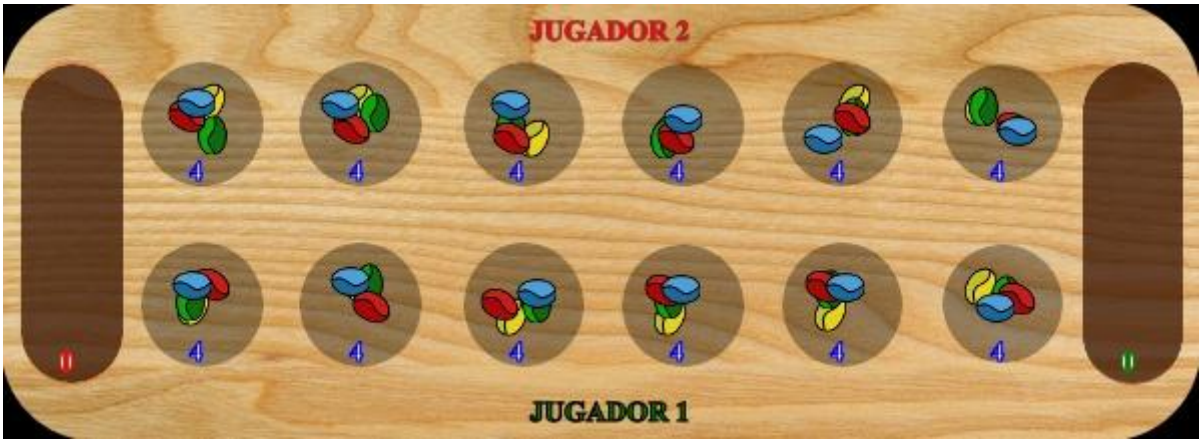
INTELIGENCIA ARTIFICIAL

E.T.S. de Ingenierías Informática y de Telecomunicación

Práctica 3

Agentes en entornos con adversario



DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

UNIVERSIDAD DE GRANADA

Curso 2017-2018

1. Introducción y objetivos

La tercera práctica de la asignatura ***Inteligencia Artificial*** consiste en el diseño e implementación de un agente deliberativo desplegado en un entorno multi-agente competitivo, en el que se encuentran dos agentes con objetivos contrapuestos que escogen sus acciones por turnos. Se trabajará con un simulador software. Para ello, se proporciona al alumno un entorno de programación, junto con el software necesario para simular el entorno.

2. Mancala

Los Mancala son una familia de juegos de tablero con fundamentos similares, procedentes de África (fundamentalmente del Antiguo Egipto) y de Asia. Su nombre deriva de la palabra árabe manqala, que significa *mover*. En este tipo de juegos, existe un ***tablero*** que representa a un campo de cultivo, con ***casillas*** u hoyos de siembra donde depositar ***semillas***, y un almacén o ***granero*** para contabilizar las semillas de cada jugador.

Hoy día, Mancala se utiliza para referirnos a una variante específica de la familia de juegos con las siguientes características:

- El campo de cultivo o tablero está compuesto por dos filas de 6 hoyos o casillas dispuestas en paralelo, junto con 2 graneros (uno en cada extremo).
- Hay 2 jugadores (J1 y J2). Cada jugador posee uno de los graneros.
- Las 6 casillas de la misma fila pertenecen al mismo jugador. Inicialmente, cada casilla contiene 4 semillas (por tanto, en total hay 48 semillas en el juego).
- Las casillas se numeran de 1 a 6, contando desde el granero del jugador al que pertenecen.

La Figura 1 muestra un ejemplo de disposición inicial del juego, donde también se muestra cada casilla de cada jugador.

Figura 1

En cada turno, comenzando por el jugador 1 (J1), cada jugador realiza un movimiento de ***siembra***. La siembra consiste en seleccionar todas las semillas de una misma casilla del jugador e ir depositándolas de una en una en las casillas consecutivas, en orden descendiente, del propio jugador hasta llegar a su granero (si es posible). Si sobrasen semillas, estas comenzarían a repartirse de una en

una por las casillas del jugador opuesto, en orden descendente de casilla hasta llegar a la casilla 1 (inclusive) de dicho jugador. Si aún quedasen semillas por repartir, se repetiría este procedimiento partiendo desde la casilla 6 del jugador que realiza el turno. **OJO:** No depositamos nunca ninguna semilla en el granero del jugador opuesto.

Por ejemplo, partiendo del estado del juego de la Figura 1, y asumiendo que le toca jugar a J1, supongamos que este decide realizar siembra sobre su casilla 3. En este caso, recogería todas las semillas de la casilla 3 y las repartiría por las casillas 2 y 1 del mismo jugador, otra semilla se depositaría en su granero, y la última en la casilla 6 del jugador J2 (ver Figura 2). Llegaría entonces el turno de J2.

Figura 2

Si partimos de la Figura 2, con J2 teniendo el turno, supongamos que realiza la siembra de su casilla 1. En tal caso, vaciaría su casilla 1 y situaría una semilla en su granero (a la izquierda), poniendo las 3 restantes en las casillas 6, 5 y 4 de J1 (ver Figura 3). Pasaría el turno al jugador J1.

Figura 3

El juego transcurre hasta que uno de los dos jugadores no tiene semillas en ninguna de sus 6 casillas. En este caso, el otro jugador recogería todas las semillas que quedan en sus casillas y las llevaría a su granero. Ganaría el juego aquel jugador que, al finalizar, contenga mayor número de semillas en su granero. La Figura 4 muestra un ejemplo de esta situación, donde le toca el turno a J2.

Sólo tiene un hoyo con semillas (el 1), que permite depositar dicha semilla en el granero de J2. Al depositar la semilla del hoyo 1 en el granero, como J2 ya no puede seguir jugando (no tiene más semillas), todas las semillas de las casillas 4, 3 y 2 que quedan en los hoyos de J1 pasarían al granero de este mismo jugador (ver Figura 5). En este caso, gana J1 por 35 semillas a 13.

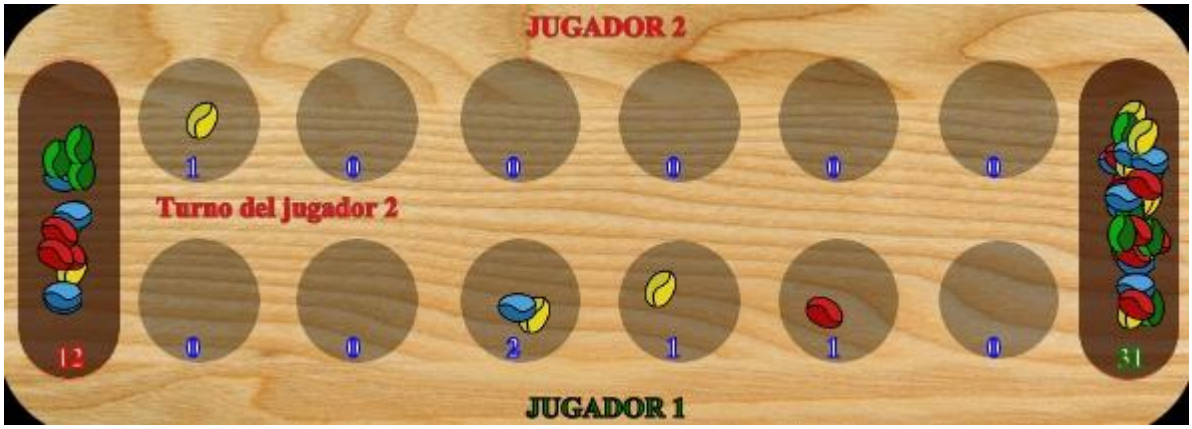


Figura 4

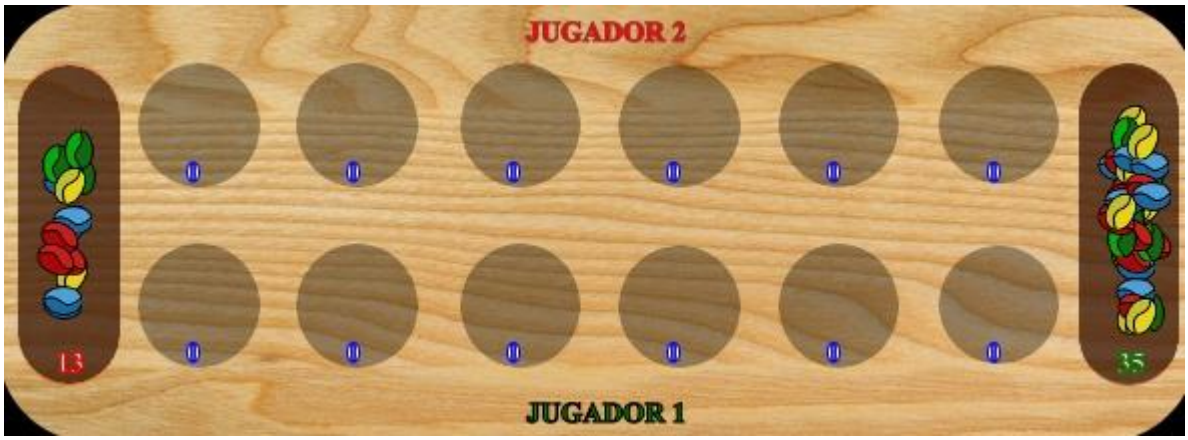


Figura 5

Aunque la mecánica general del juego es sencilla, existen un conjunto de **reglas especiales** que deben aplicarse durante el juego:

- **Inmolación**: Un jugador se inmolará si decide sembrar partiendo de un hoyo que no contiene semillas. En tal caso, ganará el jugador opuesto por 48 semillas a 0 automáticamente. Por ejemplo, si en el escenario de la Figura 4, J2 hubiese decidido sembrar su casilla 2 (la cual no tiene semillas), se habría inmolado provocando que J1 ganase por 48 semillas a 0.
- **Turno extra**: Cualquier jugador tendrá un turno extra si la última semilla que deposita se realiza en su granero. Por ejemplo, partiendo del escenario de la Figura 1, J1 podría haber decidido comenzar sembrando su casilla 4. Como hay 4 semillas, 3 de ellas se colocarían en las casillas 3, 2 y 1 respectivamente, y la última en el granero, produciendo turno extra para J1 (ver Figura 6).
- **¡Robo**!: Si un jugador realiza una siembra y la última semilla la deposita en una de sus casillas, y resulta que esta casilla está vacía pero la opuesta del otro jugador no, entonces el jugador que

está haciendo el movimiento robará su semilla y las de la casilla del contrincante, llevándolas todas a su granero. Por ejemplo, la Figura 7 muestra esta potencial situación donde, si J1 realiza la siembra sobre su casilla 6, depositará 5 semillas de una en una en las casillas 5, 4, 3, 2 y 1. Como esta última está vacía y la casilla opuesta del contrario (la 6 de J2) no lo está, ambas semillas serían robadas por J1 y llevadas a su granero (ver el resultado en la Figura 8). **Aclaración: Un robo no produce un turno extra** (faltaría más…).

