



UNIVERSIDAD
DE GRANADA



Técnicas de los Sistemas Inteligentes

Grado en Informática

Curso 2020-21. Práctica 1

Técnicas de Búsqueda Heurística

Jesús Giráldez Crú y Pablo Mesejo Santiago

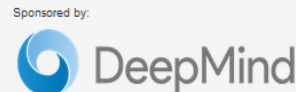
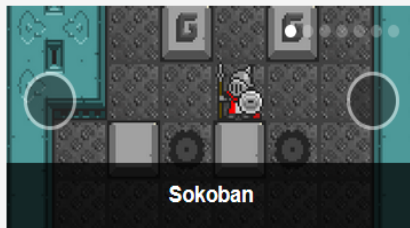
Departamento de Ciencias de la
Computación e Inteligencia Artificial

<http://decsai.ugr.es>

General Video Game AI (GVGAI)

- Website: <http://www.gvgai.net/>
- GVGA: 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 Intelligence in Games (CIG) (pp. 1-8). IEEE.



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

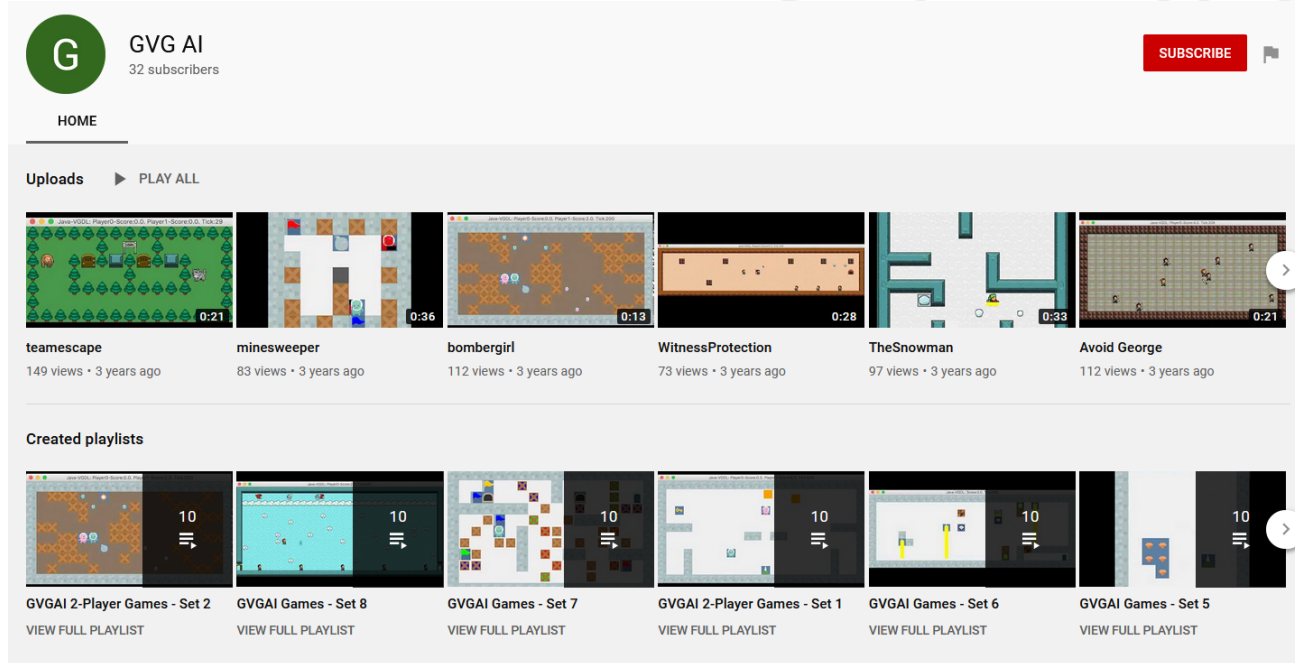
Descripciones de los videojuegos y vídeos de ejemplo

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



The screenshot shows the GVGAI YouTube channel page. At the top, there is a green circular profile picture with a white 'G' and the text 'GVG AI' and '32 subscribers'. Below this is a 'HOME' tab. The main content area is titled 'Uploads' and 'PLAY ALL'. It displays a grid of video thumbnails for various games: 'teamscape' (149 views, 3 years ago), 'minesweeper' (83 views, 3 years ago), 'bombergirl' (112 views, 3 years ago), 'WitnessProtection' (73 views, 3 years ago), 'TheSnowman' (97 views, 3 years ago), and 'Avoid George' (112 views, 3 years ago). Below the grid, there is a section titled 'Created playlists' showing six playlists: 'GVGAI 2-Player Games - Set 2', 'GVGAI Games - Set 8', 'GVGAI Games - Set 7', 'GVGAI 2-Player Games - Set 1', 'GVGAI Games - Set 6', and 'GVGAI Games - Set 5'. Each playlist has a 'VIEW FULL PLAYLIST' link.

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>

Getting Started

Create a Controller for GVGAI

1. Get the java-vgdl framework code and documentation.
2. Create a controller following the instructions.
3. Have a look at our Sample Controllers for inspiration.
4. Check framework documentation and competition rules.

Submit it and Get in the Rankings

1. Sign up in this website to participate, play and submit.
2. Submit (or update) your controller for evaluation.
3. Your controller will be introduced in the rankings.
4. Join our Google group for updates and discussions.

