UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

/UGR / decsai

GVG-IA

Técnicas de los Sistemas Inteligentes

Febrero 2019

Contents

1 GVG-IA: Instalación y sistema de carpetas

Para el desarrollo de la practica se usará el entorno de desarrollo GVG-AI. GVG-AI es un entorno creado para la *General Video Game AI Competition*, una competición anual en la que los participantes deben desarrollar un controlador (agente) capaz de resolver el mayor número de juegos posibles. Estos juegos varían en el genero (estrategia, aventuras, puzzles...) y en la

jugabilidad (entornos deterministas, NPCs enemigos, condiciones de victoria variables...).

Las siguientes secciones explicarán en detalle como descargar e instalar el entorno y como lanzar un juego de prueba. El proceso viene ilustrado con varias capturas de pantalla de la instalación de GVG-AI en Linux usando el entorno de desarrollo IntelliJ IDEA.

1.1 Instalación del entorno GVG-IA

1. Instalar Java Development Kit

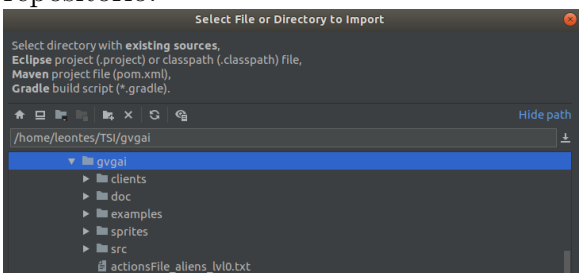
```
leontes@LeontesCitic:~$ sudo apt-get install default-jdk
```

2. Instalar IDE compatible (Eclipse o IntelliJ IDEA, recomendamos este ultimo).

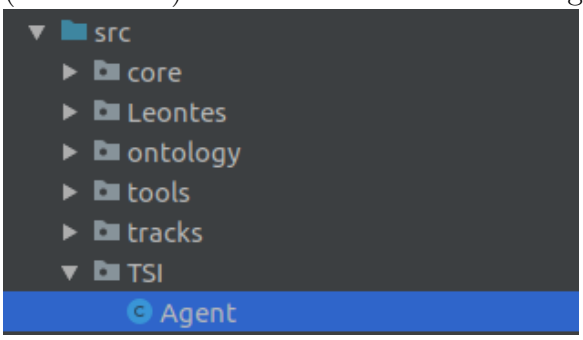
3. Descargar el entorno GVG-IA de git-hub. Recomendación: Borrar la carpeta Clients incluida en el repositorio, contiene ficheros con funciones main que suelen provocar problemas con los IDEs.

```
leontes@LeontesCitic:/mnt/Datos/Dropbox/TSI$ git clone https://github.com/GAIGResearch/GVGAI
```

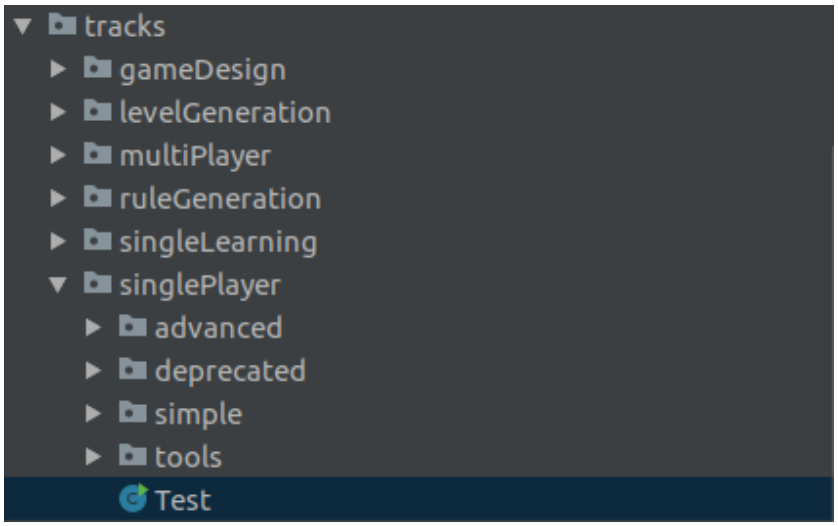
4. Crear un nuevo proyecto en el IDE a partir de los ficheros fuente del repositorio.



