UNIVERSIDAD DE GRANADA

DECSAI

Técnicas de los Sistemas Inteligentes

Grado en Informática

Curso 2021-22. Práctica 1
Técnicas de Búsqueda Heurística

Jesús Giráldez Crú y Pablo Mesejo Santiago

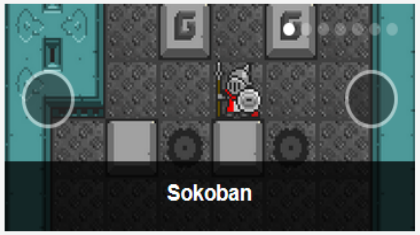
Departamento de Ciencias de la Computación e Inteligencia Artificial

http://decsai.ugr.es

General Video Game AI (GVGAI)

- Website: http://www.gvgai.net/
- GVGAI: una competición para agentes de IA para
  - Jugar a videojuegos
  - Generación de contenido (mapas de niveles y reglas de juego)
- Los agentes desarrollados se evalúan sobre videojuegos no vistos previamente
- +160 Videojuegos
- Los juegos se describen en el lenguaje VGDL (Video Game Description Language)

Schaul, T., 2013, August. A video game description language for model-based or interactive learning. In 2013 IEEE Conference on Computational Inteligence in Games (CIG) (pp. 1-8). IEEE.



Sponsored by: DeepMind

Welcome to the General Video Game AI Competition webpage. The **GVG-AI Competition** explores the problem of creating controllers for general video game playing. How would you create a single agent that is able to play any game it is given? Could you program an agent that is able to play a wide variety of games, without knowing which games are to be played? Can you create an automatic level generation that designs levels for any game is given?

In this website, you will be able to participate in the General Video Game AI Competition. You can now download the starter kit for the competition and submit your controller to be included in the rankings. For any question contact us.

Tiempo Real

- Tiempo límite reacción: 40 ms
- Si devuelves acción en [40,50] ms, se aplica NIL
- Si devuelves acción en > 50 ms, pierdes juego. Descalificado

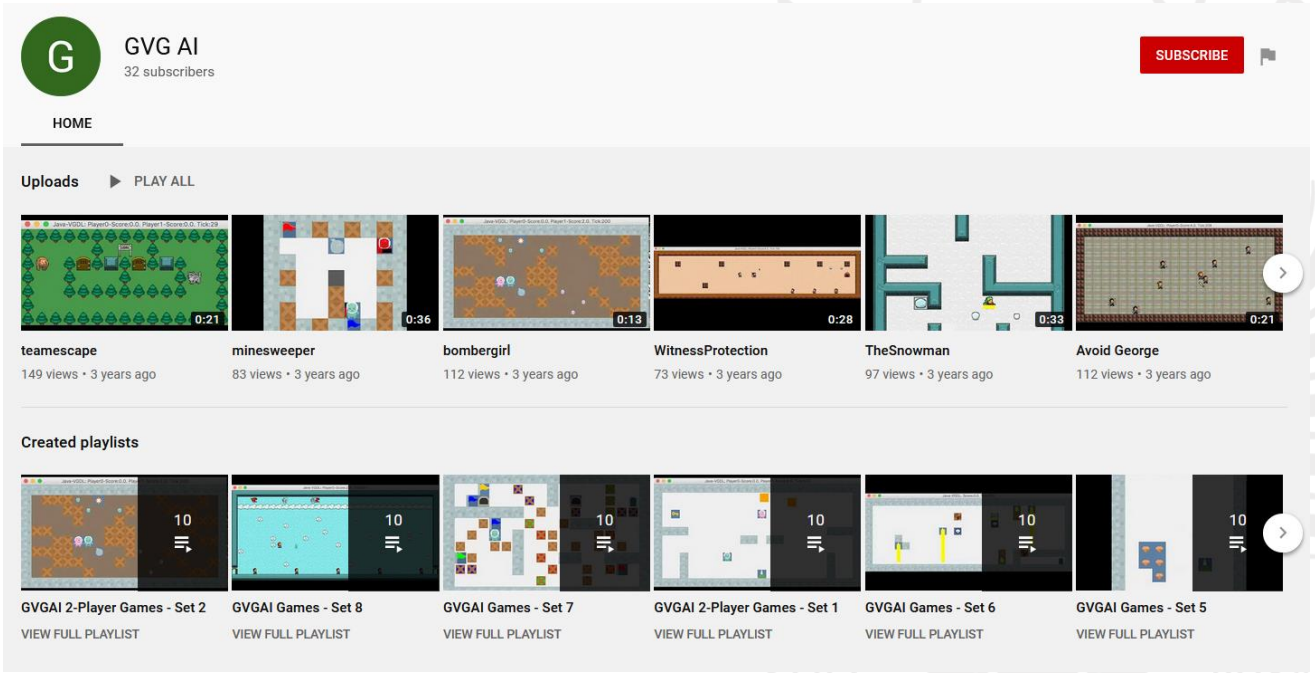
Descripciones de los videojuegos y vídeos de ejemplo

GVGAI CHANNEL

https://www.youtube.com/channel/UCMFCfXipQT55IK6R504naUQ

Tipos de juegos

- Un jugador / varios
- Total / parcialmente observables
- Puzzles vs Carreras
- Acción / Aventuras
- Recolectar, disparar, navegar
- Cooperativos / Competitivos



Varios Tracks: está concebido para probar distintas técnicas de IA en videojuegos tanto para jugar a videojuegos como para generar contenido.

Jugar

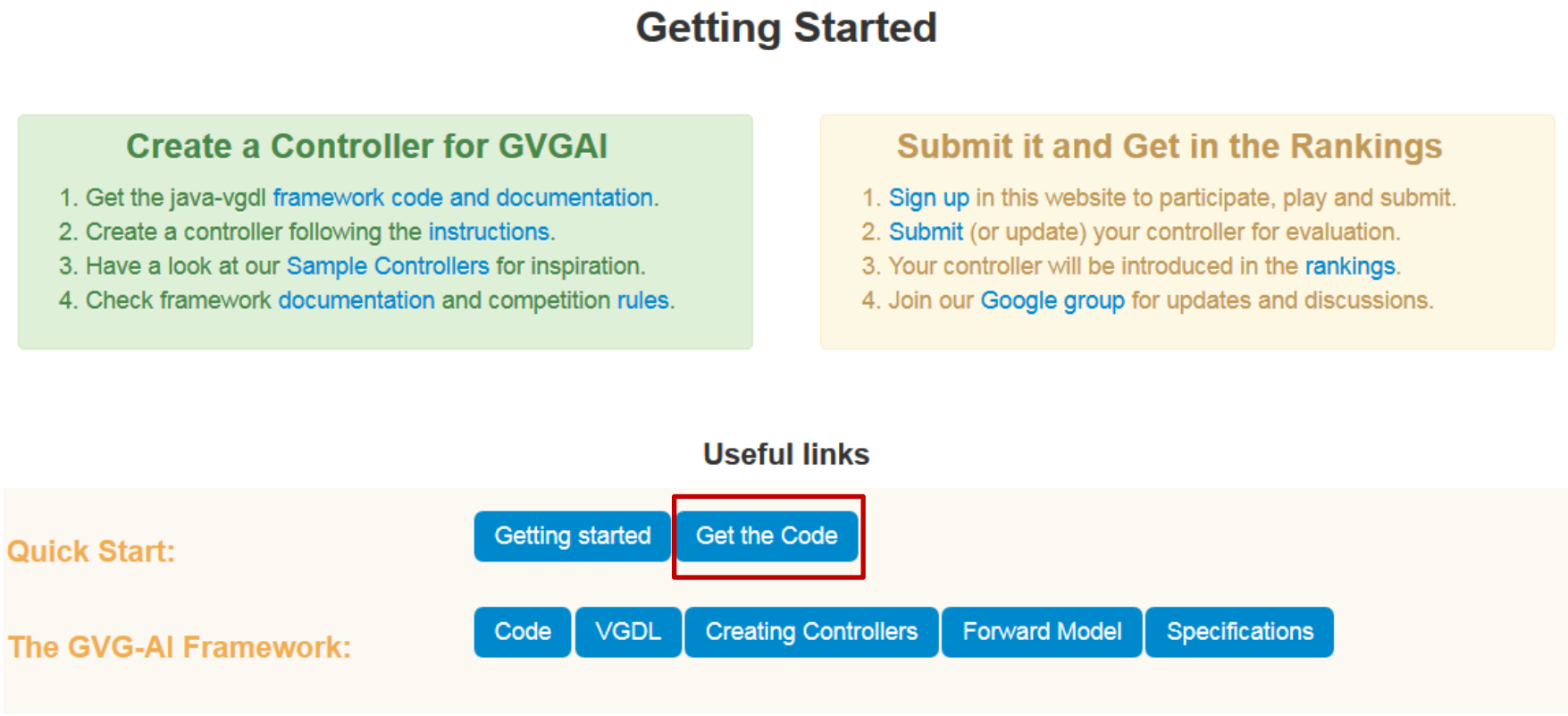
- **Single Player Track**
- Multiple Player Track
- Single Player Learning Track

