



UNIVERSIDAD  
DE GRANADA



# Inteligencia Artificial

## Grado en Ingeniería Informática

### Seminario 3

Este documento está protegido por la Ley de Propiedad Intelectual ([Real Decreto Ley 1/1996 de 12 de abril](#)).

Queda expresamente prohibido su uso o distribución sin autorización del autor.

Departamento de Ciencias de la Computación e Inteligencia Artificial  
<http://decsai.ugr.es>

### Objetivos de la práctica

- Familiarizarse con una aplicación concreta de Inteligencia Artificial (IA), dentro del ámbito de los agentes desplegados en entornos con adversario.
- Aprender a aplicar técnicas de búsqueda basadas en algoritmos para juegos.
- Implementar un agente para resolver un problema concreto en este ámbito: El juego del Mancala.



### Amplía este seminario en...

- Diapositivas de teoría, tema 4 “Búsqueda con adversario y juegos”
- Bibliografía básica y recomendada de la asignatura (ver guía docente).

### Anotación sobre estas diapositivas:

El contenido de estas diapositivas es esquemático y representa un apoyo para las clases prácticas. No se considera un sustituto de los apuntes de la asignatura.

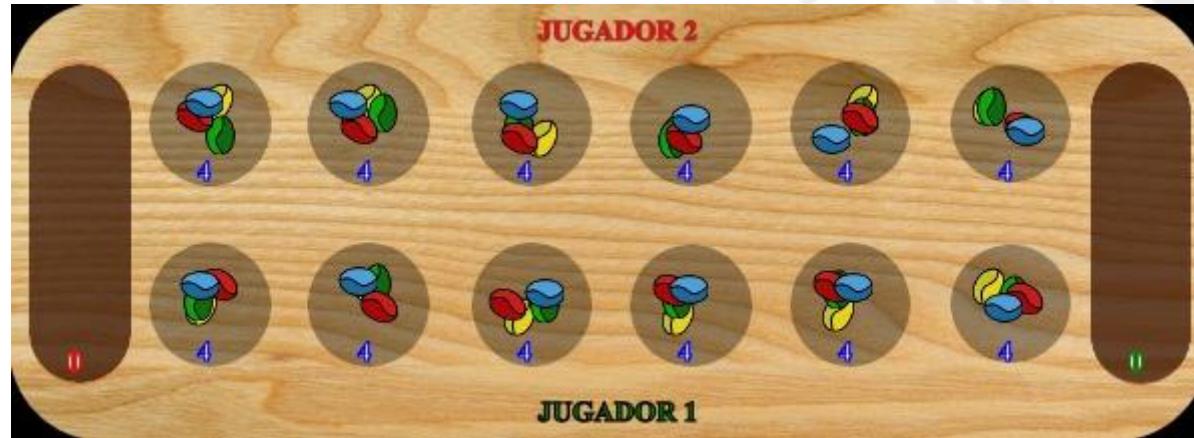
Se recomienda al alumno completar estas diapositivas con notas/apuntes propios, tomados en clase y/o desde la bibliografía principal de la asignatura.



1. El juego del Mancala
2. Objetivos de la práctica
3. El simulador
4. Creación de un bot
5. La liga oficial
6. ¿Una liga extraoficial?
7. Evaluación

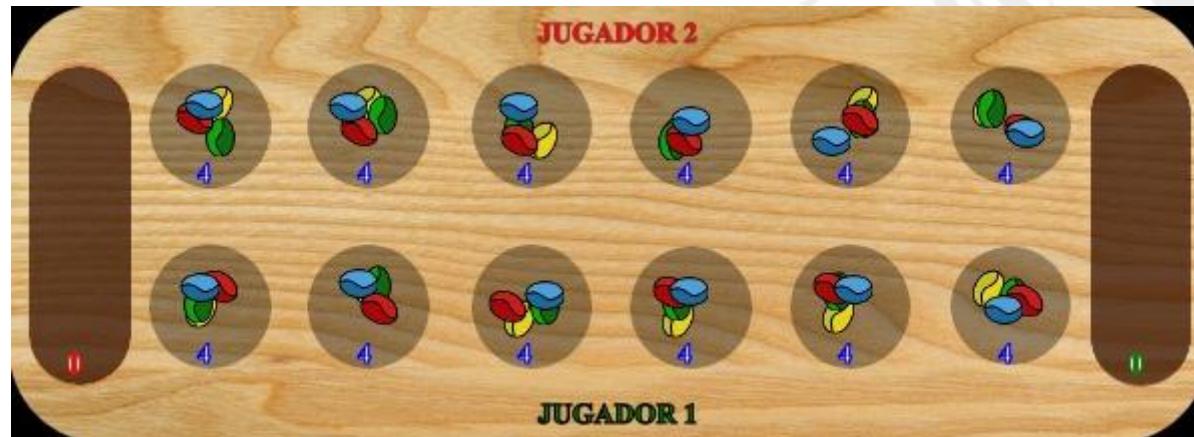
### El Mancala

- Es un juego de tablero, para 2 jugadores.
- Se juega por turnos.
- Procedentes de África (fundamentalmente del Antiguo Egipto) y de Asia.
- Hay muchas versiones y derivados del Mancala. Nosotros nos centraremos en la más popular.



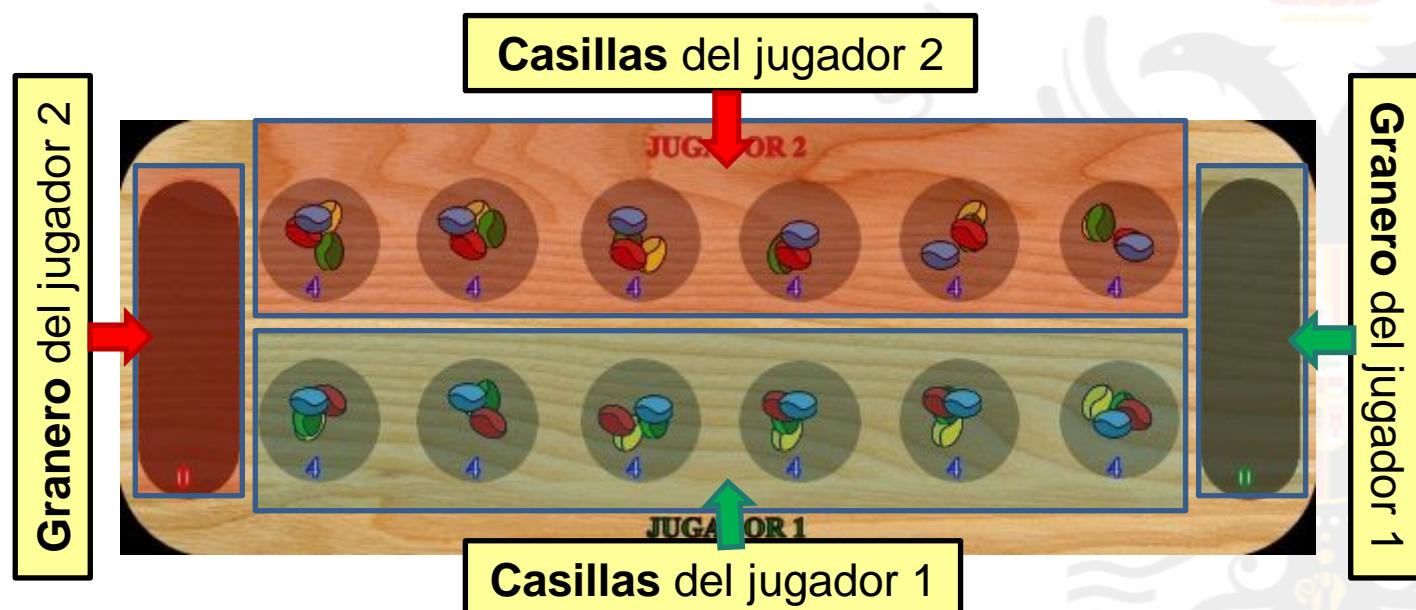
## El Mancala: Elementos del juego

- 12 **hoyos de cultivo o casillas** (6 para cada jugador), en dos filas de 6.
- 48 **semillas**, inicialmente repartidas equitativamente entre las 12 casillas (4 semillas por casilla, al principio del juego).
- 2 **graneros**, situados en los extremos del tablero. Cada uno pertenece a un jugador.



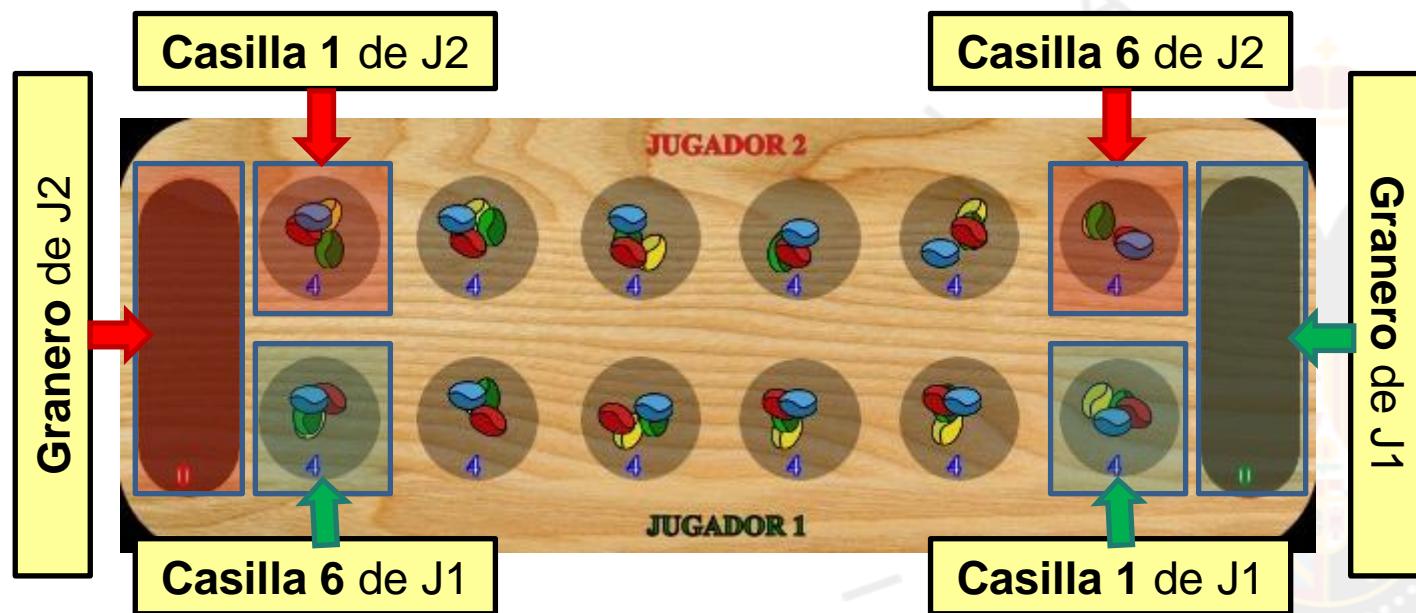
## El Mancala: Elementos del juego

- 12 **hoyos de cultivo** o **casillas** (6 para cada jugador), en dos filas de 6.
- 48 **semillas**, inicialmente repartidas equitativamente entre las 12 casillas (4 semillas por casilla, al principio del juego).
- 2 **graneros**, situados en los extremos del tablero. Cada uno pertenece a un jugador.



## El Mancala: Elementos del juego

- Las casillas de cada jugador se encuentran numeradas de 1 a 6, contando desde el granero del jugador al que nos refiramos (J1, J2).
- Así, la casilla “x” de un jugador se encuentra enfrente de la casilla “7-x” del jugador contrario en el tablero.



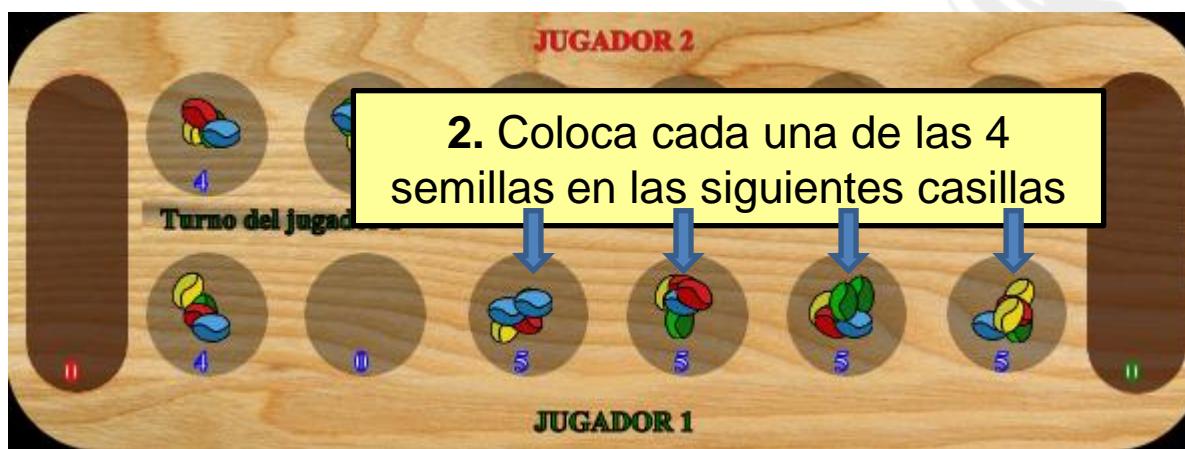
## El Mancala: Dinámica del juego

- El Mancala se juega secuencialmente, por turnos, comenzando por J1.
- En cada turno, el jugador que mueve selecciona un **movimiento de siembra**, escogiendo **una de sus casillas**.
- La **siembra** sobre una casilla consiste en quitar todas las semillas de la casilla seleccionada e ir colocando una única semilla en las casillas inmediatamente inferiores, secuencialmente, **incluido en su granero si es posible**.

Ejemplo: Estado del juego en el que le toca jugar a J1. Supongamos que J1 selecciona sembrar la casilla 5.



Ejemplo: Estado del juego en el que le toca jugar a J1. Supongamos que J1 selecciona sembrar la casilla 5.



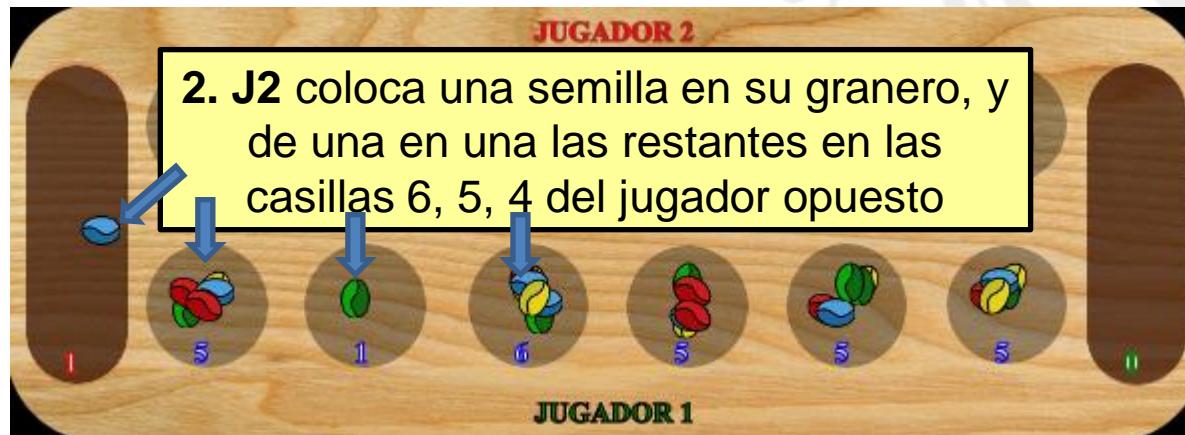
## El Mancala: Dinámica del juego

- Si tras colocar todas las semillas, aún quedasen, estas se seguirán colocando de una en una en las casillas del contrario, de mayor a menor número de casilla.
- **Nunca colocaremos una semilla en el granero del contrario.**
- Si llegamos a poner una semilla en la casilla 1 del contrario, y aún siguen quedando semillas, comenzaremos a poner de nuevo una semilla en cada una de nuestras casillas, en orden descendente.

Ejemplo: Estado del juego en el que le toca jugar a J2. Supongamos que J2 selecciona sembrar la casilla 1.



Ejemplo: Estado del juego en el que le toca jugar a J2. Supongamos que J2 selecciona sembrar la casilla 1.



## El Mancala: Objetivo del juego

- Obtener mayor número de semillas en nuestro granero que nuestro contrincante.
- **Fin del juego:** Cuando alguno de los dos jugadores no tenga semillas en ninguna de sus casillas.
- Si se llega al fin del juego y uno de los jugadores aún tiene semillas en sus casillas, automáticamente las recogerá y las sumará a su granero.

Ejemplo: Último movimiento del juego. Le toca a mover a J2.



Ejemplo: Último movimiento del juego. Le toca a mover a J2. Sólo puede hacer siembra sobre la casilla 1.

1. J2 realiza el movimiento de siembra y suma en su granero 20 semillas.
2. Se ha quedado sin semillas en sus casillas, por lo que termina el juego.
3. Todas las semillas de J1 (sólo hay una, en la casilla 3 de J1) van al granero de J1, sumando 28 semillas.
4. Gana J1 por 28 semillas a 20.



## El Mancala: reglas especiales

- **Turno extra:** Cuando un jugador siembra una casilla y la última semilla la deposita en su granero, gana un turno extra y vuelve a jugar.

Ejemplo:

Primer movimiento de J1. Si siembra la casilla 4, gana turno extra.

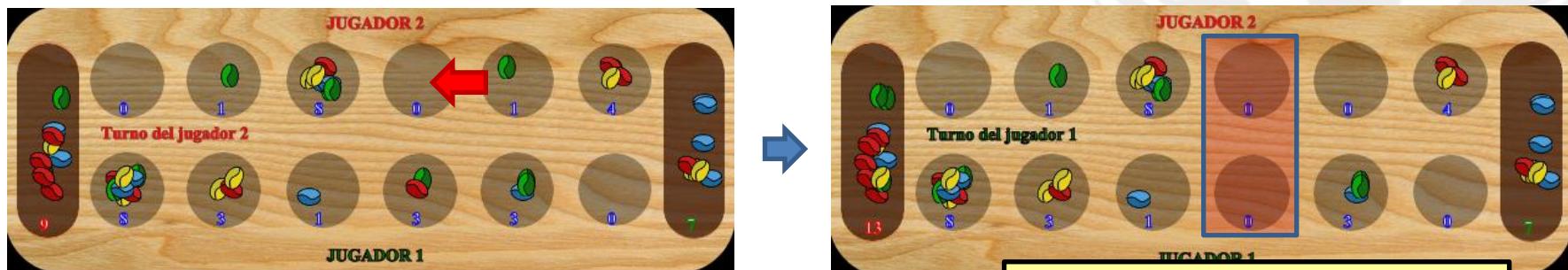


## El Mancala: reglas especiales

- **¡Robo!:**  Si al realizar una siembra, la última semilla del jugador cae en una de sus casillas que esté vacía, pero la opuesta del contrincante no, entonces el jugador lleva directamente esa semilla y las de la casilla del jugador contrario a su granero.

Ejemplo:

Turno de J2. Si decide sembrar la casilla 5 (1 semilla), esta última se depositará en la casilla 4 (que está vacía, pero la opuesta del contrario –casilla 3 de J1- no lo está). J2 ganará su semilla y las 2 que hay en la casilla 3 de J1.



J2 ha robado las semillas de su casilla 4 y de la casilla 3 de J1.

## El Mancala: reglas especiales

- Inmolación:** Si un jugador intenta hacer un movimiento no permitido (sembrar una casilla que no tiene semillas), automáticamente perderá la partida por 48 semillas a 0.

