UNIVERSIDAD DE GRANADA

DECSAI

Técnicas de los Sistemas Inteligentes

Grado en Informática

Curso 2019-20. Práctica 1
Técnicas de Búsqueda Heurística

Jesús Giráldez Crú, Pablo Mesejo Santiago y José Ángel Segura Muros

{jgiraldez,pmesejo,josesegmur}@decsai.ugr.es

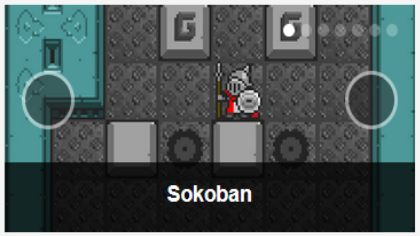
Departamento de Ciencias de la Computación e Inteligencia Artificial

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

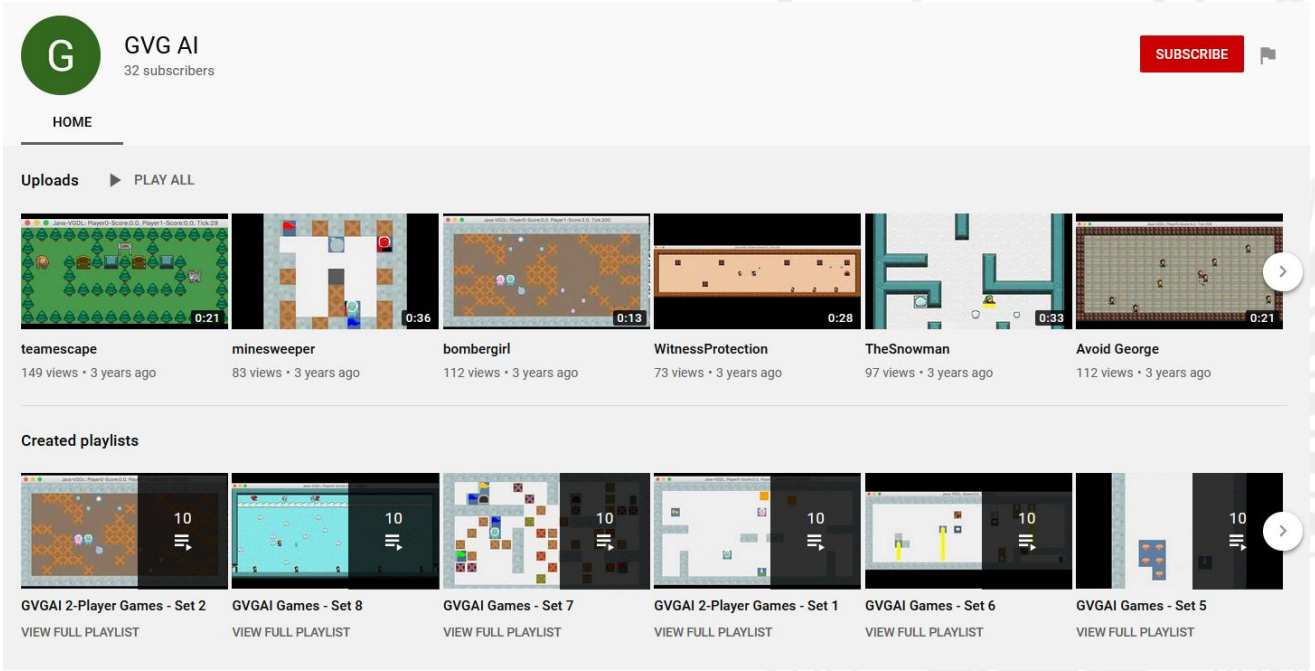
Descripciones de los videojuegos y vídeos de ejemplo

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

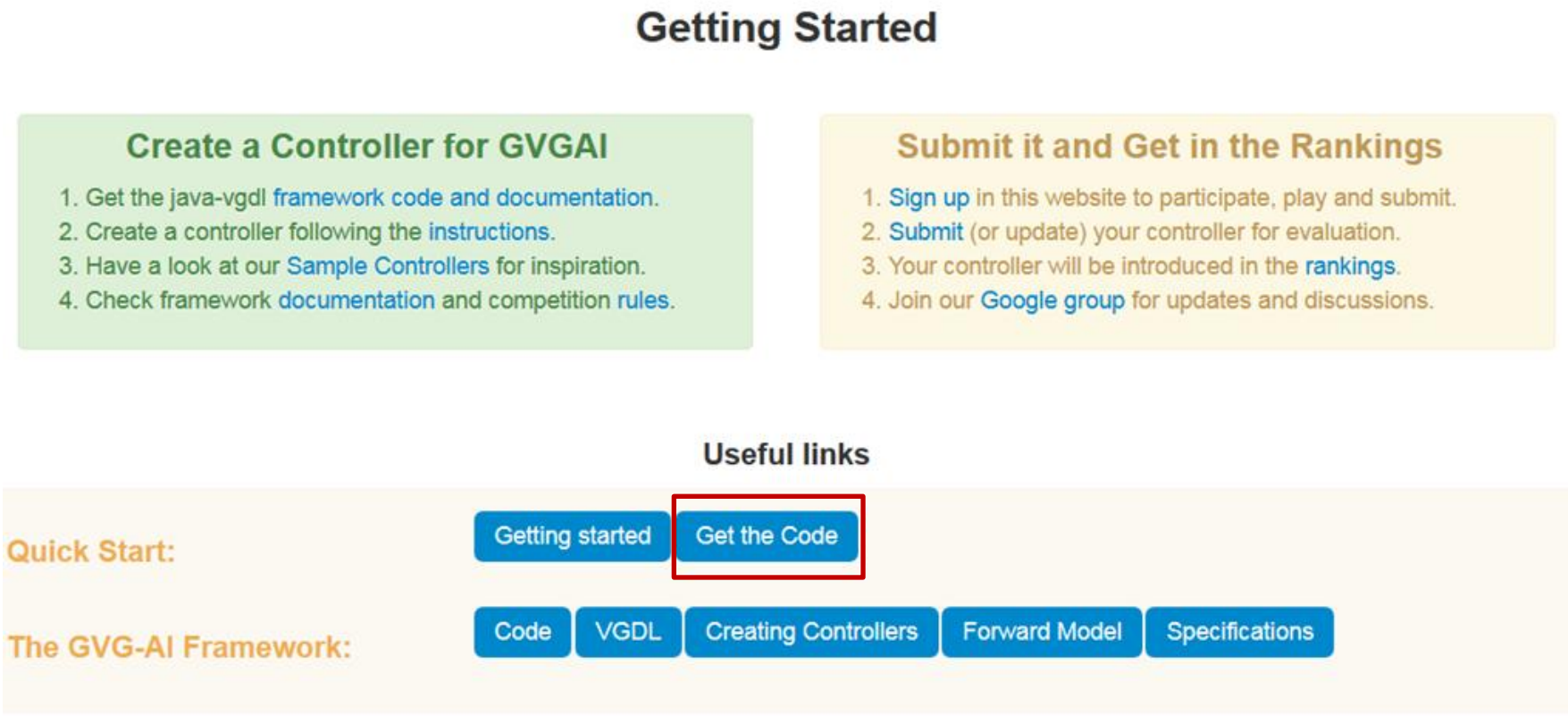
- **Single Player Track**
- Multiple Player Track
- Single Player Learning Track