Useful links

Quick Start:

[Getting started](#)
[Get the Code](#)

The GVG-AI Framework:

[Code](#)
[VGDL](#)
[Creating Controllers](#)
[Forward Model](#)
[Specifications](#)

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

The GVG-AI Competition About Research News All Rankings ▾ Log in Sign up

Software, Controllers and Documentation

The GVG-AI Competition Framework - 2018

Last code update (v2.1): 20th February 2018

You can find the framework **code** and **documentation** in a **zip** file or directly clone our Git repository: <https://github.com/GAIGResearch/GVGAI>

We are actively working on the code to improve it and fix possible bugs. If you find something suspicious, or something you think is not working properly, please do not hesitate to contact us or post a question in our [Google group](#).

The GVG-AI Competition Framework - 2016

Updated September 2016: [Download 2016 Framework](#).

- CIG 2016 Single-Player Controllers*: [Download Final CIG 2016 Single Player Controllers](#).
- CIG 2016 Two-Player Controllers: [Download Final CIG 2016 Two Player Controllers](#).

The GVG-AI Competition Framework - 2015

Updated September 2015: [Download 2015 Framework](#).

- CIG 2015 Controllers*: [Download Final CIG 2015 Controllers](#).
- CEEC 2015 Controllers: [Download Final CEEC 2015 Controllers](#).

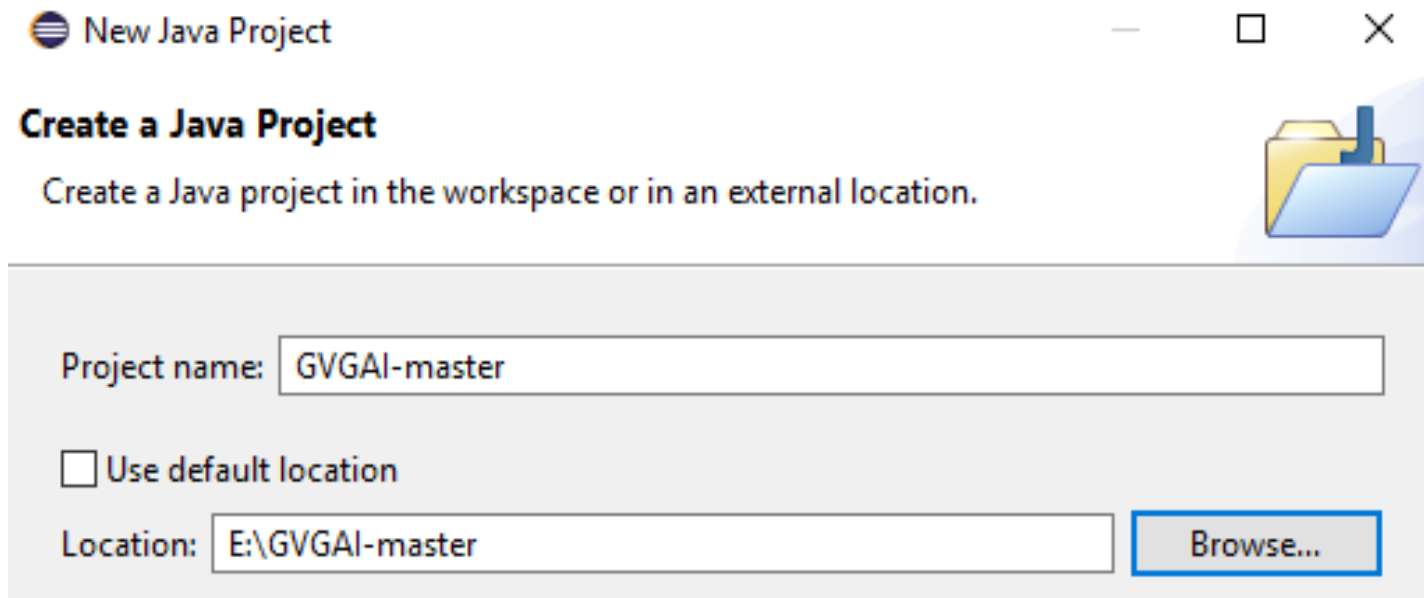
The GVG-AI Competition Framework - 2014

Updated September 2014: [Download 2014 Framework](#).