Ejemplo:

Turno de J1. Si decide sembrar la casilla 3 (0 semillas), perderá la partida, y J2 la ganará por 48 semillas a 0.



J1 pierde por intentar  
sembrar su casilla 3.

## Un ejemplo concreto:

El **videotutorial “EjemploPartidaHumanos.mp4”** explica el desarrollo de una partida entre dos jugadores humanos, exponiendo varios casos de las reglas especiales de “turno extra” y “robo”.



Se anima al alumno a jugar varias partidas humano/humano para familiarizarse con las reglas del juego.



- » 1. El juego del Mancala
- 2. Objetivos de la práctica
- 3. El simulador
- 4. Creación de un bot
- 5. La liga oficial
- 6. ¿Una liga extraoficial?
- 7. Evaluación

### Objetivo de la práctica

- Implementar un jugador artificial inteligente para el juego del Mancala, que trate de ganar.
- Deberá poder jugar como J1 o como J2, indistintamente.
- Deberá implementar comportamiento deliberativo basado en búsqueda en entornos con adversario (Minimax/Maximin, poda  $\alpha$ - $\beta$ ...).
- Tiempo de turno limitado: 2 segundos.
  - Se deberá diseñar e implementar una buena heurística para saber cómo de “bueno” es un estado para el jugador.
  - **Estimación de cara a la evaluación:** 2 segundos  $\approx$  generar 140.000 estados del juego en el simulador.

### Material

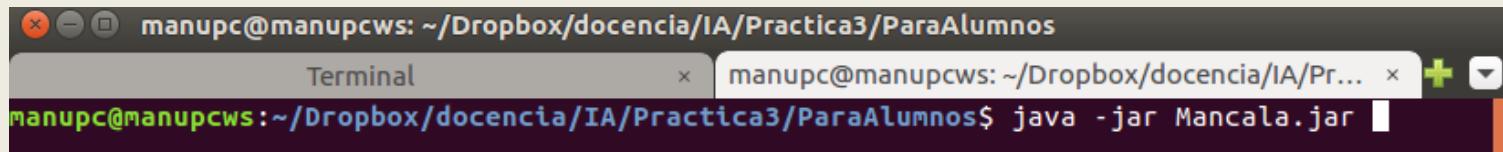
- Se proporciona al estudiante:
  - Un simulador de Mancala con entorno gráfico.
  - Un simulador de Mancala sin entorno gráfico (para hacer pruebas más rápidamente).
  - El esqueleto del agente en C++, para que el estudiante cree su Bot.
  - Dos ejemplos de bots muy simples: **RandomBot** y **GreedyBot**.



1. **El juego del Mancala**
2. **Objetivos de la práctica**
3. **El simulador**
4. **Creación de un bot**
5. **La liga oficial**
6. **¿Una liga extraoficial?**
7. **Evaluación**

## Descripción general del simulador

- El simulador ha sido construido completamente por los profesores de la asignatura. Está implementado en lenguaje Java para facilitar la portabilidad.
- **Requisitos para poder ejecutar el simulador:**
  - Java JRE o Java JDK 8 instalado (**no Java 7, ni Java 9: Debe ser 8**).
- **Ejecución del simulador con interfaz gráfica:**
  - Por línea de comandos, asumiendo que *java* está incluido en el path y que nos encontramos en la carpeta de la práctica:



```
manupc@manupcws: ~/Dropbox/docencia/IA/Practica3/ParaAlumnos
Terminal manupc@manupcws: ~/Dropbox/docencia/IA/Pr... + manupc@manupcws: ~/Dropbox/docencia/IA/Practica3/ParaAlumnos$ java -jar Mancala.jar
```

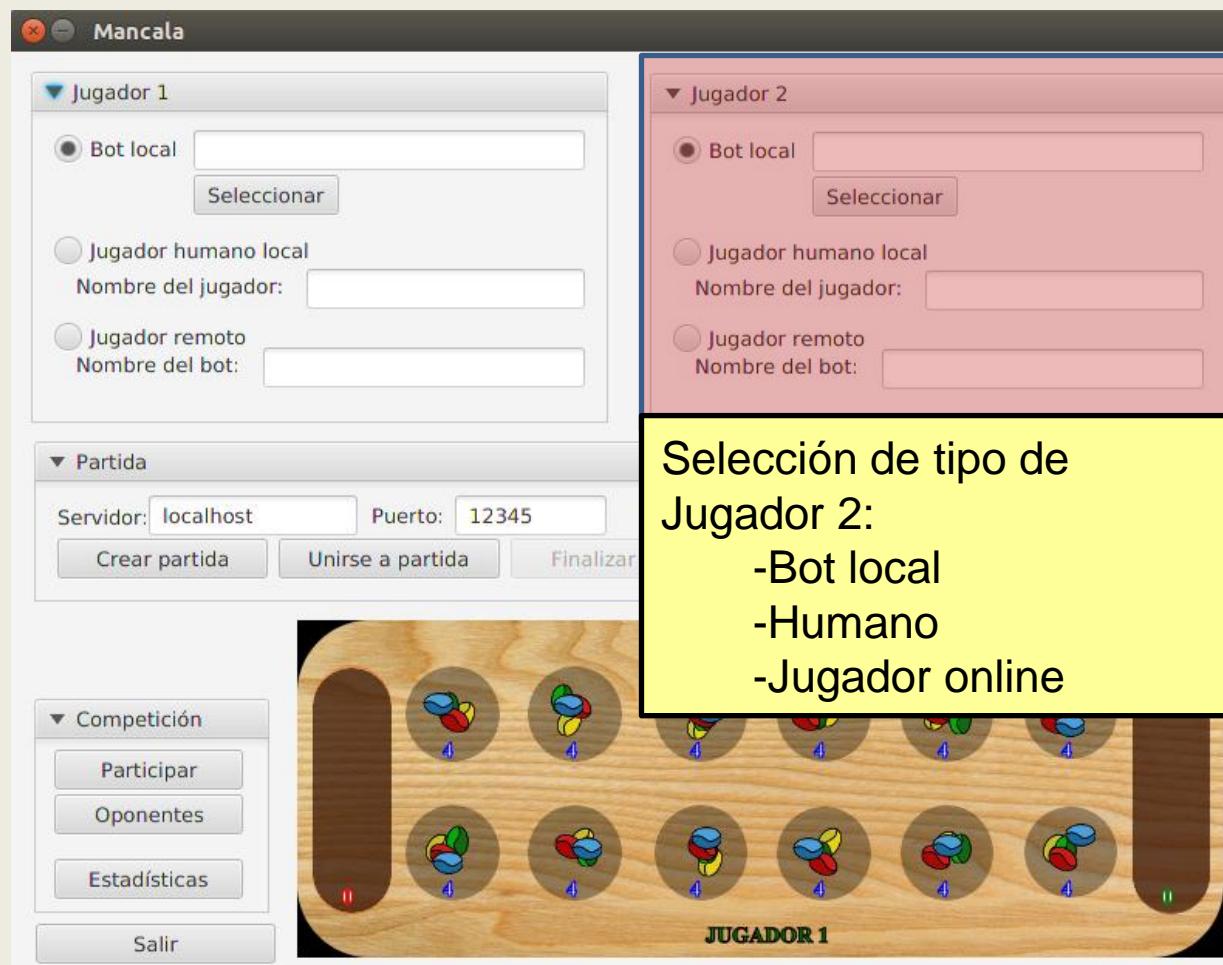
## Descripción general del simulador



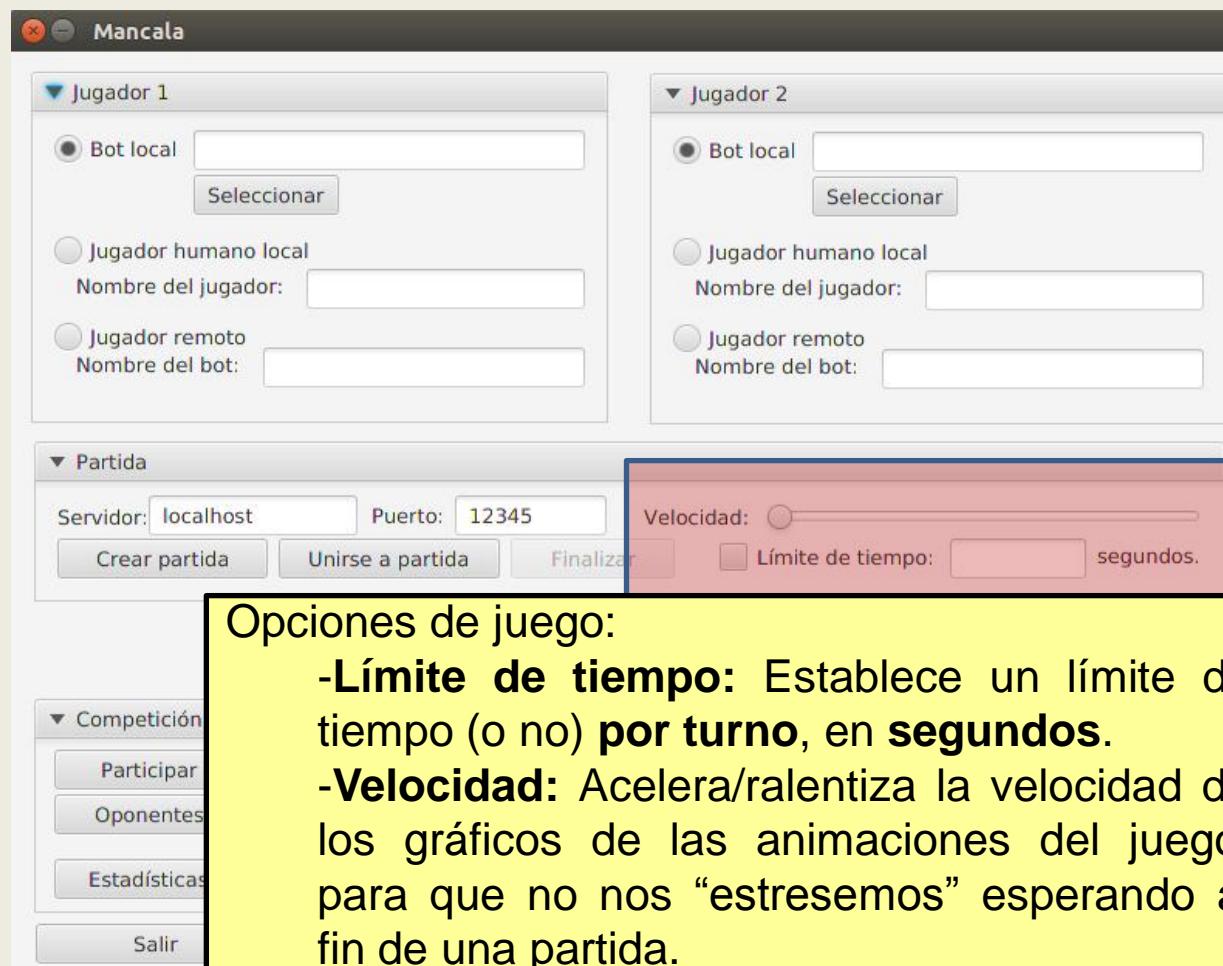
## Descripción general del simulador



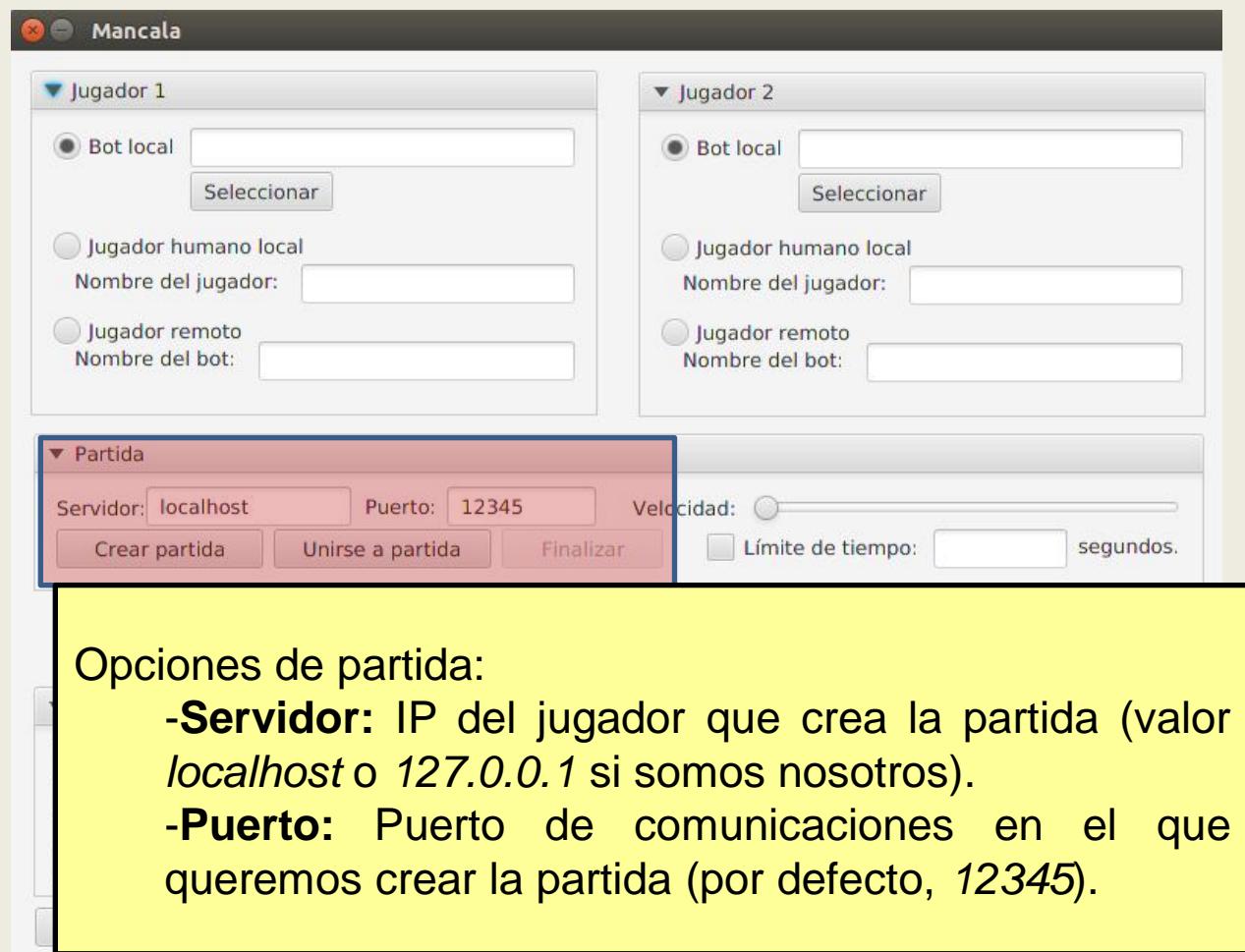
## Descripción general del simulador



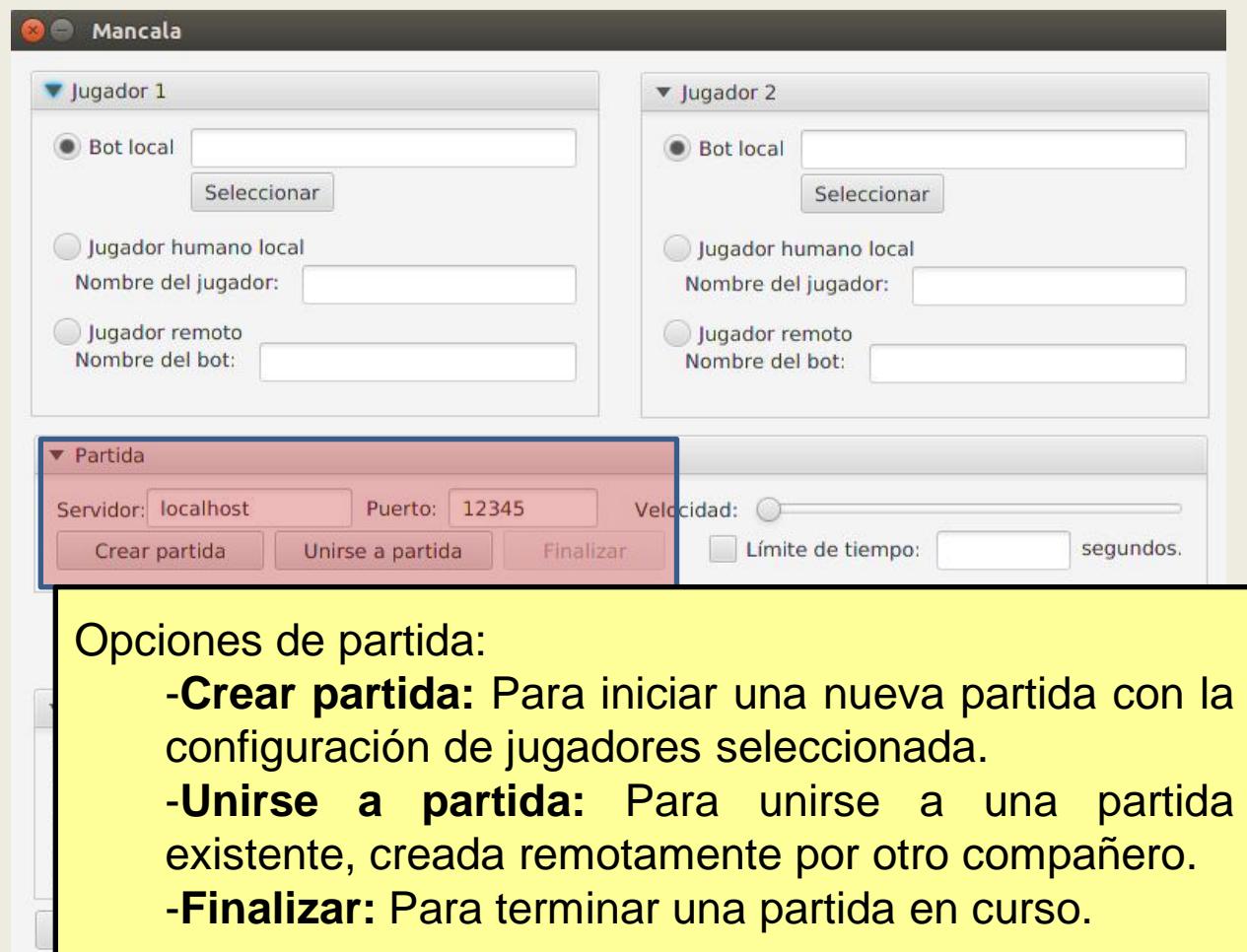
## Descripción general del simulador



## Descripción general del simulador



## Descripción general del simulador



### Descripción general del simulador

- El **videotutorial “EjemploPartidaHumanos.mp4”** explica cómo configurar una partida local entre 2 humanos, configurando la velocidad de las animaciones del juego.
- El **videotutorial “EjemploPartidaOnline.mp4”** explica cómo configurar una partida online entre estudiantes, uno con rol de servidor (crear la partida) y otro con el rol de cliente (se une a la partida).
- **¡OJO! No se podrán crear partidas online dentro de la red inalámbrica de la UGR (eduroam/cvi-ugr). El servicio de informática (CSIRC) impide acceder a los puertos de servidores no identificados (como el servidor que crea el alumno que inicia la partida de Mancala online).**

## Bots de ejemplo

- Se proporcionan dos bots de ejemplo al estudiante:
  - Carpeta “**RandomBot**”: Bot con comportamiento aleatorio.
  - Carpeta “**GreedyBot**”: Bot con comportamiento simple basado en heurística greedy.