Generar contenido

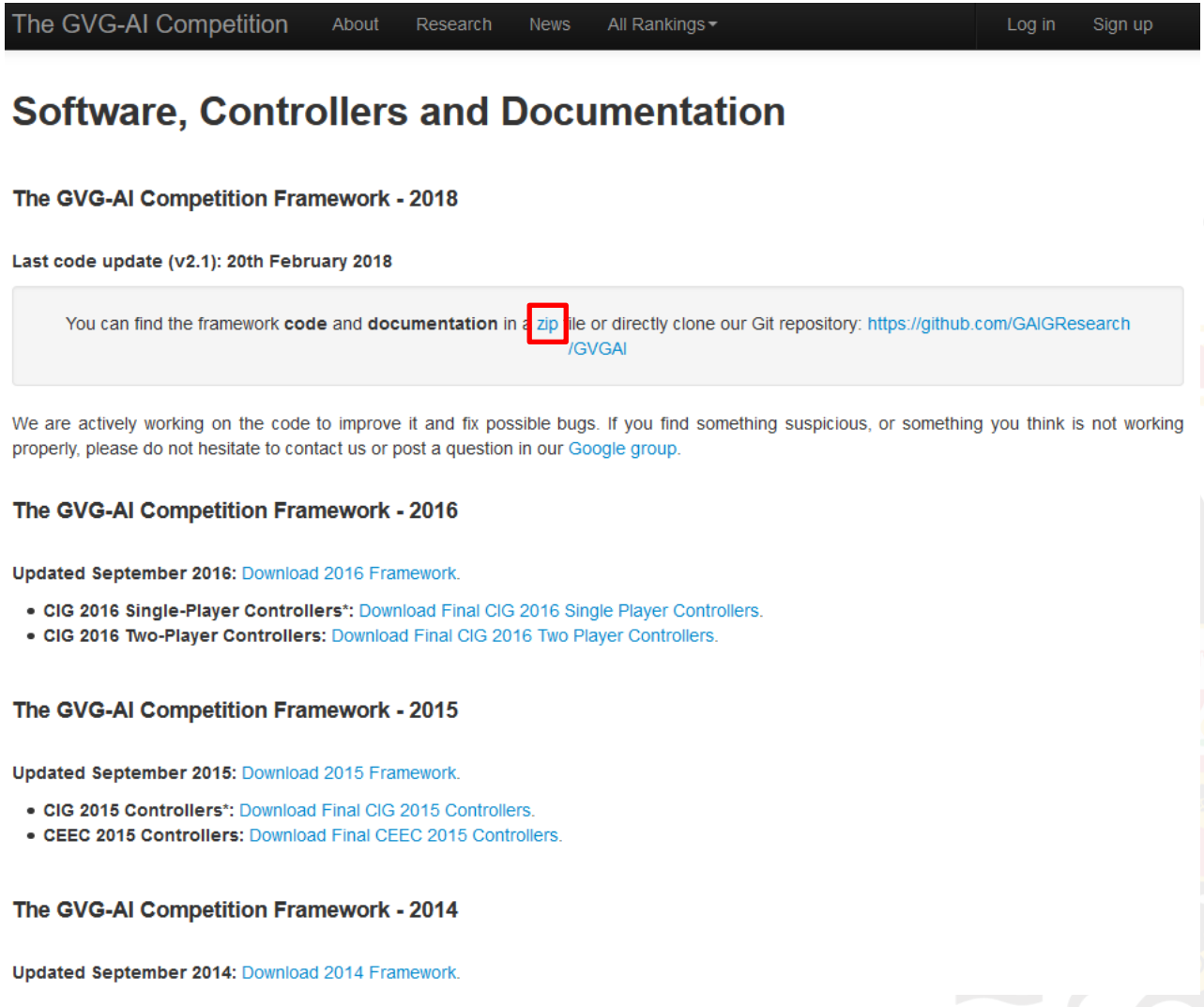
- Level Generation Track
- Rule Generation Track
- GameDesign Track

Instalación del GVG Framework

- Descargar el zip de http://www.gvgai.net/ pinchando en Useful links > Get The Code
  - https://github.com/GAIGResearch/GVGAI/archive/master.zip

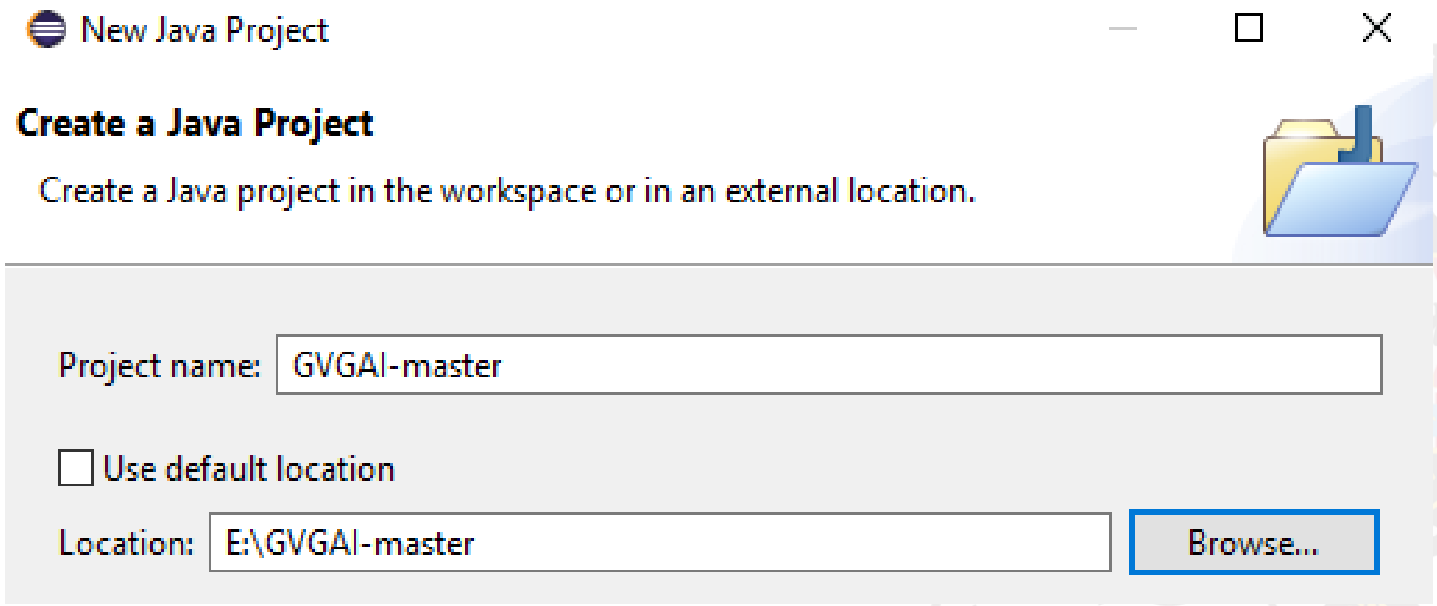


- Descargar el zip de http://www.gvgai.net/ pinchando en Useful links > Get The Code
  - https://github.com/GAIGResearch/GVGAI/archive/master.zip