Instalación del GVG Framework con ECLIPSE

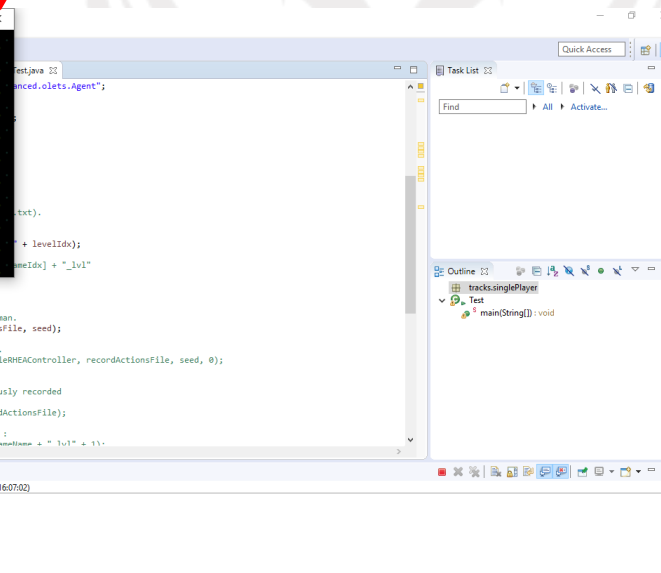
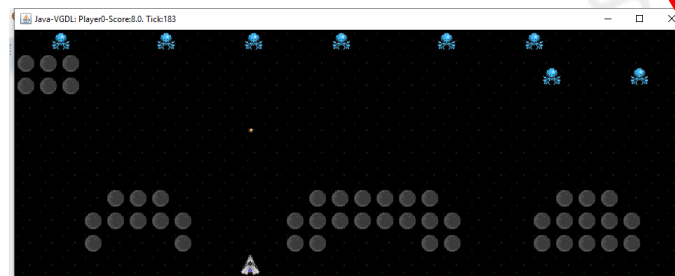
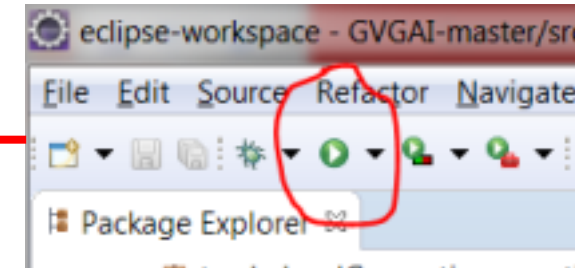
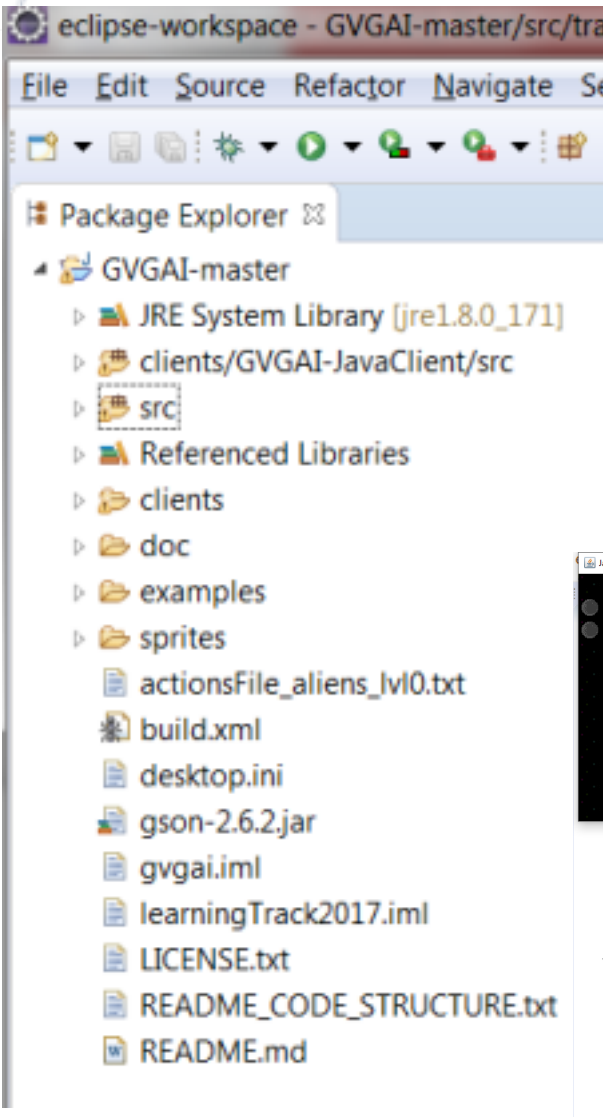
(hay que tener instalado Java y Eclipse)