Generar contenido

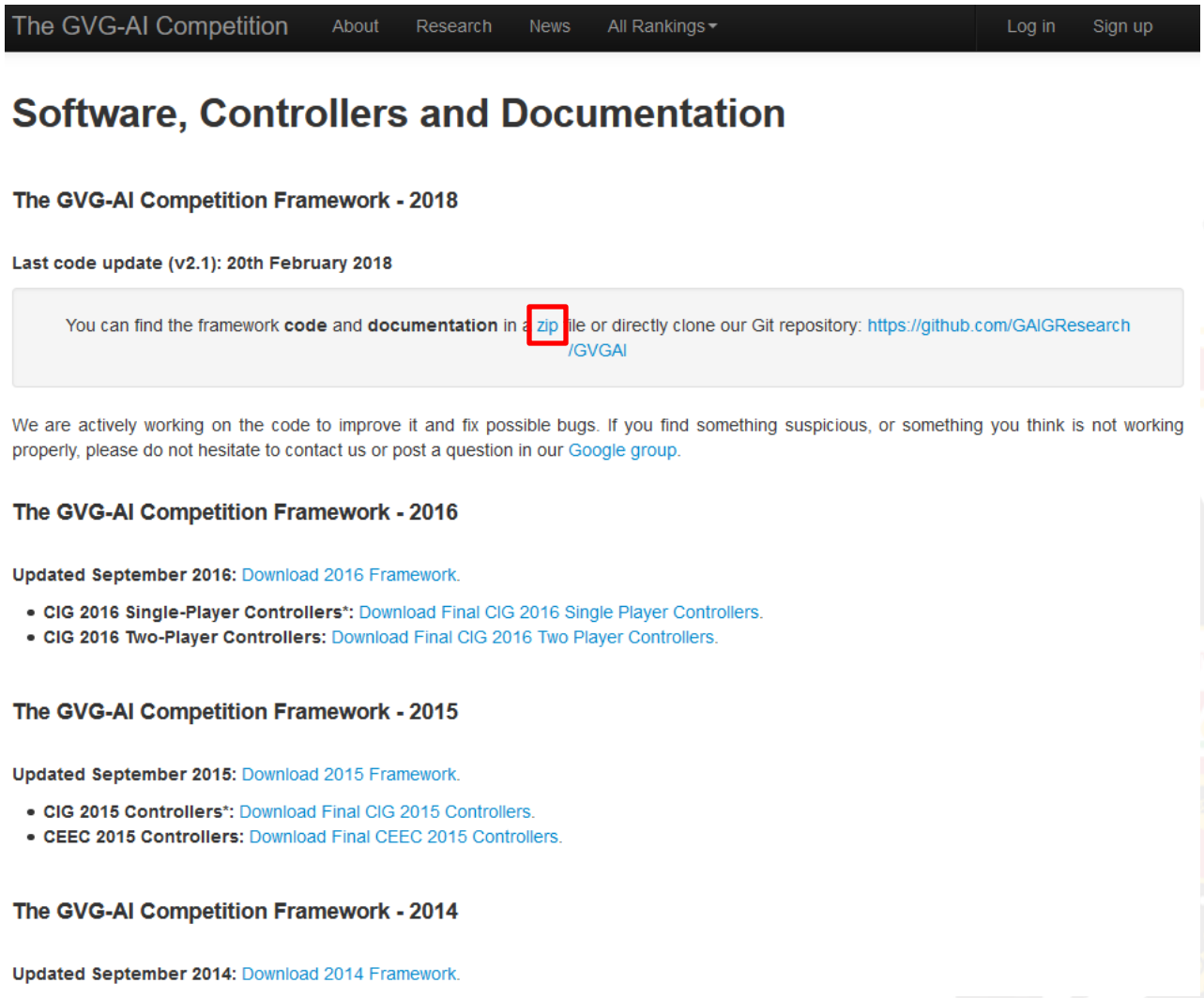
- Level Generation Track
- Rule Generation Track
- GameDesign Track