Figura 6

Figura 7

Figura 8

Para familiarizarte con las reglas del juego, puedes utilizar el simulador de la asignatura y jugar varias partidas humano contra humano/máquina.

3. Qué hacer en la práctica

En esta práctica debemos desarrollar un agente inteligente capaz de jugar y (tratar de) ganar contra otro adversario en la versión de Mancala explicada en el apartado anterior. Para ello, se proporciona al estudiante una base de programación del motor de Mancala en C++ y la estructura básica del agente que debe programar, junto con un simulador del juego que permite visualizar las acciones realizadas por dos jugadores en una partida.

Se podrán implementar técnicas de búsqueda en espacios de estados entre las estudiadas en el tema de teoría para toma de decisiones de agentes deliberativos en entornos con adversario, y ser implementadas con los algoritmos base que dan soporte a estas técnicas (Minimax, poda α-β, …). Asumiremos que el entorno no tiene componentes externas que lo modifican (como podría ser el azar) y que cada agente, cuando llega su turno, tiene conocimiento completo del mismo (qué acciones ha realizado el contrincante y cuál es el estado actual del juego).

En cada turno, el agente podrá realizar exactamente una entre 6 acciones, correspondientes a la siembra de una de sus casillas. Esta acción debe ser escogida y proceder de un modelo deliberativo de búsqueda como el descrito en el párrafo anterior, guiado siempre por una heurística que permita ganar al jugador frente al contrincante. El agente deberá ser diseñado de modo que pueda adquirir el rol de cualquier jugador: J1 o J2 indistintamente (así podremos poner a jugar a un agente contra consigo mismo, por ejemplo).

Para la realización de la práctica, se proporciona al alumno el conjunto de ficheros y carpetas que se muestran en la Figura 9.

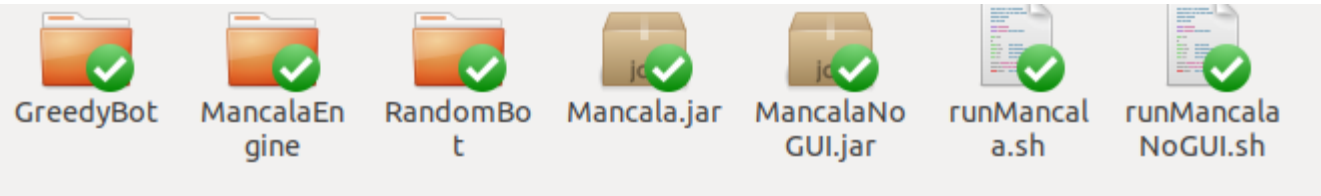


Figura 9

- Carpeta **MancalaEngine**: Contiene la base de programación en C++ para que el alumno pueda programar su bot, y que este pueda comunicarse con el simulador.
- Carpetas **GreedyBot** y **RandomBot**: Contiene dos ejemplos de bots muy simples, cada uno de ellos basados en heurísticas greedy (el primero) y decisión aleatoria (el segundo).
- Ficheros **Mancala.jar** y **runMancala.sh**: Ficheros del simulador con entorno gráfico.
- Ficheros **MancalaNoGUI.jar** y **runMancalaNoGUI.sh**: Ficheros del simulador sin entorno gráfico, y ejemplo de su ejecución, respectivamente.

3.1. El entorno de programación

La carpeta **MancalaEngine** contiene todo lo necesario para que el alumno desarrolle su agente (Figura 10).

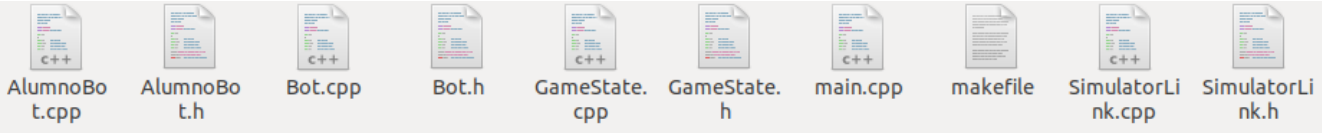


Figura 10

La descripción de los ficheros es la siguiente:

- **SimulatorLink.cpp y SimulatorLink.h**: Contiene una clase que encapsula las comunicaciones entre el simulador y el programa del agente realizado por el alumno. En particular, el simulador ejecuta automáticamente el fichero ejecutable del agente del alumno, y redirecciona las entradas y salidas estándar para utilizarlas como flujos para las comunicaciones. **Por tanto, es extremadamente importante que, durante el desarrollo de la práctica, no se utilicen los flujos de E/S estándar *cin* y *cout*** o similares (scanf, printf, etc.), dado que estos serán utilizados por el propio simulador para las comunicaciones. **El uso de cin/cout o similares en las prácticas podrá provocar el suspenso del alumno con calificación 0**, dado que la práctica no funcionará adecuadamente. En su lugar, **sí estará permitido que el alumno utilice el flujo de salida estándar de errores *cerr* para mostrar mensajes de error** durante el desarrollo de su práctica, por motivos de depuración.
- **GameState.cpp y GameState.h**: Contienen la representación de los diferentes elementos del juego:
  - **Tipo Player**: Es un tipo enumerado que indica cuál es el jugador al que se asocia un evento (por ejemplo, el turno, una acción…). Sus valores son **J1** para el jugador 1, **J2** para el jugador 2, o **NONE** para cuando el valor no está definido.
  - **Tipo Position**: Es un tipo enumerado que indica una posición de un hoyo o casilla. Sus posibles valores son **GRANERO, P1, P2, P3, P4, P5, P6**, para representar posiciones del tablero de cada jugador correspondientes a su granero o casillas desde la 1 a la 6.
  - **Tipo Move**: Es un tipo enumerado que representa la acción escogida por un jugador. Sus valores posibles son **M1, M2, M3, M4, M5, M6**, para indicar si el jugador realiza movimiento de siembra en las casillas 1, 2, 3, 4, 5 o 6, y el valor **M_NONE** para cuando el movimiento no esté definido.
  - **Tipo GameState**. Es una clase que representa un estado del juego, y contiene métodos para facilitar que el agente pueda anticipar estados y movimientos del juego. Un estado se representa siempre por el jugador al que le toca mover y el número de semillas que hay en cada hoyo y tablero del juego**. Las acciones que un agente podrá realizar con un estado son las siguientes:**
    - **Player getCurrentPlayer():** Método que indica cuál es el jugador del turno actual.
    - **unsigned char getSeedsAt(Player p, Position pos) const:** Método que indica cuántas semillas hay en la posición pos del jugador p.
    - **GameState simulateMove(Move mov) const:** Método que tiene como entrada un movimiento realizado por el jugador del turno actual, y devuelve un nuevo

estado del juego indicando cómo quedaría la partida si el jugador realizase el movimiento pasado por argumento (qué turno de qué jugador, cuántas semillas en cada hoyo y granero…).

- **bool isFinalState() const**: Método que devuelve valor verdadero si el estado es final (ningún jugador puede realizar ningún movimiento) y por tanto acaba la partida, o valor falso en caso contrario.
- **bool isValidState() const**: Método para conocer si el estado actual es válido (y nadie nos está haciendo trampas…).
- **Player getWinner() const**: Método para conocer el ganador del juego. Devuelve valor J1 o J2 si es alguno de los dos jugadores, o valor NONE si aún estamos en mitad de la partida o si hay empate.
- **int getScore(Player p) const**: Devuelve la puntuación (número de semillas en el granero) de un jugador pasado por argumento.

Estas herramientas serán las que el alumno tendrá que utilizar para poder implementar el agente dentro de la práctica y poder explorar el espacio de estados del problema.

- **Ficheros Bot.cpp y Bot.h**: Contienen la funcionalidad abstracta que cualquier agente deberá tener. El alumno deberá crear una clase que extienda de la clase **Bot** para crear su agente, e implementar los métodos obligatorios necesarios (ver punto siguiente).
- **Ficheros AlumnoBot.cpp y AlumnoBot.h**: Contienen el esqueleto de la clase que debe implementar el alumno. En particular:
  - El fichero **AlumnoBot.h** contiene la estructura del agente **AlumnoBot**, que extiende de **Bot**. **Si el alumno necesitase que su agente tuviese memoria o incorporase variables para almacenar alguna información de un turno al siguiente, deberá declararlo como campos de la clase en este fichero.**
  - El fichero **AlumnoBot.cpp** contiene la implementación de los métodos del agente **AlumnoBot**. En particular, será necesario:
    - Implementar el constructor y el destructor, si el alumno incluye memoria en el agente necesaria entre turno y turno siguiente, para inicializar/reservar y liberar las variables asociadas a la memoria.
    - Implementar el método initialize(), el cual se llama al comienzo de la partida para que el alumno pueda inicializar las variables internas de su bot.
    - Modificar el método getName() para que devuelva el nombre del bot del alumno.
    - Implementar el método nextMove, que tiene como entrada una lista de acciones realizadas por los turnos consecutivos que acaba de realizar su contrincante, y el estado actual del juego. En caso de que el turno anterior fuese del mismo agente (y este se encuentre en un turno extra), la lista de movimientos del adversario estará vacía. El método deberá devolver la acción que el agente desea realizar en el turno actual. Devolver la acción **M_NONE** supondrá la descalificación automática del agente en la partida y provocar que el contrario gane por 48 semillas a 0.
- Fichero **main.cpp**. Incluye todos los ficheros del software para poder enlazar con el simulador y ejecutar al agente del alumno. El alumno deberá modificar este fichero para incluir el fichero de cabecera de su bot, y declararlo dentro del programa principal.