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

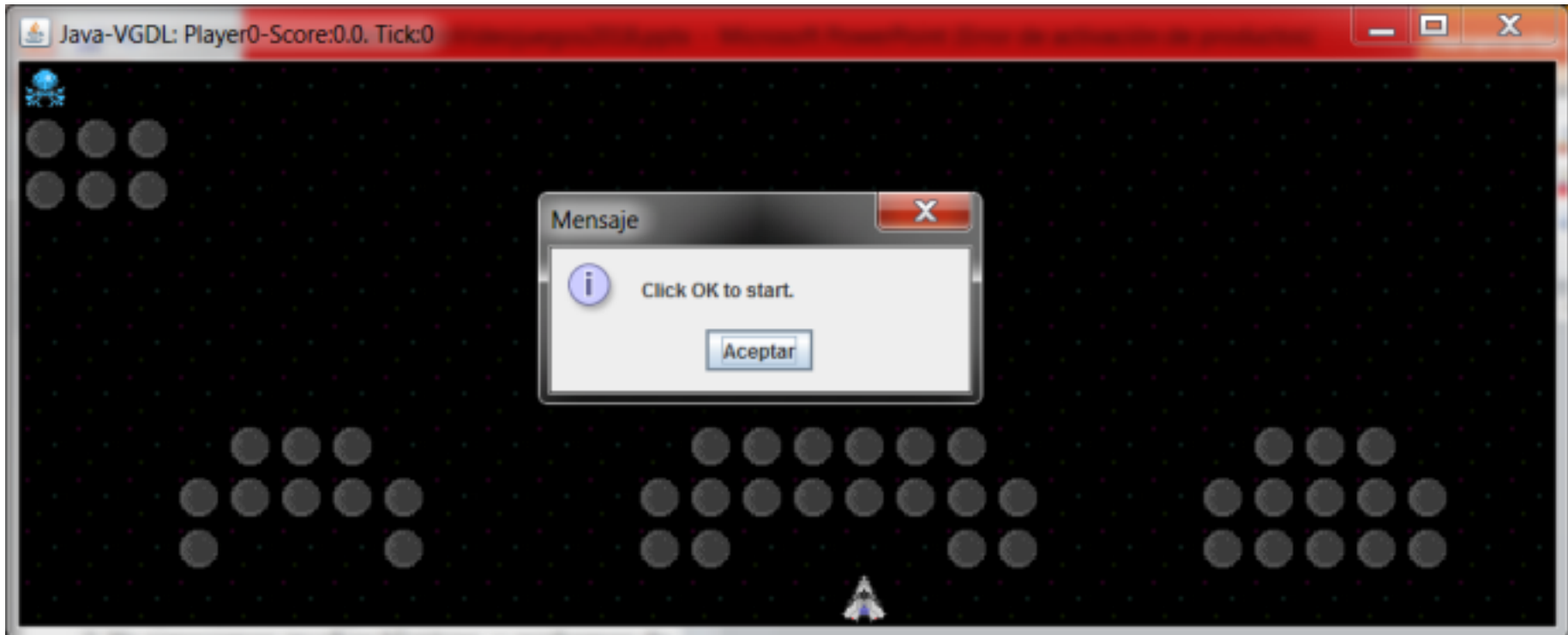


Ejecución del GVG Framework

- 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);
```

Consultar este fichero,
Hay numerosos juegos
disponibles para
SinglePlayer

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

```
// Game and level to play
```

int gameIdx = 0; Cambiar este valor y ejecutar para ver otros juegos

```
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.
```

¿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/boulcherdash.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			

```

GVGAI-master
├── JRE System Library [jdk-13.0.1]
├── clients/GVGAI-JavaClient/src
├── src
├── Referenced Libraries
├── clients
├── doc
├── examples
│   ├── 2player
│   ├── contphysics
│   ├── gameDesign
│   └── gridphysics
│       ├── aliens_ggame.txt
│       ├── aliens_gvl.txt
│       ├── aliens_lv0.txt
│       ├── ■ ■ ■
│       ├── boulderdash_ggame.txt
│       ├── boulderdash_gvl.txt
│       ├── boulderdash_lv0.txt
│       ├── boulderdash_lv1.txt
│       ├── boulderdash_lv2.txt
│       ├── boulderdash_lv3.txt
│       ├── boulderdash_lv4.txt
│       └── boulderdash.txt
│           ├── ■ ■ ■

```

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

```

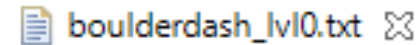
1 BasicGame
2   SpriteSet
3     background > Immovable img=oryx/backBlack hidden=True
4     wall > Immovable autotiling=true img=oryx/dirtWall
5     sword > Flicker color=LIGHTGRAY limit=1 singleton=True img=oryx/pickaxe
6     dirt > Immovable color=BROWN img=oryx/backLBrown
7     exitdoor > Door color=GREEN img=oryx/door2
8     diamond > Resource color=YELLOW limit=10 shrinkfactor=0.75 img=oryx/diamond3
9     boulder > Missile orientation=DOWN color=GRAY speed=0.2 img=oryx/mineral1
10    moving >
11      avatar > ShootAvatar stype=sword frameRate=8 img=oryx/spelunky
12      enemy > RandomNPC cons=1
13      crab > color=RED img=oryx/scorpion2
14      butterfly > color=PINK img=oryx/bat2
15
16 LevelMapping
17   . > background dirt
18   - > background
19   e > background exitdoor
20   o > background boulder
21   x > background diamond
22   c > background crab
23   b > background butterfly
24   A > background avatar
25
26 InteractionSet
27   dirt avatar sword > killSprite
28   diamond avatar > collectResource scoreChange=2
29   moving wall boulder > stepBack
30
31   avatar boulder > killIfFromAbove scoreChange=-1
32   avatar butterfly crab > killSprite scoreChange=-1
33
34   boulder dirt wall diamond boulder > stepBack
35
36   enemy dirt diamond > stepBack
37
38   crab butterfly > killSprite
39   butterfly crab > transformTo stype=diamond scoreChange=1
40   exitdoor avatar > killIfOtherHasMore resource=diamond limit=9
41
42 TerminationSet
43   SpriteCounter stype=avatar limit=0 win=False
44   SpriteCounter stype=exitdoor limit=0 win=True
45

```

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

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

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



```

1  ~~~~~
2  w...o.xx.o.....o..xox..w
3  w...ooooo.....o..o...w
4  w...xxx.....o.oxoo.ow
5  wx.....oxo...oow
6  ~~~~~o...wxxw
7  wb-...co...wxxw
8  w--.....Ao...o...wxxw
9  wooo.....-...w..w
10 w.....x...wwwx-x.oow..w
11 wc--.....x..ooxxo-...w..w
12 w---..e.....b-----..w
13 ~~~~~

```

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 sampleOLETController = "tracks.singlePlayer.advanced.olets.Agent";
```