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

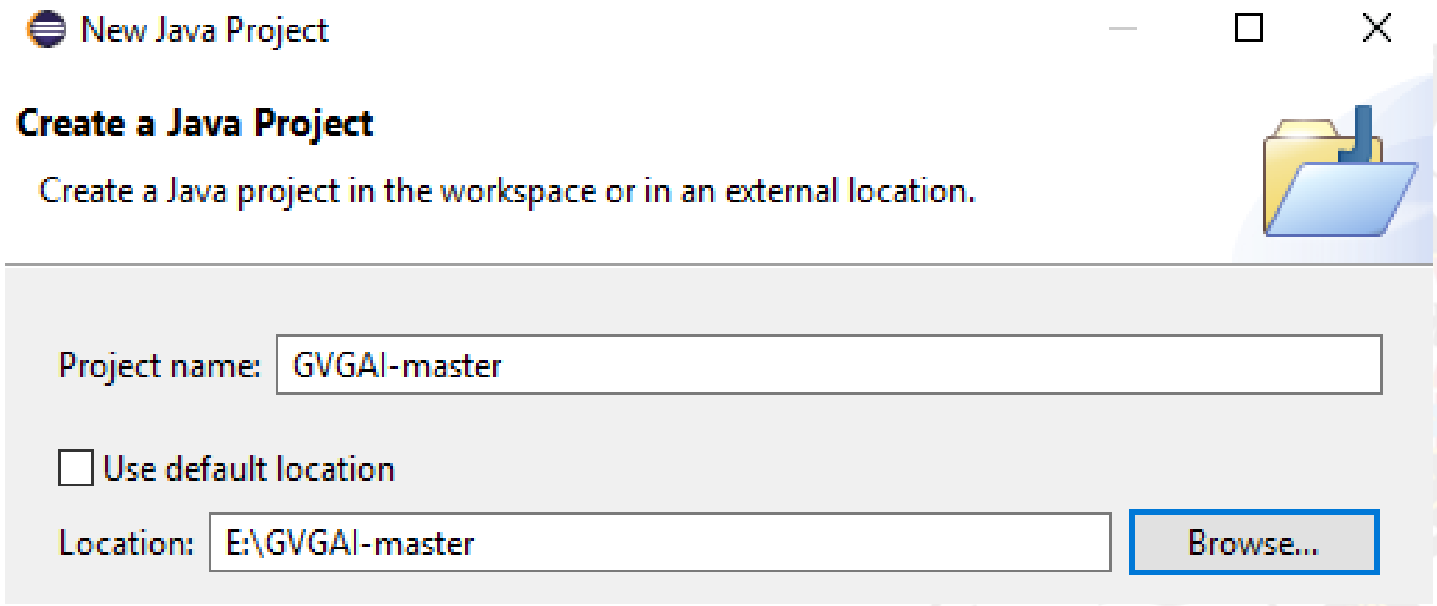


- Descargar el zip de http://www.gvgai.net/ pinchando en Useful links > Get The Code
  - https://github.com/GAIGResearch/GVGAI/archive/master.zip

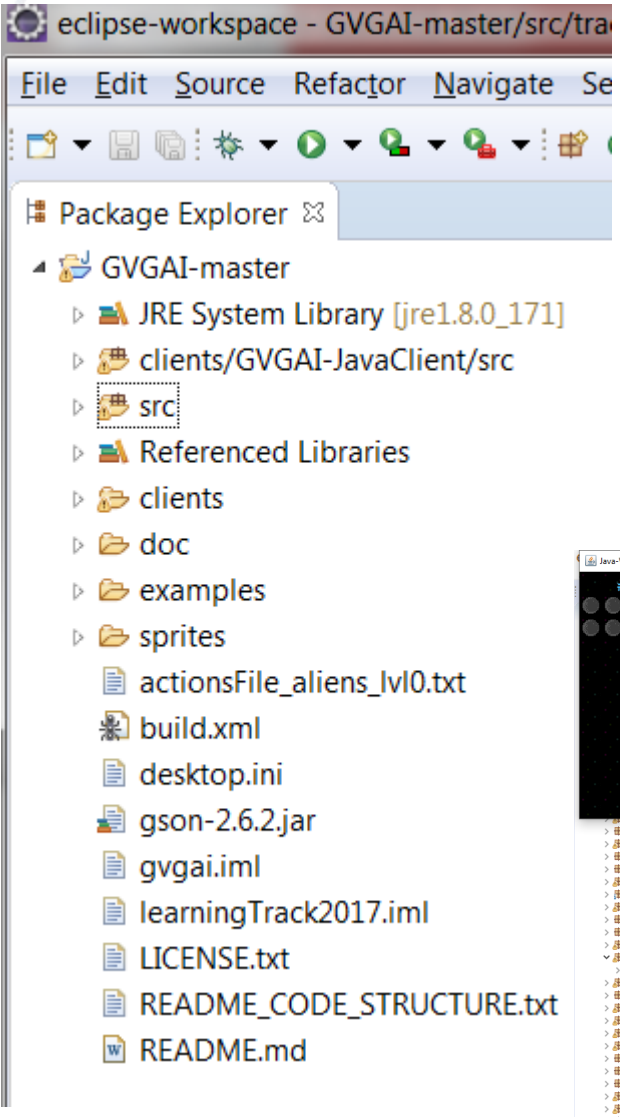


Instalación del GVG Framework con ECLIPSE
(hay que tener instalado Java y Eclipse)

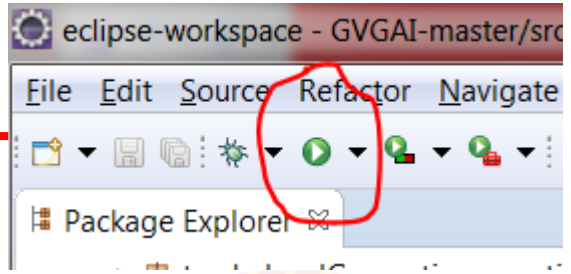
- Descomprimir en un <directorio>
- Abrir Eclipse, crear un nuevo workspace en cualquier otro directorio.
- File → New → Project… → New Java project
- Untick 'default location' y seleccionar <directorio> en Location → Finish