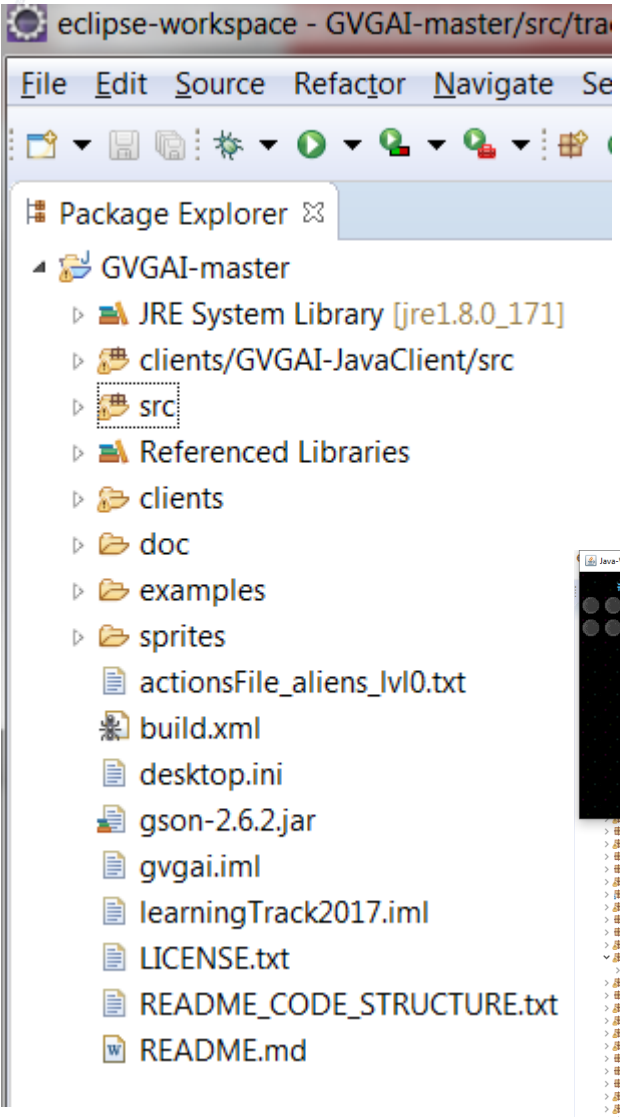


Instalación del GVG Framework con ECLIPSE
(hay que tener instalado Java y Eclipse)

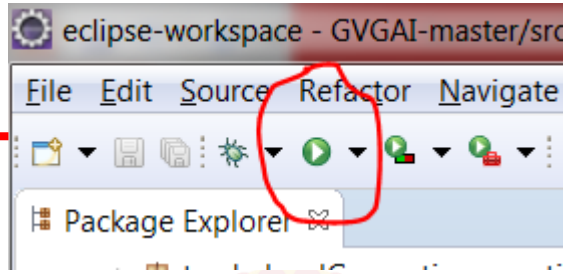
- Descomprimir en un <directorio>
- Abrir Eclipse.
- File → New → Project… → New Java project
- Untick 'default location' y seleccionar <directorio> en Location → Finish

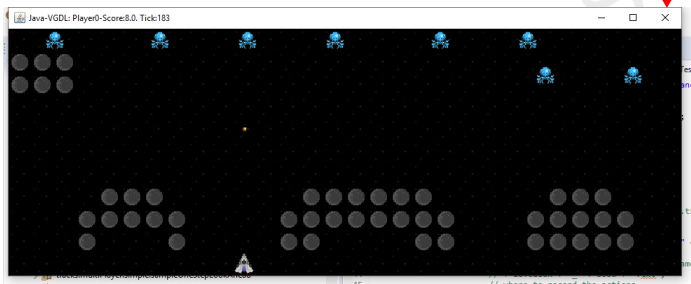


Ejecución del GVG Framework

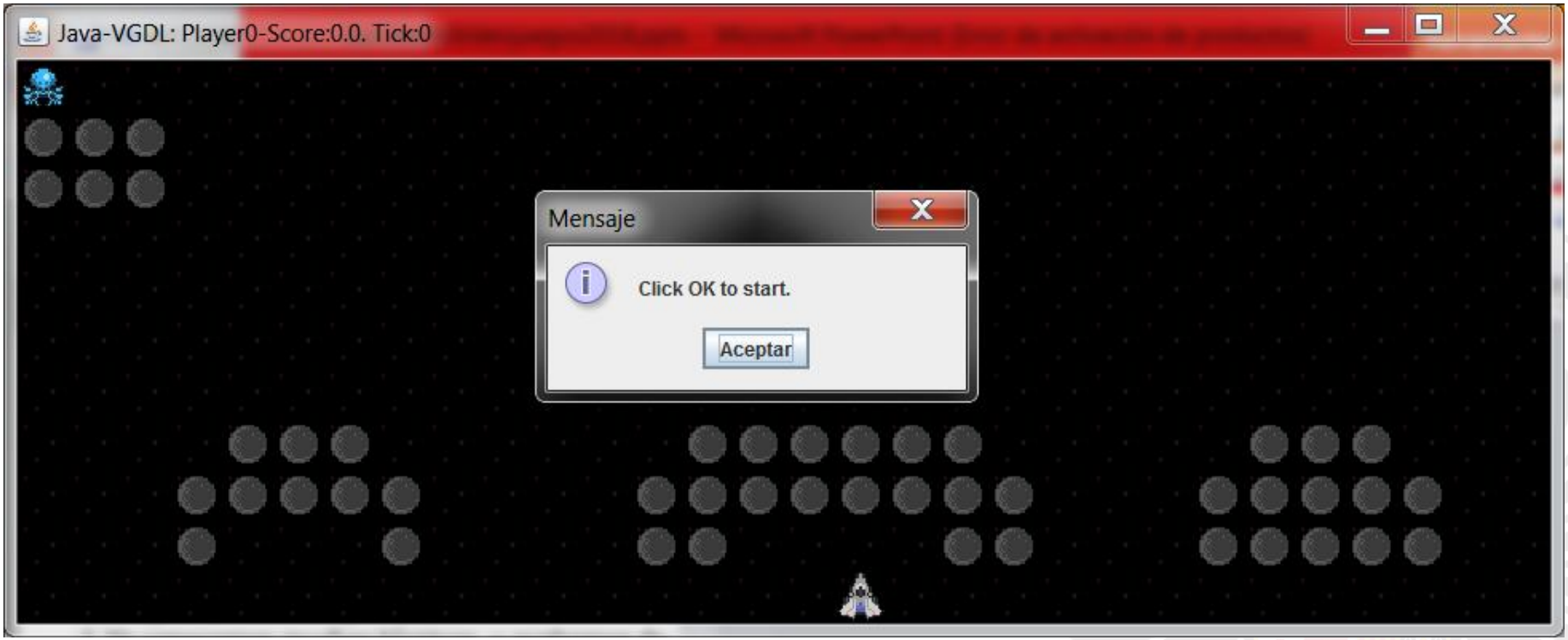


- En Package Explorer, navegar hasta
  - Src.tracks.singlePlayer → Test.java
  - Ejecutar Test.java





Ejecución del GVG Framework



- Por defecto el entorno de juego carga el juego "0" (gameIdx) y en modo jugador humano (ArcadeMachine.*playOneGame*).