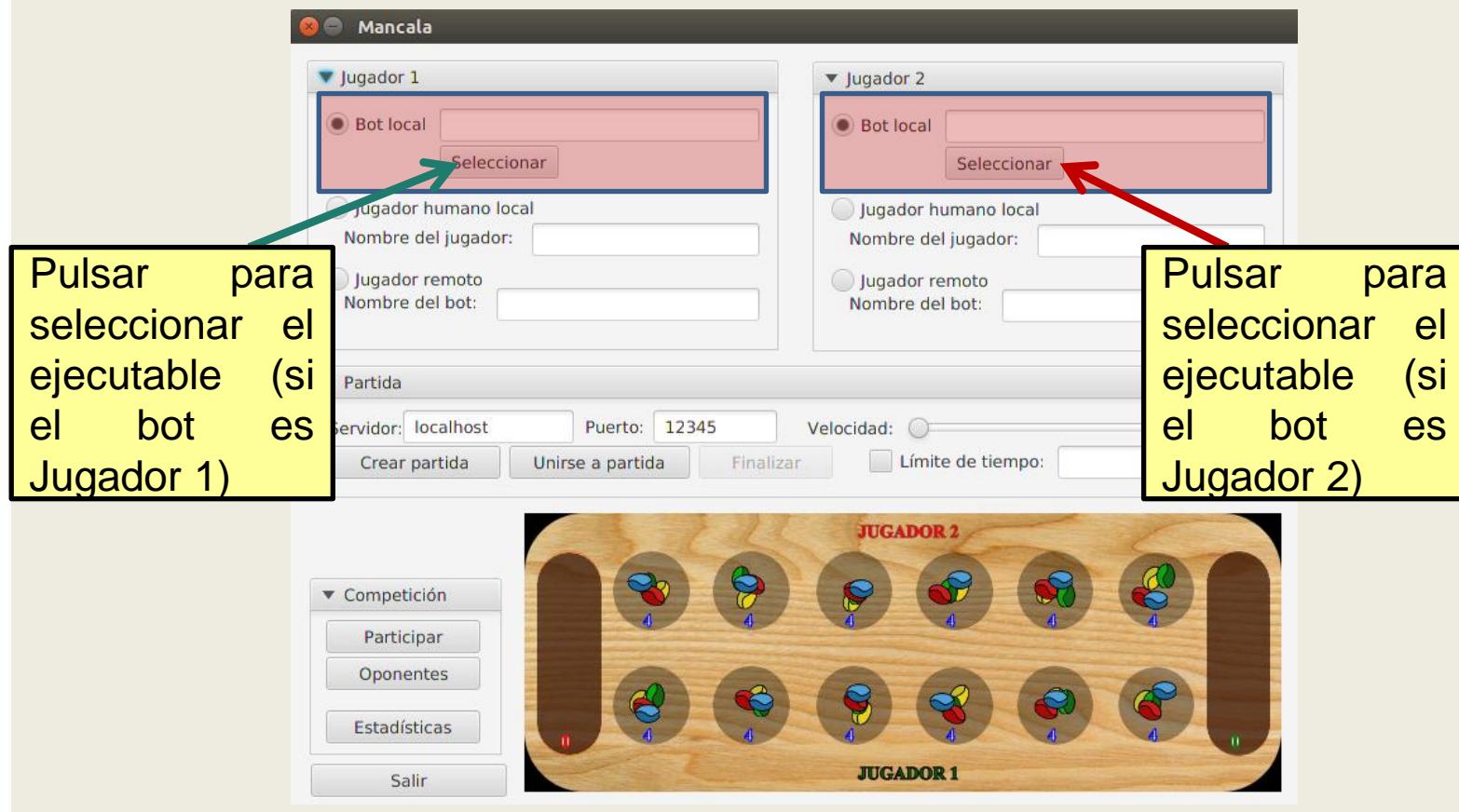
## Cómo compilar los bots de ejemplo

- Se deberá entrar a la carpeta del bot a compilar y escribir (suponiendo compilador g++ en el path, y dependiendo del bot en cuestión):

```
g++ -oRandomBot.exe Bot.cpp GameState.cpp main.cpp SimulatorLink.cpp RandomBot.cpp -I. -lm -std=c++11  
g++ -oGreedyBot.exe Bot.cpp GameState.cpp main.cpp SimulatorLink.cpp GreedyBot.cpp -I. -lm -std=c++11|
```

## Ejecución de un bot en el simulador

- En la pestaña del jugador correspondiente, seleccionaremos la opción “Bot local” y pulsaremos sobre el botón “Seleccionar” para escoger el fichero ejecutable del bot.



### Ejecución de un bot en el simulador

- El resto de pasos son equivalentes a los ejemplos anteriores para crear una partida.
- El **videotutorial “EjemploPartidaBots.mp4”** explica cómo configurar una partida local entre 2 bots (podría hacerse de forma análoga entre humano/bot o partida online bot/bot, combinando las opciones de los videotutoriales anteriores con este último).

### Ejecución del simulador sin interfaz gráfica

- Si queremos ejecutar las partidas más rápidamente, se proporciona al alumno la opción de ejecutar el simulador sin interfaz gráfica.
- La ejecución del simulador sin interfaz gráfica se debe realizar de la siguiente forma (por línea de comandos):

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

## Ejecución del simulador sin interfaz gráfica: Ejemplo

- Ejemplo de RandomBot vs GreedyBot con 1 segundo de tiempo límite por turno.

```
manupc@manupcws: ~/Dropbox/docencia/IA/Practica3/ParaAlumnos
manupc@manupcws:~/Dropbox/docencia/IA/Practica3/ParaAlumnos$ java -jar MancalaNoGUI.jar -p
1 RandomBot/RandomBot.exe -p2 GreedyBot/GreedyBot.exe -t 1
```

- El simulador mostrará la información de la partida por consola, con los resultados de la misma al finalizar.

```
manupc@manupcws: ~/Dropbox/docencia/IA/Practica3/ParaAlumnos

[Simulador]: Turno del jugador 1.
[Simulador]: El jugador 1 realiza el movimiento 3.
[Simulador]: Estado actual del tablero:
    J1: Granero=14 C1=0 C2=1 C3=0 C4=0 C5=0 C6=0
    J2: Granero=31 C1=1 C2=0 C3=0 C4=0 C5=0 C6=1

[Simulador]: Turno del jugador 2.
[Simulador]: El jugador 2 realiza el movimiento 6.
[Simulador]: Estado actual del tablero:
    J1: Granero=14 C1=0 C2=0 C3=0 C4=0 C5=0 C6=0
    J2: Granero=34 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 (RandomBot): 14
Tiempo del jugador 1 (RandomBot): 1000 milisegundos.
Puntos del jugador 2 (GreedyBot): 34
Tiempo del jugador 2 (GreedyBot): 1100 milisegundos.
Ganador: Jugador 2 (GreedyBot)
```



- 1. El juego del Mancala
- 2. Objetivos de la práctica
- 3. El simulador
- 4. Creación de un bot
- 5. La liga oficial
- 6. ¿Una liga extraoficial?
- 7. Evaluación



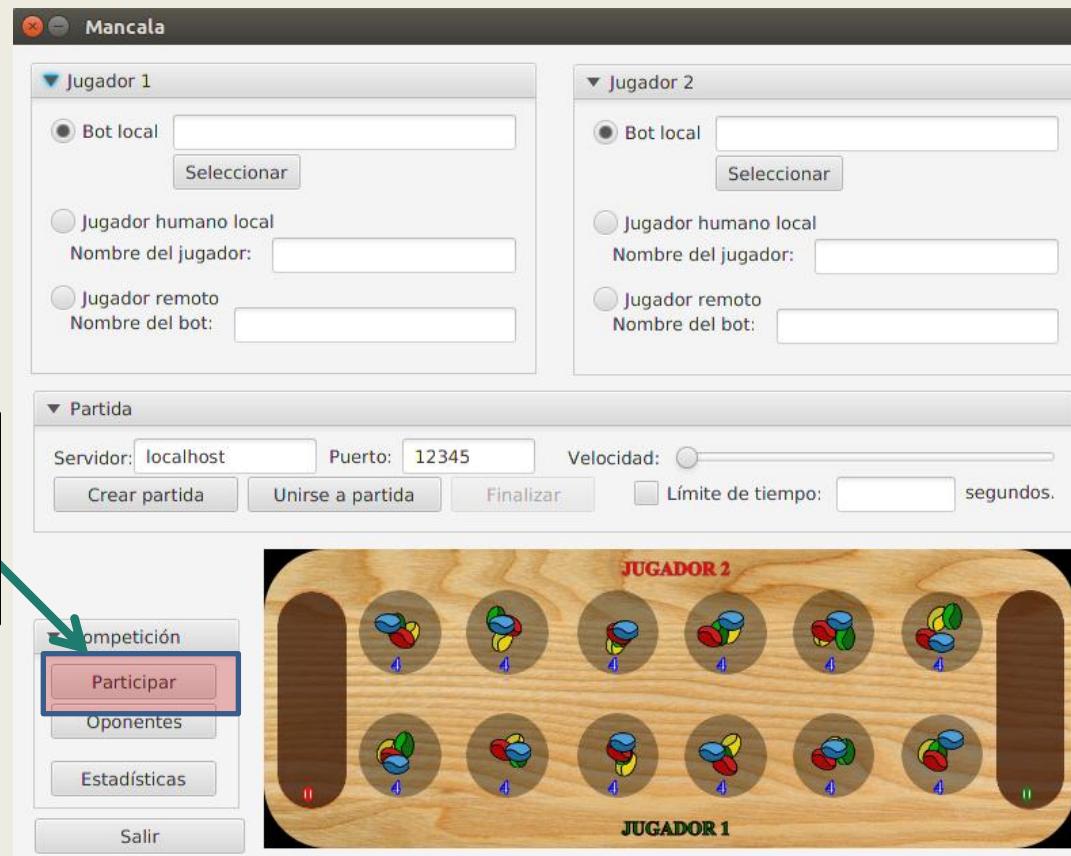
### Cómo crear un bot: Condiciones

- En primer lugar, habrá que registrar el nombre/nick de un bot en el simulador.
- **No pueden existir dos bots con el mismo nombre dentro del sistema.**
- Para ello, hay que **registrarse en la liga**.
- La liga se explicará más adelante en estas diapositivas.



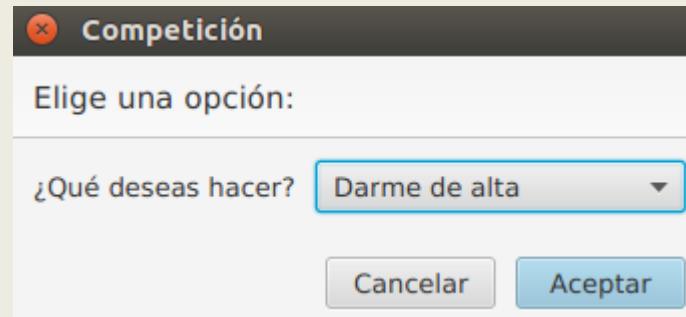
## Cómo crear un bot: Registro

- El registro en el sistema se realiza mediante el botón “Participar” del menú de “liga” del simulador gráfico.

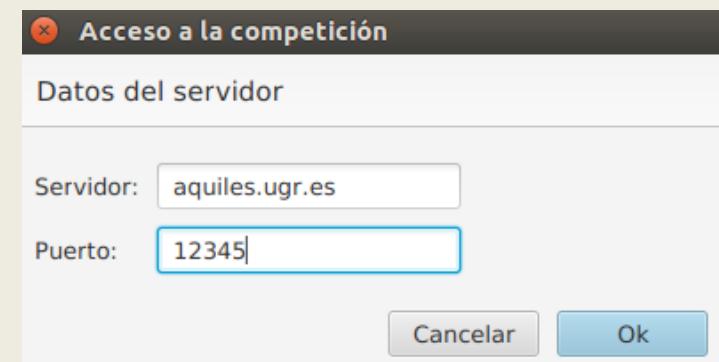


## Cómo crear un bot: Registro

- Seleccionaremos la opción “**Darme de alta**”.



- Los datos del servidor de la liga aparecen por defecto:
  - **Servidor:** aquiles.ugr.es
  - **Puerto de comunicaciones:** 12345



## Cómo crear un bot: Registro

- En el formulario, rellenaremos nuestros datos, nuestro **grupo de prácticas** y el nombre del bot, sabiendo que:
  - **El nombre del bot será único.** No podrá haber dos estudiantes cuyo bot tenga el mismo nombre.
  - Las normas para construir el nombre del bot:
    - **Cadena de caracteres alfanuméricos que comienzan por una letra.**
    - **Sensible a mayúsculas.**

Formulario de registro en la competición

Nombre:	Manuel
Apellidos:	Pegalajar Cuellar
e-Mail:	manupc@ugr.es
Nombre del bot:	ManuBot
Grupo de prácticas:	D2
Clave de acceso:	██████████████

## Cómo crear un bot: Registro

- Para seleccionar el grupo de prácticas, rígete por la siguiente tabla:

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

### Formulario de registro en la competición

Nombre:

Apellido:

Mail:

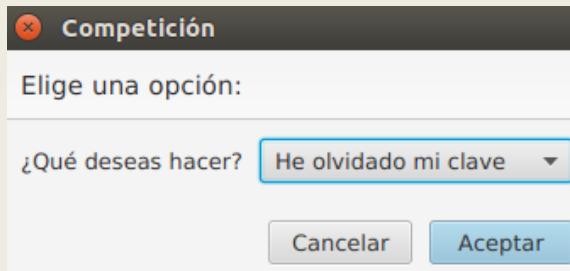
Nombre del bot:

Grupo de prácticas:

Clave de acceso:

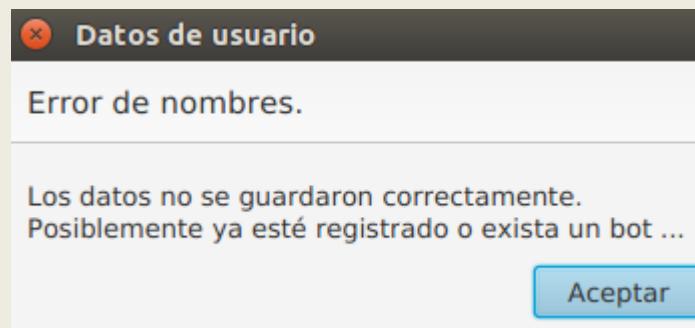
## Cómo crear un bot: Registro

- Además:
  - El e-mail deberá ser el de la cuenta **@correo.ugr.es**
  - La clave de acceso será específica para Mancala. Deberemos escoger una clave para la liga, **que no olvidemos**, y que sólo utilicemos para el simulador (por motivos de seguridad, se recomienda no utilizar una de nuestras claves de UGR o personales).
  - **¿Y si olvido mi clave?**
    - Podremos recuperarla desde el botón “**Participar**” del menú “**liga**” del simulador, seleccionando la opción “**He olvidado mi clave**” y llenando nuestro e-mail.

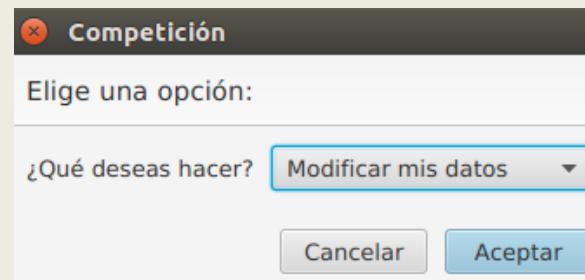


## Cómo crear un bot: Registro

- Si el nombre del bot es válido y no “está pillado” ya por otra persona, se mostrará un mensaje de confirmación. En caso contrario, se mostrará un mensaje de error:



- **¿Y si me he confundido de grupo o al llenar mis datos?**
  - Podremos modificarlos desde el botón “Participar” del menú “liga” del simulador, seleccionando la opción “Modificar mis datos” **(salvo el e-Mail y el nombre del bot).**

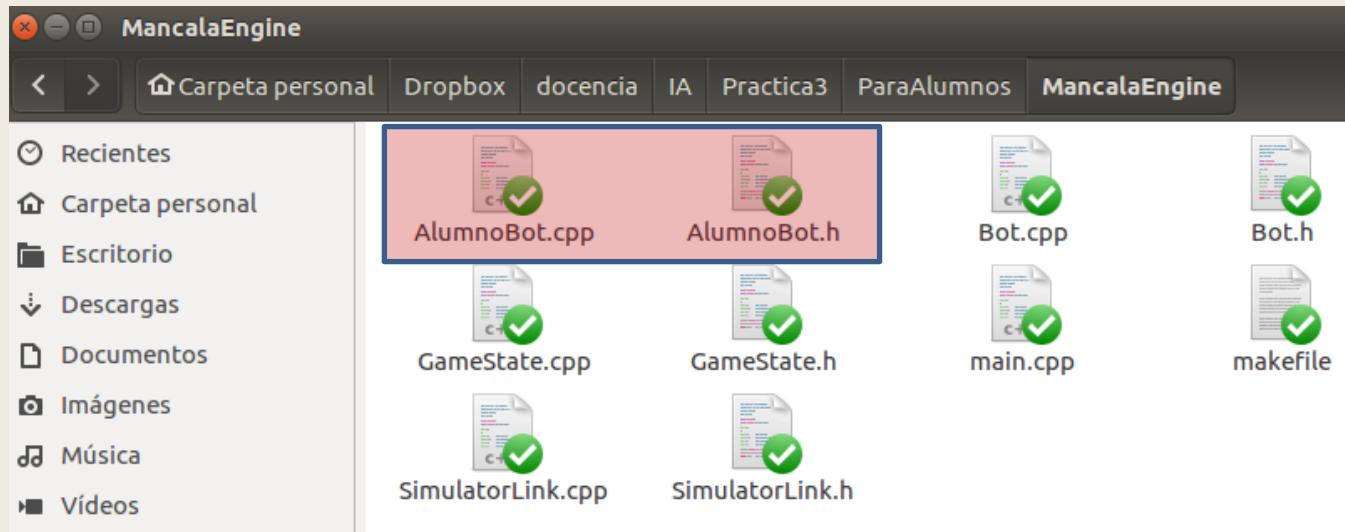


### Ya tengo mi bot registrado. Y ahora, ¿qué?

- Debemos crear dos ficheros, uno de cabecera (.h) y uno de implementación (.cpp), donde implementaremos el bot.
- Estos ficheros deberán crearse en la carpeta **MancalaEngine**, facilitada al estudiante para la realización de las prácticas.
- **El nombre de los ficheros debe ser el mismo que el del bot, tanto en mayúsculas como en minúsculas.**
- En el fichero .h, crearemos una clase que extiende de otra clase llamada **Bot**.
- **El nombre de esta clase también debe ser el mismo que el del bot.** En caso contrario, la práctica no funcionará.

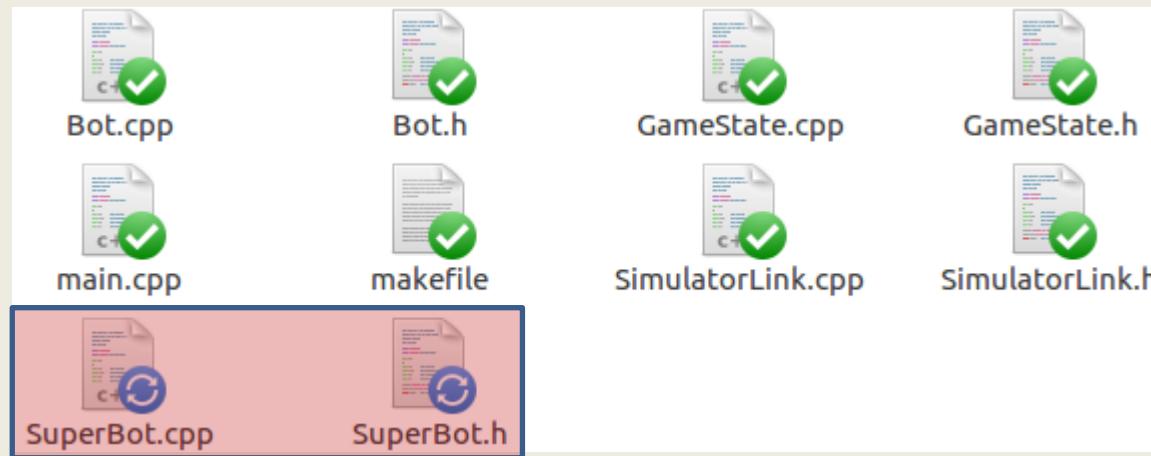
## Creación del bot (I)

- La carpeta **MancalaEngine** trae un bot de ejemplo “**AlumnoBot**”, que podemos reutilizar para facilitar la creación del bot.



