UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

/UGR/decsai

GVG-AI

Técnicas de los Sistemas Inteligentes

Febrero 2020

Contents

1 GVG-AI: Instalación y sistema de carpetas

Para el desarrollo de la práctica 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 cómo descargar e instalar el entorno y cómo 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-AI

1. Instalar Java Development Kit

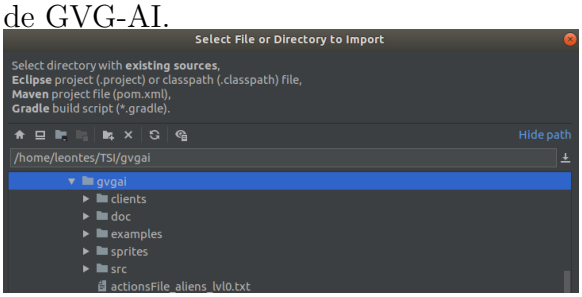
```
leontes@LeontesCitic:~$ sudo apt-get install default-jdk
```

2. Instalar IDE compatible (por ejemplo, Eclipse o IntelliJ IDEA).

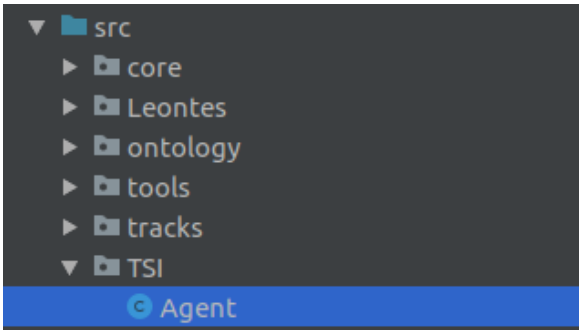
3. Descargar el entorno GVG-AI de GitHub. Recomendación: Borrar la carpeta `clients` incluida en el repositorio, contiene ficheros con funciones main que pueden provocar problemas con los IDEs.

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

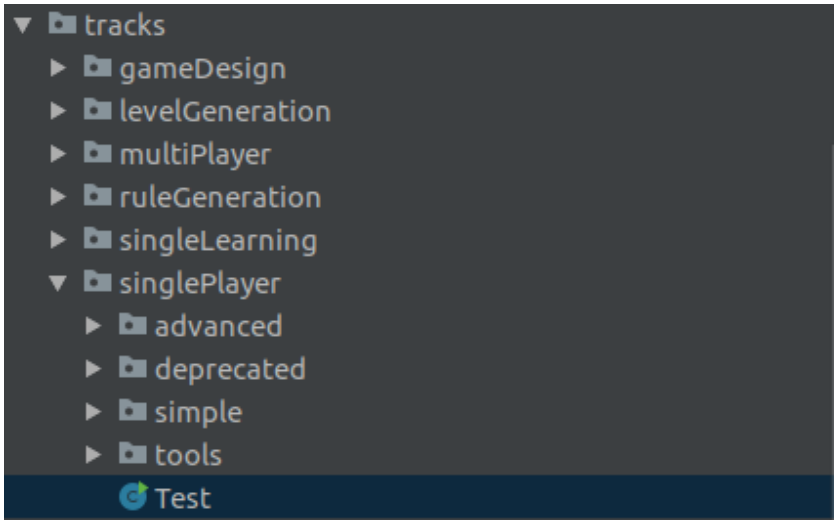
4. Crear un nuevo proyecto Java en el IDE a partir de los ficheros fuente del repositorio. Es decir, la ruta del proyecto debe apuntar al directorio de GVG-AI.



5. Crear un nuevo paquete Java en `src` (llamado, por ejemplo, `TSI`). Crear un fichero llamado `Agent.java` dentro del paquete.



Una vez instalado GVG-AI 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 GVG-AI con el juego Space Invaders para jugarlo usando el teclado (la nave se mueve usando las flechas y dispara usando el espacio).