A modo de ejemplo, los ficheros **RandomBot.cpp** y **RandomBot.h** (carpeta **RandomBot**) muestran una implementación de un agente aleatorio (ver fichero **RandomBot/main.cpp** para ver qué cambios hay que hacer en la función main para activar el bot). Otro ejemplo adicional se encuentra en los ficheros **GreedyBot.cpp** y **GreedyBot.h** (carpeta **GreedyBot**), con otra implementación que realiza una selección de acción basada en un criterio voraz (ver fichero **GreedyBot/main.cpp** para ver qué cambios hay que hacer en la función main para activar el bot).

3.2. Cómo crear mi bot

Es necesario, para que el simulador pueda trabajar correctamente con el bot implementado por el alumno, que se cumplan las siguientes condiciones:

- El nombre del bot (devuelto por la función getName() a implementar) debe coincidir con el nombre del bot que el alumno ha inscrito en la liga (ver apartado 4 de este guión).
- El nombre de la clase que implementa el bot debe ser el mismo nombre del bot (por ejemplo, apreciar que los bots RandomBot y GreedyBot se encuentran implementados en clases que se llaman RandomBot y GreedyBot, respectivamente, y que el programa main.cpp los instancia con ese mismo nombre.
- El nombre de los ficheros .h y .cpp que el implementa el bot deben tener el mismo nombre del bot (por ejemplo, apreciar que los bots RandomBot y GreedyBot se encuentran implementados en RandomBot.cpp y RandomBot.h, y GreeedyBot.cpp y GreedyBot.h, respectivamente).
- No se puede utilizar E/S estándar (es decir, no se puede usar ni cin ni cout bajo ningún concepto, ni funcionalidades similares). Sí se permite utilizar el flujo de salida de errores estándar, ***cerr***.

El incumplimiento de alguna de estas normas hará que los bots del alumno no puedan participar en la liga. Por tanto, las primeras acciones a realizar por el alumno tras darse de alta en el simulador deberán ser:

1. Renombrar los ficheros **AlumnoBot.*** para que contengan el nombre del bot del alumno,
2. Renombrar la clase **AlumnoBot** para que se llame igual que el bot del alumno (no olvidéis también modificar este nombre en el fichero cpp),
3. Modificar el main para instanciar el bot del alumno, y
4. Modificar el método **getName()** del fichero cpp para que devuelva el nombre del bot del alumno.

Por último, el alumno deberá asegurarse de que su agente puede compilar con la siguiente orden para asegurarse de que el bot puede participar en la liga, ejecutada desde la carpeta de implementación del bot (donde BotDelAlumno se sustituiría por el nombre del bot del alumno):

g++ -oSalida Bot.cpp GameState.cpp main.cpp SimulatorLink.cpp BotDelAlumno.cpp -I./ -lm -std=c++11

Por ejemplo, para el agente **GreedyBot**, las normas son que debe poder compilar y generar el ejecutable con la siguiente orden:

g++ -oSalida Bot.cpp GameState.cpp main.cpp SimulatorLink.cpp GreedyBot.cpp -I./ -lm -std=c++11

Acto seguido, el alumno podrá comprobar la ejecución de su bot en el simulador (**OJO: Recordad que no se puede usar *cin* ni *cout* en la práctica. Utilizad *cerr* si queréis mostrar errores**).

4. El simulador

El simulador del Mancala que utilizaremos en la práctica ha sido implementado íntegramente por el profesorado de la asignatura de Inteligencia Artificial del departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada, a fin de proponer las prácticas que se describen en este guión. Está desarrollado en lenguaje Java, por lo que se deberán cumplir unos requisitos previos para que el estudiante pueda ejecutarlo.

4.1. Prerrequisitos

El simulador está implementado en lenguaje Java versión 8, y utiliza las bibliotecas internas JavaFX 2.2. Por tanto, es necesario tener instalado Java 8 JRE o Java 8 JDK en el ordenador para ejecutar correctamente el programa. El simulador ha sido probado con la versión **Oracle Java 1.8.0_91** de 64 bits. Para instalar Oracle Java en Windows, basta con visitar la web de Oracle y descargar **Oracle java JDK 8** (**CUIDADO: No descargar la versión 9**). Para instalar Oracle Java en Linux Ubuntu, basta con añadir el repositorio de Oracle Java e instalar los paquetes con los siguientes comandos:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

4.2. Simulador sin interfaz gráfica

La versión más simple y rápida del simulador que el alumno puede utilizar durante el desarrollo de la práctica carece de interfaz gráfica, aunque también está limitada a que sólo puedan jugar bots contra bots. El programa se encuentra en el fichero **MancalaNoGUI.jar**, y su ejecución debe realizarse desde consola de la siguiente forma:

java –jar MancalaNoGUI.jar –p1 <ruta al bot jugador 1> –p2 <ruta al bot jugador 2> -t <tiempo límite de turno en segundos>

Se asume que, al ejecutar el simulador, existen dos bots (puede ser un único bot también que haga de jugador 1 y de jugador 2), ya compilados. El parámetro **–p1** debe estar seguido de la ruta al fichero ejecutable del bot que actuará como jugador 1. Por otra parte, el parámetro **–p2** debe estar seguido de la ruta al fichero ejecutable del bot que actuará como jugador 2. Ambos parámetros **–p1** y **–p2** son obligatorios. El tercer parámetros **–t** es opcional. En caso de no aplicarse, no se impondrá un

tiempo límite por turno para cada jugador. En caso de aplicarse, deberá ir seguido de un número natural que indique el número de segundos permitidos por turno.

Veamos un ejemplo de cómo ejecutar el simulador utilizando los dos bots que se proporcionan al alumno como ejemplo. Para ello:

1. Entraremos por consola a la carpeta de **GreedyBot** y lo compilaremos.
2. Entraremos por consola a la carpeta de **RandomBot** y lo compilaremos.
3. Ejecutaremos el simulador con **GreedyBot** como jugador 1, **RandomBot** como jugador 2, y un tiempo límite de 5 segundos por turno como máximo para cada jugador.

La Figura 11 muestra los pasos a seguir por consola para realizar estas acciones.

```
manupc@manupcws:~/Mancala$ cd GreedyBot/
manupc@manupcws:~/Mancala/GreedyBot$ make
g++ -oGreedyBot *.cpp -I./ -lm -std=c++11
manupc@manupcws:~/Mancala/GreedyBot$ cd ../RandomBot/
manupc@manupcws:~/Mancala/RandomBot$ make
g++ -oRandomBot *.cpp -I./ -lm -std=c++11
manupc@manupcws:~/Mancala/RandomBot$ cd ..
manupc@manupcws:~/Mancala$ java -jar MancalaNoGUI.jar -p1 ./GreedyBot/GreedyBot
-p2 ./RandomBot/RandomBot -t 5
```

Figura 11

Acto seguido, el simulador se ejecutará e irá mostrando en cada paso de quién es el turno, qué acción ejecuta cada agente y el estado actual tras ejecutar la acción, hasta que el juego finalice (ver Figura 12). **Si el alumno utiliza mensajes de error utilizando cerr dentro de su código, estos mensajes también aparecerán por consola**.

```
[Simulador]: Turno del jugador 1.
[Simulador]: El jugador 1 realiza el movimiento 5.
[Simulador]: Estado actual del tablero:
        J1: Granero=40 C1=0 C2=0 C3=0 C4=1 C5=0 C6=0
        J2: Granero=5 C1=2 C2=0 C3=0 C4=0 C5=0 C6=0

[Simulador]: Turno del jugador 2.
[Simulador]: El jugador 2 realiza el movimiento 1.
[Simulador]: Estado actual del tablero:
        J1: Granero=42 C1=0 C2=0 C3=0 C4=0 C5=0 C6=0
        J2: Granero=6 C1=0 C2=0 C3=0 C4=0 C5=0 C6=0

[Simulador]: Fin de la partida.

-------FIN DE LA PARTIDA
-------
Puntos del jugador 1 (GreedyBot): 42
Tiempo del jugador 1 (GreedyBot): 1150 milisegundos.
Puntos del jugador 2 (RandomBot): 6
Tiempo del jugador 2 (RandomBot): 750 milisegundos.
Ganador: Jugador 1 (GreedyBot)
```

Figura 12

Al final de la partida, se mostrará los puntos de cada jugador, el tiempo total (en milisegundos) que ha tardado cada bot en realizar todas las acciones, y el jugador ganador.

Para mayor comodidad, se ofrece al estudiante el script **runMancalaNoGUI.sh**, para que el alumno lo modifique y pueda ejecutar el simulador sin interfaz gráfica más rápidamente.

4.3. Simulador con interfaz gráfica

El simulador con interfaz gráfica ofrece muchas más posibilidades que el simulador sin interfaz gráfica, dado que permite no sólo ejecutar partidas de bots contra bots, sino también considerando jugadores humanos y partidas remotas online. Además, permite visualizar el transcurso de cada partida y participar en la liga de la asignatura (ver apartados siguientes). Para ejecutar el simulador con interfaz gráfica desde consola, se deberá escribir el siguiente comando:

java –jar Mancala.jar

Adicionalmente, se proporciona al alumno el script **runMancala.sh** para que éste pueda ejecutarlo más rápidamente haciendo doble click sobre el fichero (**NOTA: Se debe dar permisos de ejecución al fichero con el comando chmod a+x runMancala.sh**).

La ventana del simulador tiene el aspecto que se muestra en la Figura 13. En ella, podemos seleccionar qué jugadores queremos que jueguen una partida aislada (fuera de la liga) en los paneles **Jugador 1** y **Jugador 2** de la parte superior. Se presentan 3 opciones:

- **Bot local**. Si queremos que el jugador sea nuestro bot o un bot de los ejemplos, pulsaremos sobre la opción **Bot local** del jugador que queremos asignar (Jugador 1 o Jugador 2) y, tras

pulsar sobre el botón **seleccionar** buscaremos el fichero ejecutable del bot que hemos compilado previamente.

- **Jugador humano**. Seleccionando esta opción, el jugador requerido será humano, y podrá interactuar con la interfaz gráfica para jugar una partida contra otros humanos, contra bots o contra otros compañeros que se encuentren en otra localización remota. Se deberá introducir el nombre del jugador humano (o Nick) para poder crear una partida o unirse a la misma.
- **Jugador remoto**. Esta opción se utilizará cuando queramos jugar una partida contra algún otro compañero que se encuentre en otro PC, en cualquier ubicación.

