75.68 SISTEMAS DE SOPORTE PARA CELDAS DE PRODUCCIÓN FLEXIBLE

SISTEMA INTELIGENTE AUTÓNOMO QUE JUEGA AL SOKOBAN DE FORMA COPERATIVA

2do cuatrimestre, 2016

Autor: Ing. Juan Manuel Rodríguez

Índice de contenido

Introducción y motivación

El siguiente trabajo práctico está basado en la competencia: The General Video Game AI Competition" del año 2016. Dicha competencia busca promover y explorar el problema general que existe de crear un Agente Inteligente capaz de jugar diferentes vídeo juegos. Es decir que, al igual que un ser humano, se le presente un vídeo juego nuevo y sea capaz de aprender sus reglas, jugarlo y ganarlo. Explora además de los juegos de un solo jugador, la variante de los juegos cooperativos y la creación automática de niveles para cualquier juego dado. Dicha competencia está patrocinada por Google DeepMind.

Objetivo del trabajo practico

El objetivo del trabajo práctico de la materia es construir un agente inteligente autónomo que juegue al Sokoban de forma cooperativa. Se trata no solo de desarrollar un sistema capaz de jugar correctamente al Sokoban sino de que lo haga aprendiendo.

Sobre el Sokoban

Para aquellos que no estén familiarizados con este vídeo juego clásico, pueden jugarlo de manera online en el siguiente sitio: http://sokoban.info/. Se trata de ubicar unas cajas, originalmente esparcidas por el nivel, en un sitio especifico identificado con una marca. El mapa se ve desde arriba y el jugador puede moverse en dos dimensiones. Solo tiene cuatro acciones posibles: atrás, adelante, izquierda y derecha. Las cajas pueden ser empujadas pero no tiradas en el sentido de jalar o arrastrar.

Entregas

El trabajo practico se deberá ir realizando durante el cuatrimestre, una vez que él código esté terminado, es decir la lógica del agente inteligente esté correctamente codificada y este haya sido entrenado con el Sokoban el tiempo suficiente como para tener un comportamiento racional, se deberá enviar por email una copia del paquete de Java creado con las clases Agente y cualquier otra clase utilitaria creada. Para ver cómo crear el paquete y la clase Agente ver más abajo los tutoriales.

Además el conocimiento del agente deberá ser almacenado en un archivo de texto con formato JSON. Es decir, las teorías que el agente generó durante su entrenamiento. Este archivo también deberá ser entregado junto con el código.

Juntamente con las clases y el archivo JSON con el conocimiento del agente se deberá entregar un informe sencillo y conciso explicando la estrategia utilizada, las dificultades encontradas y la evolución del agente. Dijo informe deberá tener entre 10 y 15 páginas en tamaño A4 y deberá estar en el formato *lectures notes in computer science* (LNCS) que es un formato utilizado por Spring para la publicación de artículos científicos. Dejo a continuación *links* con ejemplos y plantillas tanto en Word como en Latex del formato: LNCS:

- LNCS Latex: https://goo.gl/tKiYW5
- LNCS Word: https://goo.gl/xw5i8d
- LNCS Word 2007: https://goo.gl/aa4of4

Si el trabajo está aprobado el alumno está en condiciones de rendir el coloquio, que será una defensa oral del trabajo.

Fecha de entrega

El TP podrá ser entregado en cualquier momento del cuatrimestre, pero la defensa del trabajo se hará solo en las fechas de coloquio.

Los plazos de entrega del TP son los de cualquier materia, es decir tiene un año desde finalizada la cursada.

Modo de evaluación

El paquete Java con las clases entregadas será copiado dentro de una versión limpia del proyecto para ver el comportamiento del Agente Inteligente desde cero y con las teorías que aprendió. Luego si el agente inteligente parece comportarse debidamente será puesto a trabajar de forma conjunta con otro agente inteligente creado por otro grupo para evaluar su comportamiento. Esta última prueba solo servirá para definir la nota, no es un criterio absoluto de aprobación.