5. Crear una nuevo paquete java en SRC con el nombre "practica_busqueda" (sin comillas). Crear un fichero llamado Agent.Java dentro del paquete.



Una vez instalado GVG-IA y cargado en un proyecto dentro del IDE se puede comprobar que todo esta correcto lanzando el fichero test.java localizado en src.tracks.singleplayer. Esto lanzará G-IA con el juego Space Invaders para jugarlo usando el teclado. La nave se mueve las flechas y dispara con el espacio.



Para cambiar el nivel del juego hay que modificar la variable *levelIdx* en la línea 40 en el archivo tracks.singleplayer.Test. Por otro lado, haciendo lo mismo con la variable *gameIdx* en la línea 39 de tracks.singleplayer.Test se cambia el propio juego. Una lista completa de juegos y niveles de los mismos

se puede encontrar en el fichero all_games.csv en la carpeta *examples* del repositorio de GVG-IA.

Para jugar de forma autónoma con GVG-IA usando un controlador propio se deben seguir los siguientes pasos:

1. Comentar la línea 49 de tracks.singleplayer.Test

```
1     ArcadeMachine.playOneGame(game, level1, recordActionsFile, seed);
```

2. Descomentar la línea 52 de tracks.singleplayer.Test

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

3. Crear un objeto de tipo string llamado controlador inicializado como "NombreAlumno.Agent"

```
1 String controlador = "TSI.Agent";
```

4. Cambiar la variable "sampleRHEAController" por "controlador" en la línea 52 de tracks.singleplayer.Test

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

5. Cambiar en valor de la variable "gameIdx" por 11 en la línea 39 de tracks.singleplayer.Test

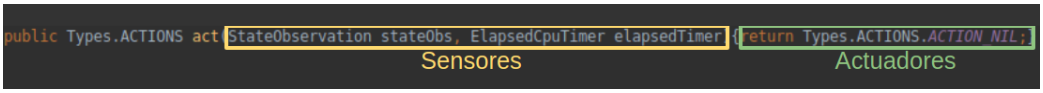
```
1 int gameIdx = 11;
```

6. Finalmente, ejecutar con el IDE.

2 Agent.java

GVG-IA controla los avatares de los distintos juegos mediante el uso de agentes inteligentes. Un agente inteligente es un componente software autónomo que recibe información sobre el entorno a través de unos sensores y aplica una acción a dicho entorno usando unos actuadores. Toda acción generada por el agente inteligente tiene el fin de alcanzar un objetivo concreto. En el caso concreto de GVG-IA los sensores se pasan por parámetro de las distintas

funciones de Agent.java, mientras que los actuadores son los posibles valores de retorno de la función act (explicada más adelante).



Para el desarrollo de la practica, el agente debe programarse usando el lenguaje de programación Java y debe heredar de la clase abstracta Abstract-Player. El agente debe implementar tres métodos: el constructor de la clase Agent, la función init y la función act.

El constructor se define como:

```
1 public Agent(StateObservation stateObs, ElapsedCpuTimer elapsedTimer){}
```

Este método recibe como parámetros información de los sensores y construye un nuevo objeto de la clase Agent. La información de los sensores puede usarse para instanciar los componentes del agente.

Opcionalmente, se puede definir un método init que se llama una sola vez al principio del juego para inicializar el agente. Principalmente, este método sirve como apoyo al constructor, funcionando de forma independiente a él y permitiendo inicializar diversos componentes del agente fuera del proceso de construcción del objeto agente.

```
1 public void init(StateObservation stateObs, ElapsedCpuTimer elapsedTimer){}
```

Finalmente, la función act es el cerebro del agente: Recibe información de los sensores, computa la mejor acción que realizar y se la notifica a GVG-IA. La función act se llama por cada tick del juego y recibe como parámetros información sobre el estado del mundo justo en el momento de ser llamada.