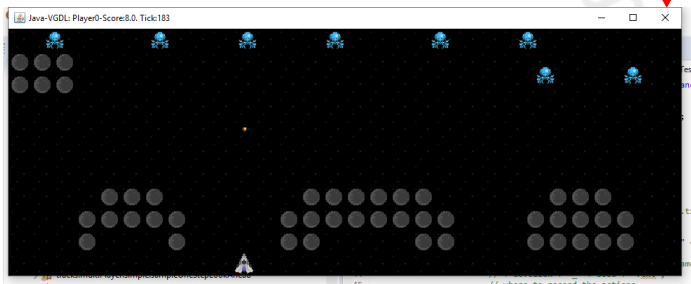


Ejecución del GVG Framework

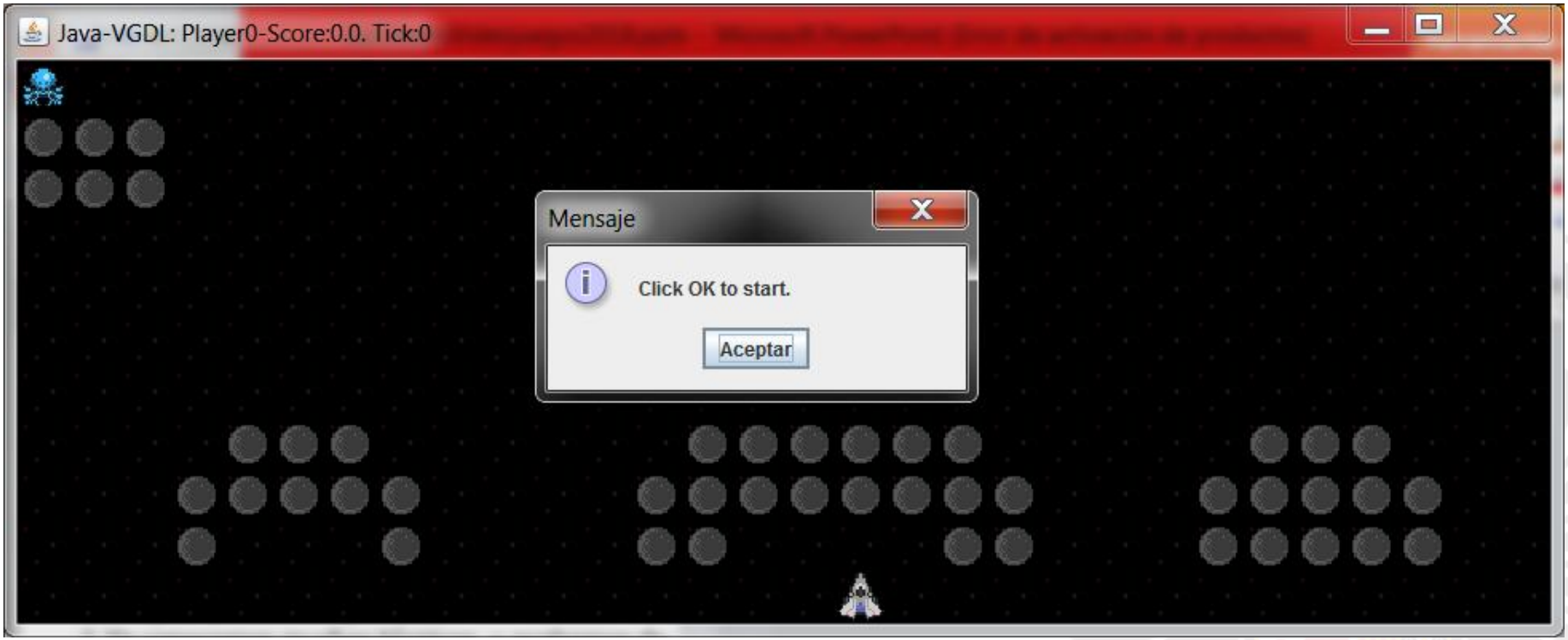


- En Package Explorer, navegar hasta
  - Src.tracks.singlePlayer → Test.java
  - Ejecutar Test.java





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.
```

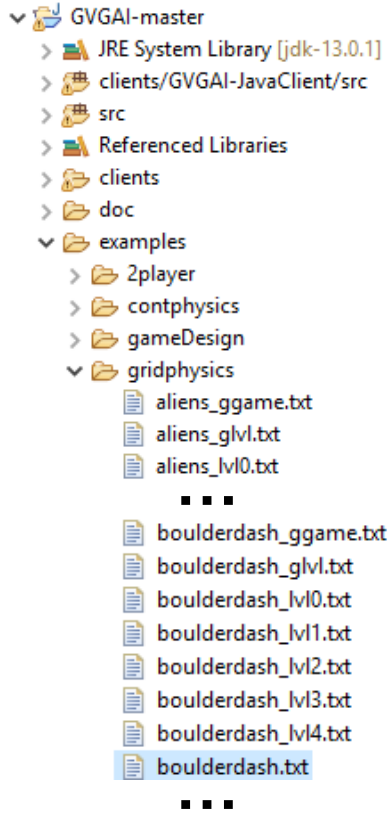
Consultar este fichero, Hay numerosos juegos disponibles para SinglePlayer

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

	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			




```
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";