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



Experimental 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";
```

```
...
```

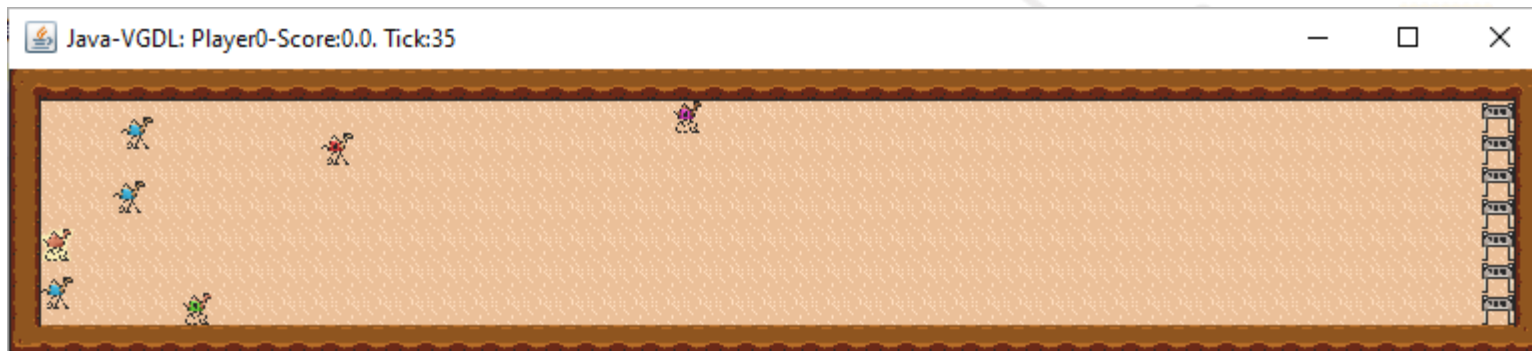
```
// 2. This plays a game in a level by the controller.
```

```
ArcadeMachine.runOneGame(game, level1, visuals,
sampleMCTSController, recordActionsFile, seed, 0);
```

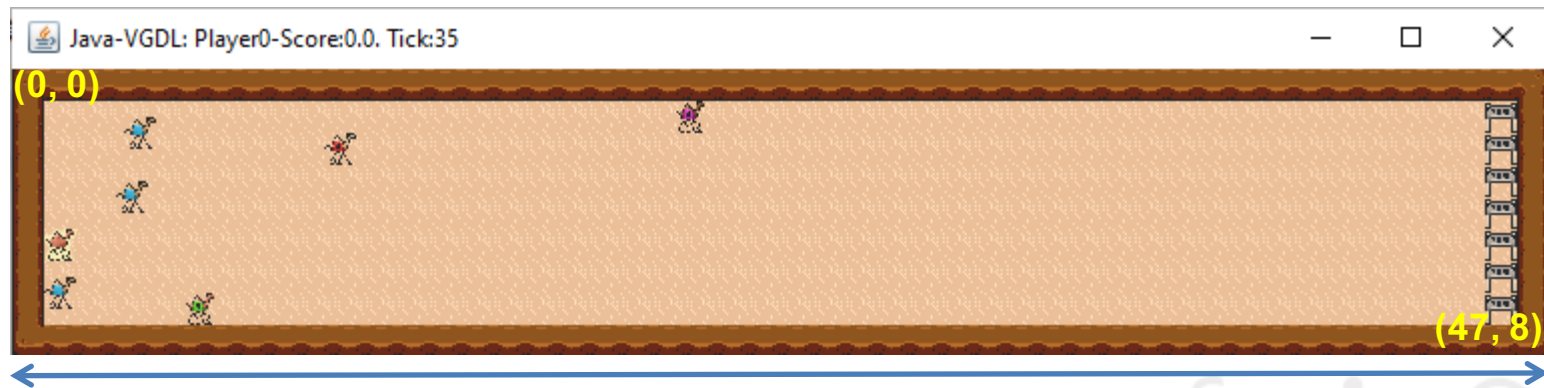
4. Run `Test.java`

Escogemos el juego de **Carrera de Camellos** (int gameldx = 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.



getWorldDimension().height : 144
getObservationGrid()[0].length : 9



stateObs.getWorldDimension().width : 768 || stateObs.getObservationGrid().length : 48

```
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 (píxeles -> 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 cercanía 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