## Creación del bot (II)

- Para crear el bot, vamos a suponer que el nombre que le hemos dado a nuestro bot es **SuperBot**. Seguiremos los siguientes pasos:
  1. Renombrar los ficheros **AlumnoBot.cpp** y **AlumnoBot.h** al nombre de nuestro bot (en el ejemplo: **SuperBot.cpp** y **SuperBot.h**).



## Creación del bot (III)

2. Abrir los dos ficheros, y reemplazar todo el texto “AlumnoBot” por el nombre de nuestro bot (en el ejemplo, “SuperBot”).

```
1 /*
2 * SuperBot.h
3 *
4 * Created on: 15 ene. 2018
5 * Author: manupc
6 */
7
8 #include "Bot.h"
9
10 #ifndef MANUPCBOT_H_
11 #define MANUPCBOT_H_
12
13 class SuperBot:Bot {
14 public:
15     SuperBot();
16     ~SuperBot();
17
18
19     void initialize();
20     string getName();
21     Move nextMove(const vector<Move> &adversary, const GameState &state);
22 };
23
24 #endif /* MANUPCBOT_H_ */
```

Ejemplo del fichero .h tras buscar y reemplazar **AlumnoBot** por **SuperBot**.

## Creación del bot (III)

2. Abrir los dos ficheros, y reemplazar todo el texto “AlumnoBot” por el nombre de nuestro bot (en el ejemplo, “SuperBot”).

```
1 /*  
2 * SuperBot.cpp  
3 *  
4 * Created on: 15 ene. 2018  
5 * Author: manupc  
6 */  
7  
8 #include "SuperBot.h"  
9  
10 #include <string>  
11 #include <cstdlib>  
12 #include <iostream>  
13 using namespace std;  
14  
15 SuperBot::SuperBot() {  
16     // Inicializar las variables necesarias para ejecutar la partida  
17 }  
18  
19  
20 SuperBot::~SuperBot() {  
21     // Liberar los recursos reservados (memoria, ficheros, etc.)  
22 }  
23  
24 void SuperBot::initialize() {  
25     // Inicializar el bot antes de jugar una partida  
26 }  
27  
28 string SuperBot::getName() {  
29     return "SuperBot"; // Sustituir por el nombre del bot  
30 }  
31  
32 Move SuperBot::nextMove(const vector<Move> &adversary, const GameState &state) {  
33  
34     Move movimiento= M_NONE;  
35  
36     // Implementar aquí la selección de la acción a realizar  
37  
38     // OJO: Recordatorio. NO USAR cin NI cout.  
39     // Para salidas por consola (debug) utilizar cerr. Ejemplo:  
40     // cerr<< "Lo que quiero mostrar"<<endl;  
41  
42  
43     // OJO: Recordatorio. El nombre del bot y de la clase deben coincidir.  
44     // En caso contrario, el bot no podrá participar en la competición.  
45     // Se deberá sustituir el nombre SuperBot como nombre de la clase por otro  
46     // seleccionado por el alumno. Se deberá actualizar también el nombre  
47     // devuelto por el método getName() acordeamente.  
48  
49     return movimiento;  
50 }
```

Ejemplo del fichero .cpp tras buscar y reemplazar **AlumnoBot** por **SuperBot**.

## Creación del bot (IV)

3. Abrir el fichero **main.cpp**. Igualmente, buscaremos el texto **AlumnoBot** y lo reemplazaremos por el nombre de nuestro bot.

```
2 #include "SimulatorLink.h"
3 #include "Bot.h"
4 #include <iostream>
5
6 // MODIFICAR: Hacer el include del fichero que contiene al Bot del alumno. Ejemplo
7 #include "SuperBot.h"
8
9
10 using namespace std;
11
12
13 int main() {
14
15     SimulatorLink sim; // Enlace con el simulador
16     Bot *bot= 0; // Bot que se ejecutará en el simulador
17
18
19     // MODIFICAR: Declarar aquí el bot del alumno. Ejemplo:
20     SuperBot *rb= new SuperBot();
21
22     // MODIFICAR: Asignar el bot del alumno a la variable bot. Ejemplo:
23     bot= (Bot *) rb;
24
25
26     // Se establece el bot del alumno como bot del simulador
27     sim.setBot(bot);
28
29     // Ejecutamos la simulación
30     bool salida= sim.run();
31
32
33     // MODIFICAR: Eliminamos el bot del alumno. Ejemplo:
34     delete rb;
35
36
37     // Ejecutamos la simulación
38     if (!salida) {
39
40         cerr<<"\n\n\tSIMULACION ABORTADA.\n\n";
41     } else {
42
43         cerr<<"\n\n\tFIN DE LA PARTIDA CON EXITO.\n\n";
44     }
45
46     return 0;
47 }
```

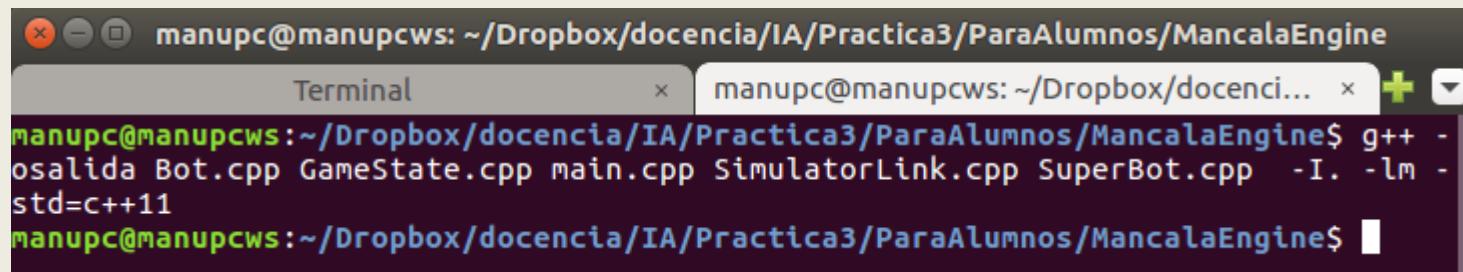
Ejemplo del fichero main.cpp  
tras buscar y reemplazar  
**AlumnoBot** por **SuperBot**.

## Creación del bot (V)

- Prueba de compilación. Nuestro bot deberá poder compilar con la siguiente orden:

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

### Compilación del ejemplo SuperBot:



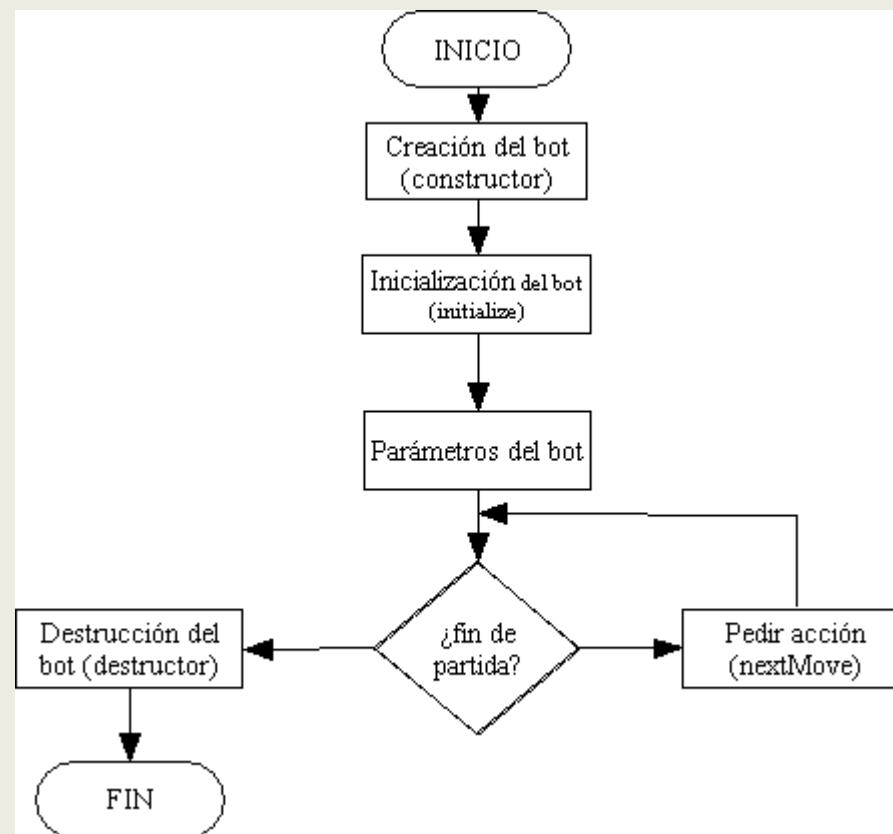
A screenshot of a terminal window titled "Terminal". The window shows a command-line interface with the following text:  
manupc@manupcws: ~/Dropbox/docencia/IA/Practica3/ParaAlumnos/MancalaEngine  
Terminal x manupc@manupcws: ~/Dropbox/docenci... +  
manupc@manupcws:~/Dropbox/docencia/IA/Practica3/ParaAlumnos/MancalaEngine\$ g++ -osalida Bot.cpp GameState.cpp main.cpp SimulatorLink.cpp SuperBot.cpp -I. -lm -std=c++11  
manupc@manupcws:~/Dropbox/docencia/IA/Practica3/ParaAlumnos/MancalaEngine\$ █

### Comunicaciones entre el simulador y el bot

- Durante el desarrollo de una partida, el simulador ejecutará el fichero ejecutable del bot, y se comunicará con él mediante E/S estándar.
- **MORALEJA: No se puede usar ni cin ni cout, porque el simulador los acapara ambos para él solito. Usar cin/cout en el bot (aunque sea para depuración) provocará que el bot no funcione.**
- **Pero sí que se puede utilizar cerr. De hecho, el log del simulador mostrará los mensajes del bot que este muestre por cerr.**

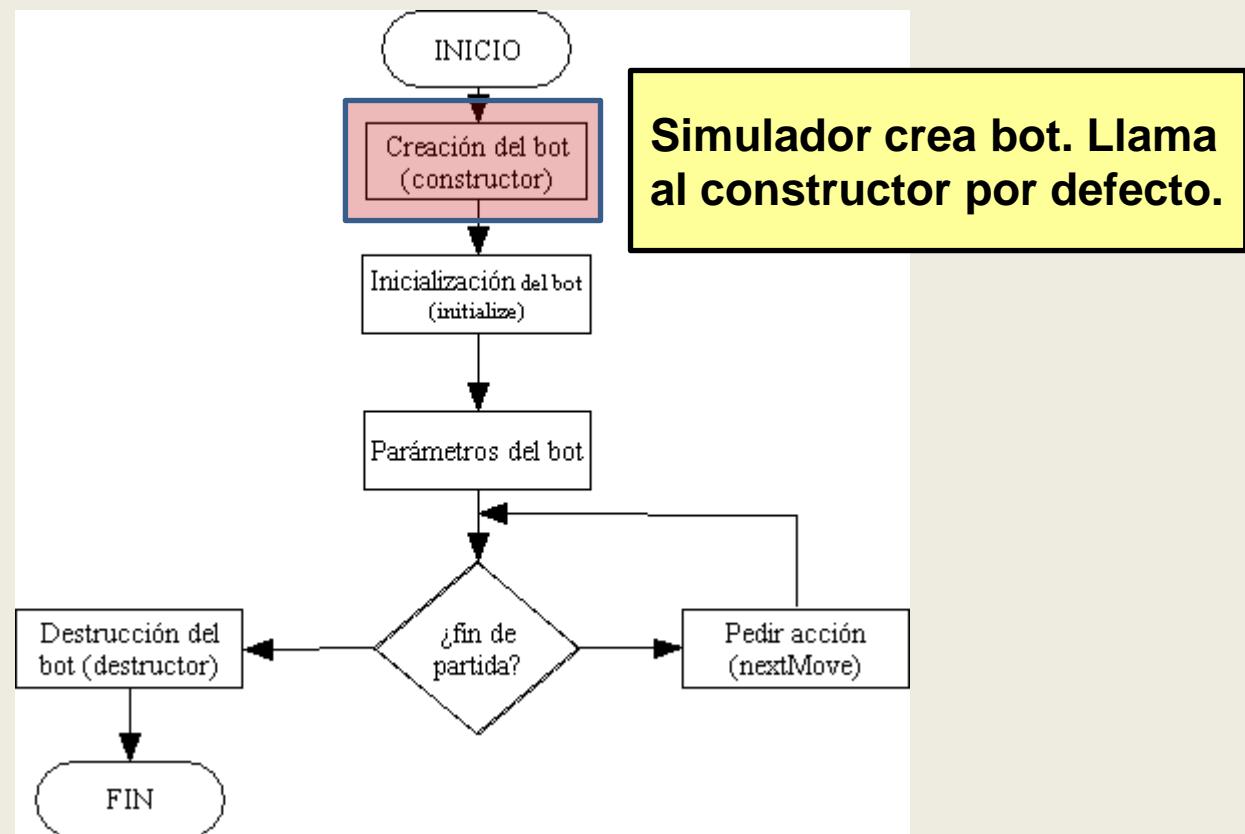
## Comunicaciones entre el simulador y el bot

- El ciclo de comunicaciones entre el simulador y el bot, durante el desarrollo de una partida, es el siguiente:



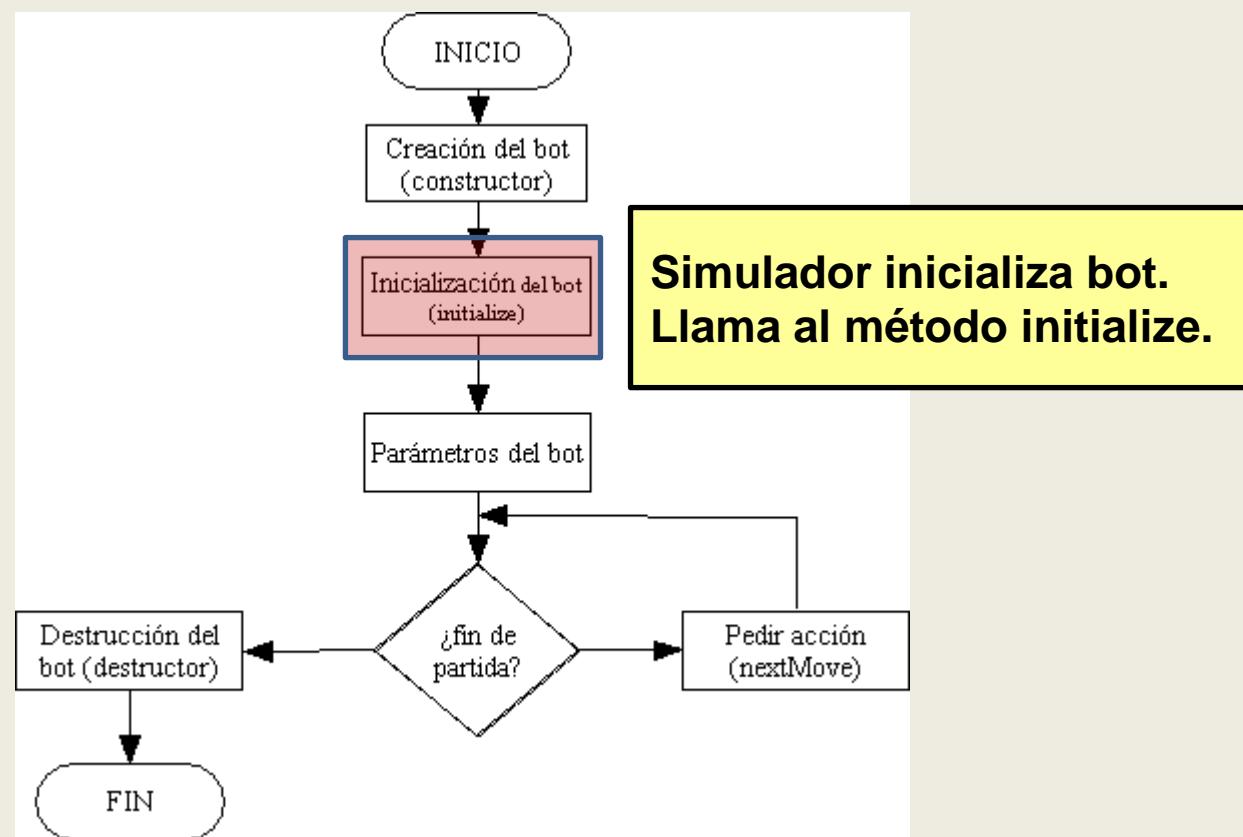
## Comunicaciones entre el simulador y el bot

- El ciclo de comunicaciones entre el simulador y el bot, durante el desarrollo de una partida, es el siguiente:



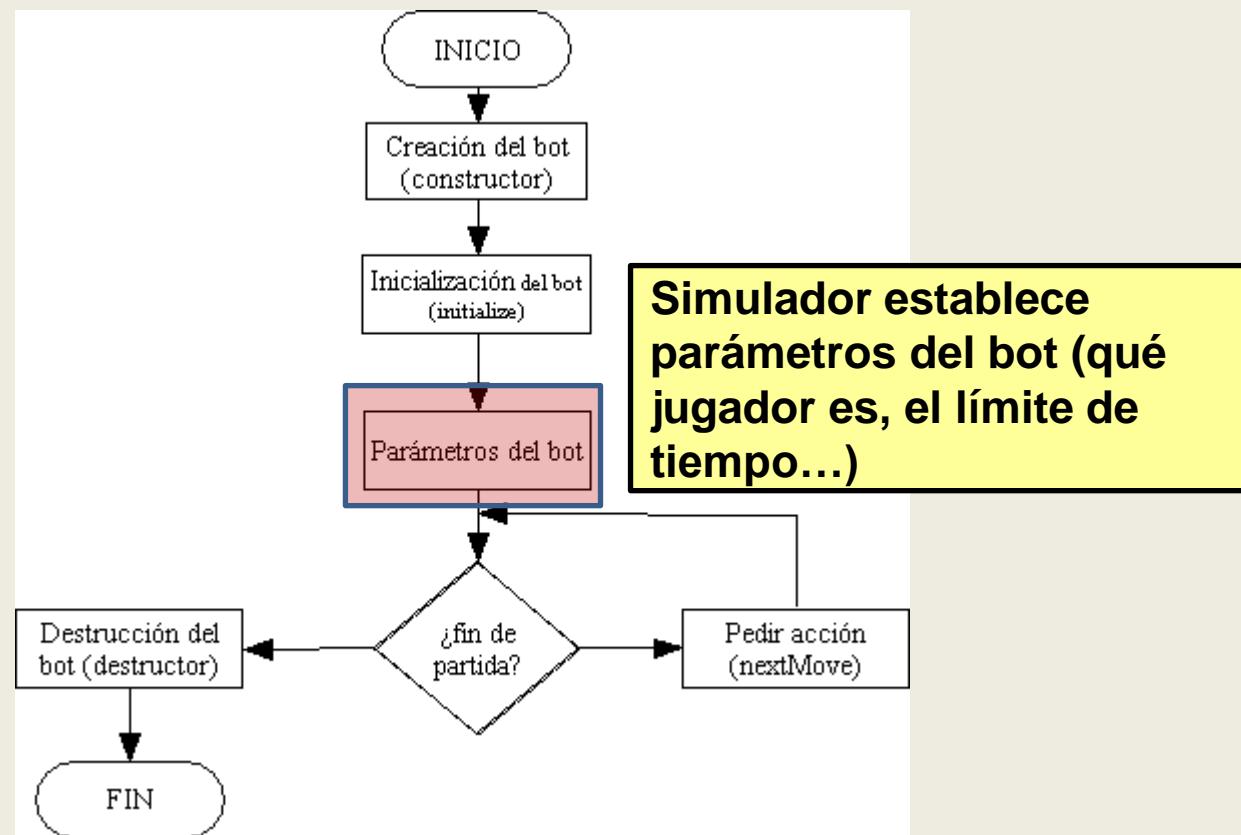
## Comunicaciones entre el simulador y el bot