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);
```

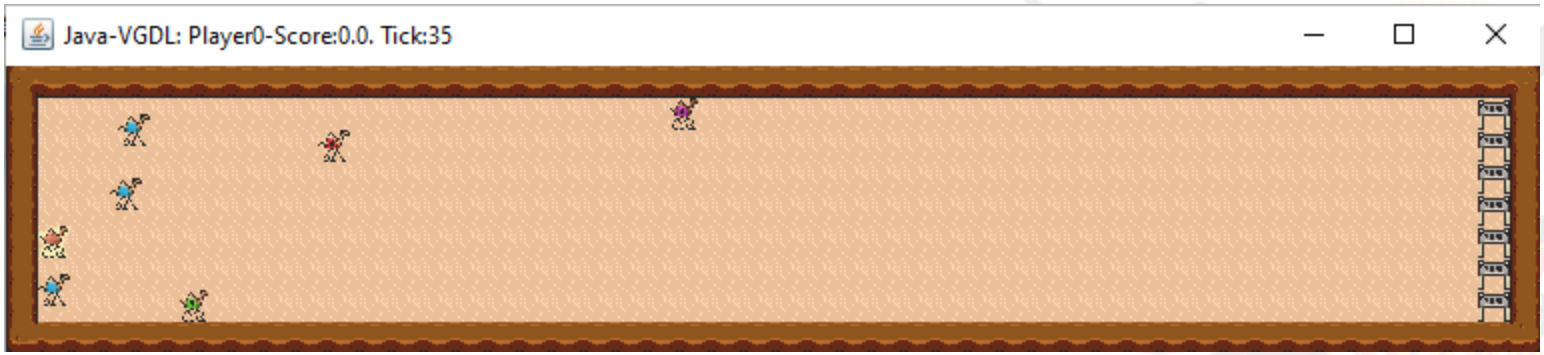
   2. Jugar como IA (con un "controller"): Usar (o crear) una variable String conteniendo el path de la clase de un agente (pre)diseñado.

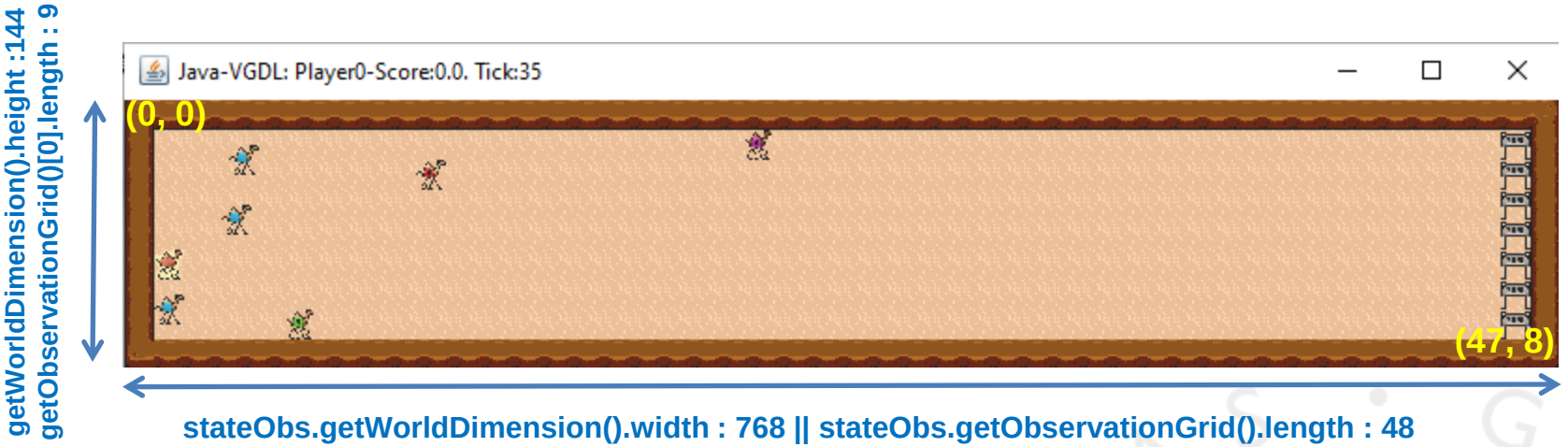
```
String sampleMCTSController =
"tracks.singlePlayer.advanced.sampleMCTS.Agent";
...
// 2. This plays a game in a level by the controller.
ArcadeMachine.runOneGame(game, level1, visuals,
sampleMCTSController, recordActionsFile, seed, 0);
```

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);
}
```