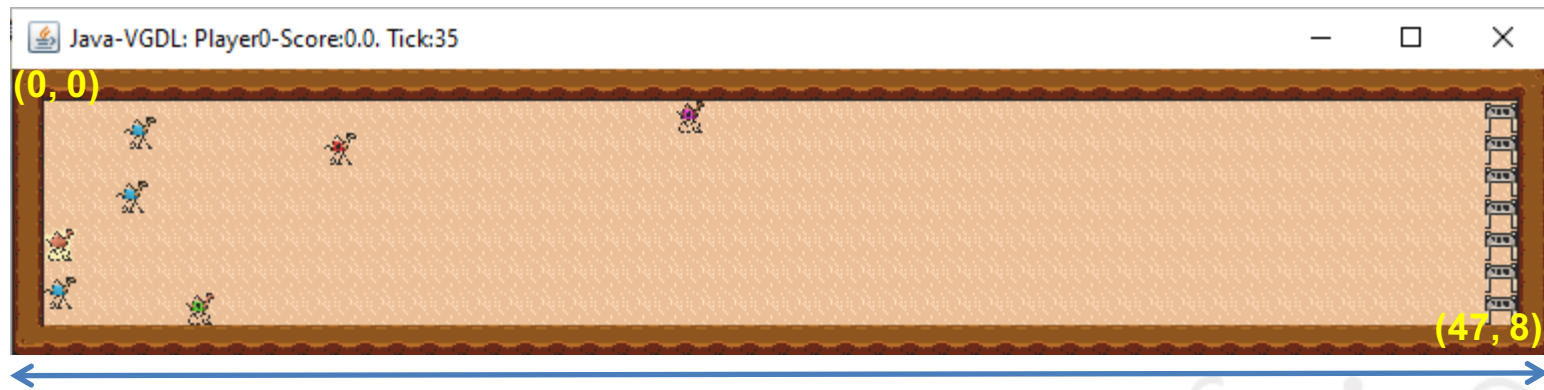
Posiciones del grid

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

getWorldDimension().height : 144
getObservationGrid()[0].length : 9



stateObs.getWorldDimension().width : 768 || stateObs.getObservationGrid().length : 48