Resumen

La entrega consiste en:

- Clases Java dentro del paquete del grupo, la clase Agent y cualquier clase auxiliar.
- El archivo JSON que almacena el conocimiento del agente en forma de teorías
- El informe en LNCS

Se aprueba si:

- El Argente inteligente funciona de forma racional y es capaz de aprender
- El informe cumple con los requisitos pedidos
- El alumno se presentó al coloquio de defensa del trabajo practico y supo responder correctamente

Especificaciones técnicas y tutoriales para empezar

El desarrollo del sistema se hará utilizando un *framework* hecho en Java provisto por la gente del concurso. Se puede utilizar cualquier IDE de desarrollo y cualquier SO. El proyecto original se provee como un proyecto de JIDEA, pero la cátedra ofrecerá una versión para Eclipse, de todos modos la decisión sobre la ide de desarrollo corresponde al alumno. En resumen:

- Lenguaje: Java
- SO: Linux, Windows, Mac, etc.
- IDE: JIDEA; Eclipse, etc.

Instrucciones para la instalación del ambiente de desarrollo utilizando Eclipse

Para este mini tutorial se utilizó la versión de Eclipse MARS, para J*ava Developers*, se muestra a continuación la versión de la IDE (se puede ver desde Help > About Eclipse):

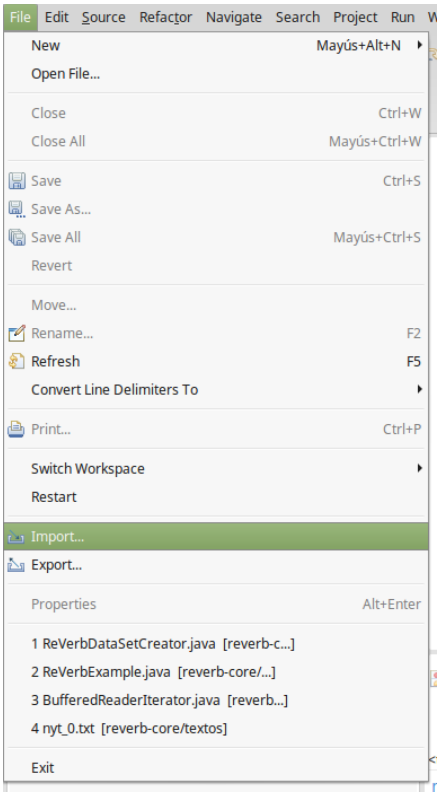
Requisitos: Tener instalada una versión de la Java Virtual Machine 1.7 o superior.

1. Descargar la versión e Eclipse Mars para el SO que cada uno utiliza de la siguiente URL: http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/marsr

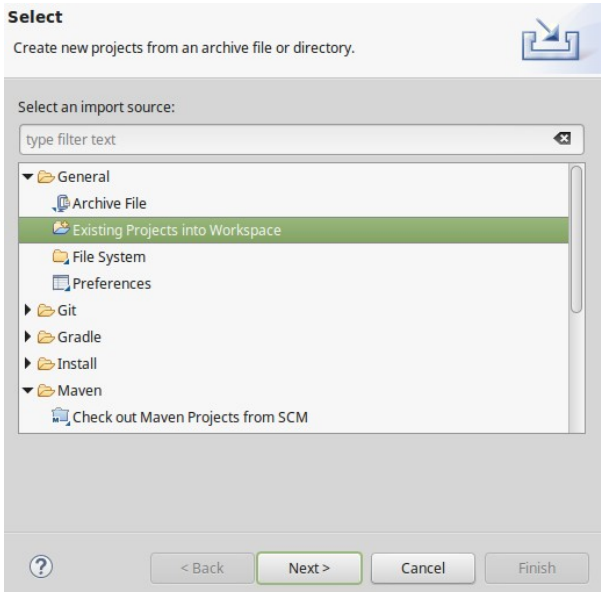
2. Descomprimir el archivo que descarguen en un directorio cualquiera del SO. Luego ejecutar

el archivo binario **eclipse** o **eclipse.exe** según sea el SO.