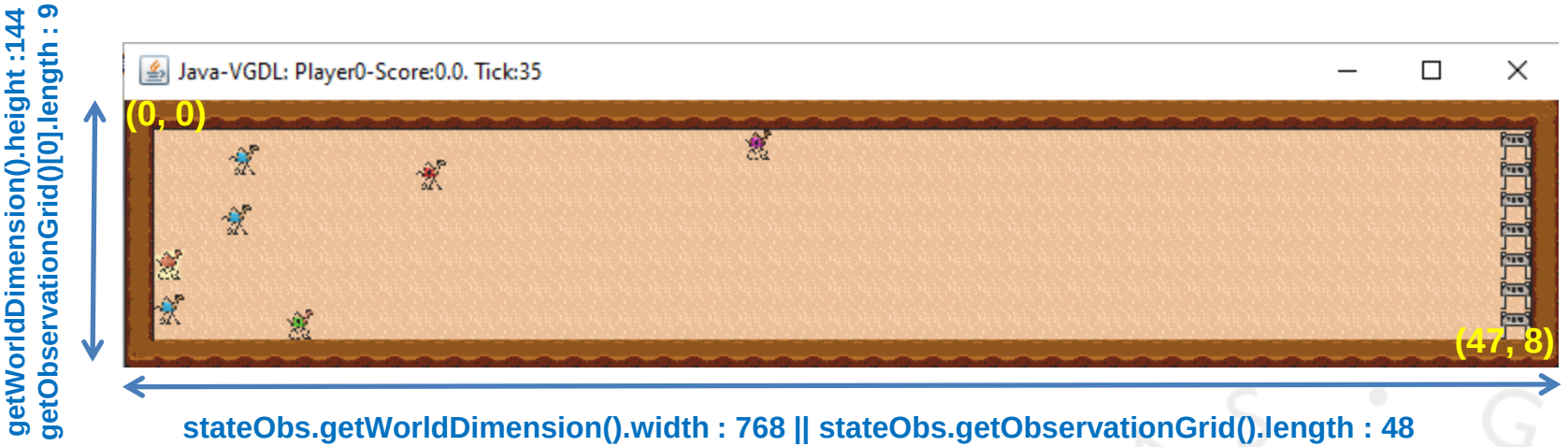
Píxeles del tablero/mundo

Posiciones del grid

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

Para obtener la posición de las gemas de la práctica

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.

```java
@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;
    }

}
```

La Práctica 1 consiste en **desarrollar un controlador, basado en A*** (o en alguna de sus variantes), **dentro del entorno GVGAI que guíe a un avatar a resolver** un juego en distintos niveles. El juego escogido es el juego con índice 11 en los tipos de juego "singleplayer", que se pueden encontrar en el fichero "examples/all_games_sp.csv" de la distribución de GVGAI, denominado **Boulder Dash.**

Acciones (IZQUIERDA, DERECHA, ARRIBA, ABAJO) y una acción de USO de la pala

El objetivo de la práctica es que los estudiantes se familiaricen con las **técnicas de búsqueda heurística, tanto en su vertiente deliberativa como reactiva**.

Para ello, se proponen 5 problemas/tareas de progresiva complejidad (en la siguiente slide se muestran posibles mapas para cada problema):

1. **Comportamiento deliberativo simple**: búsqueda del camino óptimo al portal (sin enemigos, pero con la posible presencia de obstáculos).
2. **Comportamiento deliberativo compuesto**: búsqueda de 10 gemas (en un mapa con un número superior o igual de gemas) y salida por el portal.
3. **Comportamiento reactivo simple**: mantenerse alejado de un enemigo durante un tiempo predeterminado (2000 ticks).
4. **Comportamiento reactivo compuesto**: mantenerse alejado de dos enemigos durante un tiempo predeterminado (2000 ticks).
5. **Comportamiento reactivo-deliberativo**: búsqueda de 10 gemas (en un mapa con un número superior o igual de gemas), evitando el enemigo presente en el mapa y, una vez se tengan todas, alcanzar el portal dentro de los límites de tiempo predeterminados (2000 ticks).

Diferencias con respecto al juego original:

- Todos los enemigos están liberados desde el comienzo.

- No hay rocas que puedan caer sobre el avatar.

Nota importante: se trata del mismo juego, no es necesario modificar su definición, solamente es necesario modificar los mapas.

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

1. **Comportamiento deliberativo simple**

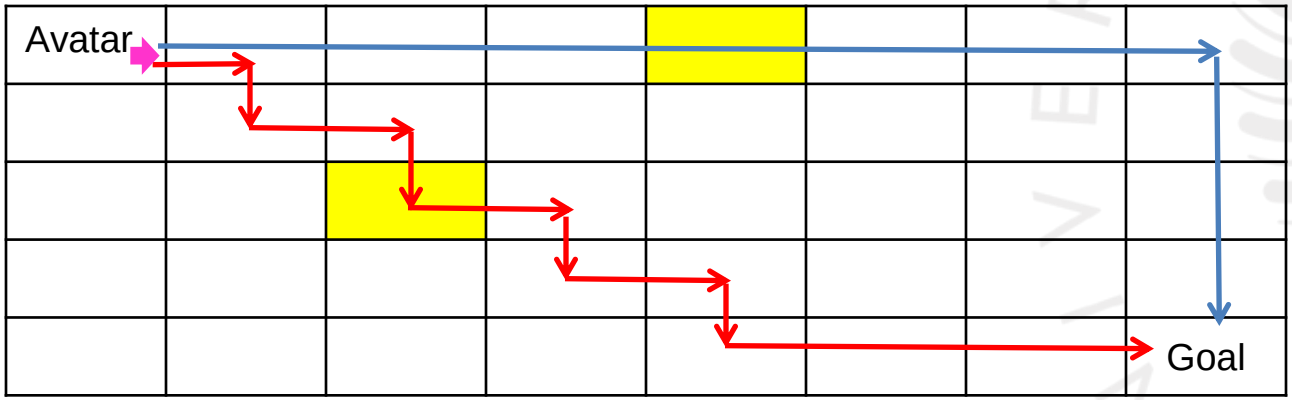
2. **Comportamiento deliberativo compuesto**

3. **Comportamiento reactivo simple**

4. **Comportamiento reactivo compuesto**

5. **Comportamiento reactivo-deliberativo**

- **No usar tools.pathfinder porque es innecesariamente complejo**:
  - Calcula el camino más corto de cada nodo a todos los demás.
  - Demasiado costoso computacionalmente.
  - Se pierde libertad a la hora de implementar A* del modo que uno considera más conveniente.
- **Importante conocer los detalles del juego:**
  - Cada tick, el juego llama a act, y si el return de act tarda en llegar más de 50ms → pierdes!
  - En BoulderDash, un cambio de dirección implica un tick. Esto es importante, de cara a utilizar el menor número ticks. Importante recordar que 2000 es el número máximo de ticks. Ejemplo:



f: distancia/ticks para llegar de la posición del Avatar al Goal

$f = \mathbf{g} + h = \mathbf{4} + 7$

$f = \mathbf{g} + h = \mathbf{7} + 7$

Porque el cambio de dirección implica un tick extra.