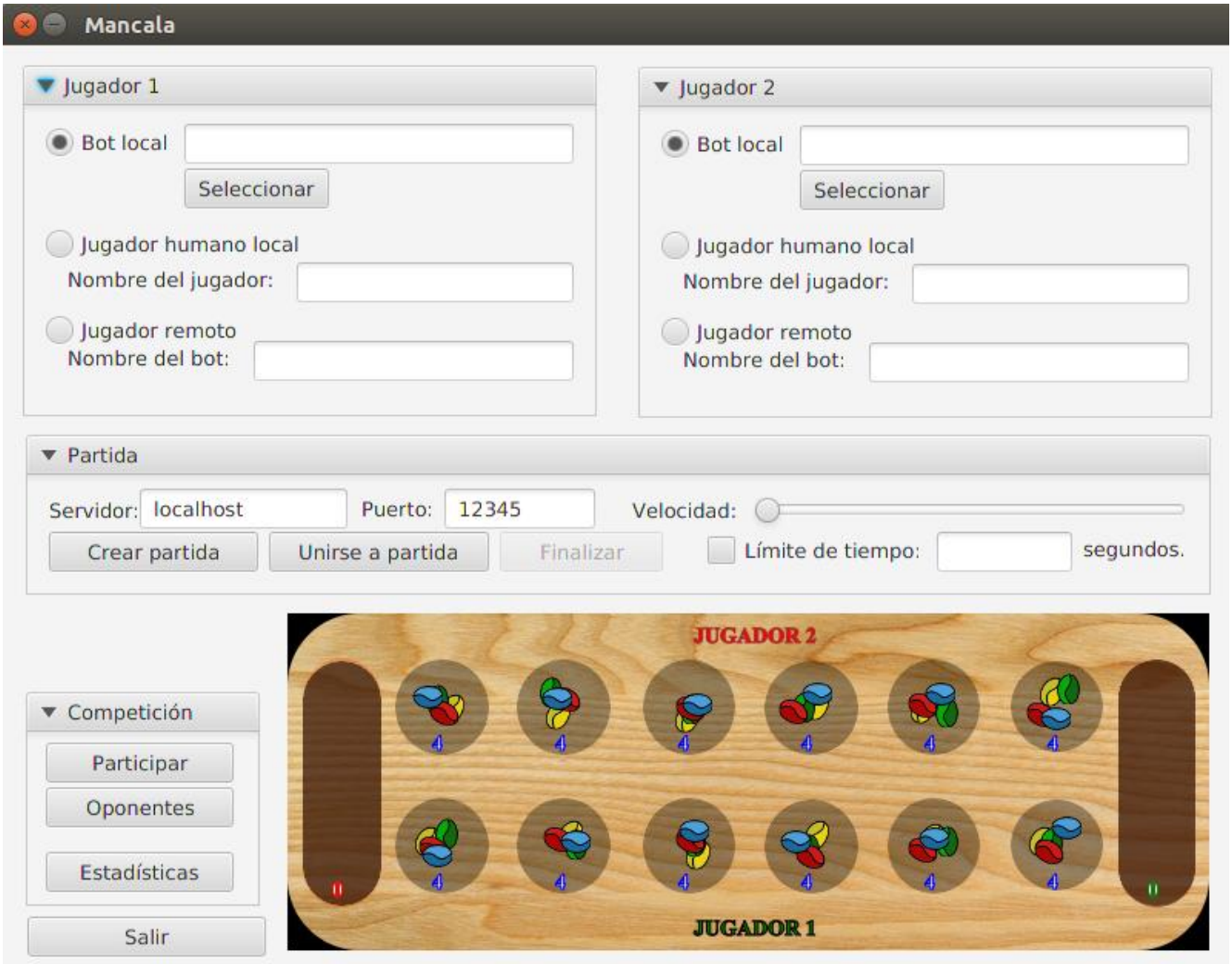


Figura 13

Por ejemplo, supongamos que el estudiante Pepito quiere jugar contra GreedyBot. En este caso, Pepito se pondría como jugador humano y GreedyBot (**EL EJECUTABLE**) como bot local (ver Figura 14). Acto seguido, pulsaríamos para crear la partida sobre el botón **"Crear partida"**.

IMPORTANTE: Para partidas locales, el campo *servidor* debe tener siempre el valor "localhost" o la IP "127.0.0.1". El puerto no debe estar en uso previamente por alguna otra aplicación del PC donde se ejecute el programa.

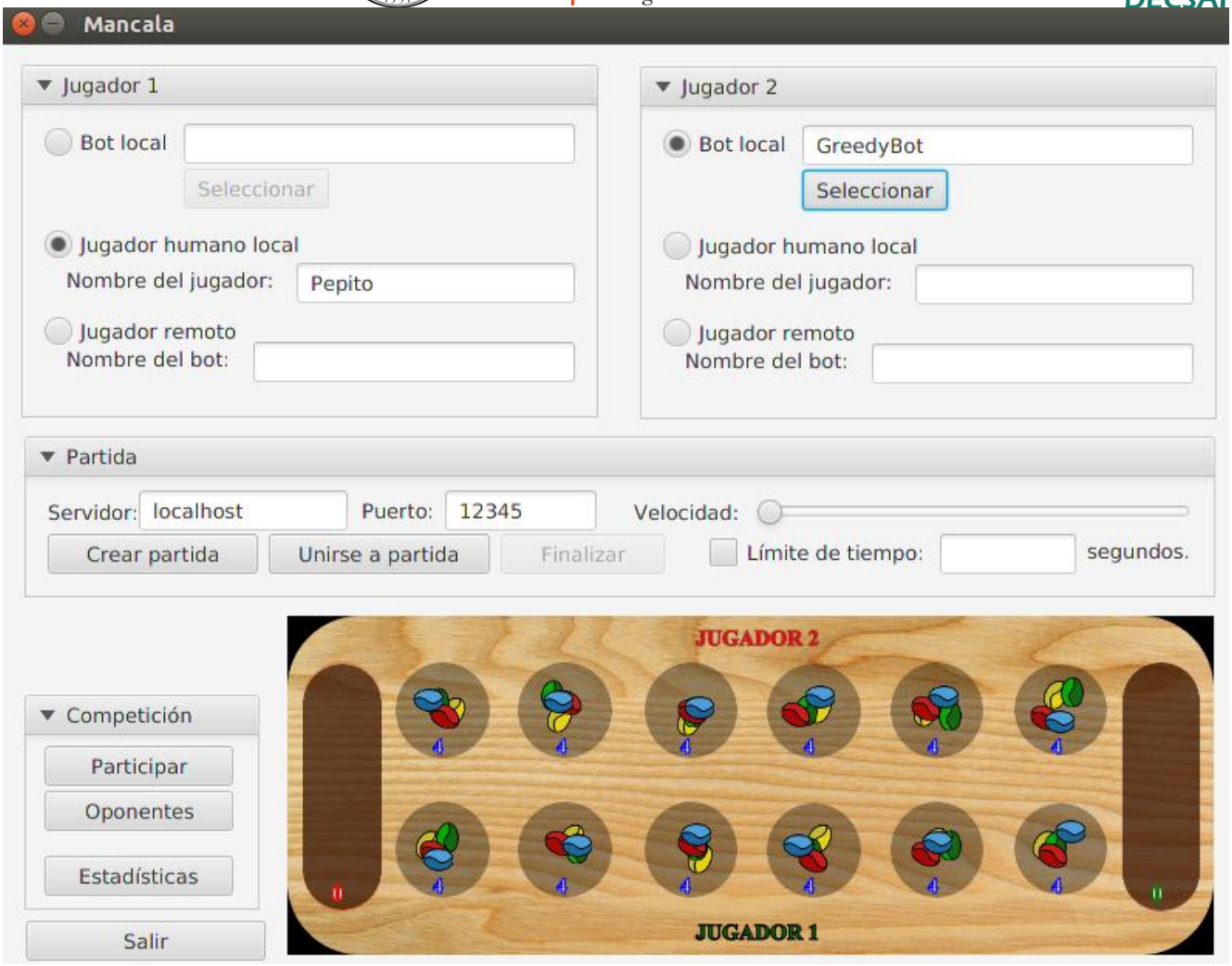


Figura 14

Una vez iniciada la partida, el jugador humano podrá realizar la siembra sobre la casilla que desee utilizando el ratón: Se deberá posicionar sobre la casilla que se desea sembrar (esta se remarcará de color verde si el jugador humano es el 1 y de color rojo si es el 2), y hacer click con el ratón. Por otra parte, tanto jugadores bots como remotos ejecutarán su acción automáticamente.

El simulador, para aumentar la experiencia de usuario, permite dos opciones adicionales:

- **Velocidad:** Aumenta o disminuye la velocidad de las animaciones del juego. Se recomienda una velocidad baja para poder visualizar correctamente toda la partida, y una velocidad alta para evitar animaciones y terminar antes la partida.
- **Límite de tiempo:** Si deseamos establecer un límite de tiempo por turno para cada jugador en la partida, haremos click sobre el check box "**Límite de tiempo**", y escribiremos en el campo de texto cuántos segundos como máximo queremos que dure cada turno. **Si un jugador excede este tiempo, la partida será ganada automáticamente por el otro jugador por 48 semillas a 0.**

4.3.1. Crear partidas aisladas online

Si un estudiante quiere probar su bot contra el de otro compañero, esta opción está barajada por el simulador como partidas online. En este caso, un estudiante actúa como servidor y el otro como cliente.

Para realizar una partida online, uno de los jugadores debe crear la partida como servidor, tras haberla configurado. Por ejemplo, si el estudiante Pepito quiere probar a **GreedyBot** contra un jugador remoto, deberá configurar los jugadores adecuadamente, estableciendo el nombre con el que el jugador remoto se unirá a la partida. Por ejemplo, Si el jugador remoto se llama **RandomBot**, el estudiante configurará un jugador con su bot local **GreedyBot** y el otro como jugador remoto **RandomBot**. Este nombre es necesario para evitar que otras personas se conecten a nuestra partida creada sin permiso previo. Además, será necesario que los campos **"Servidor**" y "**Puerto**" tengan los valores **localhost** y un entero que represente un puerto libre en la máquina donde se ejecute (por ejemplo, ver la Figura 15, donde **GreedyBot** es jugador 2, se desea jugar contra un jugador remoto **RandomBot**, y la partida se encuentra configurada con 3 segundos como máximo por turno, a una velocidad intermedia para mostrar las animaciones).