Para cambiar el nivel del juego hay que modificar la variable `levelIdx` en la línea 40 en el archivo `Test.java` del paquete `tracks.singlePlayer`. Por otro lado, cambiando el valor de la variable `gameIdx` en la línea 39 de ese fichero se cambia el propio juego. Una lista completa de juegos y niveles de los mismos se puede encontrar en el fichero `all_games_sp.csv` en la carpeta `examples` del proyecto.

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

1. Comentar la línea 49 de `tracks.singlePlayer.Test`

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

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

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

3. Crear un objeto de tipo `String` llamado `controlador` inicializado con la ruta a la clase `Agent` creada anteriormente. Siguiendo el ejemplo anterior:

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

4. Cambiar la variable `sampleRHEAController` por `controlador` en la línea 52 de `tracks.singlePlayer.Test`:

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

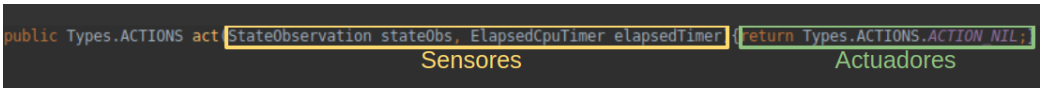
5. Cambiar en la línea 39 de `tracks.singlePlayer.Test` el valor de la variable `gameIdx=11`:

```
39   int gameIdx = 11;
```

6. Finalmente, ejecutar con el IDE.

2 Agent.java

GVG-AI 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-AI 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 práctica, el agente debe programarse usando el lenguaje de programación Java y debe implementar, al menos, dos métodos: el constructor de la clase `Agent` y la función `act`. Opcionalmente, también se puede implementar el método `init`.

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-AI. 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-AI 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 donde 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-AI junto con los ficheros fuente se adjunta un JavaDoc (dentro de la carpeta `doc` en la raíz del repositorio GVG-AI). Para poder abrirlo solo hace falta cargar el fichero `index.html`, incluido dentro de la carpeta `doc`, usando cualquier 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. En muchos casos, será conveniente traducir estas coordenadas en píxeles a las coordenadas en el grid (es decir, filas y columnas). En la última sección de este tutorial se da un ejemplo de esta conversión.

2.1.2 ElapsedCpuTimer

El parámetro `elapsedTimer` le sirve al agente para controlar el tiempo que ha invertido en su ejecución. GVG-AI controla los tiempos de ejecución de forma muy estricta, y comprueba si el agente es capaz de devolver una acción dentro del tiempo establecido (en caso contrario, es descalificado). Estos tiempos son: 1 segundo para el constructor, 1 segundo para la función `init` y 40 milisegundos para el método `act`.[1] En caso de que el agente

[1]En el caso de que el agente tarde entre 40 y 50 milisegundos en computar la acción del método `act`, GVG-AI le penaliza cambiando dicha acción por una acción nula, pero permitiéndole continuar la partida. En caso de tardar más de 50 milisegundos, entonces

sobrepase alguno de estos tiempos, 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 siempre finalizar la tarea que esté realizando.

2.2 Acciones

GVG-AI 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 Dash

La práctica se va a desarrollar usando el videojuego *BoulderDash* implementado dentro de GVG-AI. El objetivo de este juego, que no el de la práctica, es el de recopilar 10 gemas antes de escapar del mapa. El mapa está 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, recogiendo las gemas necesarias antes de salir por el portal y evitando los distintos peligros del mapa (piedas y enemigos).

La tabla 1 muestra todos los elementos del juego, así como la codificación que usa el entorno de desarrollo en su modelo interno para representarlos.