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

2.1 Sensores

Un agente en GVG-IA recibe información del entorno mediante dos parámetros definidos en todas sus funciones:

1. Un objeto de tipo StateObservation
2. Un objeto de tipo ElapsedCpuTimer

2.1.1 StateObservation

Los objetos de tipo StateObservation son pequeñas "fotos" del mundo en un instante dado. En el caso concreto de la función act, el parámetro stateobs es una representación del juego justo en el momento de llamar a la función. En el resto de funciones este parámetros contiene información sobre el estado inicial del juego.

Los objetos StateObservation contienen información sobre el avatar del jugador, sobre el mundo (localizaciones, mapa, ...) e incluso son capaces de crear una nueva observación simulando la aplicación de una acción.

Al final del documento se aporta un anexo dónde se listarán algunas de las funciones más importantes de la clase StateObservation, agrupándolas por su cometido y dando una breve descripción de su utilidad. Al descargar el entorno de desarrollo GVG-IA junto con los ficheros fuente se adjunta un JavaDoc (dentro de la carpeta doc en la raíz del repositorio GVG-IA). Para poder abrirlo solo hace falta cargar el fichero index.html, incluido dentro de la carpeta doc, usando un navegador web. Este Javadoc contiene información detallada (parámetros, valores de retorno, etc..) sobre todas las clases del entorno y sus funciones.

Nota: Todas las coordenadas obtenidas mediante los objetos de la clase StateObservation son dadas en píxeles. Para poder hacer cálculos y compararlas con la matriz mapa del mundo se debe hacer una conversión previa a filas y/o columnas.

2.1.2 ElapsedCpuTimer

El parámetro elapsedTimer sirve al agente para controlar el tiempo que ha invertido en su ejecución. GVG-IA controla los tiempos de ejecución de forma muy estricta, y si el agente es capaz de cumplir con los plazos de tiempo definidos por el entorno de desarrollo. Estos plazos de tiempo son: 1 segundo para el constructor, 1 segundo para la función init y 50 milisegundos para el método act. En caso de que no se cumpla un plazo, la ejecución se detiene y el agente queda descalificado. Un agente debe cerciorarse de que es capaz de cumplir los plazos de tiempo máximos y finalizar la tarea que este realizando siempre.

2.2 Acciones

GVG-IA tiene una serie de acciones predefinidas para todos los agentes. Estas acciones son las mismas para todos los juegos de la competición y funcionan de forma similar: ACTION_LEFT, ACTION_RIGHT, ACTION_UP y ACTION_DOWN mueven al avatar, ACTION_USE hace que el avatar realice una acción especial y ACTION_NIL sirve como acción por defecto que "no hace nada".

3 Boulder Chase

La practica se va a desarrollar usando el videojuego Boulder Chase implementado dentro de GVG-IA. El objetivo de este juego es el de recopilar 10 gemas antes de escapar del mapa. El mapa esta poblado por escorpiones y murciélagos que intentarán matar al jugador, así como piedras que pueden caer sobre él y aplastarlo. Los jugadores deben navegar entre los muros del mapa, usando su pala para abrirse camino entre la tierra que les impide avanzar y evitando los distintos peligros hasta tener suficientes gemas para poder huir.