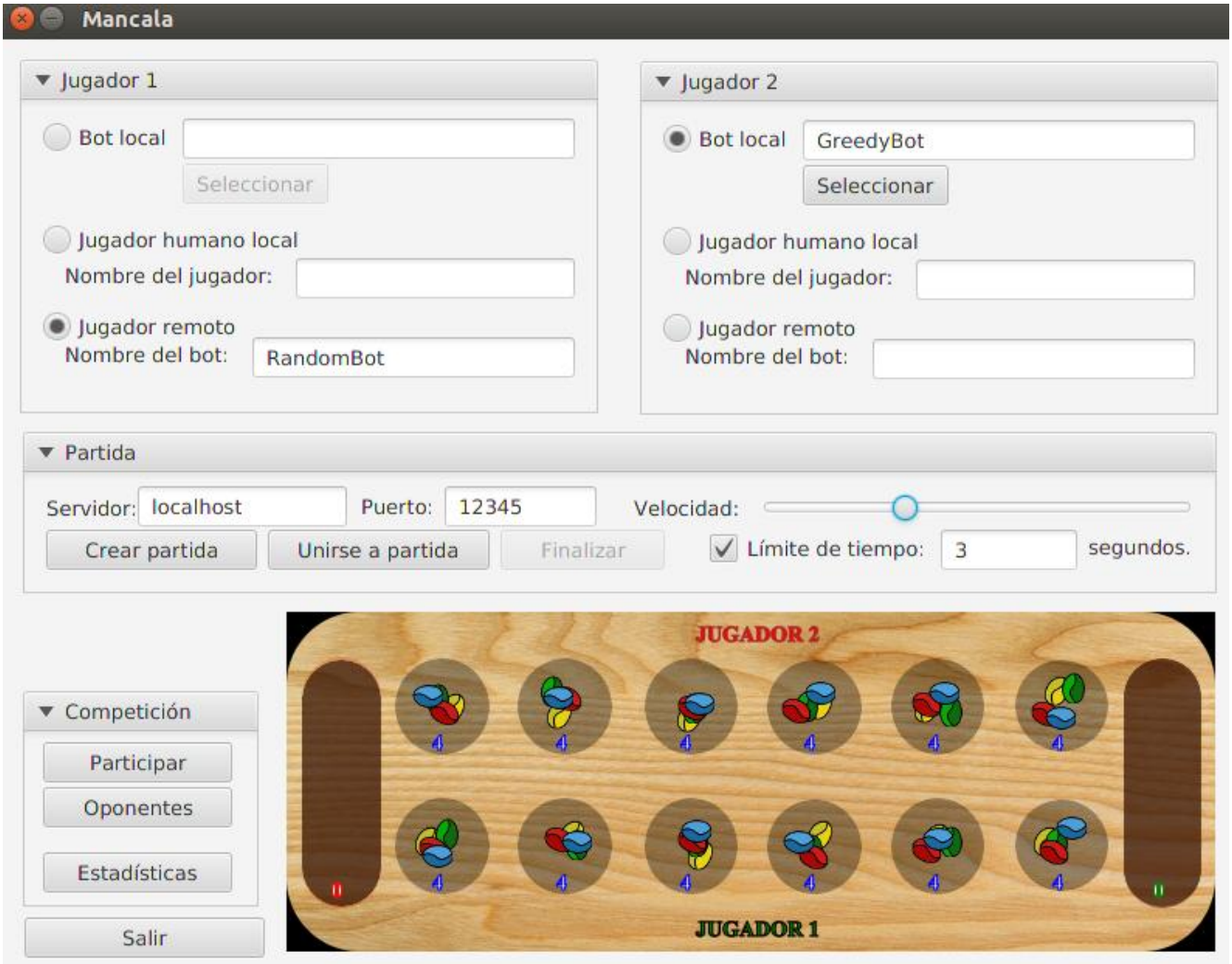


Figura 15

Acto seguido, el jugador que crea la partida pulsará sobre el botón **"Crear partida**", y saldrá una nueva ventana esperando a que el otro jugador se conecte (Figura 16).

Figura 16

MUY IMPORTANTE: Si quieres crear una partida, asegúrate de que no te encuentras detrás de un cortafuegos o router que impida el paso de comunicaciones por el puerto que has utilizado para el juego. Asegúrate, en la configuración de tu cortafuegos y de tu router, que tienes el puerto habilitado para poder crear una partida (normalmente, en opciones de *disparo de puertos* o similares).

4.3.2. Unirse a una partida aislada online

Si somos un estudiante que ha sido retado por otro compañero que ha creado una partida, podemos unirnos a la partida creada realizando los siguientes pasos:

1. El alumno que ha creado la partida debe facilitarnos su dirección IP para que la introduzcamos en el campo **"Servidor"**. El jugador que crea la partida puede conocer cuál es su IP entrando a la página web https://www.whatismyip.com/ (Por ejemplo, supongamos que su IP es 217.43.12.33). Introduciremos este valor en el campo **"Servidor"**.
2. El alumno que ha creado la partida debe facilitarnos también el puerto de comunicaciones que utiliza para crear la partida. Supongamos que nuestro compañero que crea la partida nos comunica que el puerto es 12345. Introduciremos este valor en el campo Puerto.
3. El alumno que ha creado la partida nos debe comunicar qué jugador somos, y qué nombre nos ha puesto, para poder unirnos a dicha partida. Supongamos que el nombre es **RandomBot** y que nos comunica que somos el jugador 1. En este caso, rellenaremos el panel del Jugador 1 con esos datos, y el panel del jugador 2 como jugador remoto (no hace falta el nombre en este caso).

4. Pulsaremos sobre el botón **"Unirse a partida**" para iniciarla.

La Figura 17 muestra un ejemplo de la configuración del estudiante que actúa como cliente antes de unirse a la partida.

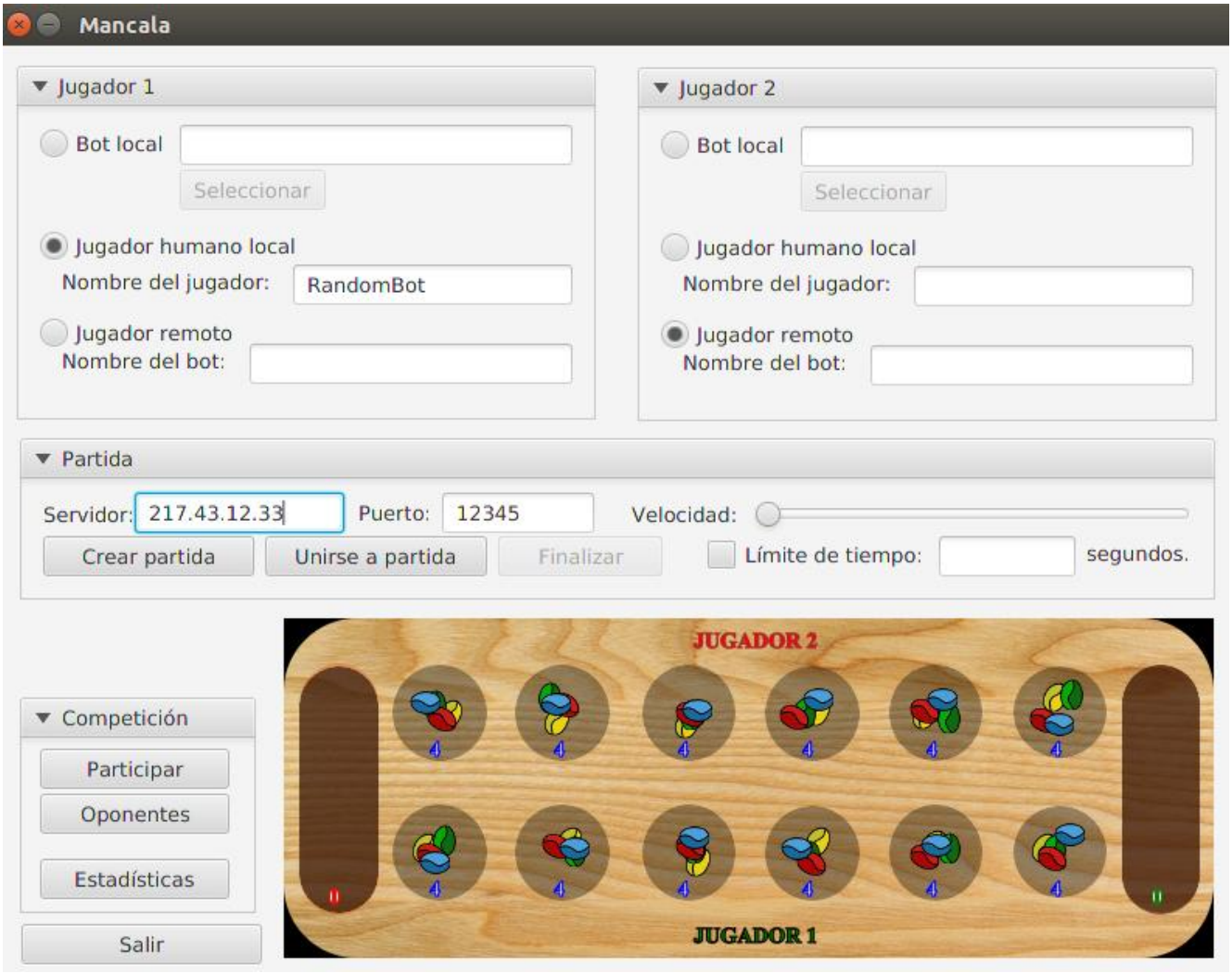


Figura 17

4.4. Modo liga

Como parte del método de evaluación de la práctica (ver apartados siguientes), se realizará una liga entre los alumnos del mismo grupo de prácticas de la asignatura. No obstante, para dar la posibilidad de que el alumno pueda validar su bot con respecto a otros compañeros, el simulador da la posibilidad de participar en la liga "oficial" y en otra liga "extraoficial". Para participar en la liga, el alumno debe dar de alta a su bot previamente.