- **Probar distintos mapas para verificar que el código funciona en distintas configuraciones específicas de un mismo problema**

- El juego viene configurado para tener que recoger 10 gemas. Si, por ejemplo, queremos salir sin recoger ninguna gema (caso del deliberativo simple), **podemos modificar boulderdash.txt**. Si no lo cambiamos, el juego no permite salir al avatar por el portal (dado que se asume que solamente puede salir si ha recogido las 10 gemas).

*exitdoor avatar > killIfOtherHasMore resource=diamond limit=**9***

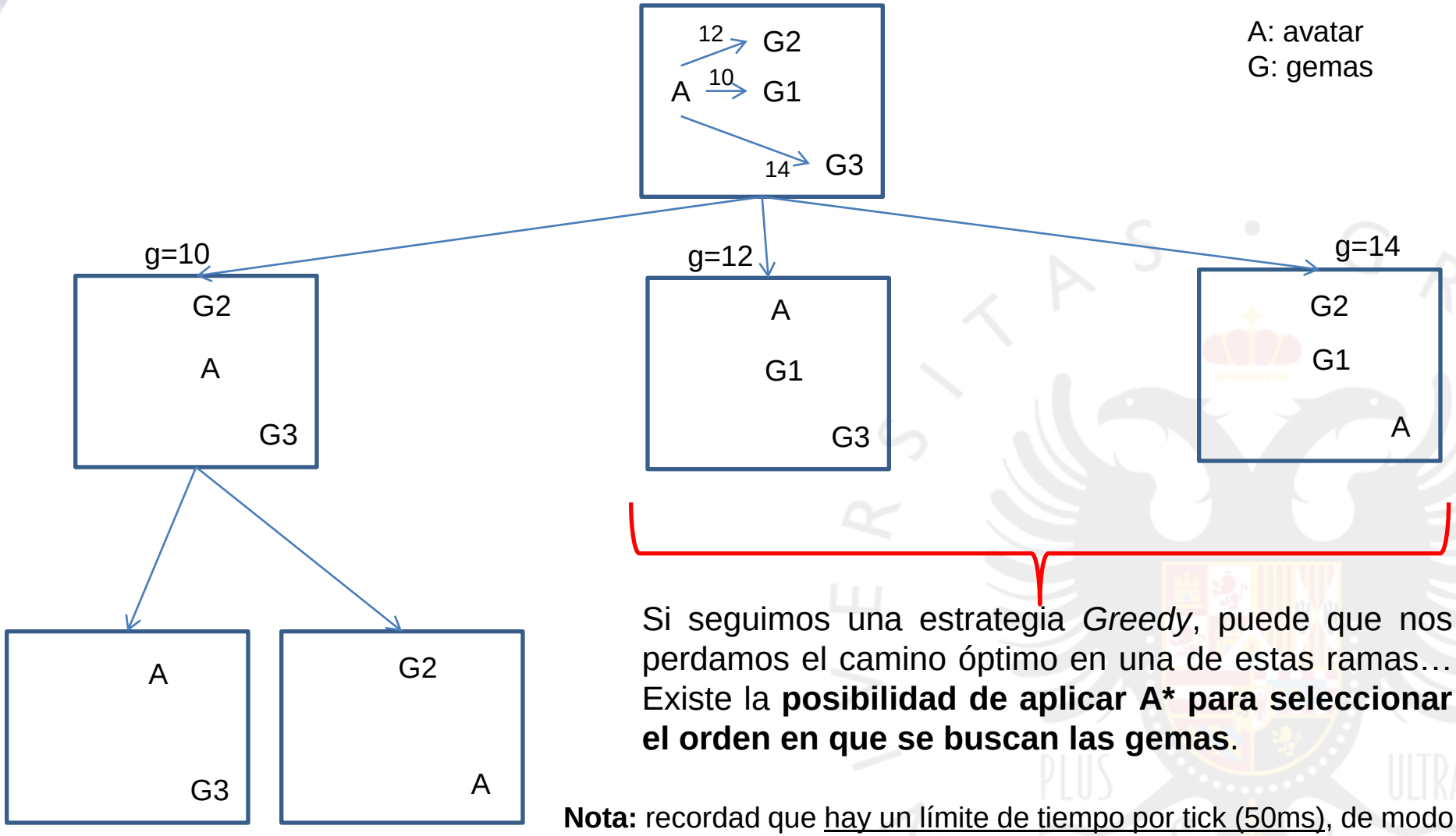
↓

*exitdoor avatar > killIfOtherHasMore resource=diamond limit=**-1***

- **¿Qué algoritmo de búsqueda debemos emplear?** A priori, **A*** parece una opción razonable.

Algunas posibilidad de algoritmos de búsqueda:	
Dijkstra's Algorithm (tb llamado Uniform Cost Search o Lowest-Cost-First Search)	**No es heurística!** No es factible para grafos grandes. Proporciona el camino más corto entre un nodo y todos los demás.
Greedy Best-First Search	**Solo usa la heurística!** Cuanto más precisa es la heurística, más rápido llegaremos al destino. Busca el camino más directo al destino.
A*	Combina Dijkstra con Greedy Best-First Search. **f(n) = g(n) + h(n)**
IDA*	**Igual que A*, pero consume menos memoria!**
Y muchos otros métodos de búsqueda (con pros y contras)…	