- El ciclo de comunicaciones entre el simulador y el bot, durante el desarrollo de una partida, es el siguiente:



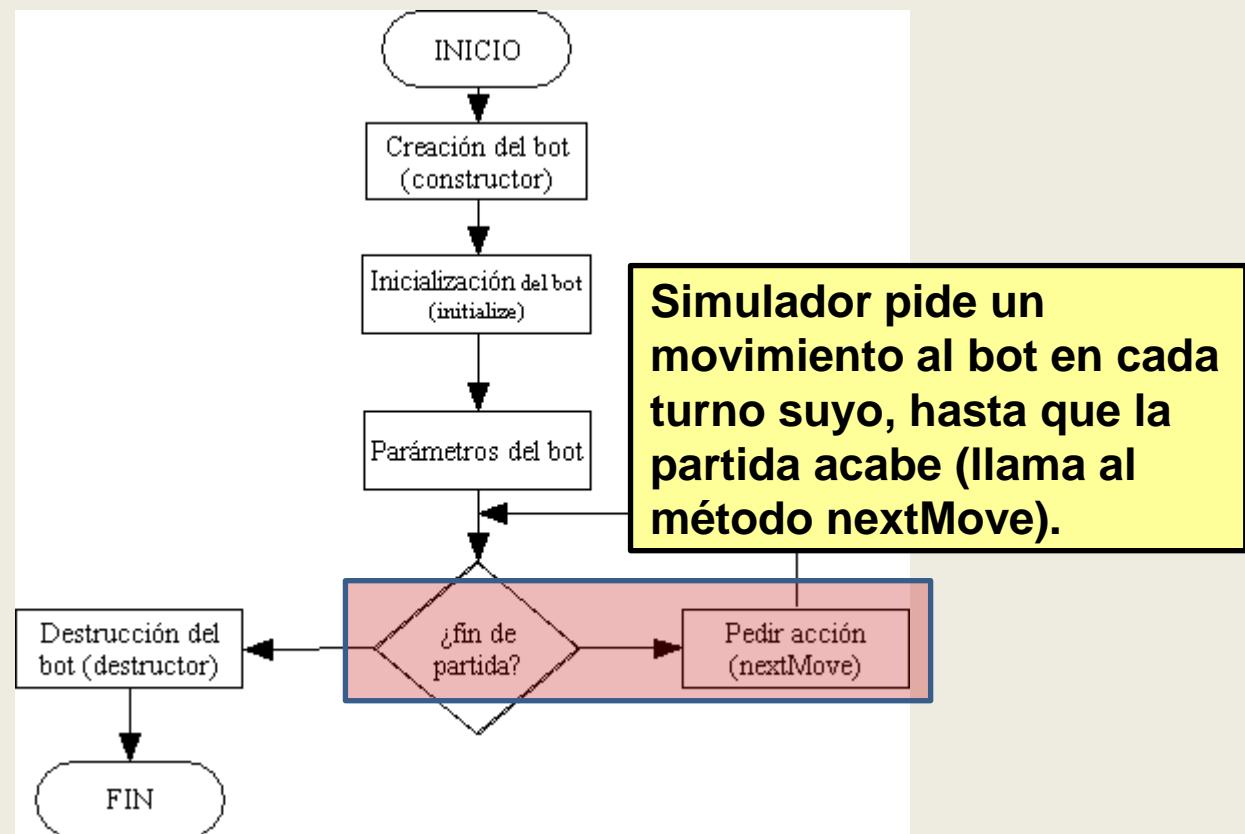
## Comunicaciones entre el simulador y el bot

- El ciclo de comunicaciones entre el simulador y el bot, durante el desarrollo de una partida, es el siguiente:



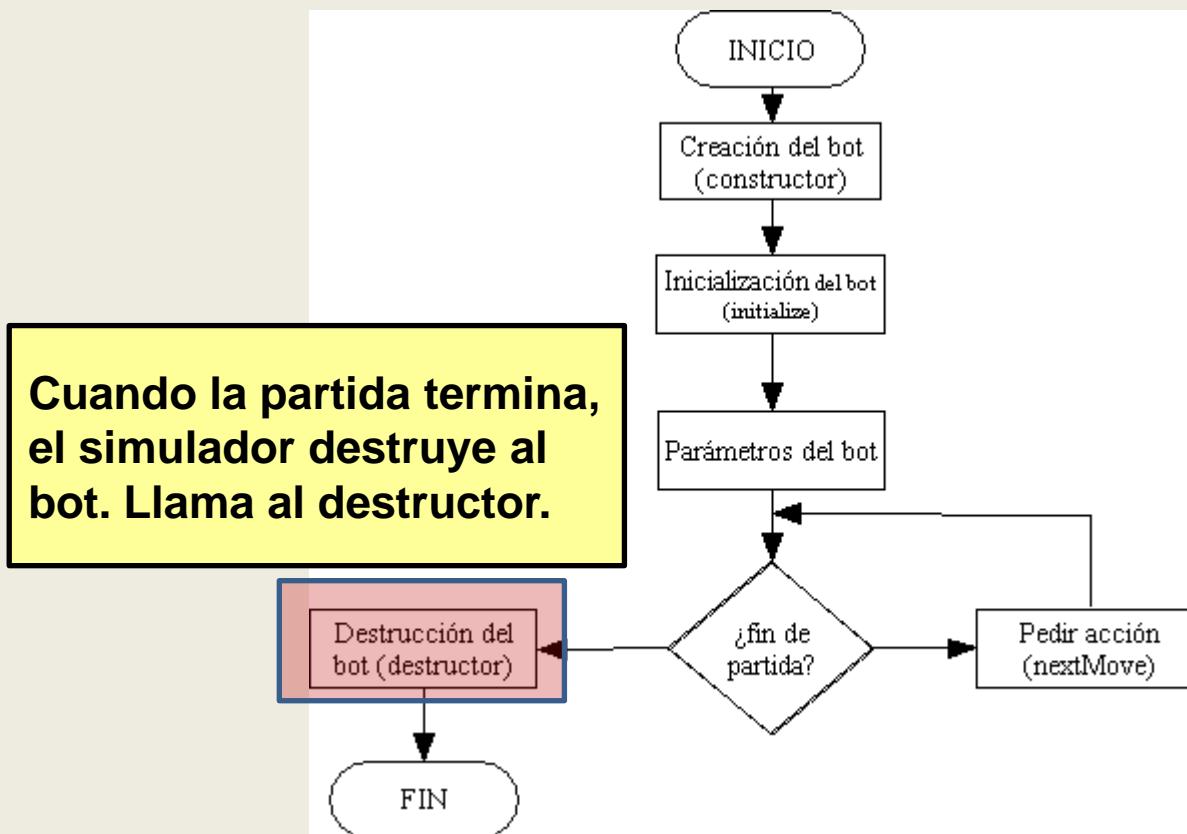
## Comunicaciones entre el simulador y el bot

- El ciclo de comunicaciones entre el simulador y el bot, durante el desarrollo de una partida, es el siguiente:



## Comunicaciones entre el simulador y el bot

- El ciclo de comunicaciones entre el simulador y el bot, durante el desarrollo de una partida, es el siguiente:



## ¿Qué funcionalidad tiene un bot?

- Un bot queda definido en el fichero **Bot.h** (que no hay que tocar en la práctica).

```
16 class Bot {  
17  
18     private:  
19         // Jugador asociado al bot (J1 o J2. Valor NONE para jugador no válido)  
20     Player player;  
21  
22     // Tiempo (en milisegundos) que el bot tendrá para responder por turno. Valor 0 para tiempo indefinido.  
23     long timeout;  
24  
25     public:  
26  
27     /**  
28         * Constructor. Crea un bot por defecto con nombre válido. Inicializa  
29         * los valores del objeto que el alumno necesite.  
30         */  
31     Bot();  
32  
33     virtual ~Bot() {}  
34  
35     /**  
36         * Establece el bot como jugador J1 o J2. Esta función se llama automáticamente  
37         * desde el simulador.  
38         */  
39     void setPlayer(const Player p);  
40  
41     /**  
42         * Indica si el bot actúa como jugador J1 o J2. Valor NONE si no está establecido  
43         */  
44     Player getPlayer();  
45  
46  
47     /**  
48         * Establece el límite de tiempo (en milisegundos) que el bot tendrá para responder por turno. Valor 0 para tiempo indefinido.  
49         */  
50     void setTimeOut(long t);  
51  
52  
53     /**  
54         * Devuelve el límite de tiempo (en milisegundos) que el bot tendrá por turno para responder. Valor 0 para tiempo indefinido.  
55         */  
56     long getTimeOut();
```

## ¿Qué funcionalidad tiene un bot?

- Se implementarán únicamente un subconjunto de funcionalidades que serán las encargadas de programar el comportamiento inteligente.

```
60  /**
61   * Función para inicializar el bot. Esta función se llama por el simulador
62   * al comienzo de cada partida, para que el alumno inicialice sus variables
63   * (en caso de que haya) antes de comenzar el juego.
64  */
65  virtual void initialize()= 0;
66
67
68 /**
69  * Devuelve el nombre del bot
70 */
71  virtual string getName()= 0;
72
73 /**
74  * Dada la serialización de un estado del juego como cadena de entrada,
75  * esta función devuelve el siguiente movimiento a realizar por el bot.
76 *
77  * El string es una secuencia de números enteros: "Turno G1 J11 J12 J13 J14 J15 J16 G2 J21 J22 J23 J24 J25 J26"
78  * turno: valor 1 para indicar que le toca mover al jugador 1; 2 para el jugador 2
79  * Gx: Piezas en el granero del jugador x
80  * Jxy: Piezas en la casilla y del jugador x (x=1,2; y=1,2,3,4,5,6)
81  * En caso de error, el motor inicializa el estado del juego a no válido
82  * Ejemplo de estado válido: "1 0 4 4 4 4 4 0 4 4 4 4 4 4 4", que es el estado inicial.
83 */
84  virtual Move nextMove(const vector<Move> &adversary, const GameState &state)= 0;
```

### ¿Qué tengo que implementar en el bot? (I)

- Sólo se pueden modificar los ficheros .h y .cpp del bot que el alumno ha creado. **No se permite modificar Bot.h/Bot.cpp.**
- **El fichero .h del bot del alumno se puede modificar para:**
  - Incorporar memoria (atributos/variables del bot dentro del class, etc.).
  - Incorporar nuevos tipos de datos (estructuras o clases que representen a un estado del espacio de búsqueda, etc.).
  - Incorporar la definición de nuevos métodos privados auxiliares que necesite el alumno para la práctica.
  - En resumen: Para añadir nuevas definiciones de variables y tipos que el alumno necesite.

## ¿Qué tengo que implementar en el bot? (II)

```
1 /*
2  * SuperBot.h
3  *
4  * Created on: 15 ene. 2018
5  * Author: manupc
6 */
7
8 #include "Bot.h"
9
10 #ifndef MANUPCBOT_H_
11 #define MANUPCBOT_H_
12
13 class SuperBot:Bot {
14 public:
15     SuperBot();
16     ~SuperBot();
17
18     void initialize();
19     string getName();
20     Move nextMove(const vector<Move> &adversary, const GameState &state);
21 };
22
23
24#endif /* MANUPCBOT_H_ */
```

Aquí: Nuevas estructuras, o clases, enums, etc.

Aquí: Nuevos atributos/variables y/o definiciones de métodos que el bot necesite para uso interno.

## ¿Qué tengo que implementar en el bot? (III)

- Sólo se pueden modificar los ficheros .h y .cpp del bot que el alumno ha creado. **No se permite modificar Bot.h/Bot.cpp.**
- **El fichero .cpp del bot del alumno se puede modificar para:**
  - Cambiar el nombre del bot que devuelve el método *getName*. Deberá devolver el nombre del bot, sabiendo que es sensible a mayúsculas y minúsculas.

```
28 string SuperBot::getName() {  
29     return "SuperBot"; // Sustituir por el nombre del bot  
30 }
```

- Implementar el constructor por defecto, que reserve memoria y/o recursos para las variables necesarias para el bot del alumno.

```
15 SuperBot::SuperBot() {  
16     // Inicializar las variables necesarias para ejecutar la partida  
17  
18 }
```

## ¿Qué tengo que implementar en el bot? (IV)

- Sólo se pueden modificar los ficheros .h y .cpp del bot que el alumno ha creado. **No se permite modificar Bot.h/Bot.cpp.**
- **El fichero .cpp del bot del alumno se puede modificar para:**
  - Implementar el método **Initialize**, que se ejecutará antes de comenzar la partida e inicializará las variables del bot (si es necesario).

```
24 void SuperBot::initialize() {  
25     // Inicializar el bot antes de jugar una partida  
26 }
```

- Implementar el destructor, que libere los recursos reservados durante la ejecución del bot.

```
20 SuperBot::~SuperBot() {  
21     // Liberar los recursos reservados (memoria, ficheros, etc.)  
22 }
```

## ¿Qué tengo que implementar en el bot? (V)

- Sólo se pueden modificar los ficheros .h y .cpp del bot que el alumno ha creado. **No se permite modificar Bot.h/Bot.cpp.**
- El fichero .cpp **del bot del alumno** se puede modificar para:
  - Implementar el método *nextMove*, para devolver la jugada que realizará el bot.

```
32 Move SuperBot::nextMove(const vector<Move> &adversary, const GameState &state) {  
33  
34     Move movimiento= M_NONE;  
35  
36     // Implementar aquí la selección de la acción a realizar  
37  
38     // OJO: Recordatorio. NO USAR cin NI cout.  
39     // Para salidas por consola (debug) utilizar cerr. Ejemplo:  
40     // cerr<< "Lo que quiero mostrar"<<endl;  
41  
42  
43     // OJO: Recordatorio. El nombre del bot y de la clase deben coincidir.  
44     // En caso contrario, el bot no podrá participar en la competición.  
45     // Se deberá sustituir el nombre SuperBot como nombre de la clase por otro  
46     // seleccionado por el alumno. Se deberá actualizar también el nombre  
47     // devuelto por el método getName() acordeamente.  
48  
49     return movimiento;  
50 }
```

## ¿Qué tengo que implementar en el bot? (VI)

- El bot, no obstante, sí que podrá hacer uso de funcionalidades de su superclase para:
  - **Conocer qué jugador es.** Utilizando el método **getPlayer**.

```
42  /**
43  * Indica si el bot actúa como jugador J1 o J2. Valor NONE si no está establecido
44  */
45 Player getPlayer();
```

- El tipo **Player** es un tipo enumerado del simulador, que el alumno puede utilizar para conocer qué jugador es el bot:

```
15 // Representación de jugadores (Jugador 1 J1, Jugador 2 J2; ninguno NONE)
16 enum Player {J1=0, J2=1, NONE=2} ;
```

- El valor J1/J2 indica que el bot es el jugador 1/jugador 2, respectivamente.
- El valor **NONE** indica que el valor del jugador aún no está asignado.
- Estas funcionalidades podrán utilizarse dentro del método **nextMove**.

## ¿Qué tengo que implementar en el bot? (VII)

- El bot, no obstante, sí que podrá hacer uso de funcionalidades de su superclase para:
  - **Conocer si hay tiempo límite por turno.** Utilizando el método **getTimeOut**.

```
/**  
 * Devuelve el límite de tiempo (en milisegundos) que el bot  
 * tendrá por turno para responder. Valor 0 para tiempo indefinido.  
 */  
long getTimeOut();
```

- El método **getTimeOut()** de **Bot** devuelve un valor positivo indicando el número de milisegundos que puede durar un turno, o valor 0 si no hay tiempo límite para que el bot escoja una acción.
- Estas funcionalidades podrán utilizarse dentro del método **nextMove**.

## El método *nextMove*

- Es la parte clave del bot.

```
Move SuperBot::nextMove(const vector<Move> &adversary, const GameState &state) {
```

- **Recibe como entrada** una lista ordenada de los últimos movimientos del adversario en su turno (o turnos, si ha tenido alguno extra), junto con el estado actual del juego.
  - **const vector<Move> &adversary** : Lista de movimientos que acaba de finalizar de realizar el adversario. Vacío si no hubo (por ser el primer turno) o si se trata de un turno extra para el bot.
  - **const GameState &state** : Estado actual del juego (a qué jugador le toca, cuántas semillas hay en cada casilla y granero...).
- **Proporciona como salida** un único movimiento, a realizar por el bot en el turno actual.

## La clase *GameState*

- Representa el estado actual del juego. **¡OJO! Es el estado del juego, no un nodo. Un nodo es un estado del espacio de estados.**
- Contiene información sobre:
  - A qué jugador le toca el turno.
  - Cuántas semillas hay en cada granero.
  - Cuántas semillas hay en cada casilla.
- Además, nos proporciona funcionalidad para conocer si el juego ha terminado, o simular un movimiento para poder explorar correctamente el espacio de estados.

