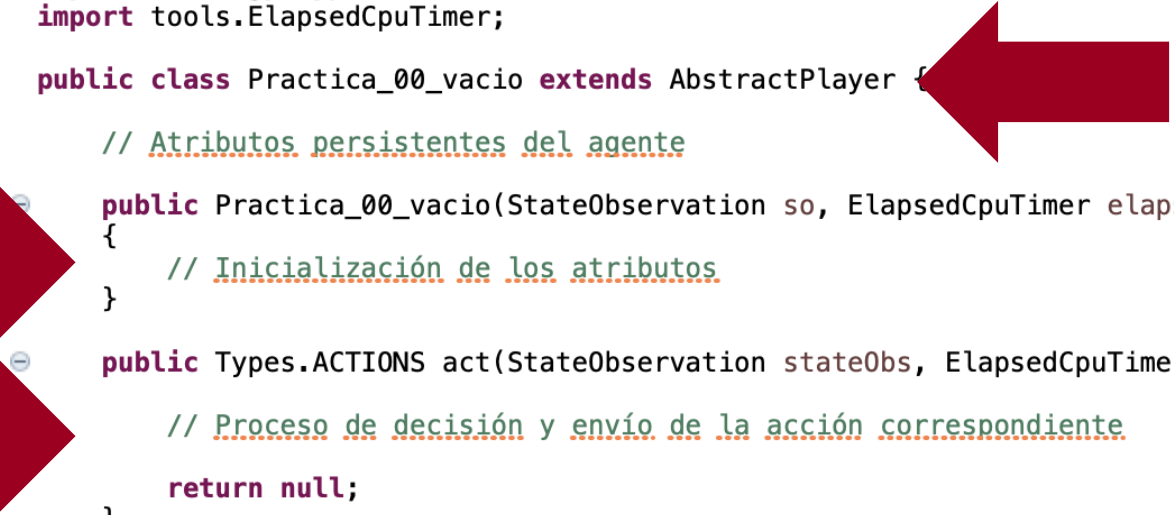
    // Atributos persistentes del agente

    public Practica_00_vacio(StateObservation so, ElapsedCpuTimer elapsedTimer)
    {
        // Inicialización de los atributos
    }

    public Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer) {

        // Proceso de decisión y envío de la acción correspondiente

        return null;
    }
}
```



# El jugador básico: Ejecución

## Pasos:

1. Nombre del jugador implementado con paquetes.
2. Con Entorno gráfico o no
3. Elegir número de juego
4. Y nivel
5. Juego: Modo humano
6. Juego: Modo nuestro jugador

## Practica XX exe.java