La liga oficial y los datos de la misma se llevarán a cabo en las mismas condiciones para todos los bots, ejecutándose en un servidor externo cuyos datos son los siguientes:

Servidor: aquiles.ugr.es

Puerto: 12345

4.4.1. Darme de alta y gestión de mis datos

Pulsando sobre el botón **"Participar"** de la ventana principal, se nos dará las opciones de conexión al servidor (el servidor y el puerto aparecen por defecto). Acto seguido, se nos presentarán varias opciones:

- **Entrar.** Nos permite acceder a la liga. Deberemos entrar con nuestro email y clave de Mancala cada vez que deseemos participar.
- **Darme de alta**. Nos permite introducir nuestros datos para dar de alta a nuestro bot en la liga:
  - **Nombre y apellidos**: Los del alumno:
  - **eMail**: El email @correo.ugr.es del estudiante. **Este campo no podrá ser modificado posteriormente.**
  - **Nombre del bot**: El nombre de nuestro bot. **IMPORTANTE: El nombre del bot no será posible cambiarlo una vez que lo hayamos dado de alta. ¡Pensad muy bien el nombre del bot antes de registraros!**. Como el nombre del bot también debe ser el nombre de la clase que lo implemente y los ficheros .h y .cpp que se entregan, este nombre debe seguir rigurosamente las normas para nombrar identificadores dentro del lenguaje C/C++.
  - **Grupo de prácticas**: El grupo de prácticas en el que estamos matriculados. Será uno de los mostrados en la Tabla 1, dependiendo del grado que estemos cursando y del campus.
  - **Clave de acceso**: La clave de acceso a Mancala. Se recomienda utilizar una clave propia para el simulador y no reutilizar una de las que tenemos para nuestras cuentas, por motivos de seguridad.

Tabla 1

Titulación	Grupo de prácticas	Grupo en Mancala
Grado en Ingeniería Informática (Campus de Aynadamar)	Grupo A, prácticas A1	A1
	Grupo A, prácticas A2	A2
	Grupo A, prácticas A3	A3
	Grupo B, prácticas B1	B1
	Grupo B, prácticas B2	B2
	Grupo B, prácticas B3	B3
	Grupo C, prácticas C1	C1
	Grupo C, prácticas C2	C2
	Grupo C, prácticas C3	C3
	Grupo D, prácticas D1	D1
	Grupo D, prácticas D2	D2
Doble Grado en Ingeniería Informática y Matemáticas	Grupo A, prácticas A1	M1

	Grupo A, prácticas A2	M2
Grado en Ingeniería Informática (Campus de Ceuta)	Grupo A, Prácticas A	CE

- **Modificar mis datos**. Se podrán modificar los datos personales (nombre, apellidos y clave) y el grupo de prácticas en el que estamos matriculados. **No podrá cambiarse ni el nombre del bot ni el e-mail del estudiante**.
- **Subir bot**. Se nos permitirá subir los ficheros .h y .cpp de nuestro bot para participar en la liga oficial. **IMPORTANTE:** Hay que asegurarse de que las normas de creación del bot cumplen con los requisitos expuestos en el apartado 3.2 de este guión. En otro caso, el bot quedará desclasificado. Adicionalmente, el simulador nos avisará si existe algún problema a la hora de subir los ficheros .h y .cpp.
- **He olvidado mi clave**. Si hemos olvidado nuestra contraseña de Mancala, introduciremos nuestro e-mail y se nos enviará un correo automático desde una cuenta de correo creada expresamente para estas prácticas, pasados unos minutos, con la clave de acceso. Los profesores no responderán a ningún correo electrónico que se envíe a esta cuenta.
- **Salir**. Permite salir del servidor de Mancala.

4.4.2. La liga oficial

La liga oficial se actualiza cada 3 días, ejecutando los bots de cada grupo de prácticas en el servidor. La clasificación del bot dentro de esta liga se tendrá en cuenta para la evaluación, en la última actualización de la liga que se realice en la asignatura.

Para participar en la liga, debemos dar de alta a nuestro bot siguiendo los pasos comentados en el apartado 4.4.1, y subir el código fuente (ficheros .h y .cpp) del bot al sistema. Para ello, realizaremos las siguientes acciones:

- En la ventana principal del simulador de Mancala, pulsaremos en "**Participar"**.
- Si no estamos dentro de la liga, pulsaremos sobre **"Entrar**" para *loguearnos* en el sistema.
- En caso de que ya estemos registrados, seleccionaremos la opción **"Subir bot"**, y aparecerá la ventana de la Figura 18. Buscaremos los ficheros .h y .cpp de nuestro bot y pulsaremos sobre el botón de **"Ok"**. El simulador nos avisará si ha habido algún problema al subir el bot o al compilarlo en el servidor. Podremos realizar esta acción tantas veces como queramos antes de llegar a la última actualización de la liga, incorporando todas las actualizaciones que deseemos del bot.

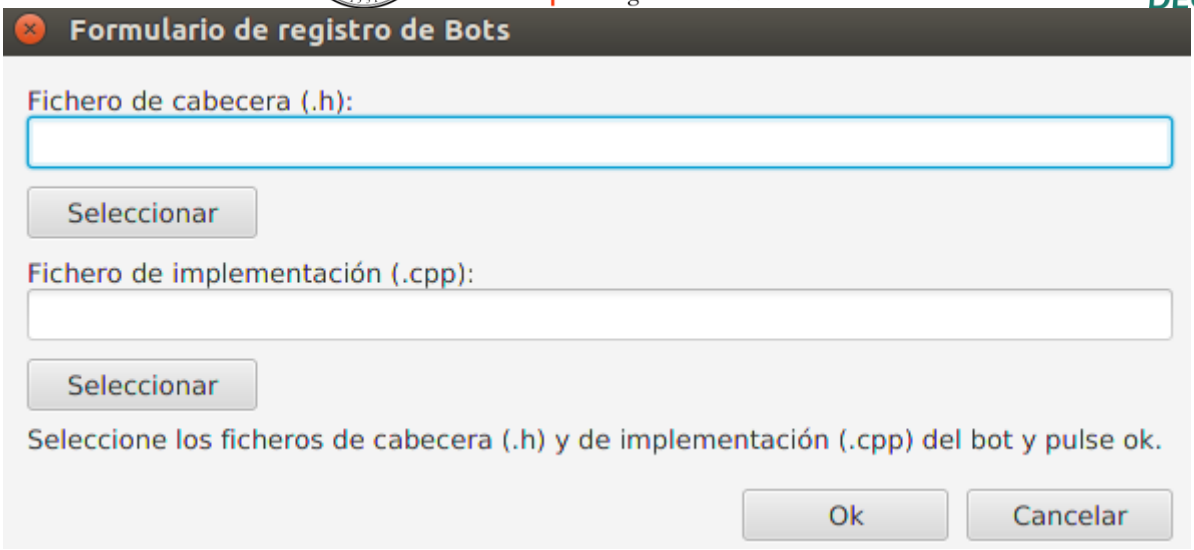


Figura 18

4.4.2.1. Normas de la liga oficial

La liga oficial se actualizará con los bots existentes cada 3 días. Para la participación en la liga, sólo se tendrán en cuenta los bots existentes en el sistema en el comienzo de la misma. Por tanto, si un alumno sube una actualización del bot durante el desarrollo de la liga, esta no se tendrá en cuenta hasta la próxima ejecución de la liga, a los 3 días siguientes.

Durante la ejecución de una actualización de la liga, todos los bots del mismo grupo de prácticas compiten entre sí, jugando por parejas e intercambiándose entre jugador 1 y jugador 2. Al finalizar, la clasificación se crea de acuerdo a las siguientes normas:

- El mejor bot será aquel que más puntuación haya conseguido, considerando la puntuación como 3*(nº de partidas ganadas)+(nº de partidas empatadas).
- En caso de empate, el mejor bot será aquel que menos partidas haya perdido por exceso de límite de tiempo por turno.
- En caso de empate, el mejor bot será el más rápido (aquel que menos tiempo (milisegundos) haya acumulado entre todas sus partidas en tomar la decisión en cada turno).
- En caso de empate, el mejor bot será aquel que mejor puntuación global haya obtenido en todas sus partidas (semillas ganadas/puntos a favor – semillas perdidas/puntos en contra).
- En caso de empate, el mejor bot será aquel que mayor número de semillas haya acumulado entre todas sus partidas.
- En caso de empate, el profesor revisará la implementación de los bots.

Las normas de ejecución para competir son simples:

- Cada bot se ejecutará dos veces contra cada uno de los bots del mismo grupo de prácticas: Una actuando como jugador 1 y otra actuando como jugador 2.

- Existe un límite de tiempo de 2 segundos para efectuar el turno. Dado que la intensidad de los algoritmos de búsqueda depende del procesador donde se ejecuten estos, como información adicional, se indica al estudiante que el servidor que ejecutará la liga es capaz de generar 140.000 estados en los dos segundos, aproximadamente. Se permite el uso de técnicas de medición del tiempo de ejecución dentro del propio bot (funciones time, funciones del estándar C++11 como chrono::system_clock, etc.). Si un bot sobrepasa este límite, perderá la partida por timeout por 48 semillas a 0.
- Para la calificación final, sólo se tendrá en cuenta la posición de cada bot dentro de la tabla de clasificación en la última actualización de la liga.
- La última actualización de la liga se realizará el día del examen.

4.4.3. La liga extraoficial

Mientras que la liga oficial se realiza entre los alumnos de un mismo grupo de prácticas, se proporciona la opción de una liga extraoficial entre todos los alumnos de todos los grupos de todos los grados donde se imparte esta asignatura. Esta liga extraoficial se facilita para que los estudiantes validen y prueben sus bots contra otros compañeros, y no se tendrá en cuenta para la calificación final de la asignatura.

Para participar en la liga extraoficial, deberemos estar *logueados* en el sistema según se ha explicado en la sección 4.4.1 de este guión. Acto seguido, se nos mostrará una ventana donde deberemos elegir el fichero ejecutable (ya compilado) del bot con el que queremos participar (Figura 19).

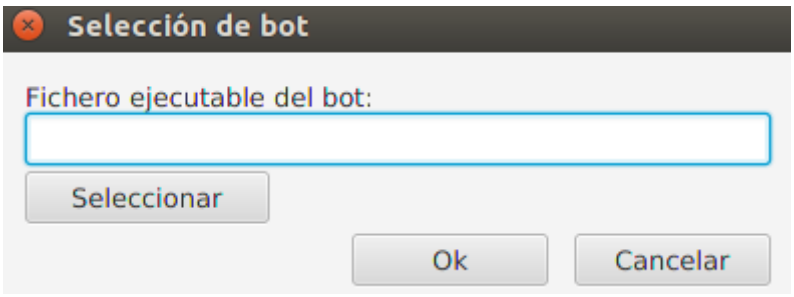


Figura 19

Tras seleccionar el ejecutable, entraremos a la ***antesala de los valientes*** (Figura 20), donde podremos buscar oponentes que se encuentren dentro del sistema. La parte izquierda de la ventana contiene una lista con todos los bots que se encuentran actualmente online en el sistema. Podremos mandar mensajes a todos los bots (o a sólo uno de ellos) seleccionando el bot (o los bots) destino en la lista, y escribiendo un texto en el campo **"Escribe tu texto aquí"**. Cuando tengamos nuevos mensajes de algún bot, su nombre se remarcará en verde.