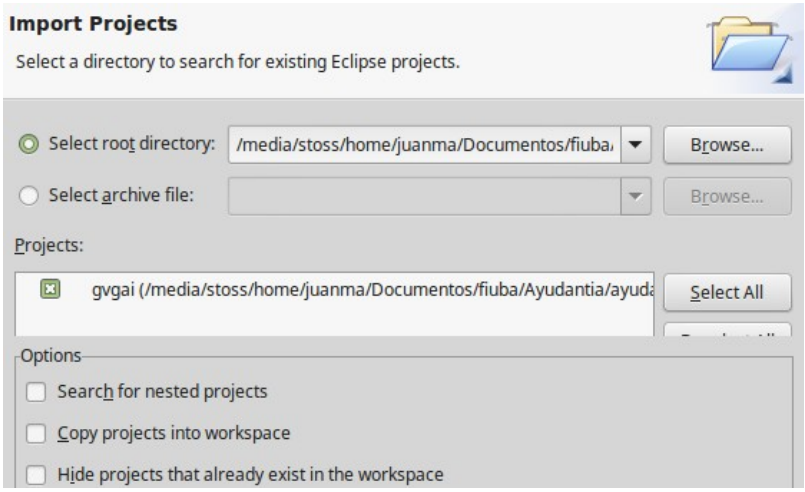
3. Bajar el *framework* desde alguno de los siguientes links:
    a. Versión para Eclipse: https://goo.gl/lhGDKf
    b. Versión JIDEA:https://goo.gl/xzJq4R
    Utilizar la versión de Eclipse para este tutorial.
4. El archivo anterior es un ZIP. Descomprimirlo en un directorio cualquiera. Creará el subdirectorio “gvgai”
5. Ir al eclipse e importar el proyecto anterior, ver los pasos a continuación:
    a. File => Import



    b. Elegir la opción “Existing projects into Workspace”

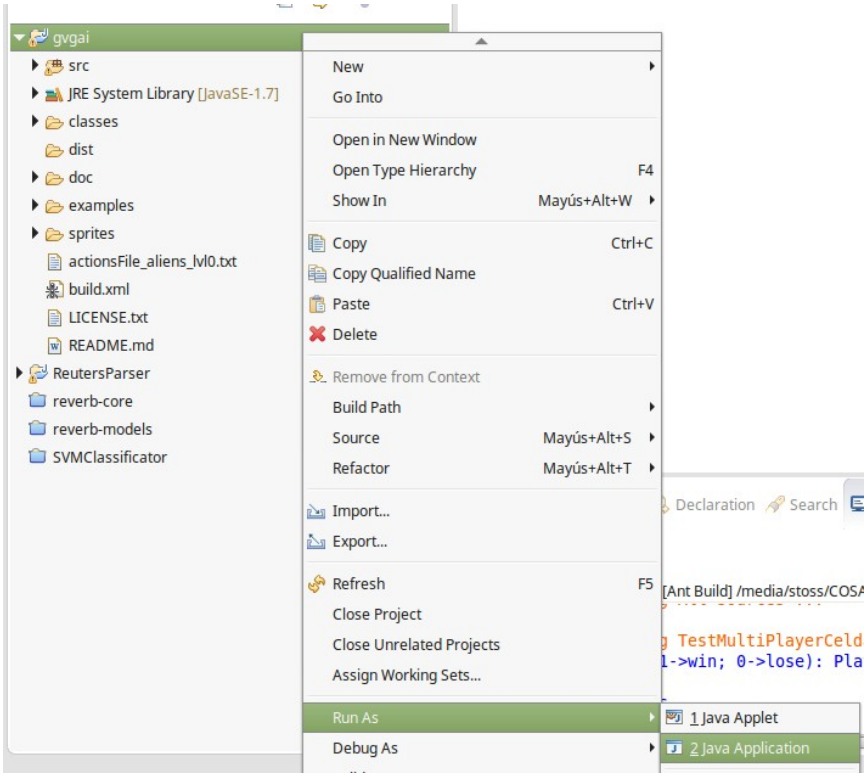


    c. En el siguiente dialogo buscamos el directorio en el cual descomprimimos la carpeta: “gvgai”