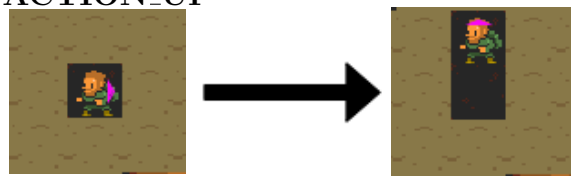
Id	Elemento del Grid	Id	Elemento del Grid
0	Muro	6	Gema
1	Jugador	7	Piedra
4	Tierra	10	Escorpión
5	Portal	11	Murciélago

Table 1: ID de los elementos utilizados en el juego *BoulderDash*.

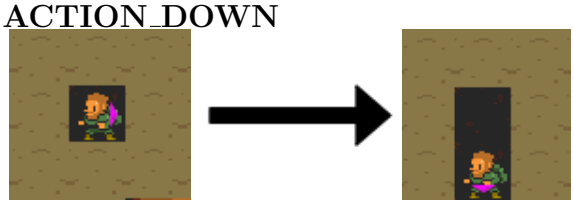
queda descalificado.

Finalmente, las acciones permitidas en el juego son todas las permitidas por el entorno GVG-AI: moverse en las cuatro direcciones, y estarse quieto. Un poco más abajo en este documento se detallan las acciones del avatar necesarias para realizar la práctica, así como su resultado en el mundo.

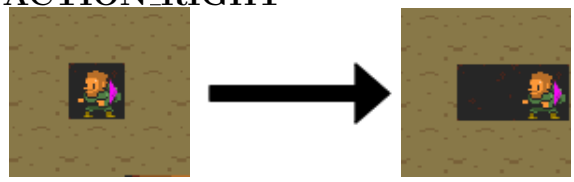
- **ACTION_UP**



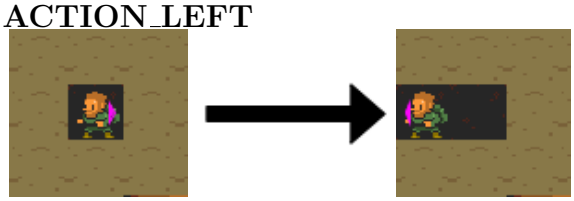
- **ACTION_DOWN**



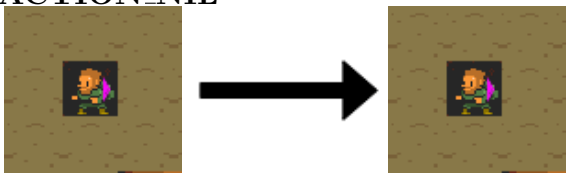
- **ACTION_RIGHT**



- **ACTION_LEFT**



- **ACTION_NIL**



4 Ejemplo de agente

Las siguientes líneas muestran un ejemplo de agente desarrollado por los profesores de prácticas de la asignatura para un juego de GVG-AI. En concreto, como ejemplo, se utiliza el primer nivel del juego *CamelRace* (`gameIdx = 15; levelIdx = 0`).

Este agente está diseñado para buscar de modo *greedy* (voraz o avaro, en castellano) la mejor acción para llegar lo antes posible a la puerta de salida más cercana. Se trata de un juego/mapa sin obstáculos, ni enemigos que puedan dañarte. De este modo, lo primero que se debe hacer es estima cuál es la puerta más cercana. Y, a continuación, se debe calcular a qué distancia de la puerta nos quedaríamos si tomásemos alguna de las cuatro acciones disponibles para desplazarse (ACTION_UP, ACTION_DOWN, ACTION_LEFT, ACTION_RIGHT). La acción que debe encontrar es sistemáticamente ACTION_RIGHT, dado que la puerta más próxima es aquella que se encuentra al fondo del mapa a la derecha, justo frente al agente/avatar. Se trata de un controlador muy sencillo en un entorno muy simple, pero que nos ayudará a familiarizarnos con algunas de las principales funcionalidades necesarias para llevar a cabo esta práctica. Es importante remarcar que este agente NO hace uso de ningún algoritmo de búsqueda heurística.