La estructura del código y documentación básica está en
https://github.com/EssexUniversityMCTS/gvgai/wiki/Code-Structure

Modificación del GVG Framework
src.tracks.singlePlayer.Test.java

```
//Load available games
String spGamesCollection =  "examples/all_games_sp.csv";
String[][] games = Utils.readGames(spGamesCollection);

//Game settings
boolean visuals = true;
int seed = new Random().nextInt();

// Game and level to play
int gameIdx = 0;
int levelIdx = 0; // level names from 0 to 4 (game_lvlN.txt).
String gameName = games[gameIdx][1];
String game = games[gameIdx][0];
String level1 = game.replace(gameName, gameName + "_lvl" + levelIdx);

String recordActionsFile = null;// "actions_" + games[gameIdx] + "_lvl"
                // + levelIdx + "_" + seed + ".txt";
                // where to record the actions
                // executed. null if not to save.
```

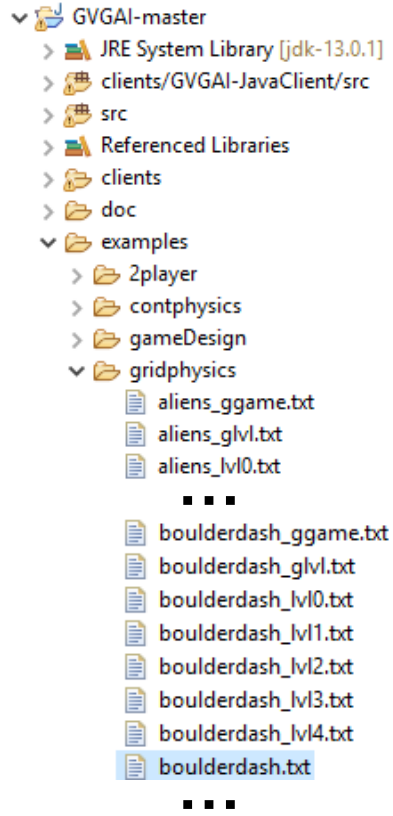
Consultar este fichero, Hay numerosos juegos disponibles para SinglePlayer

Cambiar este valor y ejecutar para ver otros juegos

¿Cómo podemos saber cómo se definen, por ejemplo, los mapas del juego BoulderDash?

(1) Abrir all_games_sp.csv → (2) Abrir la ruta relativa a boulderdash.txt

	A	B	C	D
1	0,examples/gridphysics/aliens.txt			
2	1,examples/gridphysics/angelsdemons.txt			
3	2,examples/gridphysics/assemblyline.txt			
4	3,examples/gridphysics/avoidgeorge.txt			
5	4,examples/gridphysics/bait.txt			
6	5,examples/gridphysics/beltmanager.txt			
7	6,examples/gridphysics/blacksmoke.txt			
8	7,examples/gridphysics/boloadventures.txt			
9	8,examples/gridphysics/bomber.txt			
10	9,examples/gridphysics/bomberman.txt			
11	10,examples/gridphysics/boulderchase.txt			
12	11,examples/gridphysics/boulderdash.txt			
13	12,examples/gridphysics/brainman.txt			
14	13,examples/gridphysics/butterflies.txt			
15	14,examples/gridphysics/cakybaky.txt			
16	15,examples/gridphysics/camelRace.txt			
17	16,examples/gridphysics/catapults.txt			
18	17,examples/gridphysics/chainreaction.txt			



(3) Ver en boulderdash.txt cómo se definen los objetos del mapa

```
BasicGame
    SpriteSet
        background > Immovable img=oryx/backBlack hidden=True
        wall > Immovable autotiling=true img=oryx/dirtWall
        sword > Flicker color=LIGHTGRAY limit=1 singleton=True img=oryx/pickaxe
        dirt > Immovable color=BROWN img=oryx/backLBrown
        exitdoor > Door color=GREEN img=oryx/door2
        diamond > Resource color=YELLOW limit=10 shrinkfactor=0.75 img=oryx/diamond3
        boulder > Missile orientation=DOWN color=GRAY speed=0.2 img=oryx/mineral1
        moving >
            avatar  > ShootAvatar   stype=sword frameRate=8 img=oryx/spelunky
            enemy > RandomNPC cons=1
                crab > color=RED img=oryx/scorpion2
                butterfly > color=PINK img=oryx/bat2

    LevelMapping
        . > background dirt
        - > background
        e > background exitdoor
        o > background boulder
        x > background diamond
        c > background crab
        b > background butterfly
        A > background avatar

    InteractionSet
        dirt avatar sword > killSprite
        diamond avatar > collectResource  scoreChange=2
        moving wall boulder > stepBack

        avatar boulder > killIfFromAbove scoreChange=-1
        avatar butterfly crab > killSprite scoreChange=-1

        boulder dirt wall diamond boulder > stepBack

        enemy dirt diamond > stepBack

        crab butterfly > killSprite
        butterfly crab > transformTo stype=diamond scoreChange=1
        exitdoor avatar > killIfOtherHasMore resource=diamond limit=9

    TerminationSet
        SpriteCounter stype=avatar limit=0 win=False
        SpriteCounter stype=exitdoor limit=0 win=True
```

¿Cómo podemos saber cómo se definen, por ejemplo, los mapas del juego BoulderDash?

(3) Ver en boulderdash.txt cómo se definen los objetos del mapa

```
LevelMapping
    . > background dirt
    - > background
    e > background exitdoor
    o > background boulder
    x > background diamond
    c > background crab
    b > background butterfly
    A > background avatar
```

(4) Ver el mapa del nivel 0 de BoulderDash y cómo se visualiza en el juego en la práctica

boulderdash_lvl0.txt

```
1  wwwwwwwwwwwwwwwwwwwwwwwwww
2  w...o.xx.o......o..xoxx..w
3  w...oooooo........o..o...w
4  w....xxx.........o.oxoo.ow
5  wx..............oxo...oow
6  wwwwwwwww........o...wxxw
7  wb-...co..............wxxw
8  w--........Ao....o....wxxw
9  wooo.............-....w..w
10 w......x....wwww-x.oow..w
11 wc--.....x..ooxxo-....w..w
12 w---..e..........b------..w
13 wwwwwwwwwwwwwwwwwwwwwwwwww
```

Single Player Planning: GVGAI proporciona agentes básicos para jugar (llamados "controllers") que pueden intercambiarse fácilmente.

```
// Available tracks:
String sampleRandomController = "tracks.singlePlayer.simple.sampleRandom.Agent";
String doNothingController = "tracks.singlePlayer.simple.doNothing.Agent";
String sampleOneStepController = "tracks.singlePlayer.simple.sampleonesteplookahead.Agent";
String sampleFlatMCTSController = "tracks.singlePlayer.simple.greedyTreeSearch.Agent";
                                  tracks.singlePlayer.evaluacion.blanca
String sampleMCTSController = "tracks.singlePlayer.advanced.sampleMCTS.Agent";
String sampleRSController = "tracks.singlePlayer.advanced.sampleRS.Agent";
String sampleRHEAController = "tracks.singlePlayer.advanced.sampleRHEA.Agent";
String sampleOLETSController = "tracks.singlePlayer.advanced.olets.Agent";
```

- Técnicas:
  - Aleatorio
  - No hacer nada
  - Avanzar una etapa (greedy)
  - Algoritmo genético
  - MCTS (Monte Carlo Tree Search)
  - ---

Experimentar con un agente deliberativo (single player planning agent)

1. Encontrar la clase src.tracks.singlePlayer.Test.java

2. Asignar `gameIdx` y `levelIdx` con distintos valores

3. Seleccionar modo de ejecución:
   1. Jugar como humano:

```
// 1. This starts a game, in a level, played by a human.
ArcadeMachine.playOneGame(game, level1, recordActionsFile, seed);
```

   2. Jugar como IA (con un "controller"): Usar (o crear) una variable String conteniendo el path de la clase de un agente (pre)diseñado.