## Métodos de la clase *GameState* que se pueden usar

```
// Devuelve el jugador al que le toca jugar el turno  
Player getCurrentPlayer() const;
```

```
// Devuelve las semillas existentes en una posición de un jugador.  
unsigned char getSeedsAt(Player p, Position pos) const;
```

```
// Simula un movimiento de entrada, proporcionando como salida  
// un estado resultante de que el jugador que tiene el turno  
// ejecute el movimiento dado como entrada  
GameState simulateMove(Move mov) const;
```

```
// Comprueba si el estado es final  
bool isFinalState() const;
```

```
// Devuelve true si el estado actual del juego es válido  
bool isValidState() const;
```

```
// Devuelve la puntuación del jugador pasado por argumento.  
// Valor 0 si el jugador no es válido.  
int getScore(Player p) const;
```

## Un ejemplo: RandomBot

- **Comportamiento:** Escoge una acción de siembra aleatoria entre las casillas que tienen semillas.

```
Move RandomBot::nextMove(const vector<Move> &adversary, const GameState &state) {  
  
    Player turno= this->getPlayer(); // Sin uso. Sólo código de ejemplo  
    long timeout= this->getTimeOut(); // Sin uso. Sólo código de ejemplo  
  
    Move movimiento= M_NONE;  
  
    // Contamos cuántas casillas tienen semillas  
    for (int i= 1; i<=6;i++) {  
        if (state.getSeedsAt(turno, (Position) i) >0)  
            available++;  
    }  
  
    // Generamos casilla al azar  
    int n= 1+ (rand()%available);  
    int aux= 0;  
  
    // Seleccionamos el movimiento de la casilla escogida  
    for (int i= 1; i<=6 && movimiento == M_NONE; i++) {  
        if (state.getSeedsAt(turno, (Position) i) >0)  
            aux++;  
        if (aux == n)  
            movimiento= (Move)i;  
    }  
  
    return movimiento;  
}
```

## Un ejemplo: RandomBot

- **Comportamiento:** Escoge una acción de siembra aleatoria entre las casillas que tienen semillas.

```
Move RandomBot::nextMove(const vector<Move> &adversary, const GameState &state) {  
  
    Player turno= this->getPlayer(); // Sin uso. Sólo código de ejemplo  
    long timeout= this->getTimeOut(); // Sin uso. Sólo código de ejemplo  
  
    Move movimiento;  
  
    // Contamos cuántas casillas tienen semillas  
    for (int i= 1; i<=6;i++) {  
        if (state.getSeedsAt(turno, (Position) i) >0)  
            available++;  
    }  
  
    // Generamos casilla al azar  
    int n= 1+ (rand()%available);  
    int aux= 0;  
  
    // Seleccionamos el movimiento de la casilla escogida  
    for (int i= 1; i<=6 && movimiento == M_NONE; i++) {  
        if (state.getSeedsAt(turno, (Position) i) >0)  
            aux++;  
        if (aux == n)  
            movimiento= (Move)i;  
    }  
  
    return movimiento;  
}
```

Miramos cuántas semillas hay en la casilla i del jugador



## Un ejemplo: RandomBot

- **Comportamiento:** Escoge una acción de siembra aleatoria entre las casillas que tienen semillas.

```
Move RandomBot::nextMove(const vector<Move> &adversary, const GameState &state) {  
  
    Player turno= this->getPlayer(); // Sin uso. Sólo código de ejemplo  
    long timeout= this->getTimeOut(); // Sin uso. Sólo código de ejemplo  
  
    Move movimiento= M_NONE;  
  
    // Contamos cuántas casillas tienen semillas  
    for (int i= 1; i<=6; i++) {  
        if  
            Generamos casilla al azar, entre 1 y available  
    }  
  
    // Generamos casilla al azar  
    int n= 1+ (rand()%available);  
    int aux= 0;  
  
    // Seleccionamos el movimiento de la casilla escogida  
    for (int i= 1; i<=6 && movimiento == M_NONE; i++) {  
        if (state.getSeedsAt(turno, (Position) i) >0)  
            aux++;  
        if (aux == n)  
            movimiento= (Move)i;  
    }  
  
    return movimiento;  
}
```

## Un ejemplo: RandomBot

- **Comportamiento:** Escoge una acción de siembra aleatoria entre las casillas que tienen semillas.

```
Move RandomBot::nextMove(const vector<Move> &adversary, const GameState &state) {  
  
    Player turno= this->getPlayer(); // Sin uso. Sólo código de ejemplo  
    long timeout= this->getTimeOut(); // Sin uso. Sólo código de ejemplo  
  
    Move movimiento= M_NONE;  
  
    // Contamos cuántas casillas tienen semillas  
    for (int i= 1; i<=6;i++) {  
        if (state.getSeedsAt(turno, (Position) i) >0)  
            available++;  
    }  
  
    // Generamos casilla al azar  
    int n= 1+ (rand()%available);  
    int aux= 0;  
  
    // Seleccionamos el movimiento de la casilla escogida  
    for (int i= 1; i<=6 && movimiento == M_NONE; i++) {  
        if (state.getSeedsAt(turno, (Position) i) >0)  
            aux++;  
        if (aux == n)  
            movimiento= (Move)i;  
    }  
  
    return movimiento;  
}
```

Seleccionamos n-ésima casilla con semillas



## Otro ejemplo: GreedyBot

- **Comportamiento: Escalada por máxima pendiente.** Comprueba las acciones posibles, simula un movimiento con cada acción y finalmente escoge aquella acción que le dé una mayor puntuación.

```
Move GreedyBot::nextMove(const vector<Move> &adversary, const GameState &state) {  
  
    Move movimiento= M_NONE;  
    int puntos= -10000;  
  
    // Recorremos los 6 movimientos posibles  
    for (int i= 1; i<=6; i++) {  
  
        // Comprobamos si hay semillas en la casilla i  
        if (state.getSeedsAt(this->getPlayer(), (Position) i) >0) {  
  
            // Si las hay, intentamos ver cómo queda el tablero si  
            // hacemos el movimiento de la casilla i  
            GameState hijo= state.simulateMove( (Move) i);  
  
            // Si nos da más puntos de los que tenemos  
            // con respecto al contrario, lo hacemos  
            if (hijo.getScore(this->getPlayer()) > puntos) {  
                puntos= hijo.getScore(this->getPlayer());  
                movimiento= (Move) i;  
  
                // O si nos da los mismos puntos pero tiramos de nuevo,  
                // cogemos el movimiento i  
            } else if (hijo.getScore(this->getPlayer()) == puntos) {  
                if (hijo.getCurrentPlayer() == this->getPlayer())  
                    movimiento= (Move)i;  
            }  
        }  
    }  
    return movimiento;  
}
```

## Otro ejemplo: GreedyBot

- **Comportamiento: Escalada por máxima pendiente.** Comprueba las acciones posibles, simula un movimiento con cada acción y finalmente escoge aquella acción que le dé una mayor puntuación

```
Move GreedyBot::nextMove(const GameState &state) {  
  
    Move movimiento= M_NONE;  
    int puntos= -10000;  
  
    // Recorremos los 6 movimientos posibles  
    for (int i= 1; i<=6; i++) {  
  
        // Comprobamos si hay semillas en la casilla i  
        if (state.getSeedsAt(this->getPlayer(), (Position) i) >0) {  
  
            // Si las hay, intentamos ver cómo queda el tablero si  
            // hacemos el movimiento de la casilla i  
            GameState hijo= state.simulateMove( (Move) i);  
  
            // Si nos da más puntos de los que tenemos  
            // con respecto al contrario, lo hacemos  
            if (hijo.getScore(this->getPlayer()) > puntos) {  
                puntos= hijo.getScore(this->getPlayer());  
                movimiento= (Move) i;  
  
                // O si nos da los mismos puntos pero tiramos de nuevo,  
                // cogemos el movimiento i  
            } else if (hijo.getScore(this->getPlayer()) == puntos) {  
                if (hijo.getCurrentPlayer() == this->getPlayer())  
                    movimiento= (Move)i;  
            }  
        }  
    }  
    return movimiento;  
}
```

**Recorremos todos los movimientos posibles**



## Otro ejemplo: GreedyBot

- **Comportamiento: Escalada por máxima pendiente.** Comprueba las acciones posibles, simula un movimiento con cada acción y finalmente escoge aquella acción que le dé una mayor puntuación.

```
Move GreedyBot::nextMove(const vector<Move> &adversary, const GameState &state) {  
    Move movimiento= M_NONE;  
    int puntos= -10000;  
  
    // Recorremos los 6 movimientos posibles  
    for (int i= 1; i<=6; i++) {  
  
        // Comprobamos si hay semillas en la casilla i  
        if (state.getSeedsAt(this->getPlayer(), (Position) i) >0) {  
  
            // Si las hay, intentamos ver cómo queda el tablero si  
            // hacemos el movimiento de la casilla i  
            GameState hijo= state.simulateMove( (Move) i);  
  
            // Si nos da más puntos de los que tenemos  
            // con respecto al contrario, lo hacemos  
            if (hijo.getScore(this->getPlayer()) > puntos) {  
                puntos= hijo.getScore(this->getPlayer());  
                movimiento= (Move) i;  
  
                // O si nos da los mismos puntos pero tiramos de nuevo,  
                // cogemos el movimiento i  
            } else if (hijo.getScore(this->getPlayer()) == puntos) {  
                if (hijo.getCurrentPlayer() == this->getPlayer())  
                    movimiento= (Move)i;  
            }  
        }  
    }  
    return movimiento;  
}
```

Miramos si el jugador tiene semillas en la casilla *i*



## Otro ejemplo: GreedyBot

- **Comportamiento: Escalada por máxima pendiente.** Comprueba las acciones posibles, simula un movimiento con cada acción y finalmente escoge aquella acción que le dé una mayor puntuación.

```
Move GreedyBot::nextMove(const vector<Move> &adversary, const GameState &state) {
    Move movimiento= M_NONE;
    int puntos= -10000;

    // Recorremos los 6 movimientos
    for (int i= 1; i<=6; i++) {
        // Comprobamos si hay semillas en la casilla i
        if (state.getSeedsAt(this->getPlayer(), (Position) i) >0) {

            // Si las hay, intentamos ver cómo queda el tablero si
            // hacemos el movimiento de la casilla i
            GameState hijo= state.simulateMove( (Move) i);

            // Si nos da más puntos de los que tenemos
            // con respecto al contrario, lo hacemos
            if (hijo.getScore(this->getPlayer()) > puntos) {
                puntos= hijo.getScore(this->getPlayer());
                movimiento= (Move) i;

                // O si nos da los mismos puntos pero tiramos de nuevo,
                // cogemos el movimiento i
            } else if (hijo.getScore(this->getPlayer()) == puntos) {
                if (hijo.getCurrentPlayer() == this->getPlayer())
                    movimiento= (Move)i;
            }
        }
    }
    return movimiento;
}
```

**Comprobamos cómo queda el estado del juego tras sembrar la casilla *i***



## Otro ejemplo: GreedyBot

- **Comportamiento: Escalada por máxima pendiente.** Comprueba las acciones posibles, simula un movimiento con cada acción y finalmente escoge aquella acción que le dé una mayor puntuación.

```
Move GreedyBot::nextMove(const vector<Move> &adversary, const GameState &state) {  
  
    Move movimiento= M_NONE;  
    int puntos= -10000;  
  
    // Recorremos los 6 movimientos  
    for (int i= 1; i<=6; i++) {  
  
        // Comprobamos si hay semillas  
        if (state.getSeedsAt(this->getMove(i))>0){  
  
            // Si las hay, intentamos el movimiento  
            GameState hijo= state.simulateMove(i);  
  
            // Si nos da más puntos de los que tenemos  
            // con respecto al contrario, lo hacemos  
            if (hijo.getScore(this->getPlayer()) > puntos) {  
                puntos= hijo.getScore(this->getPlayer());  
                movimiento= (Move)i;  
  
                // O si nos da los mismos puntos pero tiramos de nuevo,  
                // cogemos el movimiento i  
            } else if (hijo.getScore(this->getPlayer()) == puntos) {  
                if (hijo.getCurrentPlayer() == this->getPlayer())  
                    movimiento= (Move)i;  
            }  
        }  
    }  
    return movimiento;  
}
```

**Si el estado nos da más puntos de los que teníamos, seleccionamos sembrar *i***



## Otro ejemplo: GreedyBot

- **Comportamiento: Escalada por máxima pendiente.** Comprueba las acciones posibles, simula un movimiento con cada acción y finalmente escoge aquella acción que le dé una mayor puntuación.

```
Move GreedyBot::nextMove(const vector<Move> &adversary, const GameState &state) {  
  
    Move movimiento= M_NONE;  
    int puntos= -10000;  
  
    // Recorremos los 6 movimientos posibles  
    for (int i= 1; i<=6; i++) {  
  
        // Comprobamos si hay semillas en la casilla i  
        if (state.getSeedsAt(this->getPlayer(), (Position) i) >0) {  
  
            // Si las hay, intentamos movernos a la siguiente  
            // hacemos el movimiento  
            GameState hijo= state.siguienteMovimiento(i);  
  
            // Si nos da más puntos, lo cogemos  
            // con respecto al contrario  
            if (hijo.getScore(this->getPlayer()) > puntos) {  
                puntos= hijo.getScore(this->getPlayer());  
                movimiento= (Move) i;  
  
                // O si nos da los mismos puntos pero tiramos de nuevo,  
                // cogemos el movimiento i  
            } else if (hijo.getScore(this->getPlayer()) == puntos) {  
                if (hijo.getCurrentPlayer() == this->getPlayer())  
                    movimiento= (Move)i;  
            }  
        }  
    }  
    return movimiento;  
}
```

**Si el estado nos da iguales puntos, pero nos da turno extra, seleccionamos sembrar i**



## Pistas para crear el bot de la práctica

- Se recomienda:
  - Hacer un diseño de cómo representar un **estado del espacio de búsqueda (nodo del árbol)**.
  - Implementar la estructura de un estado del espacio de búsqueda en el fichero .h del bot del alumno (como struct, class, o como se desee).
  - Diseñar cómo hacer la búsqueda, sabiendo que habrá un límite de tiempo de 2 segundos/turno (140.000 llamadas a **GameState::SimulateMove**, aprox.).
  - Que el bot actúe como J1 o como J2 da igual: Lo importante es generar el plan correctamente con el estado con el que el simulador llama a **nextMove** del bot.
  - Diseñar (**antes de implementar**) cómo se explorará el espacio.
  - Establecer criterio de parada del algoritmo de búsqueda, para dejar de explorar por exceso de límite de tiempo.

### Pistas para crear el bot de la práctica

- Se recomienda:
  - Al parar la búsqueda por límite de tiempo, hay que asignar a los nodos del último nivel una heurística que permita discriminar cuál de ellos es más deseable para el agente.
  - Pensar varias posibilidades de heurísticas: Maximización vs minimización → Algoritmos minimax/maximin.
  - Ejemplos:
    - ¿Maximizar mis puntos?
    - ¿Minimizar los puntos del contrario?
    - ¿Considerar la disposición de las semillas en las casillas?
    - ¿Dar un peso/importancia al número semillas en proporciona a la casilla donde se encuentren?
    - ... y un largo etc., que podéis ingeniar.

### Pistas para crear el bot de la práctica

- Otras cuestiones:
  - ¿Haré una búsqueda desde el estado de entrada a **nextMove** cada vez que el simulador llame a la función, o guardaré el árbol generado para luego seguir expandiéndolo?
  - ¿Es necesario guardar el plan de acciones? Si es así, ¿cómo lo guardo?
- **Las decisiones de diseño que se piensen y se escojan influirán directamente en su implementación y, por tanto, en la velocidad del bot y su eficacia.**
- Se recomienda buscar un consenso entre una heurística más/menos costosa, una buena estrategia de búsqueda y un código exquisitamente eficiente.



- » 1. El juego del Mancala
- 2. Objetivos de la práctica
- 3. El simulador
- 4. Creación de un bot
- 5. La liga oficial
- 6. ¿Una liga extraoficial?
- 7. Evaluación

## liga oficial como evaluación

- Se realizará una liga entre los bots de un grupo de prácticas, como medio relevante dentro de la evaluación.
- El acceso a la liga estará disponible desde el primer día de la práctica, aunque sólo se tendrá en cuenta el resultado del último día de clase (ver apartado de “**Evaluación**” de estas diapositivas).
- La liga se realiza mediante una liga entre todos los bots de un grupo de prácticas, donde se juega por parejas 2 partidas (una con un bot como jugador 1 y el otro como jugador 2, y viceversa).
- La liga se actualizará cada 3 días de forma automática, de modo que los alumnos puedan observar la evolución de su bot con respecto al resto.
- Hay una restricción de tiempo de 2 segundos para cada turno de cada jugador, en cada partida.

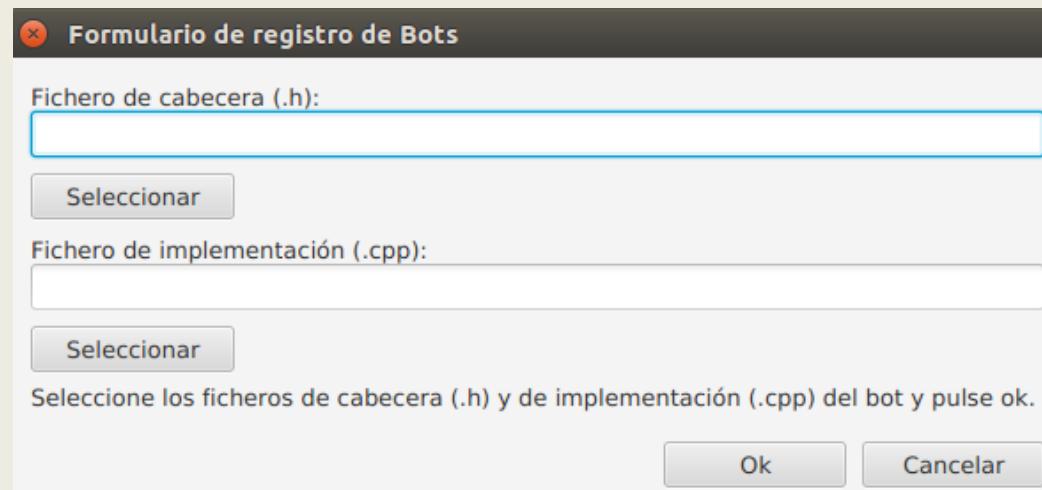
## Acceso a la liga

- El acceso a la liga se realiza desde el entorno gráfico del simulador, menú “liga”, botón “Participar”.
- Se deberán subir **sólo los ficheros .h y .cpp del bot del alumno**, utilizando la opción “Subir bot” (hay que estar registrado y entrar antes con nuestro e-Mail y nuestra clave de Mancala):



## Acceso a la liga

- Al pulsar sobre la opción de “**Subir bot**”, se nos abrirá una nueva ventana para que seleccionemos los ficheros .h y .cpp de nuestro bot.
- Seguidamente, estos ficheros se almacenarán en el servidor y tratarán de compilarse en el mismo.
- Se nos mostrará un mensaje de error si no se pudo subir algún fichero, si el bot no se pudo compilar según las instrucciones que se han expuesto para compilación de bots, o un mensaje de información indicando que la subida se realizó con éxito.



## Acceso a la liga

- Un bot subido por un alumno entrará a formar parte de la liga en la actualización siguiente que se realice de la misma.
- La actualización de la liga se realiza cada 3 días: Se eliminan todos los resultados anteriores, y se vuelve a ejecutar la liga completa con todos los bots que haya actualmente en el sistema.
- Los resultados finales se actualizarán en 24h aproximadamente (o cuando el servidor termine de ejecutar la liga, por lo que... ¡Ya sabéis! A hacer bots eficientes que no hagan gastar la energía del servidor y que tarden poco en dar los resultados ☺ ).

## Resultados de la liga

- Para visualizar los resultados de la liga, deberemos estar *logueados* en el sistema, pulsando sobre el botón “**Participar**” del menú “**Liga**” del simulador, opción “**Entrar**”.
- Una vez dentro, pulsaremos sobre el botón “**Estadísticas**”.



- Aparecerá la ventana con los resultados de todas las competiciones de los diferentes grupos de prácticas.