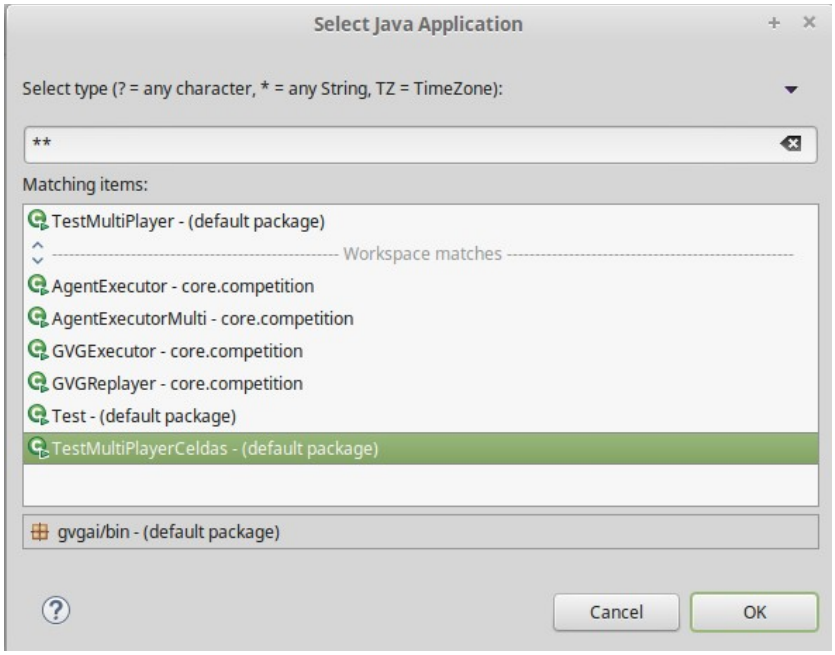


d. Luego hacemos click en “Finish”

6. El proyecto aparecerá en el arbol de la izquierda. Hacer click derecho sobre el mismo, y buscar la opción “Run As...” y escoger “Java Aplication”

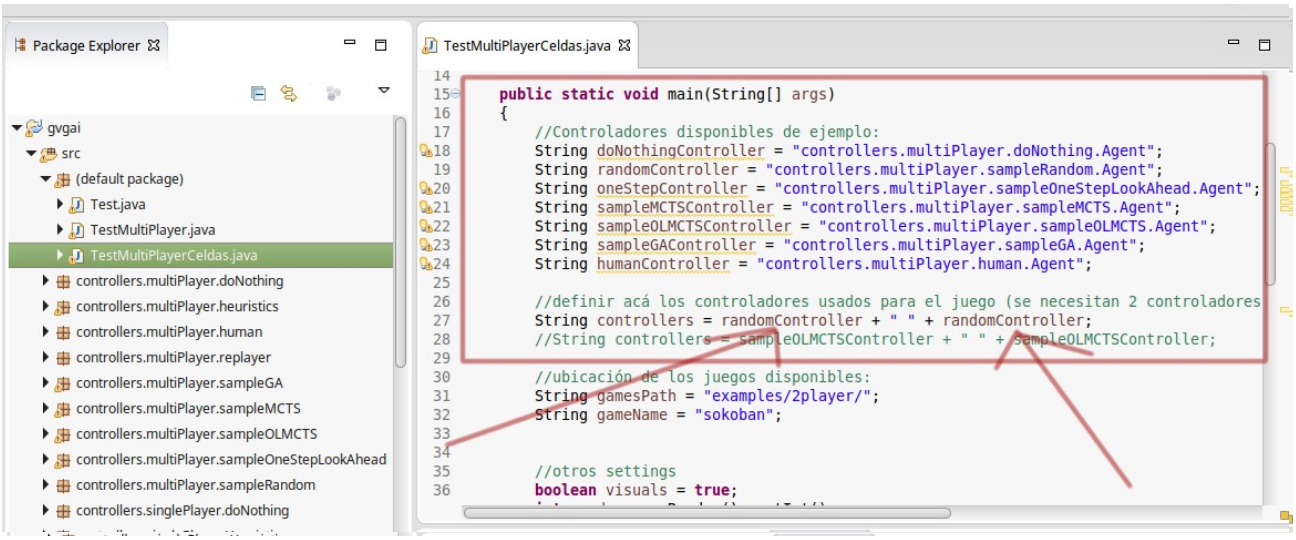


7. En el dialogo que aparece, elegimos la clase: “TestMultiPlayerCeldas” y hacemos click en “OK”. Esto servirá para lazar la aplicación y observar un comportamiento de ejemplo.