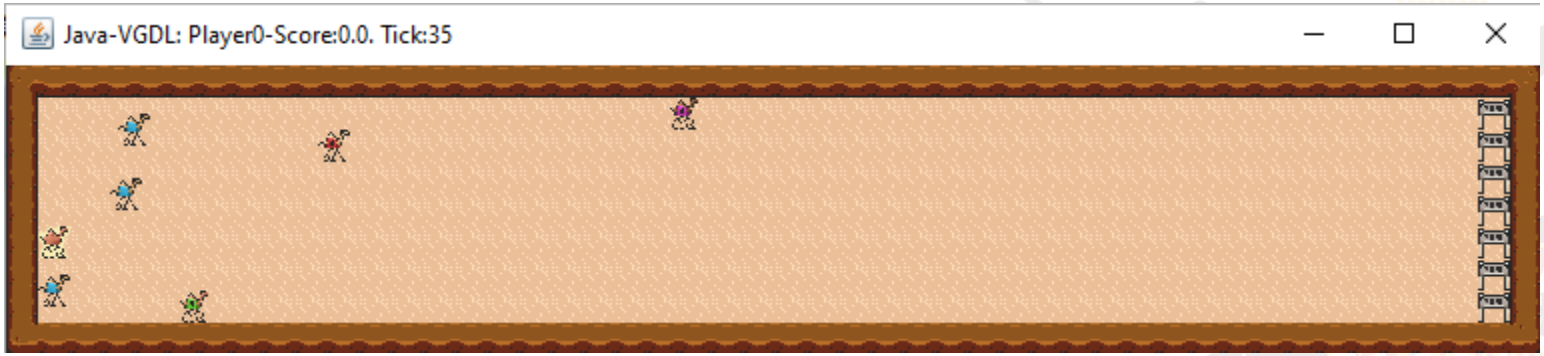
```
String sampleMCTSController =
"tracks.singlePlayer.advanced.sampleMCTS.Agent";
...
// 2. This plays a game in a level by the controller.
ArcadeMachine.runOneGame(game, level1, visuals,
sampleMCTSController, recordActionsFile, seed, 0);
```

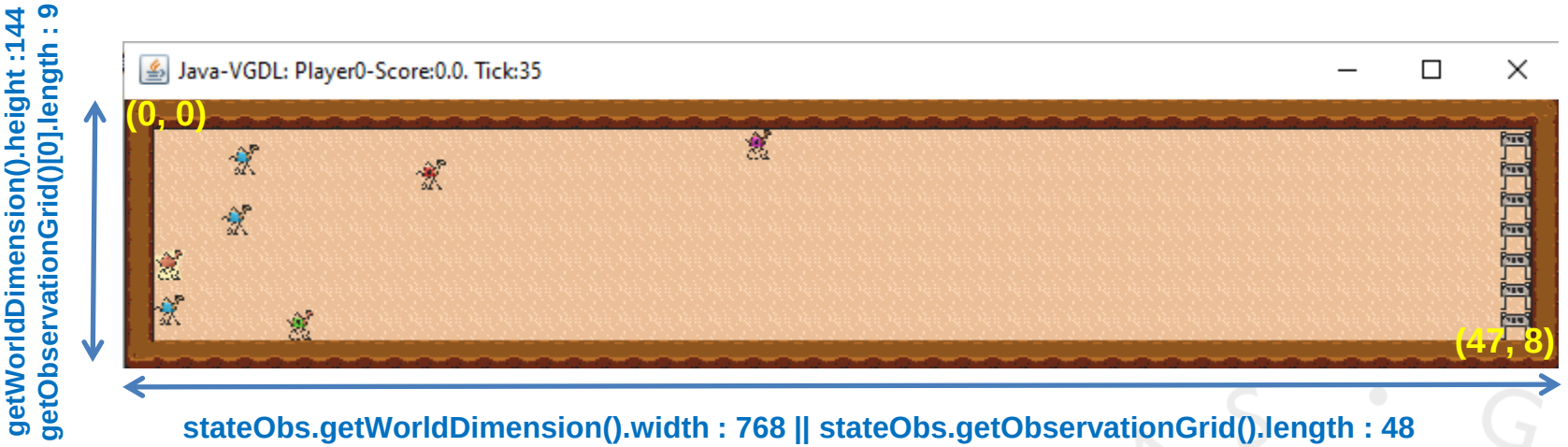
parámetro a cambiar para el controlador

4. Run Test.java

Escogemos el juego de **Carrera de Camellos** (int gameIdx = 15) y el **primer nivel de juego** (int levelIdx = 0).

Queremos que el agente escoja el portal de salida más cercano, y luego, de modo voraz, escoja la acción que le acerque más al portal. Es decir, queremos escoger en todo momento, y de modo inmediato, la acción más conveniente para alcanzar el objetivo de llegar al destino y ganar la carrera.






```
Vector2d fescala;
Vector2d portal;

/**
 * initialize all variables for the agent
 * @param stateObs Observation of the current state.
 * @param elapsedTimer Timer when the action returned is due.
 */
public myAgent_Camel(StateObservation stateObs, ElapsedCpuTimer elapsedTimer){
    //Calculamos el factor de escala entre mundos (pixeles -> grid)
    fescala = new Vector2d(stateObs.getWorldDimension().width / stateObs.getObservationGrid().length ,
            stateObs.getWorldDimension().height / stateObs.getObservationGrid()[0].length);

    //Se crea una lista de observaciones de portales, ordenada por cercania al avatar
    ArrayList<Observation>[] posiciones = stateObs.getPortalsPositions(stateObs.getAvatarPosition());
    //Seleccionamos el portal mas proximo
    portal = posiciones[0].get(0).position;
    portal.x = Math.floor(portal.x / fescala.x);
    portal.y = Math.floor(portal.y / fescala.y);
}
```