```

1 package p0.gonzaloarandatti;
2
3 import java.util.Random;
4
5 public class Practica_XX_exe {
6
7     public static void main(String[] args) {
8
9         String p0 = "p0.gonzaloarandatti.a.Practica_00_aleatorio";
10
11         //Load available games
12         String spGamesCollection = "examples/all_games_sp.csv";
13         String[][] games = Utils.readGames(spGamesCollection);
14
15         //Game settings
16         boolean visuals = true;
17         int seed = new Random().nextInt();
18
19         // Game and level to play
20         int gameIdX = 0;
21         int levelIdx = 0;
22
23         String gameId = games[gameIdX][0];
24         String gameIdN = gameId.substring(4, gameId.length() - 4);
25         String gameIdLvlN = gameIdN + gameIdLvlN.txt).
26
27         String gameIdN = gameIdN.substring(0, gameIdN.length() - 1);
28         String gameIdLvl = gameIdN + gameIdLvlN;
29         String gameIdLvlN = gameIdLvl + gameIdLvlN;
30
31         // 1. This starts a game, in a level, played by a human.
32         ArcadeMachine.playOneGame(game, gameIdLvl, null, seed);
33
34         // 2. This plays a game in a level by the controller.
35         ArcadeMachine.runOneGame(game, gameIdLvl, visuals, p0, null, seed, 0);
36
37         System.exit(0);
38     }
39 }
40
41 }
42

```



# El jugador básico: Ejecución

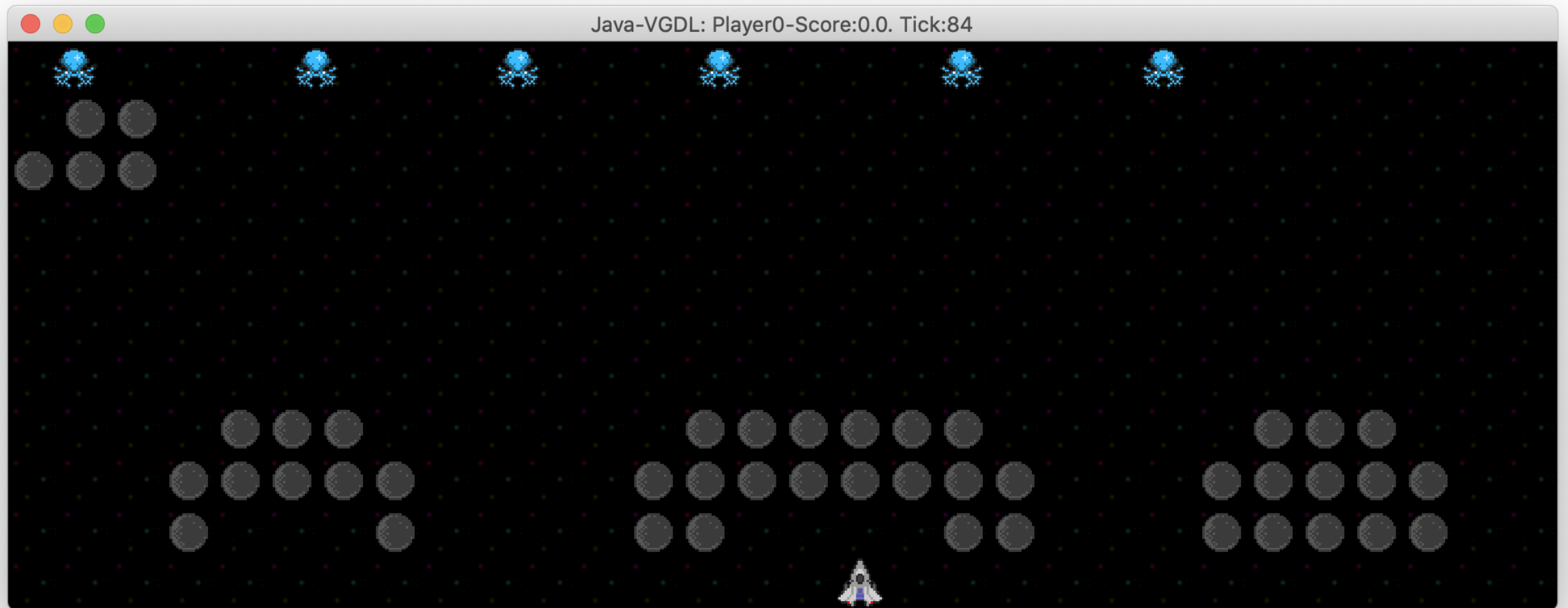
Pasos:

RUN!! (as Java Application)

Practica\_XX\_exe.java