En un primer momento, se inicializan todas las variables del agente en el constructor. En concreto:

- Se calcula el factor para transformar las coordenadas píxel (en las que, por defecto, se proporcionan las posiciones de los avatares, portales, gemas,...) en coordenadas del *grid*. Es decir, coordenadas que podemos comprender visualmente observando el tablero del juego.

- Se recupera la lista de posiciones de portales, ordenada de modo ascendente por cercanía al avatar.
- Se selecciona el portal más próximo al avatar.

```
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);
}
```

Cuando al agente se le pide actuar, éste utiliza el factor de escala para conocer la posición del avatar (en este caso, del camello), prueba las cuatro acciones de desplazamiento a su disposición y verifica cuál de ellas, de acuerdo a la distancia Manhattan, le acerca más al portal de salida. Es decir, calcula las cuatro nuevas posiciones hipotéticas, estima la distancia, y devuelve la acción correspondiente. La principal dificultad de este sencillo ejemplo consiste en:

- Transformar las coordenadas píxel a coordenadas grid, para lo cual calculamos el factor de escala anteriormente mencionado.
- Comprender el sistema de coordenadas del mundo representado, en donde las Xs corresponden a las columnas, las Ys a las filas, y en donde el punto (0, 0) se encuentra en la esquina superior izquierda del tablero. En este caso concreto, se trata de un mapa con 48 columnas y 9 filas, en donde la esquina inferior derecha se corresponde con el punto (47, 8).

```
@Override
public ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer
  ) {
```

```
3         //Posicion del avatar
4         Vector2d avatar =  new Vector2d(stateObs.getAvatarPosition().x /
    fescala.x,
5                 stateObs.getAvatarPosition().y / fescala.y);
6
7         //Probamos las cuatro acciones y calculamos la distancia del nuevo
    estado al portal.
8         Vector2d newPos_up = avatar, newPos_down = avatar, newPos_left =
    avatar, newPos_right = avatar;
9         if (avatar.y - 1 >= 0) {
10          newPos_up = new Vector2d(avatar.x, avatar.y-1);
11        }
12        if (avatar.y + 1 <= stateObs.getObservationGrid()[0].length-1) {
13          newPos_down = new Vector2d(avatar.x, avatar.y+1);
14        }
15        if (avatar.x - 1 >= 0) {
16          newPos_left = new Vector2d(avatar.x - 1, avatar.y);
17        }
18        if (avatar.x + 1 <= stateObs.getObservationGrid().length - 1) {
19          newPos_right = new Vector2d(avatar.x + 1, avatar.y);
20        }
21
22        //Manhattan distance
23        ArrayList<Integer> distances = new ArrayList<Integer>();
24        distances.add((int) (Math.abs(newPos_up.x - portal.x) + Math.abs(
    newPos_up.y-portal.y)));
25        distances.add((int) (Math.abs(newPos_down.x - portal.x) + Math.abs(
    newPos_down.y-portal.y)));
26        distances.add((int) (Math.abs(newPos_left.x - portal.x) + Math.abs(
    newPos_left.y-portal.y)));
27        distances.add((int) (Math.abs(newPos_right.x - portal.x) + Math.abs(
    newPos_right.y-portal.y)));
28
29        // Nos quedamos con el menor y tomamos esa accion.
30        int minIndex = distances.indexOf(Collections.min(distances));
31        switch (minIndex) {
32          case 0:
33            return Types.ACTIONS.ACTION_UP;
34          case 1:
35            return Types.ACTIONS.ACTION_DOWN;
36          case 2:
37            return Types.ACTIONS.ACTION_LEFT;
38          case 3:
39            return Types.ACTIONS.ACTION_RIGHT;
40          default:
41            return Types.ACTIONS.ACTION_NIL;
42        }
43
44    }
```

UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

/UGR/decsai

Anexo información StateObservation

Table 2: **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

Table 3: **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

Table 4: **Acciones del jugador**

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

Table 5: **Mapa del juego**

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

Table 6: **Observaciones del mundo**

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