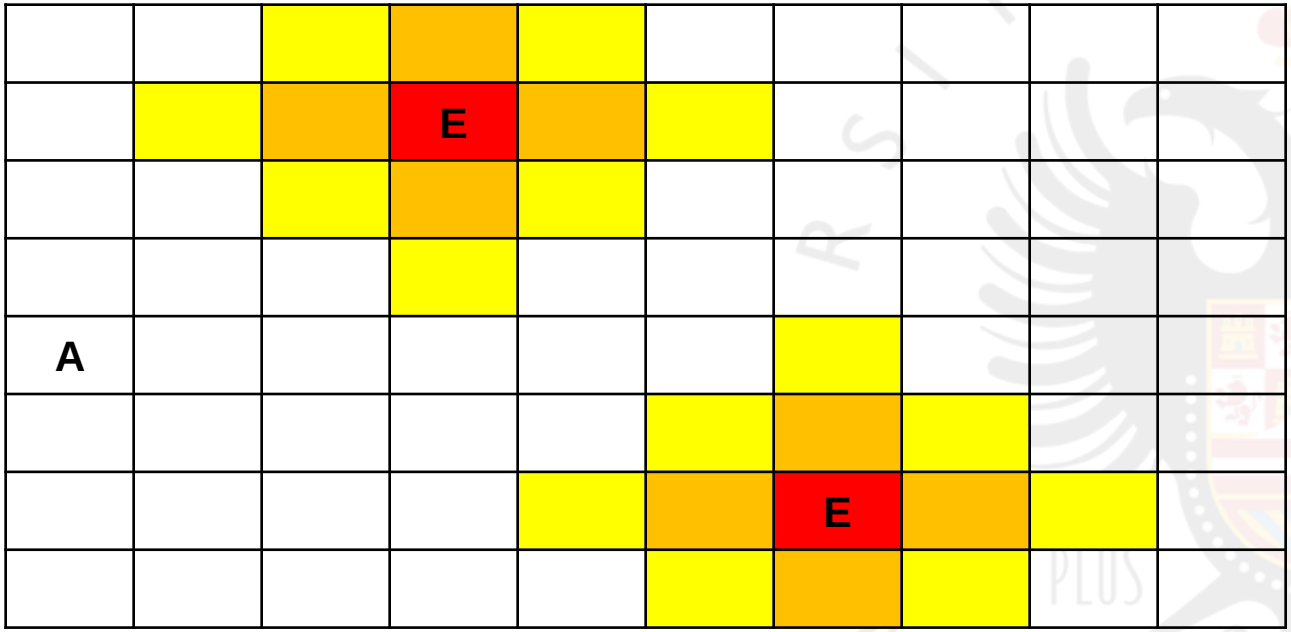
- **Posibilidad de cara a resolver de modo óptimo el problema del deliberativo compuesto:**



Si seguimos una estrategia *Greedy*, puede que nos perdamos el camino óptimo en una de estas ramas… Existe la **posibilidad de aplicar A* para seleccionar el orden en que se buscan las gemas**.

Nota: recordad que hay un límite de tiempo por tick (50ms), de modo que la computación de todo esto puede requerir más de un tick.

- **De cara a enfrentarse al problema reactivo, se pueden emplear mapas de calor que reflejen la proximidad de los enemigos:**
  - La idea es mantenerse lo más alejado posible de las zonas calientes
    - Evitar atraversarlas
    - Huir si se acerca un enemigo



A: avatar
E: enemigos

- Características
  - Plantearse varios objetivos
  - Priorizar objetivos
  - Alcanzar un objetivo es un problema de búsqueda heurística
    - No solo planificación de caminos teniendo en cuenta la distancia como heurística. Otros aspectos a considerar:
      - *presencia de enemigos*
      - *límite de tiempo*
  - Integración de reactivo y deliberativo
  - El plan elaborado puede fallar: replanificación
  - Incluso replanteamiento de objetivos.

- 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
     - 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: desde el apartado/problema 1 al 5 (va incrementando la dificultad progresivamente).

- El material a entregar a través de PRADO será un fichero ZIP con el siguiente contenido:
  - Una carpeta denominada "src_<apellido1>_<apellido2>_<nombre>" que incluya el **código fuente en java** cumpliendo las siguientes restricciones:
    - Un paquete java cuyo nombre sea el mismo que el de la carpeta.
    - Contener, al menos, un fichero "Agent.java" en el que se defina la clase que implementa el controlador, tal y como se describe en los tutoriales que se entregan como material de la práctica o en los tutoriales del entorno.
      - *Podrán entregarse otros ficheros fuente adicionales si así lo considera el alumno.*
  - Un fichero en **PDF** con la memoria de la práctica. La memoria debe incluir:
    - Un apartado por cada uno de los comportamientos solicitados:
      - *Comportamiento deliberativo (simple y compuesto)*
      - *Comportamiento reactivo (simple y compuesto)*
      - *Comportamiento reactivo-deliberativo*
    - En todos ellos debe proporcionarse con claridad una descripción de la solución propuesta por el estudiante: qué se hace, cómo se hace y por qué.

A los alumnos se les facilitarán mapas para resolver cada uno de los apartados/problemas de la práctica, pero **serán evaluados en mapas diferentes** (idénticos en dificultad)

PROBLEMA	PUNTUACIÓN
Comportamiento deliberativo simple	1 pto
Comportamiento deliberativo compuesto	1 pto
Comportamiento reactivo simple	1 pto
Comportamiento reactivo compuesto	1 pto
Comportamiento reactivo-deliberativo	3 ptos

+ 2 ptos la memoria: claridad, justificación de las decisiones y buena presentación
(0.75 deliberativo, 0.75 reactivo, 0.5 reactivo+deliberativo)

El **punto restante, para aquellos que alcancen un 9,** se obtiene según tiempo de ejecución (ticks). Crearemos cuatro segmentos:

4º Cuartil → 9
3º Cuartil → 9,35
2º Cuartil → 9,7
1º Cuartil → 10

Fecha de entrega:

29 de Marzo a las 14:00 horas