Para obtener la posición de los enemigos

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

Para obtener la posición de las gemas de la práctica

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.

```
@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 deliberativas (incluyendo búsqueda heurística) y reactivas**.

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 9 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 9 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).

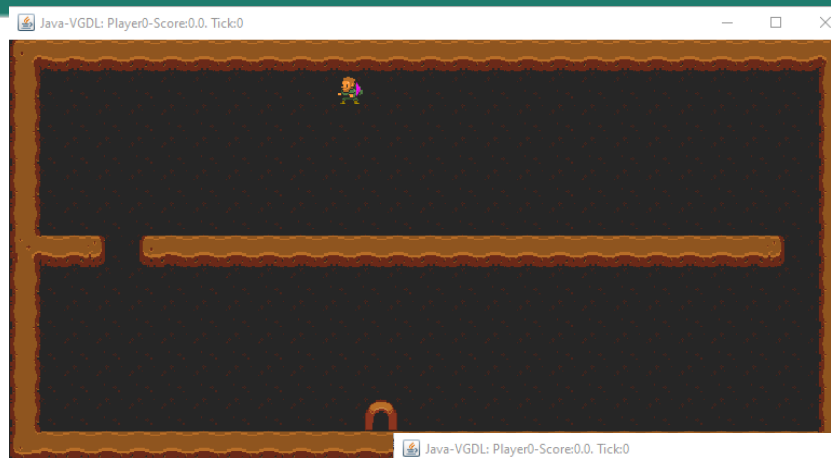
Diferencias con respecto al juego original:

- 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.

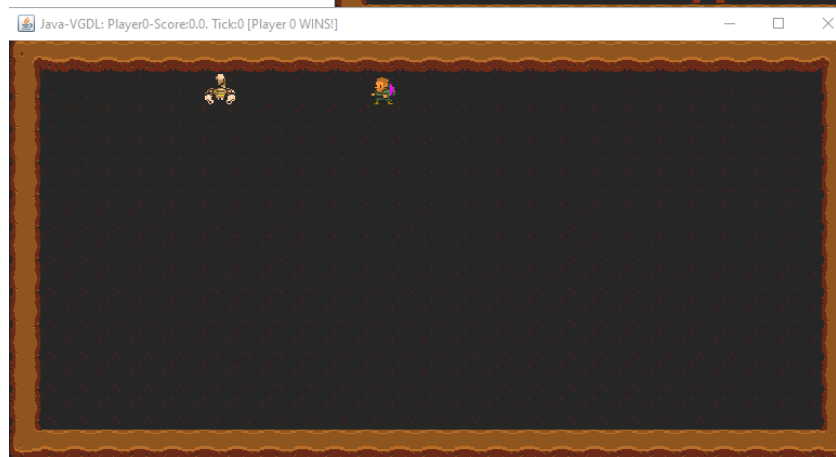
1. Comportamiento deliberativo simple



2. Comportamiento deliberativo compuesto



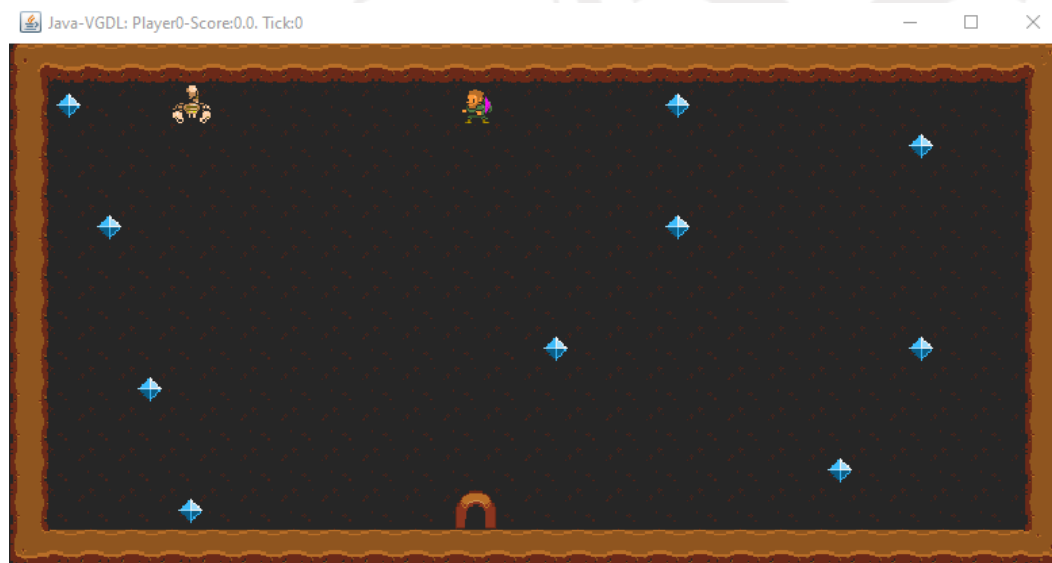
3. Comportamiento reactivo simple



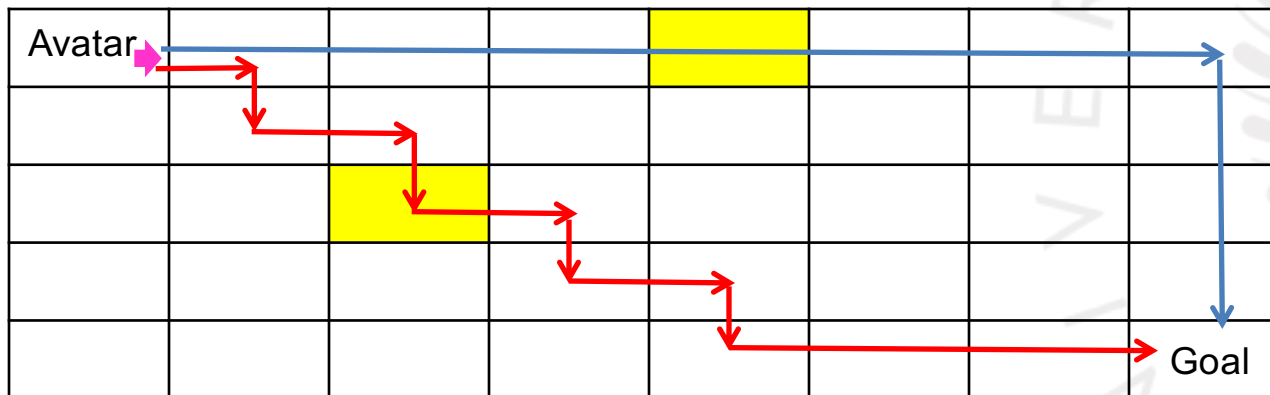
4. Comportamiento reactivo compuesto



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

$$f = g + h = 4 + 7$$

$$f = g + h = 7 + 7$$

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 9 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 9 gemas).

exitdoor avatar > killIfOtherHasMore resource=diamond limit=9



exitdoor avatar > killIfOtherHasMore resource=diamond limit=0

- **¿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)...	

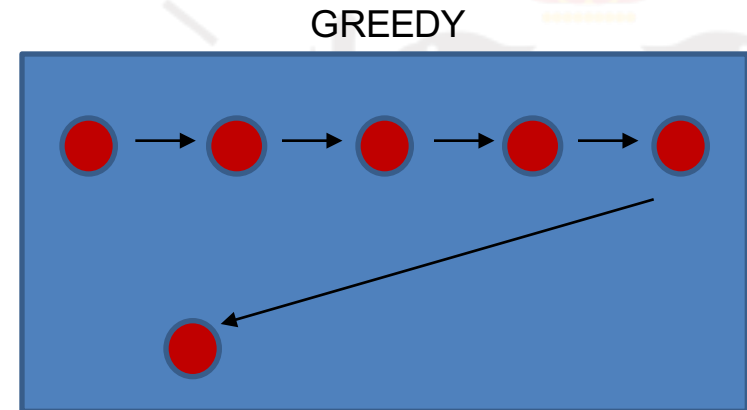
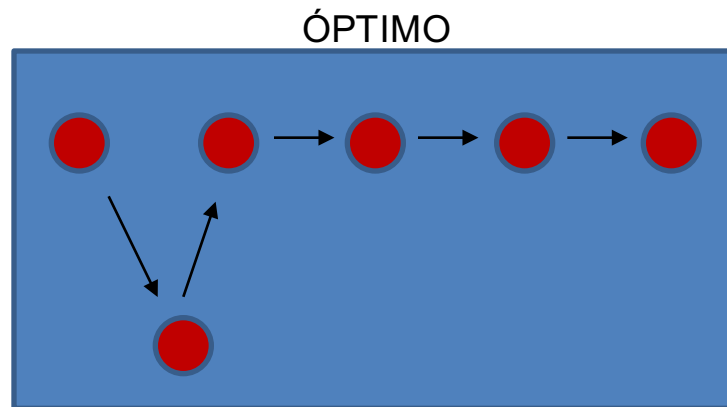
▪ Planteamiento para el problema del deliberativo compuesto:

Este problema se puede **descomponer en dos sub-problemas**:

1. Decidir a qué gema (portal) dirigirse en cada momento
2. Decidir el camino para dirigirse hasta ella

Para el **primer sub-problema**, se empleará una estrategia greedy:

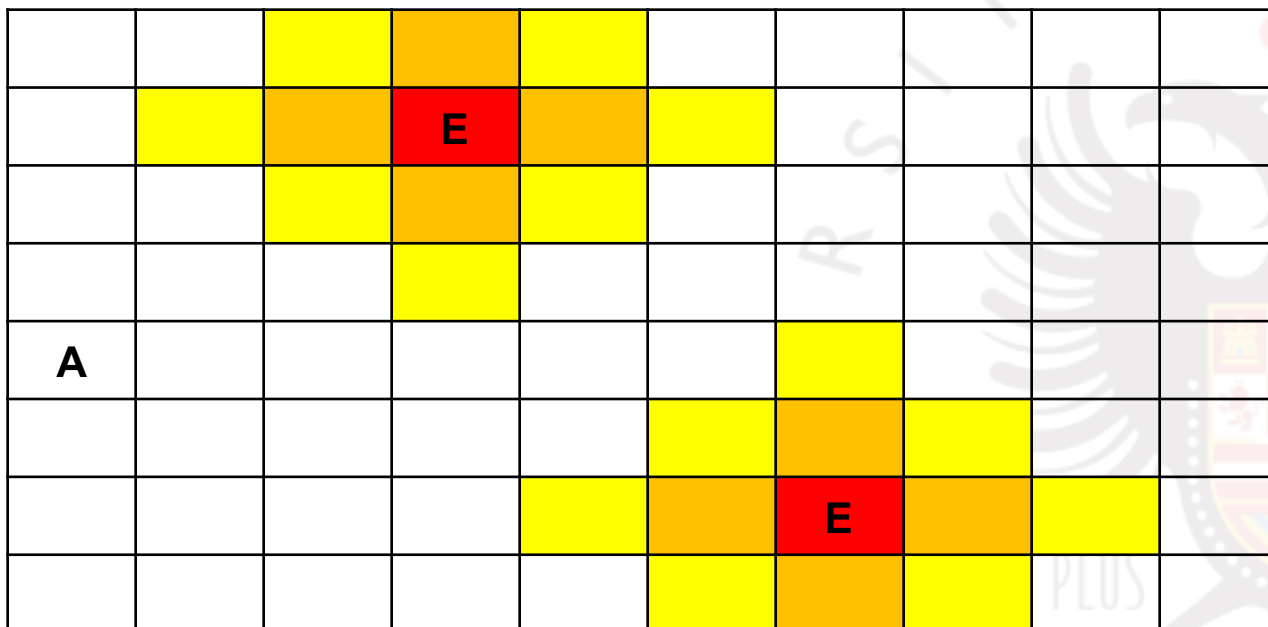
- En cada momento, dirigirse a la gema (portal) más cercana según la **distancia Manhattan**
- **NO** es una estrategia óptima, pero tiene mayor capacidad adaptativa (p.ej, en presencia de enemigos)



Para el **segundo sub-problema**, usaremos el camino óptimo calculado con A* para el problema deliberativo simple

Nota: recordad que hay un límite de tiempo por tick (50ms), por lo que la implementación debe ser suficientemente eficiente para no superar este tiempo.