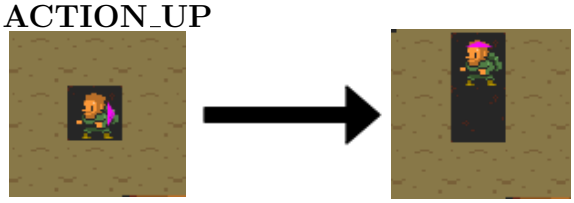
La tabla 1 muestra todos los elementos del juego, así como la codificación interna que usa el entorno de desarrollo en su modelo interno para representarlos. Finalmente, las acciones permitidas en el juego son todas las permitidas por el entorno GVG-IA: Moverse en las cuatro direcciones, usar(en este caso la pala) y estarse quieto. Un poco más abajo en este documento se detallan las acciones que puede realizar el avatar durante el juego, así como su resultado en el mundo.

ID de los elementos utilizados en el juego BoulderChase.

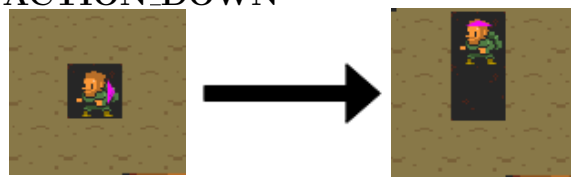
Id	Elemento del Grid	Id	Elemento del Grid
0	Muro	6	Gema
1	Jugador	7	Piedra
4	Tierra	10	Escorpión
5	Portal	11	Murciélago

3.1 Acciones del avatar

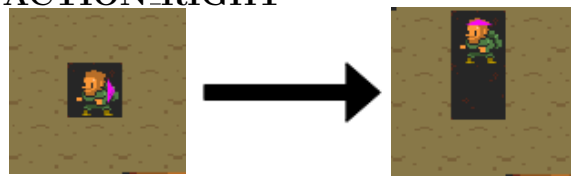
- **ACTION_UP**



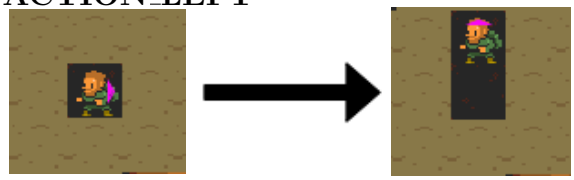
- **ACTION_DOWN**



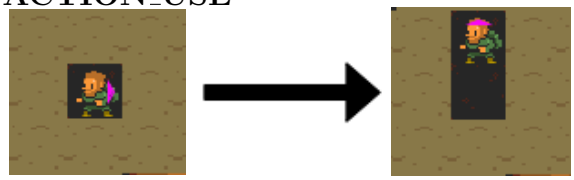
- **ACTION_RIGHT**



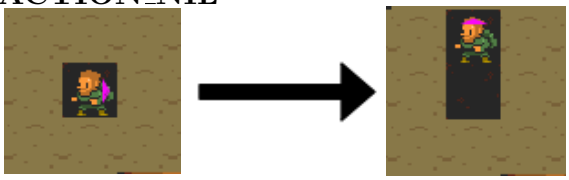
- **ACTION_LEFT**



- **ACTION_USE**



- **ACTION_NIL**



4 Ejemplo de agente

Las siguientes líneas muestran un ejemplo de agente desarrollado por los profesores de practicas de la asignatura. Este agente esta diseñado para efectuar la rutina más simple posible siguiendo una estrategia voraz: Busca la gema más cercana, traza un camino hacia ella, la coge y repite el proceso hasta tener 10. Una vez se han cogido 10 gemas el agente buscara el portal más cercano y tratará de ir hacia él. Este agente NO tiene en cuenta ni enemigos, ni piedras, ni ningún tipo de imprevisto, por lo que si ocurre algo que se salga de su control se quedará atascado o morirá. Este agente hace uso de un algoritmo de Búsqueda de Caminos ya implementado en GVG-IA.