## Resultados de la liga

- #### • La ventana de estadísticas:

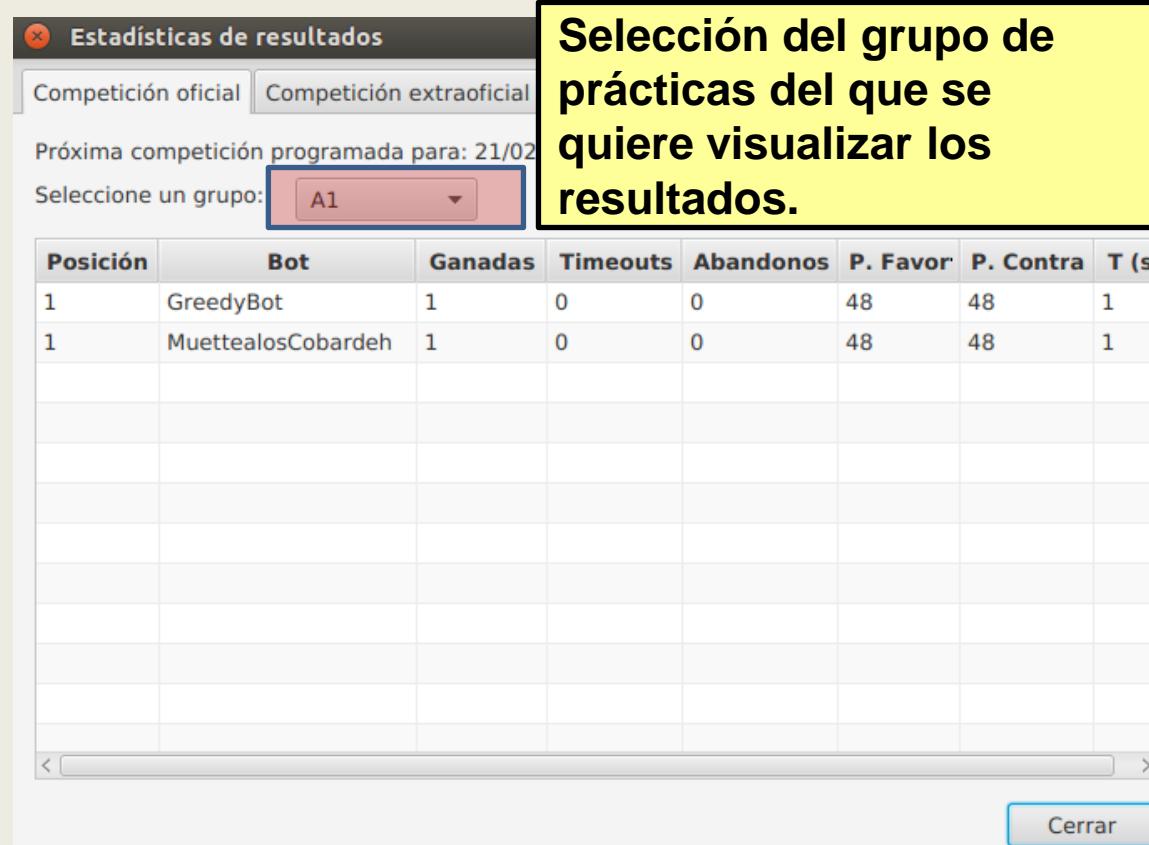
## Resultados de la liga

- #### • La ventana de estadísticas:

**Fecha programada para la  
próxima actualización de  
la liga.**

## Resultados de la liga

- #### • La ventana de estadísticas:



## Resultados de la liga

- La ventana de estadísticas:

**Estadísticas de resultados**

Competición oficial    Competición extraoficial

Próxima competición programada para: 21/02/2018 01:00:00

Seleccione un grupo: A1

Posición	Bot	Ganadas	Timeouts	Abandonos	P. Favor	P. Contra	T (s)
1	GreedyBot	1	0	0	48	48	1
1	MuettealosCobardeh	1	0	0	48	48	1

**Posición de cada bot en la liga (se puede compartir posición si los bots tienen la misma puntuación).**

Cerrar

## Resultados de la liga

- La ventana de estadísticas:

Estadísticas de resultados

Competición oficial    Competición extraoficial

Próxima competición programada para: 21/02/2018 01:00:00

Seleccione un grupo: A1

Posición	Bot	Ganadas	Timeouts	Abandonos	P. Favor	P. Contra	T (s)
1	GreedyBot	1	0	0	48	48	1
1	MuettealosCobardeh	1	0	0	48	48	1

Nombre de cada bot

Cerrar

## Resultados de la liga

- #### • La ventana de estadísticas:

**Estadísticas de resultados**

Competición oficial | Competición extraoficial

Próxima competición programada para: 21/02/2018 01:00:00

Seleccione un grupo: A1

Posición	Bot	Ganadas	Timeouts	Abandonos	P. Favor	P. Contra	T (s)
1	GreedyBot	1	0	0	48	48	1
1	MuettealosCobardeh	1	0	0	48	48	1

**Partidas ganadas por cada bot**

## Resultados de la liga

- #### • La ventana de estadísticas:

**Estadísticas de resultados**

Competición oficial | Competición extraoficial

Próxima competición programada para: 21/02/2018 01:00:00

Seleccione un grupo: A1

Posición	Bot	Ganadas	Timeouts	Abandones	P. Favor	P. Contra	T (s)
1	GreedyBot	1	0	0	48	48	1
1	MuettealosCobardeh	1	0	0	48	48	1

**Partidas perdidas por cada bot, por pasarse del límite de tiempo.**

< >

Cerrar

## Resultados de la liga

- #### • La ventana de estadísticas:

Posición	Bot	Ganadas	Timeouts	Abandones	P. Favor	P. Contra	T (s)
1				0	48	48	1
1				0	48	48	1

Partidas perdidas por cada bot, por finalizar una partida de forma inesperada (segmentation fault, ejem, ejem) o selección de acción incorrecta.



## Resultados de la liga

- #### • La ventana de estadísticas:

**Número de semillas total que ha ganado en todas las partidas**

Posición	Bot	Ganadas	Timeouts	Abandonos	P. Favor	P. Contra	T (s)
1	GreedyBot	1	0	0	48	48	1
1	MuettealosCobardeh	1	0	0	48	48	1



## Resultados de la liga

- #### • La ventana de estadísticas:

**Número de semillas total que ha perdido en todas las partidas**

Posición	Bot	Ganadas	Timeouts	Abandonos	P. Favor	P. Contra	T (s)
1	GreedyBot	1	0	0	48	48	
1	MuettealosCobardeh	1	0	0	48	48	



## Resultados de la liga

- #### • La ventana de estadísticas:

Estadísticas de resultados

Competición oficial | Competición extraoficial

Próxima competición programada para: 21/02/2018 01:00:00

Seleccione un grupo: A1

Posición	Bot	Ganadas	Timeouts	Abandonos	P. Favor	P. Contra
1	GreedyBot	1	0	0	48	48
1	MuettealosCobardeh	1	0	0	48	48

Tiempo total (en segundos) empleado en jugar todas las partidas

Cerrar



## Resultados de la liga

- Los criterios usados en la ventana de los resultados de la clasificación se calculan de la siguiente forma:
  1. Máximas partidas ganadas.
  2. En caso de empate: Mínimas partidas perdidas por timeout.
  3. En caso de empate: Mínimo tiempo empleado.
  4. En caso de empate: Máxima puntuación global (P. favor – P. Contra).
  5. En caso de empate: Máximos puntos a favor.
  6. En caso de empate: ¿has copiado la práctica?



1. El juego del Mancala
2. Objetivos de la práctica
3. El simulador
4. Creación de un bot
5. La liga oficial
6. ¿Una liga extraoficial?
7. Evaluación



### liga extraoficial

- **La liga extraoficial no entra dentro de la evaluación.**
- Sólo sirve para dar la posibilidad de que los estudiantes compitan todos contra todos y valoren cómo de buenos son sus bots, e incluso que se organicen para establecer competiciones paralelas. A esto se le ha denominado “**liga extraoficial de Mancala**”.
- La dinámica de la liga extraoficial es simple:
  - El simulador facilita un chat.
  - Los estudiantes entran al chat y buscan oponentes para su bot.
  - Se crea una partida online entre dos estudiantes.
  - Los resultados de la partida se almacenan en las estadísticas de los dos jugadores.
  - Todos los alumnos pueden acceder a una tabla de clasificación.

## Participar en la liga extraoficial

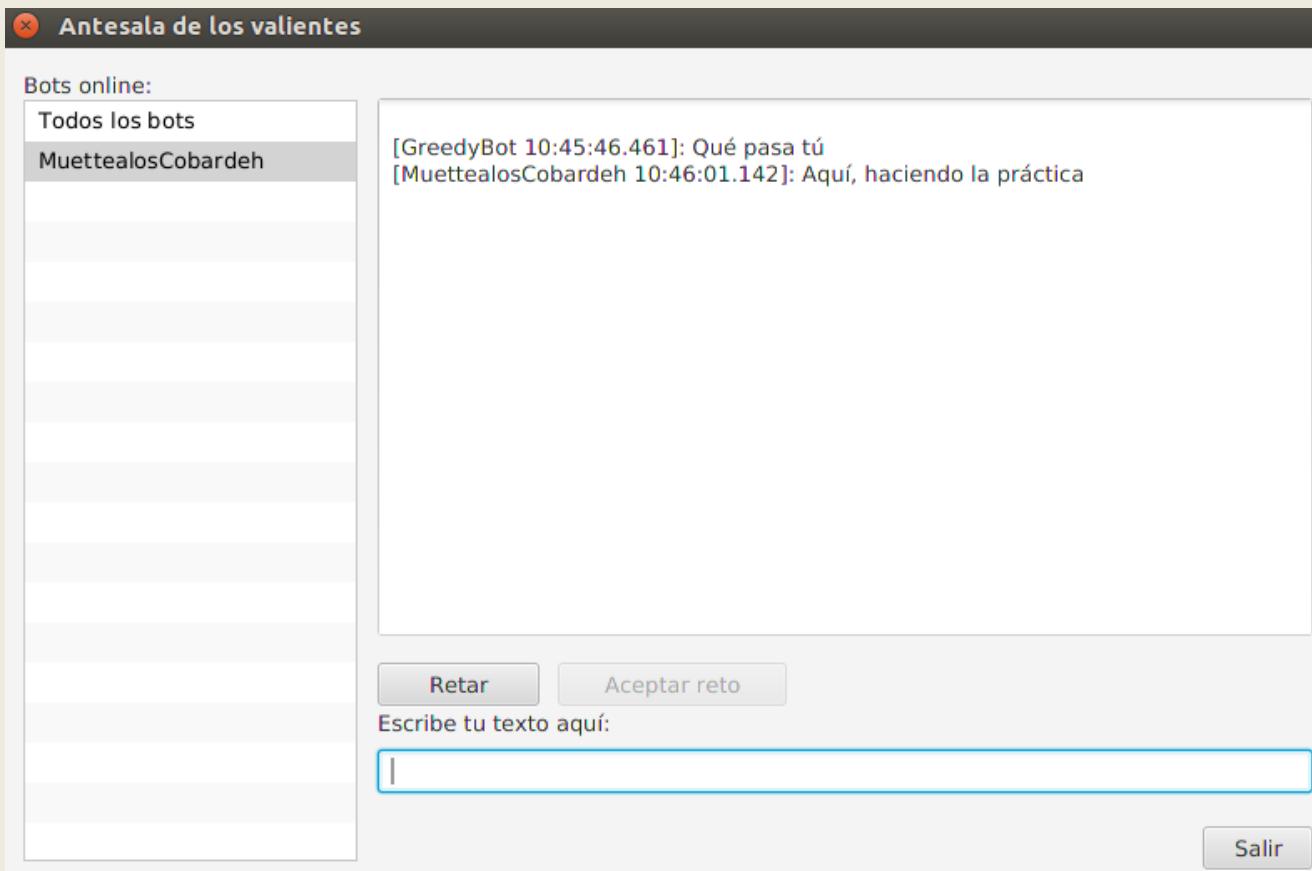
- Para participar en la liga extraoficial, hay que pulsar sobre el botón “Oponentes” del menú “liga” del simulador.



- Acto seguido, se nos pedirá que seleccionemos **el fichero ejecutable de nuestro bot**. Por tanto, tendremos que compilarlo antes de entrar a la liga extraoficial.

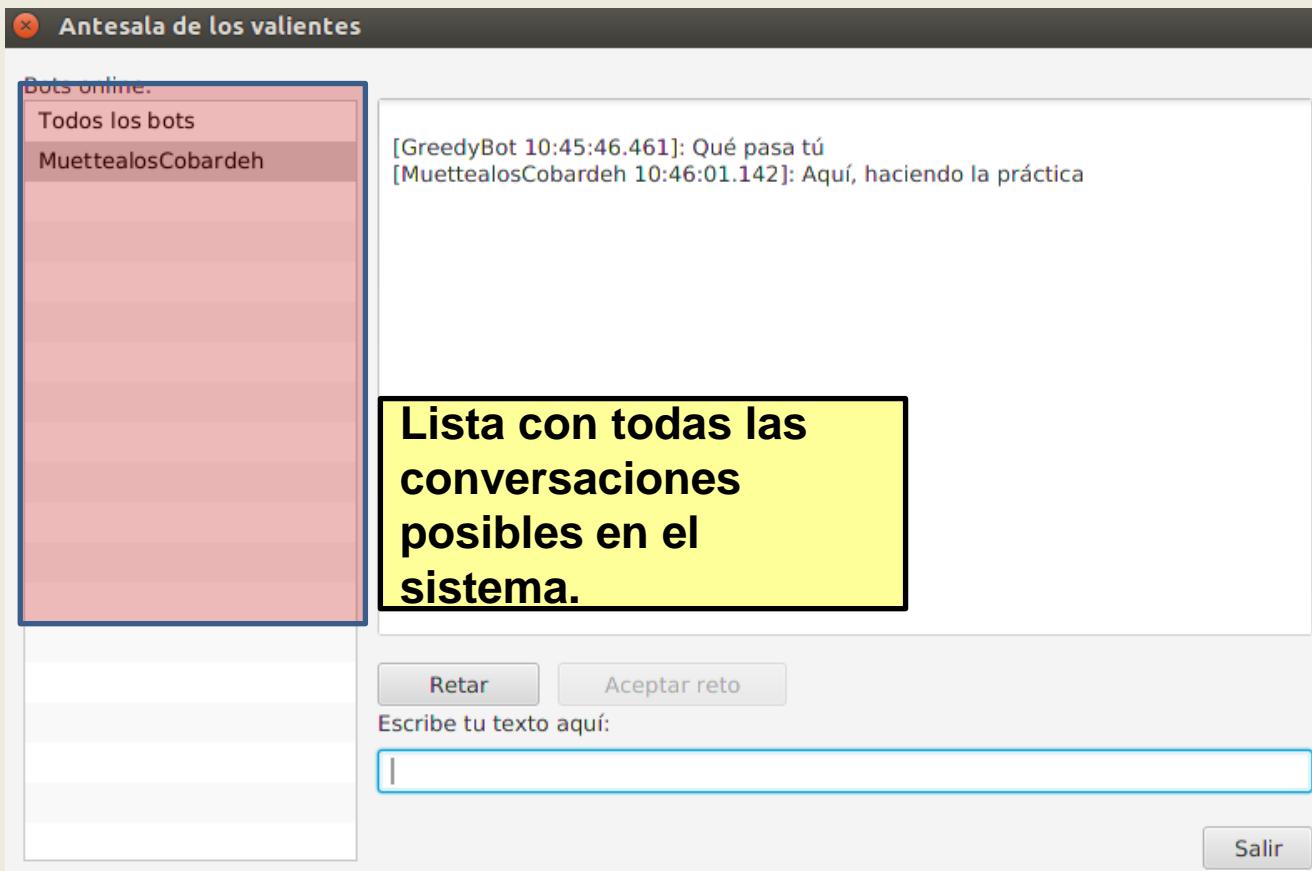
## Participar en la liga extraoficial

- Tras seleccionar el fichero ejecutable, entraremos a la “**Antesala de los valientes**”.



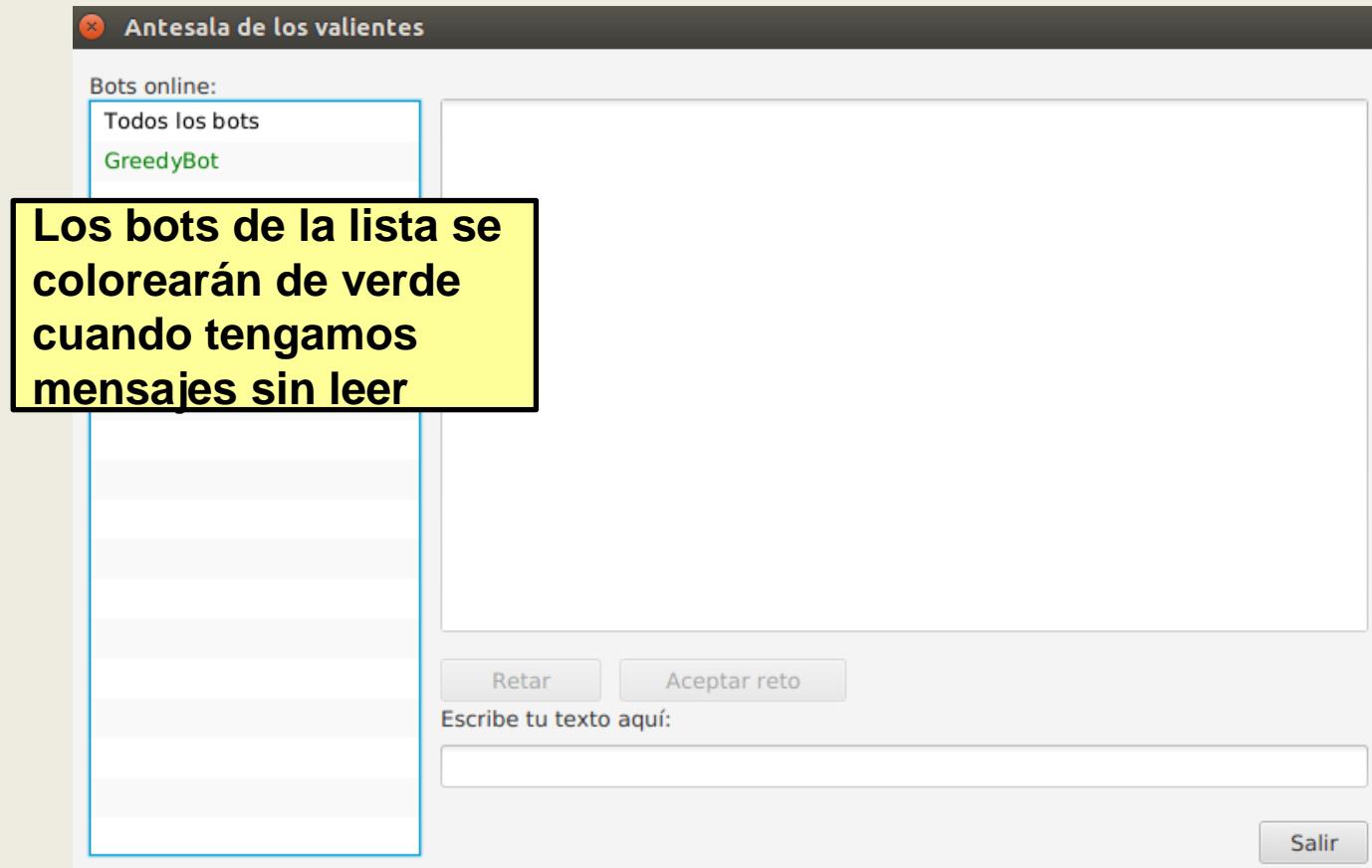
## Participar en la liga extraoficial

- Tras seleccionar el fichero ejecutable, entraremos a la “**Antesala de los valientes**”.



## Participar en la liga extraoficial

- Tras seleccionar el fichero ejecutable, entraremos a la “Antesala de los valientes”.