Píxeles del tablero/mundo

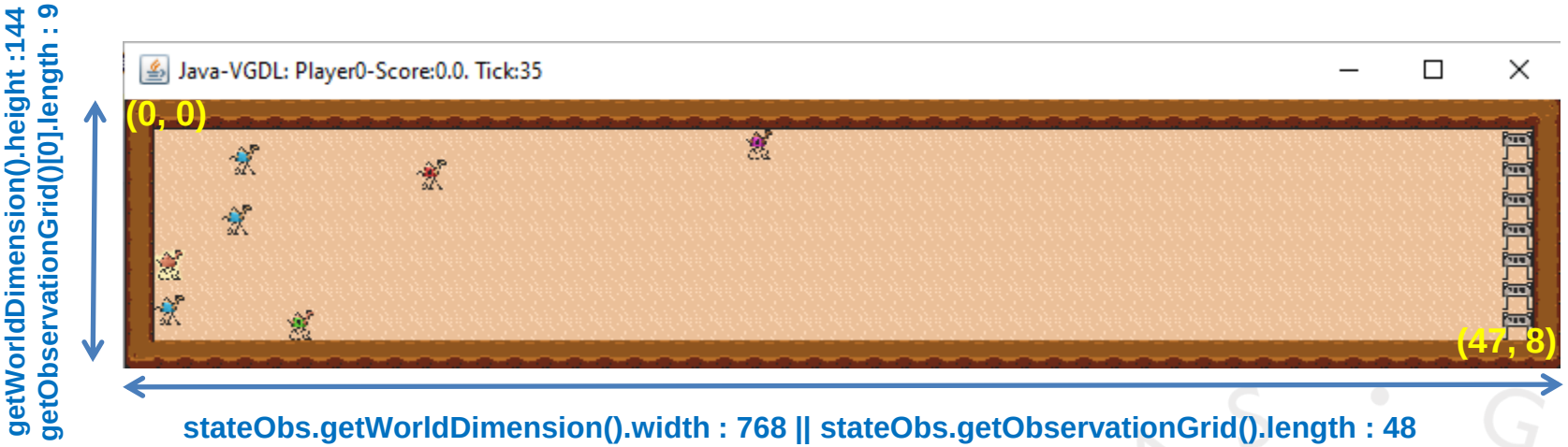
Posiciones del grid

devuelve ordenada por cercanía

Posición de los portales en coordenadas píxel

Posición del avatar en coordenadas píxel

Del primer tipo de portal (podría haber varios) cogemos el primer elemento



Para obtener la posición de los enemigos, si estos existiesen

esta es la función a usar

Para obtener, por ejemplo, la posición de las gemas si se usase Boulderdash

Del mismo modo que podemos acceder a los portales, podemos acceder a todo tipo de recursos/objetos:

Función	Descripción
getNPCPositions	Devuelve la lista de posiciones de los NPCs
getMovablePositions	Devuelve la lista de posiciones de objetos móviles
getImmovablePositions	Devuelve la lista de posiciones de objetos inmóviles
getResourcesPositions	Devuelve la lista de posiciones de recursos
getPortalsPositions	Devuelve la lista de posiciones de los portales

Nota: Revisad el tutorial de GVG-AI que adjuntamos con la práctica. Contiene esta información y mucha más.

Ejemplo Código

CAMEL RACE

```
@Override
public ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer) {
    //Posicion del avatar
    Vector2d avatar =  new Vector2d(stateObs.getAvatarPosition().x / fescala.x,
            stateObs.getAvatarPosition().y / fescala.y);

    //Probamos las cuatro acciones y calculamos la distancia del nuevo estado al portal.
    Vector2d newPos_up = avatar, newPos_down = avatar, newPos_left = avatar, newPos_right = avatar;
    if (avatar.y - 1 >= 0) {
        newPos_up = new Vector2d(avatar.x, avatar.y-1);
    }
    if (avatar.y + 1 <= stateObs.getObservationGrid()[0].length-1) {
        newPos_down = new Vector2d(avatar.x, avatar.y+1);
    }
    if (avatar.x - 1 >= 0) {
        newPos_left = new Vector2d(avatar.x - 1, avatar.y);
    }
    if (avatar.x + 1 <= stateObs.getObservationGrid().length - 1) {
        newPos_right = new Vector2d(avatar.x + 1, avatar.y);
    }

    //Manhattan distance
    ArrayList<Integer> distances = new ArrayList<Integer>();
    distances.add((int) (Math.abs(newPos_up.x - portal.x) + Math.abs(newPos_up.y-portal.y)));
    distances.add((int) (Math.abs(newPos_down.x - portal.x) + Math.abs(newPos_down.y-portal.y)));
    distances.add((int) (Math.abs(newPos_left.x - portal.x) + Math.abs(newPos_left.y-portal.y)));
    distances.add((int) (Math.abs(newPos_right.x - portal.x) + Math.abs(newPos_right.y-portal.y)));

    // Nos quedamos con el menor y tomamos esa accion.
    int minIndex = distances.indexOf(Collections.min(distances));
    switch (minIndex) {
        case 0:
            return Types.ACTIONS.ACTION_UP;
        case 1:
            return Types.ACTIONS.ACTION_DOWN;
        case 2:
            return Types.ACTIONS.ACTION_LEFT;
        case 3:
            return Types.ACTIONS.ACTION_RIGHT;
        default:
            return Types.ACTIONS.ACTION_NIL;
    }

}
```

En esencia:

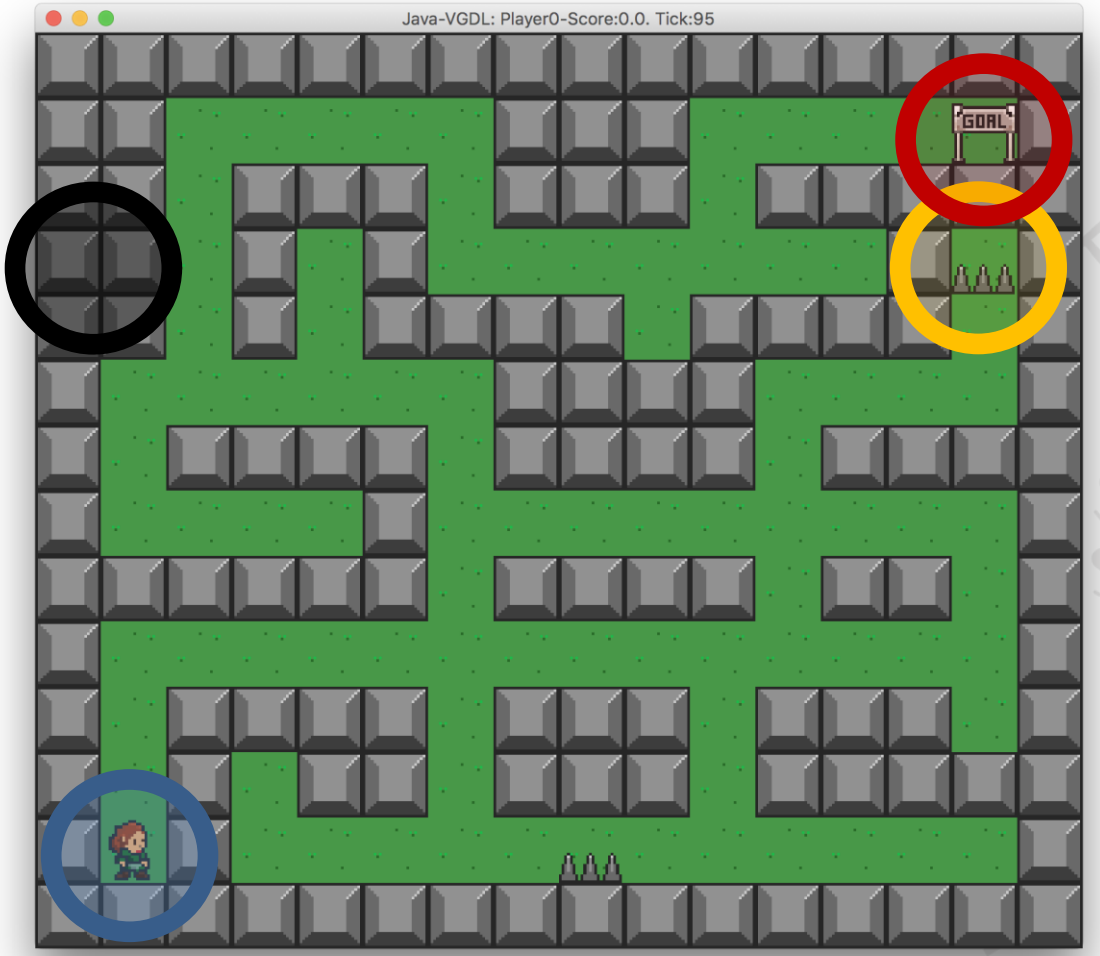
1) Calculamos la posición del avatar en coordenadas grid.

2) Vemos a dónde se movería con todas y cada una de las 4 acciones posibles.

3) Calculamos la distancia de las nuevas posiciones al portal.

4) Escogemos la acción asociada con la distancia más corta. Es decir, escogemos la acción que nos acerca de modo inmediato al objetivo.