Aparte de mantener conversaciones, la antesala de los valientes nos permite retar a otros bots para jugar una partida (o aceptar retos). Podremos retar a un bot seleccionándolo de la lista y pulsando sobre el botón **"retar"**. En caso de que otro bot nos rete, su nombre quedará remarcado en rojo en la lista de bots. Podremos aceptar el reto seleccionando el bot y pulsando sobre el botón **"Aceptar reto"**. Si un reto no se acepta pasados unos segundos, se entenderá que el destinatario no acepta el reto. Un bot sólo puede retar a otro bot simultáneamente.

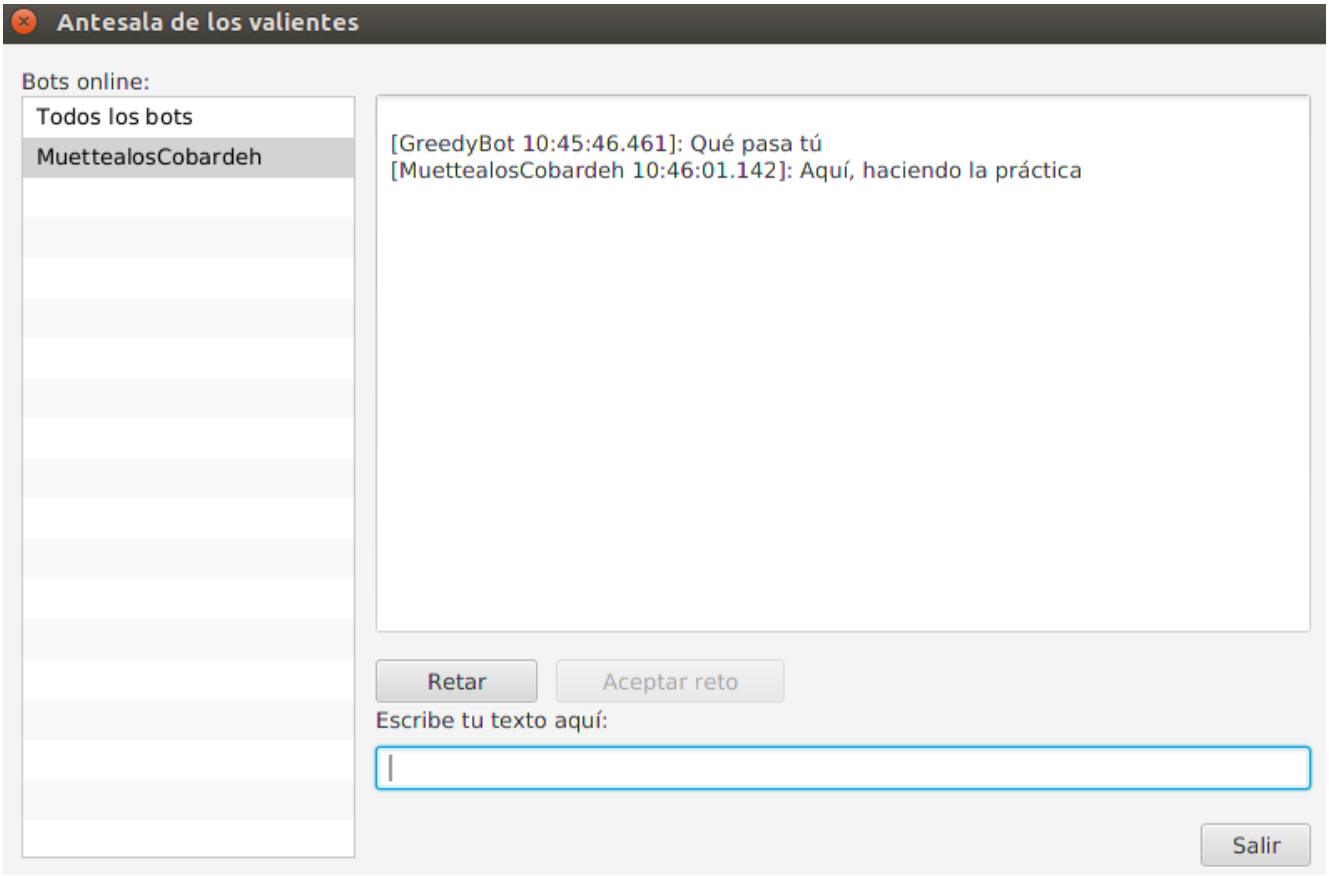


Figura 20

Tras la aceptación de un reto, la ventana de la ***antesala de los valientes*** se cerrará, y comenzará la partida. Cuando esta finalice, los resultados de la partida se enviarán al servidor para actualizar las estadísticas en la liga extraoficial de cada jugador involucrado.

4.4.4. Estadísticas y clasificaciones.

Para conocer los resultados de las competiciones oficial y extraoficial es necesario encontrarse *logueados* en el sistema (ver apartados anteriores). En la ventana principal de Mancala, pulsaremos sobre el botón **"Estadísticas"** para acceder a la información requerida. Nos aparecerá la ventana de la Figura 21. En ella, podremos seleccionar entre ver los resultados de la liga oficial (pestaña **"Liga oficial"**) o los de la extraoficial.

En la pestaña de la liga oficial nos aparecerá un mensaje con la fecha para la que está programada la siguiente ejecución de la liga, y podremos seleccionar cualquier grupo de prácticas de cualquier centro y grado que participe. La tabla inferior mostrará los resultados de la clasificación de la liga en curso.

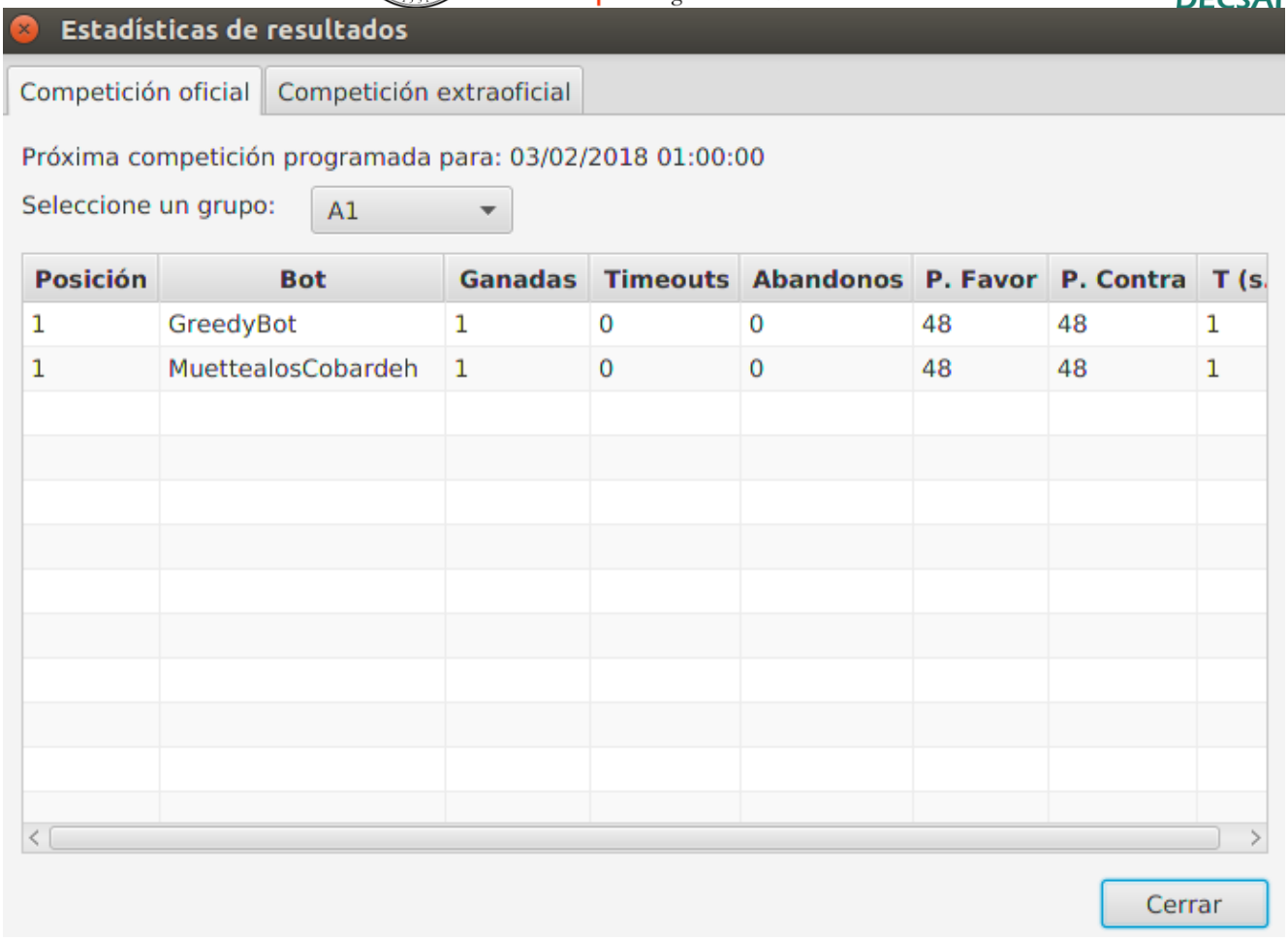


Figura 21

Por otra parte, la pestaña **"Liga extraoficial"** (Figura 22) nos mostrará las estadísticas de todos los grupos mezclados, atendiendo a las partidas que cada bot haya jugado en la ***antesala de los valientes***.