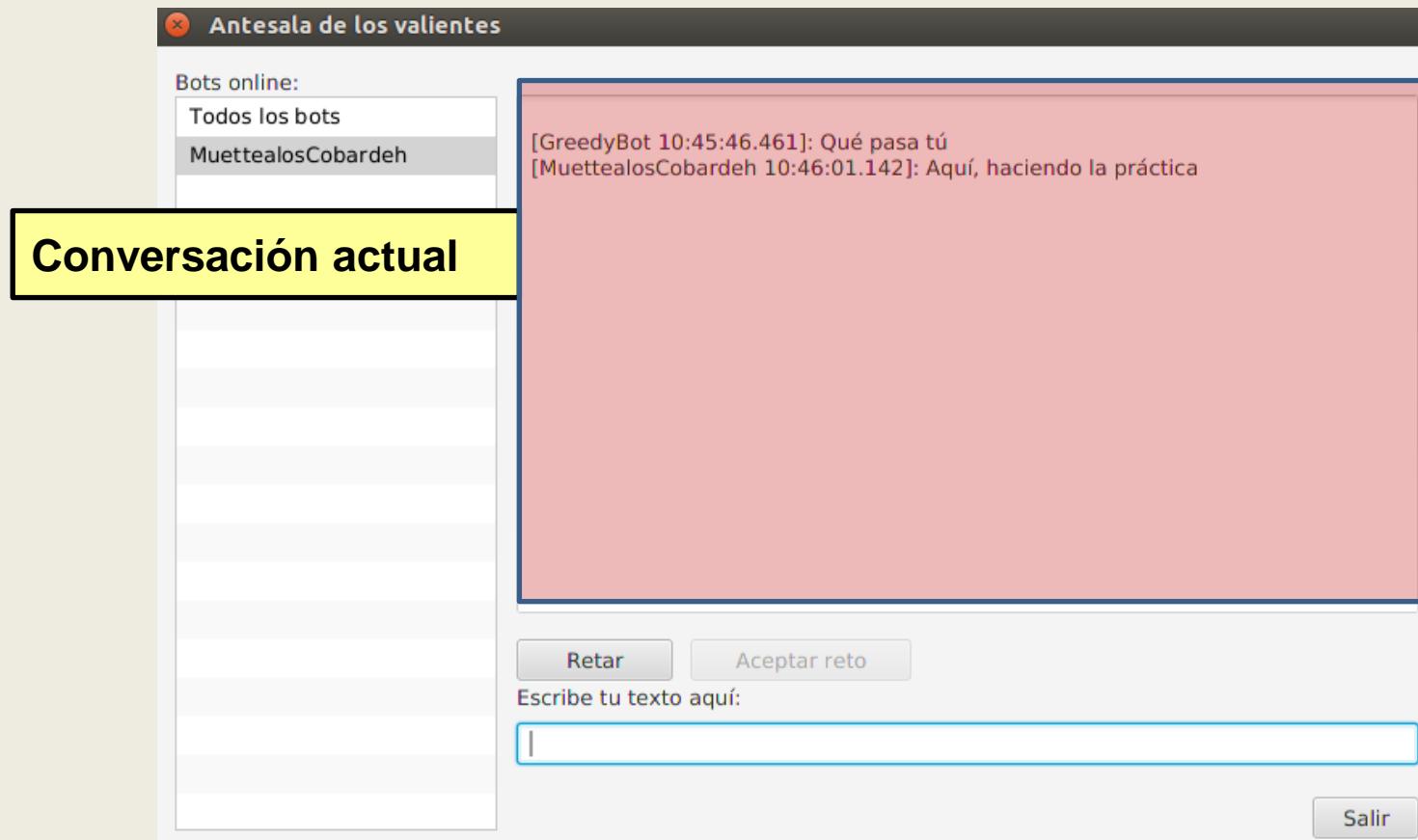
## Participar en la liga extraoficial

- Tras seleccionar el fichero ejecutable, entraremos a la “Antesala de los valientes”.



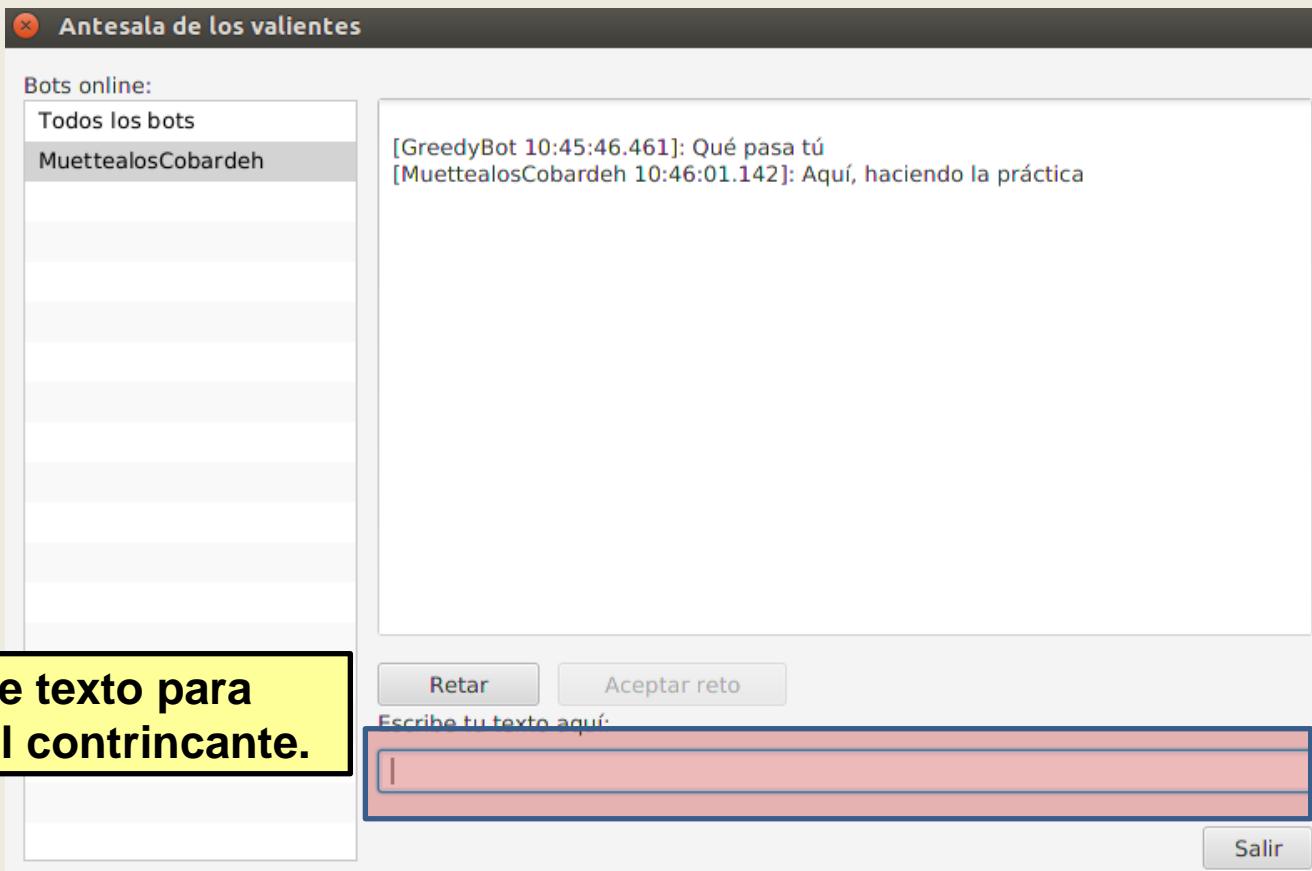
## Participar en la liga extraoficial

- Tras seleccionar el fichero ejecutable, entraremos a la “**Antesala de los valientes**”.



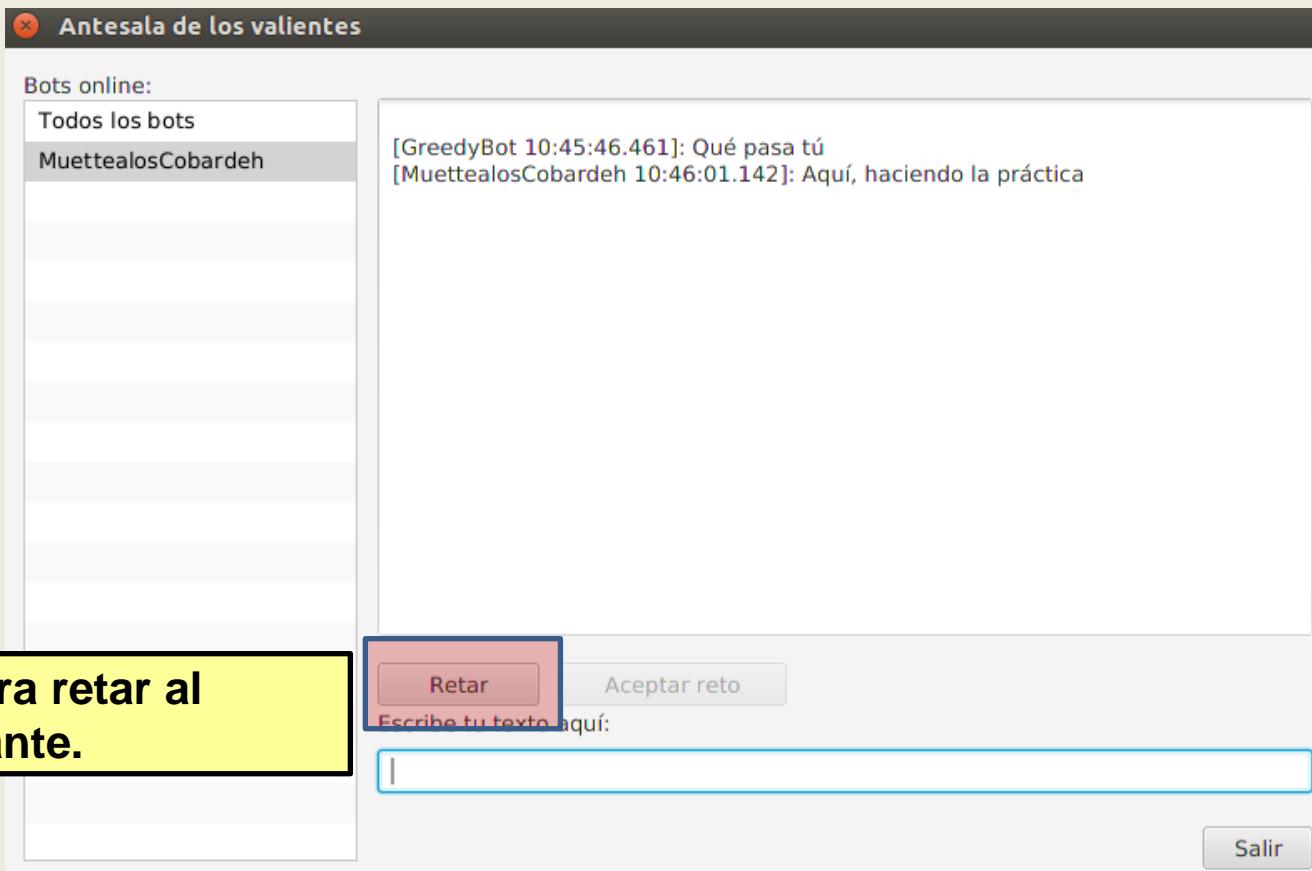
## Participar en la liga extraoficial

- Tras seleccionar el fichero ejecutable, entraremos a la “Antesala de los valientes”.



## Participar en la liga extraoficial

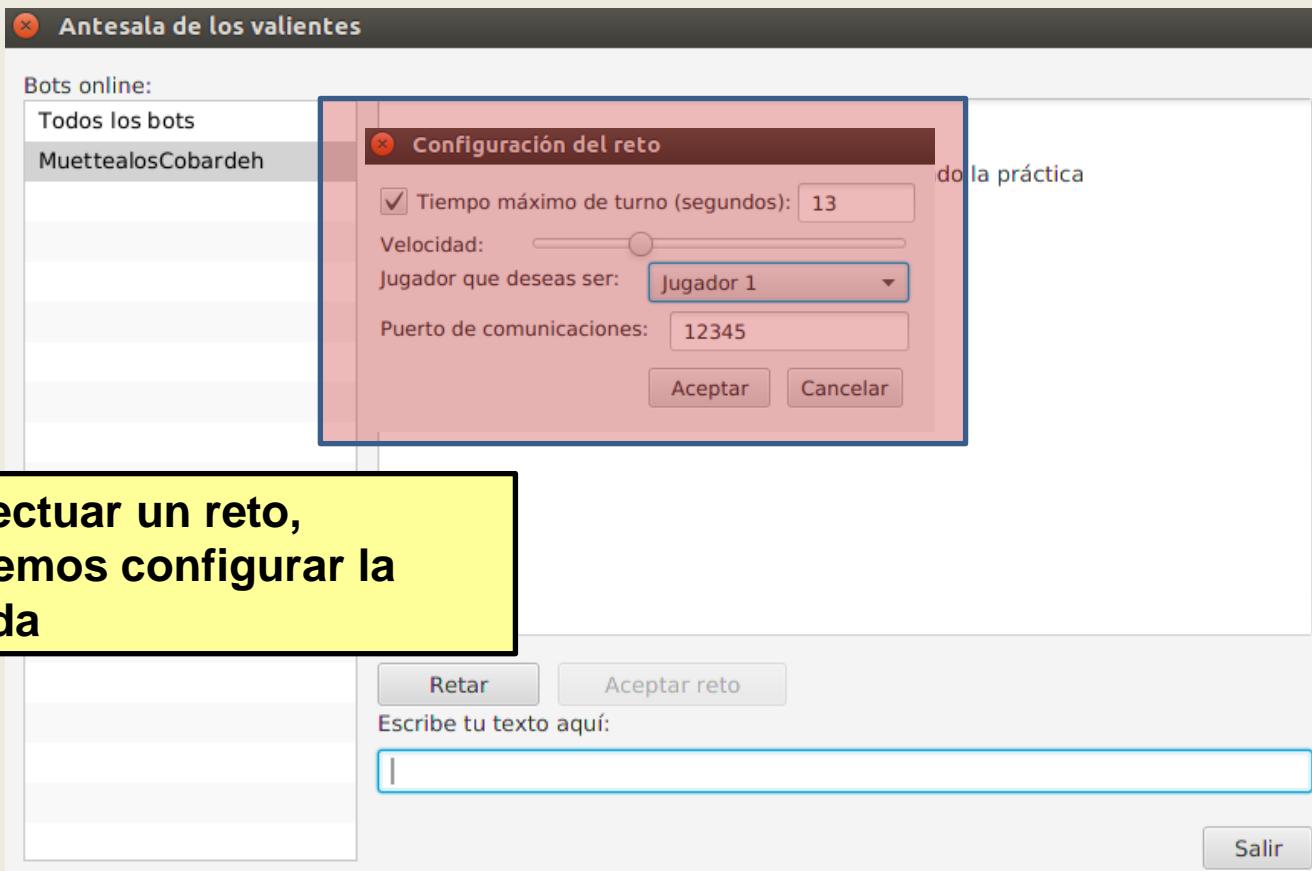
- Tras seleccionar el fichero ejecutable, entraremos a la “**Antesala de los valientes**”.



**Botón para retar al contrincante.**

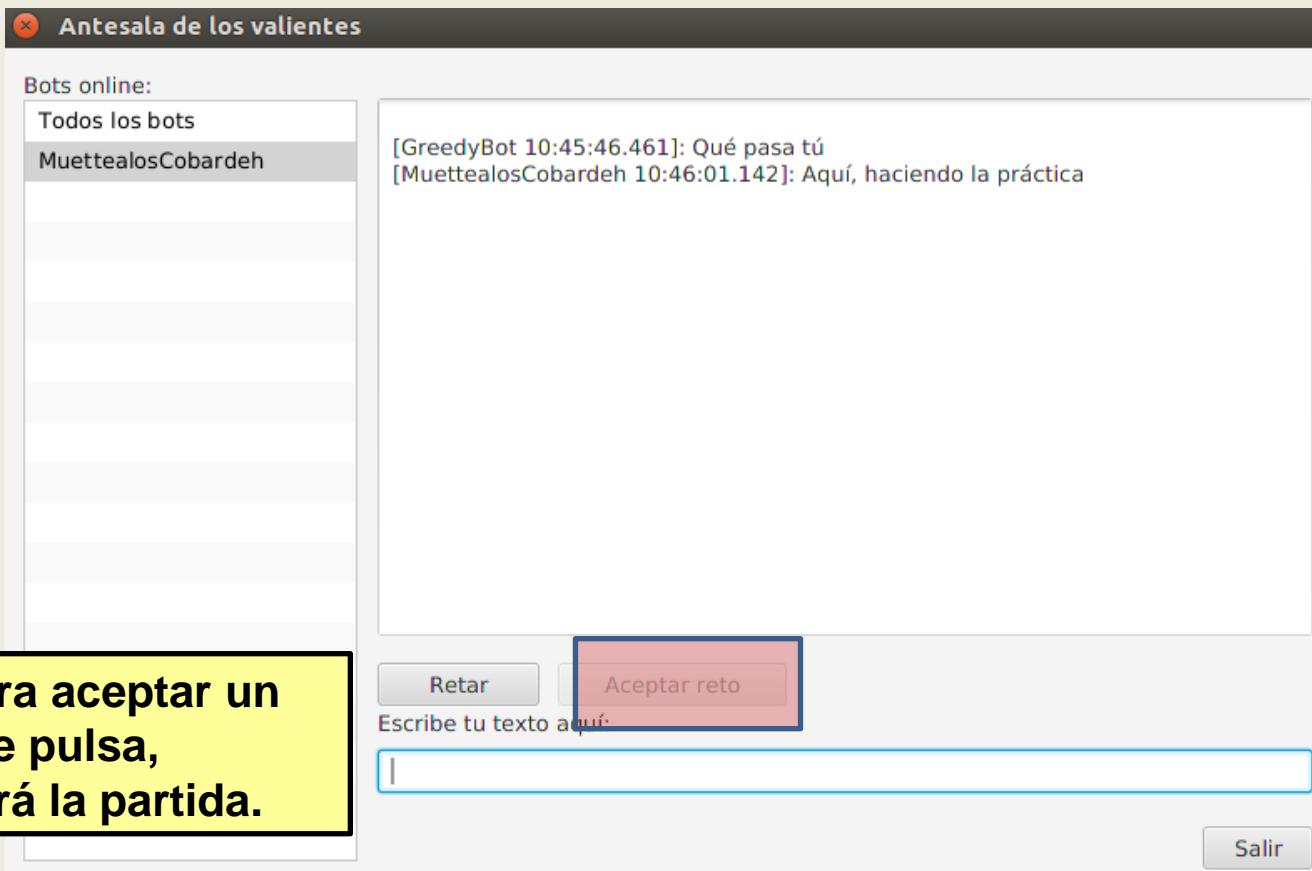
## Participar en la liga extraoficial

- Tras seleccionar el fichero ejecutable, entraremos a la “Antesala de los valientes”.



## Participar en la liga extraoficial

- Tras seleccionar el fichero ejecutable, entraremos a la “Antesala de los valientes”.





# Resultados de la liga extraoficial

- Tras aceptar un reto, el simulador saldrá de la *Antesala de los Valientes* y ejecutará una partida online normal.
  - Al finalizar la partida, los resultados de la misma servirán para actualizar el perfil *extraoficial* de cada alumno, con las nuevas puntuaciones.
  - Se podrá visualizar en todo momento la clasificación global (entre todos los alumnos de todos los grupos de todos los grados) en el botón “**Estadísticas**” del menú “**liga**” del simulador.



1. El juego del Mancala
2. Objetivos de la práctica
3. El simulador
4. Creación de un bot
5. La liga oficial
6. ¿Una liga extraoficial?
7. Evaluación



## Evaluación de la práctica 3

- Valor entre 0 y 10. Se entregan los ficheros .h y .cpp del bot del alumno, junto con una memoria de prácticas.
  - La memoria de prácticas se evalúa de 0 a 3 (valor M). Ver guión de prácticas para mayor detalle del contenido de la memoria.
  - El bot del alumno se ejecutará contra dos bots con una inteligencia simple, desarrollados por los profesores:
    - **GreedyBot:** El mismo que el que se proporciona al estudiante.
    - **MuerteaLosCobardes:** No se proporciona al alumno.
  - Valor B: Si el bot gana en 2 partidas (como J1 y como J2) a **MuerteaLosCobardes**, el alumno tendrá +3 puntos. En caso contrario, si el bot del alumno gana en 2 partidas (como J1 y como J2) a **GreedyBot**, tendrá +1,5 puntos.
  - Valor C: Puntuación en la liga: +x (x= valor entre 0 y 4). Valor que dependerá de los resultados del bot en la liga oficial.

### Evaluación de la práctica 3

- Calificación final:

**Nota de la práctica: Valor M+Valor B+Valor C**

**Por supuesto, esta calificación está sujeta a que el estudiante siga las normas expuestas en el guión de prácticas y que no copie. En caso contrario, la calificación de la práctica será de 0 puntos.**



UNIVERSIDAD  
DE GRANADA



# Inteligencia Artificial

## Grado en Ingeniería Informática

### Seminario 3

Este documento está protegido por la Ley de Propiedad Intelectual ([Real Decreto Ley 1/1996 de 12 de abril](#)).

Queda expresamente prohibido su uso o distribución sin autorización del autor.

Departamento de Ciencias de la Computación e Inteligencia Artificial  
<http://decsai.ugr.es>