La Práctica 1 consiste en **desarrollar cinco controladores basados en técnicas deliberativas de búsqueda dentro del entorno GVGAI que guíen a un avatar a resolver** un juego en distintos niveles. El juego escogido es el juego con índice 58 en los tipos de juego "singleplayer", que se pueden encontrar en el fichero "examples/all_games_sp.csv" de la distribución de GVGAI, denominado **Labyrinth.**



4 acciones:
- Izquierda
- Derecha
- Arriba
- Abajo

4 tipos de casilla:
- **Avatar**
- **Objetivo**
- **Trampas**
- **Muros**

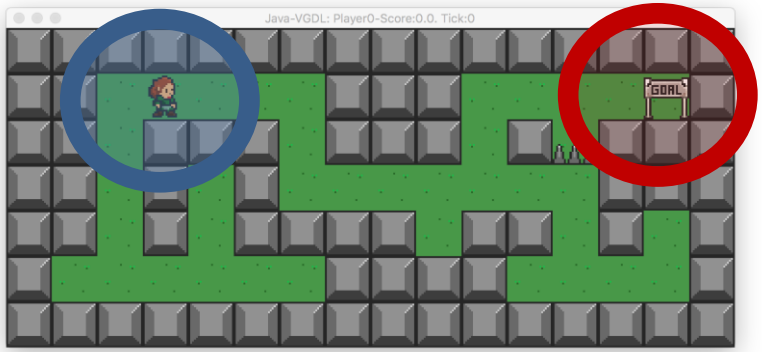
El objetivo de la práctica es que los estudiantes se familiaricen con las **técnicas deliberativas de búsqueda**.

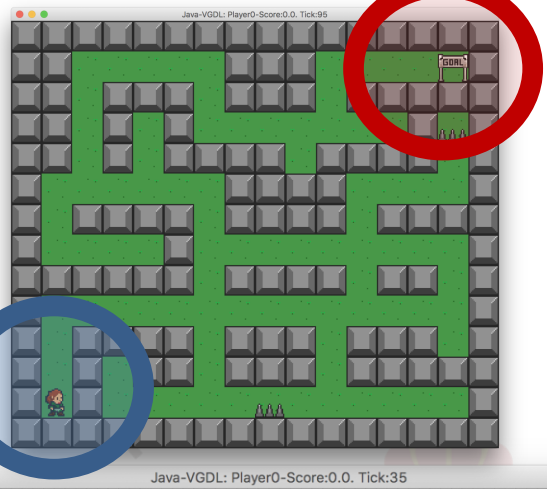
Para ello, el avatar deberá resolver 4 mapas de progresiva dificultad con los siguientes algoritmos:

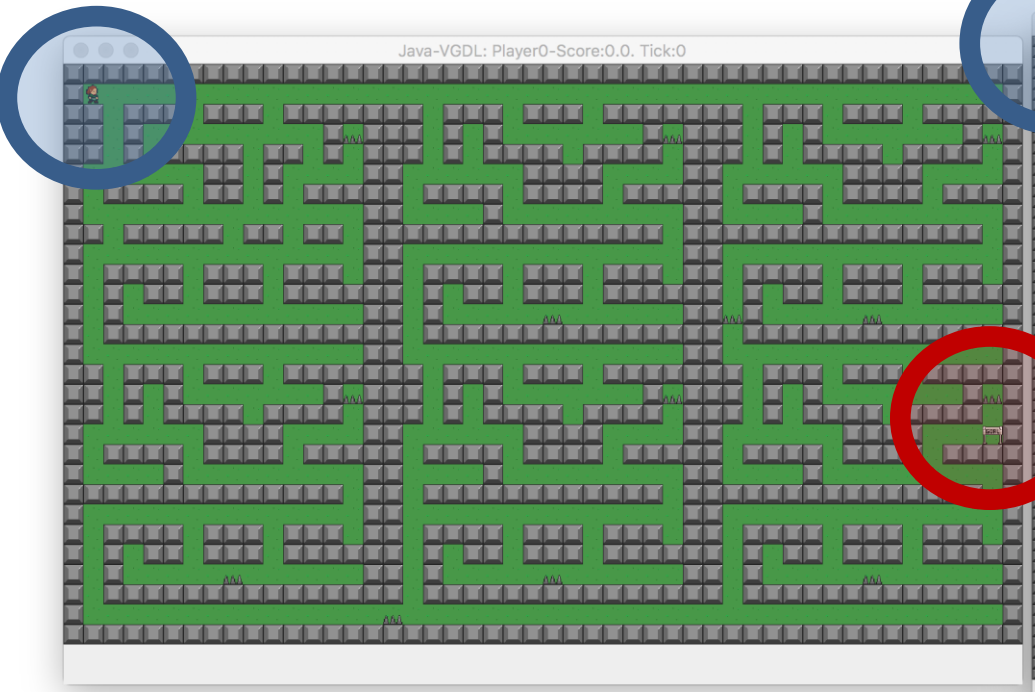
1. **Búsqueda en anchura (BFS)**
2. **Búsqueda en profundidad (DFS)**
3. **Búsqueda heurística (offline) con A***
4. **Búsqueda heurística (offline) con memoria acotada con IDA***
5. **Búsqueda heurística (online) en tiempo real con RTA***

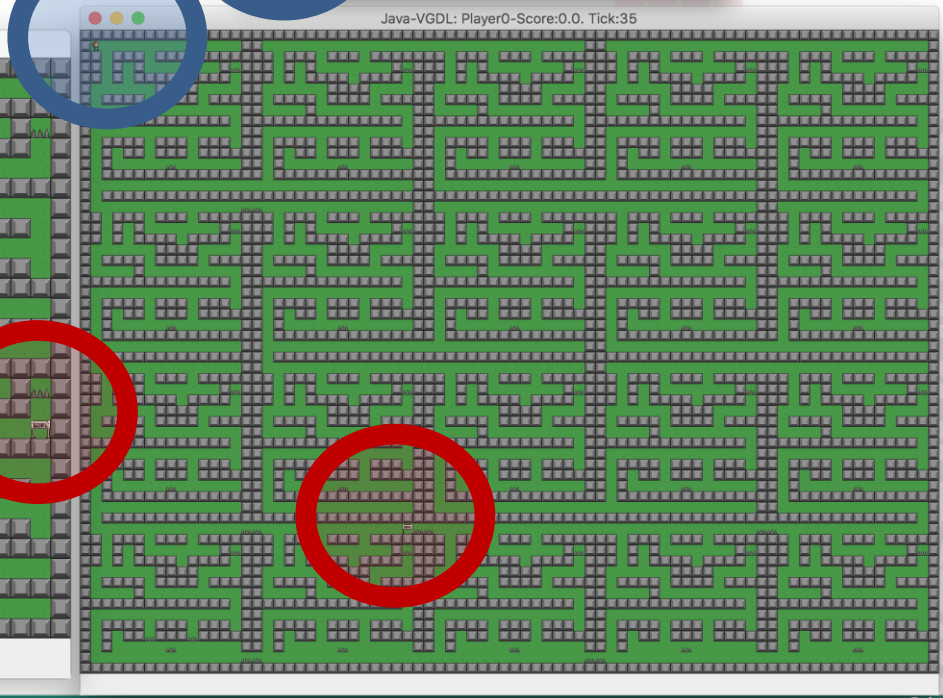
Todos estos algoritmos se repasarán en las próximas sesiones de prácticas

Los mapas son proporcionados por los profesores de prácticas, aunque se recomienda encarecidamente que los estudiantes creen sus propios mapas y verifiquen el comportamiento de sus algoritmos en ellos.