- De cara a enfrentarse al problema reactivo, se pueden emplear mapas de calor, o campos de potencial, que reflejen la proximidad de los enemigos:
 - La idea es mantenerse lo más alejado posible de las zonas calientes
 - Evitar atravesarlas
 - 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.
 - Que no imprima ningún mensaje por pantalla!
 - *Nosotros, a la hora de corregir los ejercicios, generaremos unos informes automáticos. Si vuestro código imprime algo, dificulta el análisis de dichos informes.*
 - 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.*
 - **El código debe estar bien comentado.**

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)

NIVEL	PUNTUACIÓN SI SE SUPERA EL NIVEL
Comportamiento deliberativo simple	1.5 pts
Comportamiento deliberativo compuesto	1.5 pts
Comportamiento reactivo simple	1.5 pts
Comportamiento reactivo compuesto	1.5 pts
Comportamiento reactivo-deliberativo	3 pts

El **punto restante, para aquellos que alcancen un 9**, se obtiene según el tiempo de ejecución (ticks) del mapa reactivo-deliberativo, según cuatro segmentos:

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

- Deliberativos simple y compuesto: se da toda la puntuación si se llega al portal en el número mínimo de ticks según la estrategia deliberativa de cada nivel (A* para el deliberativo simple y A*+greedy para el deliberativo compuesto). Si no se consigue, 0 puntos.
- Reactivo simple: se da toda la puntuación si se sobrevive en ≥ 9 ejecuciones (sobre 10); 0 si es 6 o menos; calificación lineal de 6 a 9:
 - 6 veces (0 ptos)
 - 7 veces (0.5 ptos)
 - 8 veces (1 pto)
 - 9 veces (1.5 ptos)
- Reactivo compuesto: toda la puntuación si se sobrevive en ≥ 7 ejecuciones (sobre 10); 0 si es 4 o menos; calificación lineal de 4 a 7:
 - 4 veces (0 ptos)
 - 5 veces (0.5 ptos)
 - 6 veces (1 pto)
 - 7 veces (1.5 ptos)
- Reactivo-deliberativo: toda la puntuación si se sobrevive + se recogen todas las gemas necesarias + se llega al portal en ≥ 7 ejecuciones (sobre 10); 0 si es 4 o menos; calificación lineal de 4 a 7:
 - 4 veces (0 ptos)
 - 5 veces (1 pto)
 - 6 veces (2 ptos)
 - 7 veces (3 ptos)

Fecha de entrega:

18 de abril a las 23:59 horas