Para poder usar este algoritmo se debe crear una variable de clase de tipo Pathfinder e inicializarla en el constructor indicándole el código de los obstáculos del juego. En el caso concreto de nuestro agente solo tendrá en cuenta como obstáculos los objetos inmóviles del juego (en este caso solo hay paredes) ya que las piedras están contempladas como misiles en vez de objetos. Esta lista de "Ids a evitar" se pasa al constructor del objeto Pathfinder en cual precalcula los distintos caminos posibles que se pueden realizar en el mapa. Finalmente, el constructor calcula también cual es el factor de escala que hay que aplicar a las coordenadas píxel que devuelven las distintas funciones de StateObservation para obtener coordenadas del grid del mapa de juego.

```
1     public Agent(StateObservation stateObs, ElapsedCpuTimer elapsedTimer) {
2         //Creamos una lista de IDs de obstaculos
3         ArrayList<Observation>[] obstaculos = stateObs.getImmovablePositions
      ();
4
5         ArrayList<Integer> tiposObs = new ArrayList<Integer> ();
6         for(ArrayList<Observation> obs:obstaculos){
7             tiposObs.add(obs.get(0).obsID);
8         }
9
10        tiposObs.add((int)'o');
11
12        //Se inicializa el objeto pf con las ids de los obstaculos
13        pf = new PathFinder(tiposObs);
14        pf.VERBOSE = false; // <- Activa o desactiva el modo la impresion
      del log
15
16        //Se lanza el algoritmo de pathfinding para poder ser usado en ACT
17        pf.run(stateObs);
18
```

```
19          //Calculamos el factor de escala entre mundos
20          fescala = new Vector2d((double) stateObs.getWorldDimension().width /
     stateObs.getObservationGrid().length ,
21                                 (double) stateObs.getWorldDimension().
    height / stateObs.getObservationGrid()[0].length);
22
23          //Ultima posicion del avatar
24          ultimaPos = new Vector2d(stateObs.getAvatarPosition().x / fescala.x,
     stateObs.getAvatarPosition().y / fescala.y);
25      }
```

Cuando al agente se le pide actuar este chequea si tiene un plan de ruta calculado hacia un objetivo. Si tiene uno, lo intenta ejecutar, en caso de no tenerlo calcula uno. Antes de comprobar esto, primero, comprueba si su localización actual es la que debe ser, en caso de que sea correcta se actualiza el plan de ruta actual(tras mandar una acción de movimiento el juego primero actualiza la orientación, por lo que a veces hay que mandar dos veces seguidas la misma orden para que el avatar se mueva). Si debe calcular un nuevo plan de ruta, el agente cuenta el número de gemas que lleva. Si es menor que 10 intenta buscar la gema más cercana y crea un plan de ruta hacía ella, si es igual a 10 lo que hace es buscar el portal más cercano.