Instrucciones para la creación del Agente Inteligente, usando Eclipse

1. Si buscamos en el árbol de la izquierda, la clase: “TestMultiPlayerCeldas”, dentro de la carpeta “src”, podremos ver la lógica que sirve para invocar a los agentes inteligentes que están siendo usados.

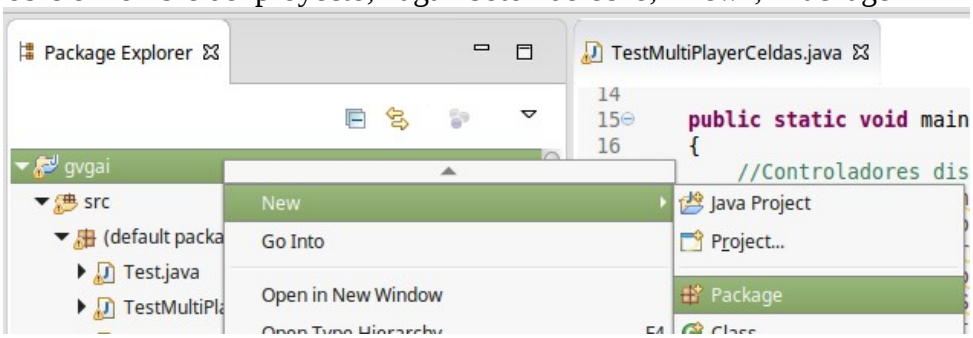


En la imagen anterior podrán ver que al principio aparecen muchas clases listadas, escritas como Strings, estas clases corresponden a Agentes Inteligentes (más o menos inteligentes en verdad) ya que hay una que no hace nada, otra que solo da un paso adelante, una que mueve al azar, etc. Son solo clases de ejemplos, para que entiendan como se construyen.

En la línea siguiente, hay otro String que concatena el nombre de dos de las clases anteriores, en este ejemplo son “randomController” ambas. Eso quiere decir que cuando ejecutamos el juego el comportamiento que estamos viendo para cada uno de los agentes es el que está implementado en la clase “randomController”

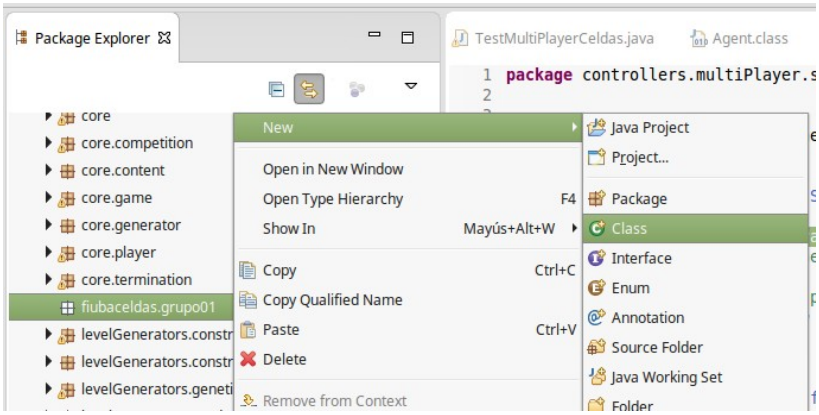
2. Para crear una clase propia primeramente vamos a crear un paquete. Creen un paquete utilizando el número de grupo. Por ejemplo si son el grupo: “01” creen un paquete llamado: “fiubaceldas.grupo01”. Para ello sigan los pasos a continuación:

    1. Sobre el nombre del proyecto, hagan botón derecho, “New”, “Package”

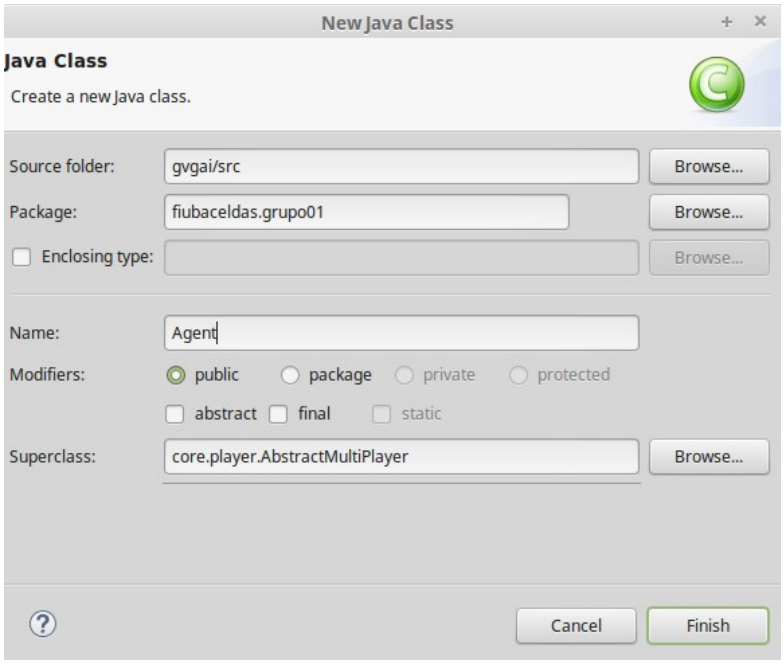


    2. En la ventana de dialogo que aparece ingresamos el nombre del paquete

3. El paso anterior debería haber creado un paquete vacío en el árbol de la izquierda. Ahora debemos crear una clase dentro del paquete.
    1. Ubicamos el paquete llamado: “fiubaceldas.grupo01” y hacemos click con el botón derecho para crear una nueva clase:



2. Llamamos a la clase “Agent” y especificamos que heredará de “core.player.AbstractMultiPlayer”



4. Buscamos la clase recién creada en el árbol de la izquierda, veremos que implementa un solo método “act” y que devuelve una acción:

```
package fiubaceldas.grupo01;

import core.game.StateObservationMulti;
import core.player.AbstractMultiPlayer;
import ontology.Types.ACTIONS;
import tools.ElapsedCpuTimer;

public class Agent extends AbstractMultiPlayer {

	@Override
	public ACTIONS act(StateObservationMulti stateObs, ElapsedCpuTimer elapsedTimer) {
		// TODO Auto-generated method stub
		return null;
	}

}
```

Como se puede observar el método está recibiendo dos parámetros:

- StateObs: El estado actual.
- ElapsedTime: El tiempo de CPU que transcurrió desde la última observación.

Por último vemos que está devolviendo “null” lo cual no es una ACCION valida. Por ello hay que

cambiar “null” por **ACTIONS.ACTION_NIL** que sí es una acción valida. Equivale a no hacer nada.

5. Creamos un constructor, ya que si no agregamos un constructor explícitamente, nos arrojará un error más adelante. Añadimos las siguientes líneas de código en la línea 12:

```
public Agent(StateObservationMulti stateObs, ElapsedCpuTimer elapsedTimer, int playerID) {
}
```

6. Guardamos los cambios con CTRL+S y buscamos el archivo que habíamos visto anteriormente: “TestMultiPlayerCeldas” (Una forma rápida de ubicar archivos es usando el atajo de teclado CTRL+SHIFT+R)
7. Agregamos la siguiente línea debajo de todos los Strign con nombres de controladores de ejemplo, (línea 25):

```
String grupo01Controller = "fiubaceldas.grupo01.Agent";
```

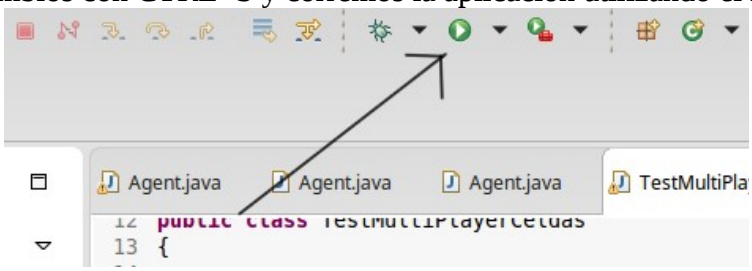
8. Cambiamos los dos controladores que se estaban usando para probar, en la línea 28, donde estaba:

```
String controllers = randomController + " " + randomController;
```

Lo cambios por:

```
String controllers = grupo01Controller + " " + grupo01Controller;
```

9. Guardamos los cambios con CTRL+S y corremos la aplicación utilizando el icono verde de play:



El juego debería arrancar, pero nuestros agentes no van a hacer nada. Solo estarse quietos, ya que nuestro método ACT devuelve siempre la misma acción que es no hacer nada.