```
Practica_XX_exe.java
1 package p0.gonzaloarandadti;
2
3 import java.util.Random;
4
5 public class Practica_XX_exe {
6
7     public static void main(String[] args) {
8
9         String p0 = "p0.gonzaloarandadti.a.Practica_00_aleatorio";
10
11         //Load available games
12         String spGamesCollection = "examples/all_games_sp.csv";
13         String[][] games = Utils.readGames(spGamesCollection);
14
15         //Game settings
16         boolean visuals = true;
17         int seed = new Random().nextInt();
18
19         // Game and level to play
20         int gameIdx = 0;
21         int levelIdx = 0; // level names from 0 to 4 (game_lv1N.txt).
22
23         String gameName = games[gameIdx][1];
24         String game = games[gameIdx][0];
25         String level1 = game.replace(gameName, gameName + "_lv1" + levelIdx);
26
27         // 1. This starts a game, in a level, played by a human.
28         ArcadeMachine.playOneGame(game, level1, null, seed);
29
30         // 2. This plays a game in a level by the controller.
31         // ArcadeMachine.runOneGame(game, level1, visuals, p0, null, seed, 0);
32
33         System.exit(0);
34     }
35 }
```

# El jugador básico



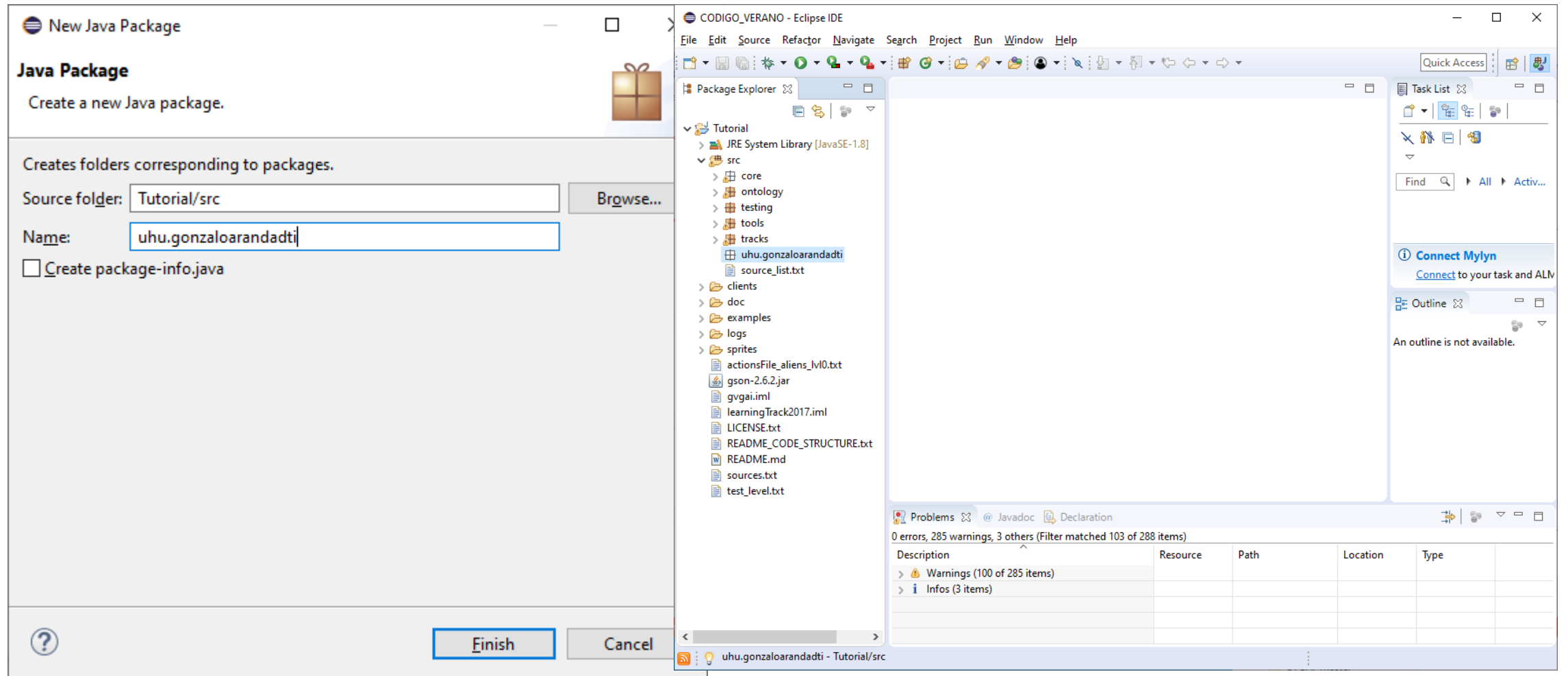
# Ejercicio 1

- Arrancar, con un jugador vacío, el juego 50 y el nivel 3.

# Jugador Prácticas

- Los distintos jugadores que vamos a usar durante las prácticas van a tener TODOS la misma forma de construir el nombre.
  - Nombre: si2022.pXX.nombreusuario.YYYYYYYYYY
  - donde:
    - XX es el número de práctica
    - nombreusuario: el usuario de vuestras aplicaciones UHU PERO SIN PUNTOS.
      - Ej: mi usuario es: gonzaloarandadti

# Primer controlador



# Ejercicio 2

- Arreglar el ejercicio 1 para que se ajuste a la nomenclatura de la asignatura

# Ejercicio 3

- Crear un usuario que envíe una acción aleatoria

# Crear Jugador aleatorio

```
Agente.java
1 package uhu.gonzaloarandadi;
2
3 import java.util.ArrayList;
4 import java.util.Random;
5 import core.game.StateObservation;
6 import core.player.AbstractPlayer;
7 import ontology.Types;
8 import ontology.Types.ACTIONS;
9 import tools.ElapsedCpuTimer;
10
11 /**
12  * @author GonzaloAAranda
13  *
14  */
15 public class Agente extends AbstractPlayer {
16
17     private Random randomGenerator;
18     ArrayList<Types.ACTIONS> actions = null;
19
20     /**
21      * initialize all variables for the agent
22      * @param stateObs Observation of the current state.
23      * @param elapsedTimer Timer when the action returned is due.
24      */
25     public Agente(StateObservation stateObs, ElapsedCpuTimer elapsedTimer){
26         randomGenerator = new Random();
27         actions = stateObs.getAvailableActions();
28     }
29
30     /**
31      *
32      */
33     @Override
34     public ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer) {
35         Types.ACTIONS action = Types.ACTIONS.ACTION_NIL;
36         int index = randomGenerator.nextInt(actions.size());
37         action = actions.get(index);
38         return action;
39     }
40 }
41
42
43
44
```



## Ejercicio 4

- Crear un jugador que envíe la cadena de acciones correctas para resolver el problema.