Una vez con un plan de ruta el agente toma su posición actual y la posición destino y calcula la acción necesaria para ir de una a la otra. En caso de que haya había algún problema con el algoritmo de Búsqueda de Caminos se devuelve ACTION_NIL por defecto.

```
1       public Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer
    elapsedTimer){
2
3           //Obtenemos la posicion del avatar
4           Vector2d avatar =  new Vector2d(stateObs.getAvatarPosition().x /
    fescala.x, stateObs.getAvatarPosition().y / fescala.y);
5
6           //Actualizamos el plan de ruta
7           if(((avatar.x != ultimaPos.x) || (avatar.y != ultimaPos.y)) && !path
    .isEmpty()){
8               path.remove(0);
9           }
10
11          //Calculamos el numero de gemas que lleva encima
12          int nGemas = 0;
13          if(!stateObs.getAvatarResources().isEmpty()){
14              nGemas = stateObs.getAvatarResources().get(6);
15          }
16
17          //Si no hay un plan de ruta calculado...
18          if(path.isEmpty()){
```

```
19          //Si ya tiene todas las gemas se calcula uno al portal mas
    cercano. Si no se calcula a la gema mas cercana
20          if(nGemas == 10){
21              Vector2d portal;
22
23              //Se crea una lista de observaciones de portales, ordenada
    por cercania al avatar
24              ArrayList<Observation>[] posiciones = stateObs.
    getPortalsPositions(stateObs.getAvatarPosition());
25
26              //Se seleccionan el portal mas cercano
27              portal = posiciones[0].get(0).position;
28
29              //Se le aplica el factor de escala para que las coordenas de
     pixel coincidan con las coordenadas del grid
30              portal.x = portal.x / fescala.x;
31              portal.y = portal.y / fescala.y;
32
33              //Calculamos un camino desde la posicion del avatar a la
    posicion del portal
34              path = pf.getPath(avatar, portal);
35          }
36          else{
37              Vector2d gema;
38
39              //Se crea una lista de observaciones, ordenada por cercania
    al avatar
40              ArrayList<Observation>[] posiciones = stateObs.
    getResourcesPositions(stateObs.getAvatarPosition());
41
42              //Se selecciona la gema mas cercana
43              gema = posiciones[0].get(0).position;
44
45              //Se le aplica el factor de escala para que las coordenas de
     pixel coincidan con las coordenadas del grig
46              gema.x = gema.x / fescala.x;
47              gema.y = gema.y / fescala.y;
48
49
50              //Calculamos un camino desde la posicion del avatar a la
    posicion de la gema
51              path = pf.getPath(avatar, gema);
52          }
53      }
54
55
56      if(path != null){
57          Types.ACTIONS siguienteAccion;
58          Node siguientePos = path.get(0);
59
60          //Se determina el siguiente movimiento a partir de la posicion
    del avatar
61          if(siguientePos.position.x != avatar.x){
```

```
62                  if (siguientePos.position.x > avatar.x) {
63                      siguienteAccion = Types.ACTIONS.ACTION_RIGHT;
64                  }else{
65                      siguienteAccion = Types.ACTIONS.ACTION_LEFT;
66                  }
67              }else{
68                  if(siguientePos.position.y > avatar.y){
69                      siguienteAccion = Types.ACTIONS.ACTION_DOWN;
70                  }else{
71                      siguienteAccion = Types.ACTIONS.ACTION_UP;
72                  }
73              }
74
75              //Se actualiza la ultima posicion del avatar
76              ultimaPos = avatar;
77
78              //Se devuelve la accion deseada
79              return siguienteAccion;
80          }
81          else{
82              //Salida por defecto
83              return Types.ACTIONS.ACTION_NIL;
84          }
85
86      }
87
88  }
```

UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

/UGR/decsai

Anexo información StateObservation

Información sobre el avatar

Función	Descripción
getAvatarHealthPoints	Devuelve el número de puntos de vida actual del avatar
getAvatarMaxHealthPoints	Devuelve el número de puntos de vida máximos del avatar
getAvatarLimitHealthPoints	Devuelve el número de puntos de vida limite del avatar
getAvatarLastAction	Devuelve la última acción del avatar
getAvatarPosition	Devuelve la posición del avatar
getAvatarOrientation	Devuelve la orientación del avatar
getAvatarSpeed	Devuelve la velocidad del avatar
getAvatarResources	Devuelve el inventario del avatar
getAvatarType	Devuelve el código que representa al avatar en el modelo del mundo

Información sobre el juego

Función	Descripción
getGameState	Devuelve el estado del juego
isGameOver	Devuelve si el juego ha terminado o no
getGameWinner	Devuelve si el jugador ha ganado o no

Acciones del jugador

Función	Descripción
getAvailableActions	Devuelve una lista de las acciones posibles
advance	Devuelve el resultado de aplicar una acción una observación

Mapa del juego

Función	Descripción
getWorldDimension	Devuelve el tamaño del mundo
getObservationGrid	Devuelve el mapa del mundo

Observaciones del mundo

Función	Descripción
getNPCPositions	Devuelve la lista de NPCs de la observación
getMovablePositions	Devuelve la lista de objetos móviles de la observación
getImmovablePositions	Devuelve la lista de objetos inmóviles de la observación
getIResourcesPositions	Devuelve la lista de recursos de la observación