- La **temporización sugerida** para este práctica:

  - *Semana 14 de Marzo:* Presentación Práctica 1
    - Instalación de GVGAI, revisar materiales, ejecutar códigos de prueba (Camel Race)
  - *Semana 21 de Marzo:* Búsqueda no informada y búsqueda heurística
    - Implementación de BFS, DFS y A*
  - *Semana 28 de Marzo*: Búsqueda online
    - Implementación de IDA* y RTA*
  - *Semana 04 de Abril:* Seguimiento/Dudas
    - Preparación de la memoria
  - *Semana del 11 de Abril – SEMANA SANTA*
    - **Entrega P1**: 17 de Abril de 2022 hasta las 23:59

- Para el desarrollo de la práctica, cada estudiante creará el siguiente **Java package**:
  - src/tracks/singlePlayer/evaluacion/src_APELLIDO1_APELLIDO2_NOMBRE

lo que se envía es el zip de la carpeta src_APELLI…

  - Los nombres de paquete son case-sensitive, por lo que:
    - Nombre y apellidos irán en mayúscula, sin espacios ni guiones, y separados por una barra baja

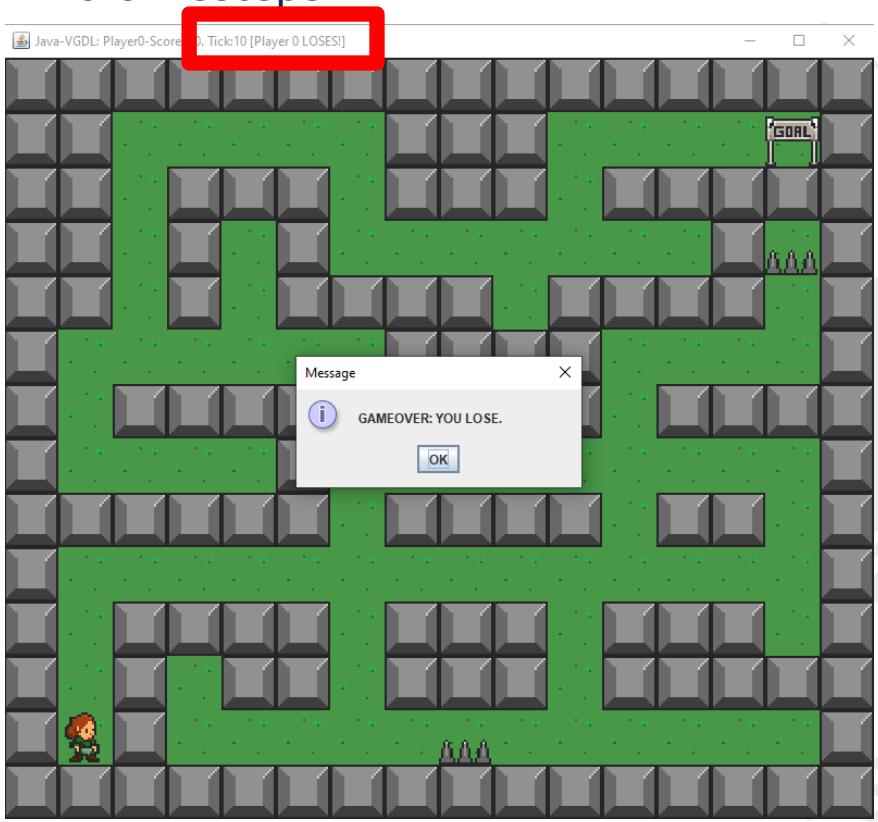
  - Dentro de este paquete se crearán, al menos, las siguientes **clases Java**:
    - AgenteBFS
    - AgenteDFS
    - AgenteAStar
    - AgenteIDAStar
    - AgenteRTAStar

A parte se puede incluir ALGORITMO.java y que este se llame en el agente.

  - Se pueden incluir otras clases auxiliares que sean utilizadas por las anteriores.
  - Los nombres de clase también son case-sensitive, por lo que se deben respetar las mayúsculas/minúsculas

Es **REQUISITO** de la práctica respetar esta nomenclatura!!!

- Para la correcta ejecución de todos los mapas y algoritmos, se debe **modificar el número máximo de timesteps**, tanto en la definición del juego como en el entorno de GVGAI.
  - Los timesteps (o ticks) representan el tiempo máximo permitido para jugar. Si se alcanza ese valor, y no se ha conseguido el objetivo del juego, se pierde la partida.
  - Ejemplo con 10 timesteps:



- El número de timesteps sugerido es de **10.000 timesteps**.

- Para modificarlo:
  - En examples/gridphysics/labyrinth.txt:
    - Timeout limit=**10000** win=False

  - En src/core/competition/CompetitionParameters.java:
    - public static final int MAX_TIMESTEPS = **10000**;

Con cambiarlo en uno basta, por seguridad cambiarlo en los dos

- Todos los algoritmos propuestos tienen un **comportamiento determinista**
  - Todas las **implementaciones correctas** darán los **mismos resultados** (mismo recorrido, mismo número de nodos expandidos, etc.), a excepción del tiempo de ejecución, que variará (ligeramente) de una máquina a otra

- Para ello **SIEMPRE** se **expandirán** los vecinos de un nodo en el siguiente **orden**:

  ARRIBA, ABAJO, IZQUIERDA, DERECHA

esto hace que el algoritmo sea determinista

  - Algunos algoritmos visitan los nodos siguiendo el orden de expansión. Ej: BFS o DFS
  - Otros algoritmos visitan los nodos usando información heurística
    - Por ejemplo, en A* los nodos se visitan según el valor de f(n), desempatando por g(n) ¿Y si también hay un empate en g(n)?
    - En la práctica **se usará el orden de expansión arriba indicado para cualquier desempate final**

- GVGAI **descalifica** al agente si:
  - El tiempo empleado en el **constructor** es mayor que 1s
  - El tiempo empleado en el **método "act"** es mayor que 50ms
    - Si el tiempo de act está en [40,50]ms, se ignora la acción calculada y realiza un ACTIONS.NIL

- En la práctica **NO se permite** el uso del **constructor** para **ninguna ejecución** relacionada con los **algoritmos de búsqueda**, todo el algoritmo de búsqueda se realizará en el método "act"
  - En el constructor sí se podría inicializar alguna variable (factor de escala, posición de la casilla objetivo, etc…) o tarea menor que no tenga relación con el algoritmo de búsqueda

- Por tanto, una **estructura de datos eficiente** es crucial para el correcto funcionamiento de los agentes

a estrella priority que ordenada por la izquierda

- Pasos a seguir:
  1. Descargar e instalar el entorno GVGAI.
  2. Seguir las indicaciones anteriores para probar varios juegos y niveles.
  3. Consultar y revisar los materiales proporcionados con la práctica:
     - Esta presentación
     - Enunciado de la práctica
     - Tutorial sobre GVGAI texto
     - Consultar la documentación sobre el código en caso de que sea necesario.
       - *La estructura del código y documentación básica está en https://github.com/EssexUniversityMCTS/gvgai/wiki/Code-Structure*
  4. Explorar y ejecutar el juego de la carrera de camellos (Camel Race), del que se proporciona un script sencillo (comentado en el tutorial y en estas diapositivas).
  5. Comenzar la práctica: se recomienda implementar los algoritmos en el orden propuesto

- El material a **entregar** a través de PRADO será un **fichero ZIP** con el siguiente contenido:
  - Un fichero **PDF** con la **memoria**
  - La **carpeta** del **paquete Java** anteriormente descrito y atendiendo a las siguientes consideraciones:
    - Únicamente debe **imprimir por pantalla** los mensajes especificados en el enunciado.
    - **El código debe estar bien comentado**.

- **Fecha de entrega:**
  - **17 de abril a las 23:59 horas**

NOTA: se recomienda encarecidamente **revisar las restricciones de la entrega en el enunciado de la práctica**, y seguir escrupulosamente todas las indicaciones allí incluidas.