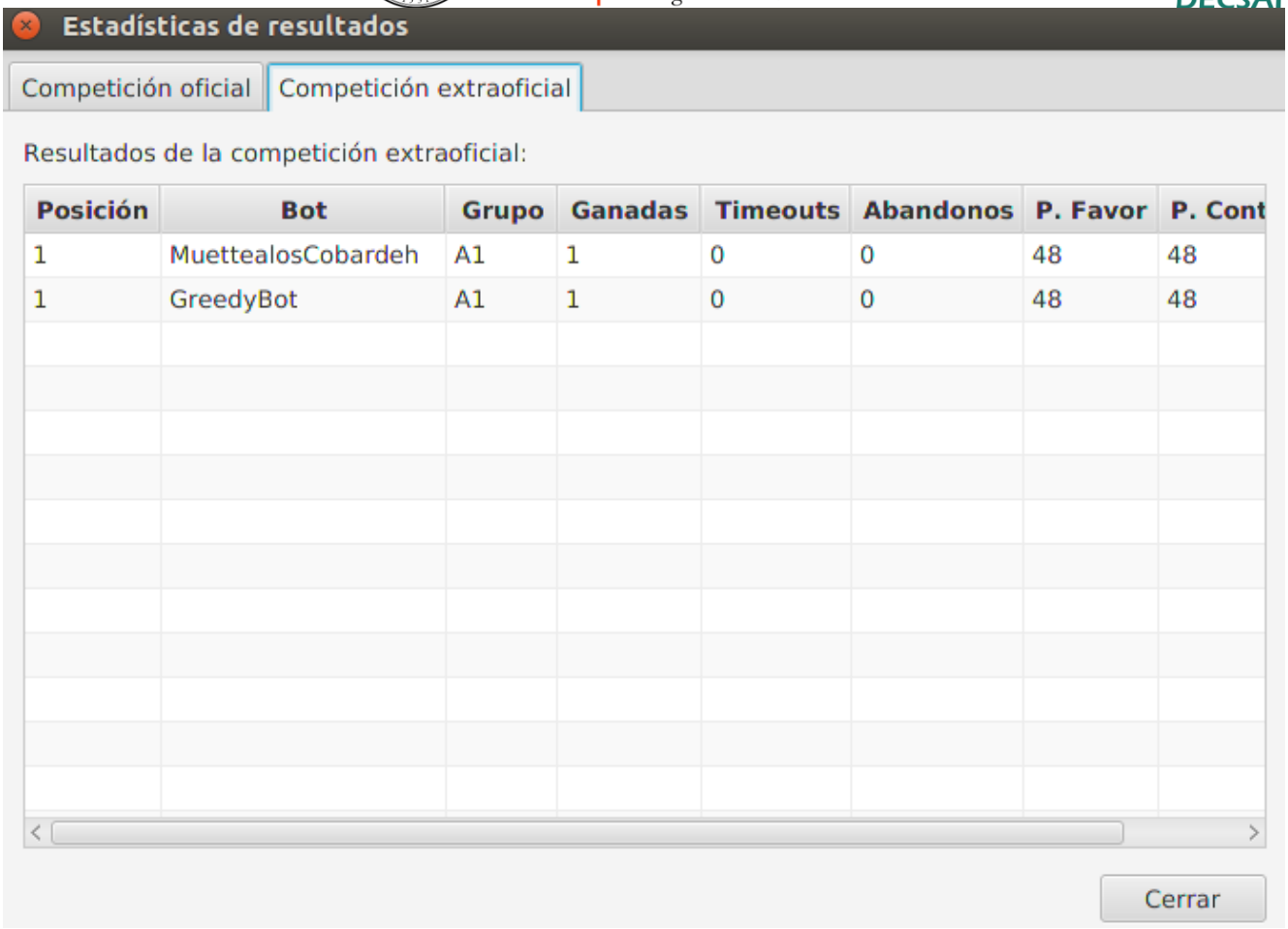


Figura 22

5. Objetivo de la práctica

El objetivo de esta práctica es dotar de un comportamiento inteligente a nuestro bot, usando agentes deliberativos en entornos con adversario para definir el curso de acciones que éste deberá tomar a lo largo de una partida, tanto si le toca jugar como jugador 1 o como jugador 2.

Nuestro bot implementará una de las técnicas estudiadas para agentes deliberativos en entornos competitivos con información perfecta, implementada con base en los algoritmos Minimax o poda α-β (el alumno deberá escoger una de las estrategias, y no podrá entregar varias soluciones).

El tiempo límite de cada turno del jugador estará limitado a 2 segundos (exploración de 140.000 estados/turno aproximadamente). En cada turno, el agente deberá devolver la acción más prometedora para sí mismo, atendiendo a la búsqueda realizada.

Como el espacio de estados es muy grande, no dará tiempo a explorarlo en el margen dado para cada turno. Por tanto, será necesario parar la búsqueda en algún punto (según el diseño que realice el alumno) y dar, como pagos (valores de los nodos) a los nodos "hoja" del árbol de búsqueda creado por el estudiante, valores heurísticos que intenten representar cómo de beneficioso es dicho estado para futuras acciones en el juego. La heurística a usar es libre y debe ser ideada por el estudiante. Recordemos también que el cálculo de una heurística debe ser sencillo computacionalmente. Por tanto, se podrá hacer uso de cualquier información que el agente pueda recabar durante el juego para calcular esta heurística (puntuación de cada personaje, número de semillas en alguna o algunas casillas, etc.).

6. Método de evaluación y entrega de prácticas

Se deberán entregar los dos ficheros ".h" y ".cpp", junto con un documento elaborado por el alumno (memoria de prácticas). Deberán compilar según se ha indicado en este guión de prácticas, en cualquier PC con g++ instalado. El alumno elegirá entre abordar el problema basándose en una de las técnicas estudiadas en el tema de teoría correspondiente a agentes con adversario (juegos), sabiendo que:

- La práctica se evaluará de 0 a 10.
- La memoria de prácticas se evalúa de 0 a 3 (valor **M**). La redacción de la memoria tiene formato libre, pero deberá responder a las siguientes cuestiones:
  - Explicación del diseño de "estado" para realizar la búsqueda. Relación de las estructuras de datos que lo implementan con el diseño especificado. Indicar dónde se encuentran implementadas dichas estructuras de datos dentro de la implementación (**no se requiere código fuente**, se pide una descripción del diseño realizado por el alumno e indicar dónde está implementada cada estructura de datos en los ficheros .h/.cpp).
  - Explicación del diseño del algoritmo implementado. **Aquí no se requiere** que el alumno describa la teoría ya explicada en clase sobre el algoritmo minimax, poda α-β, etc., sino que explique qué esquema sigue, cómo ha diseñado el algoritmo para explorar el espacio de estados según el diseño de estado del apartado anterior, cuándo realiza la búsqueda, cómo se almacena el plan, etc. Realizar, además, una explicación de cómo este diseño se plasma en la implementación (**no se requiere código fuente**, sino una explicación de cómo el algoritmo finalmente ha sido implementado, si han sido necesarias estructuras de datos adicionales, qué funciones/procedimientos están relacionados con el mismo en la implementación, etc.).
  - Explicación de la heurística utilizada. Se requiere exponer la fórmula o una descripción de la heurística seleccionada por el alumno, justificando el porqué de cada elemento que compone dicha heurística.
- Ejecución de la solución del alumno frente a bots propuestos por el profesorado (evaluado de 0 a 3 puntos, valor **B**). La solución del alumno se ejecutará contra dos bots por parte del profesorado:
  - **GreedyBot.** El alumno **dispondrá de la implementación** este bot desde el comienzo de las prácticas. Es un algoritmo simple que decide la acción a realizar basándose en una heurística voraz. La solución del alumno se ejecutará contra **GreedyBot** actuando como jugador 1 y como jugador 2. Límite de tiempo por turno: 2 segundos (aproximadamente 140.000 estados generados).
  - **MuerteaLosCobardes.** El alumno **no dispondrá de la implementación** de este bot. La solución del alumno se ejecutará contra **MuerteaLosCobardes** actuando como jugador 1 y como jugador 2. Límite de tiempo por turno: 2 segundos (aproximadamente 140.000 estados generados).
  - **La calificación de este apartado se calculará de la siguiente forma:**
    - 1,5 puntos si la solución del alumno es capaz de ganar a **GreedyBot** como jugador 1 y como jugador 2. Calificación 0 si no gana como jugador 1 o como jugador 2.

- 3 puntos si la solución del alumno es capaz de ganar a **MuerteaLosCobardes** como jugador 1 y como jugador 2. Calificación 0 si no gana como jugador 1 o como jugador 2.
- **Se entiende que, si el alumno es capaz de ganar a MuerteaLosCobardes, también será capaz de ganar a GreedyBot**, dado que el primero de ellos es capaz de ganar a este segundo.

- Liga oficial (valor **C**, de 0 a 4 puntos). Los resultados de la liga oficial se reflejarán en la calificación, considerando la puntuación de la solución de cada alumno, donde los alumnos que obtengan una mayor puntuación tendrán una calificación de valor numérico 4, y los alumnos que no consigan ningún punto la mínima calificación. Dicha puntuación se calculará como 3*(nº de partidas ganadas)+(nº de partidas empatadas).

La nota final de la práctica se calculará de la siguiente forma:

Calificación= valor M + valor B + valor C

Obviamente, las prácticas que se detecten copiadas (incluso parcialmente) implicarán un suspenso de la práctica tal como se detalla en la normativa de exámenes vigente, con calificación 0.

6.1. Entrega de prácticas

Se pide desarrollar un programa (modificando el código de los ficheros del simulador 'AlumnoBot.cpp' y 'AlumnoBot.hpp') con el comportamiento requerido para el agente. Estos ficheros deberán entregarse mediante la plataforma web de la asignatura, en un fichero ZIP que no contenga carpetas ni subcarpetas. El archivo ZIP deberá contener sólo el código fuente de estos dos ficheros con la solución del alumno así como un fichero de documentación en formato PDF que describa el comportamiento implementado con un máximo de 5 páginas.

No se evaluarán aquellas prácticas que contengan ficheros ejecutables o virus

6.2. Fechas Importantes

La fecha de entrega final de la práctica será a final de curso, según el calendario académico oficial para el curso 2017-2018. El profesor comunicará en la clase de prácticas la fecha límite específica de entrega para cada grupo de prácticas. La entrega deberá efectuarse a través de la plataforma docente de la asignatura.

Nota: Las fechas límite de entrega representan el concepto de hasta esa fecha. Por consiguiente, es posible hacer la entrega antes de la fecha indicada.