UNIVERSIDAD DE GRANADA

DECSAI

Inteligencia Artificial

Seminario 1: Agentes reactivos

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

© Raúl Pérez
fgr@decsai.ugr.es
Departamento de Ciencias de la Computación e Inteligencia Artificial
http://decsai.ugr.es

Evaluación de la parte de prácticas

En Convocatoria Ordinaria

Calificación de varias prácticas/pruebas (100%):

Práctica 1	Agente Reactivos	25%
Práctica 2	Resolución de problemas con agentes reactivos/deliberativos	25%
Práctica 3	Resolución de un juego basado en técnicas de búsqueda	25%
Examen de problemas		25%

¡¡¡No es obligatorio entregar todas las prácticas!!!

¡Es necesario alcanzar un 3 para hacer media con la teoría!

Temporización Inicial Aproximada

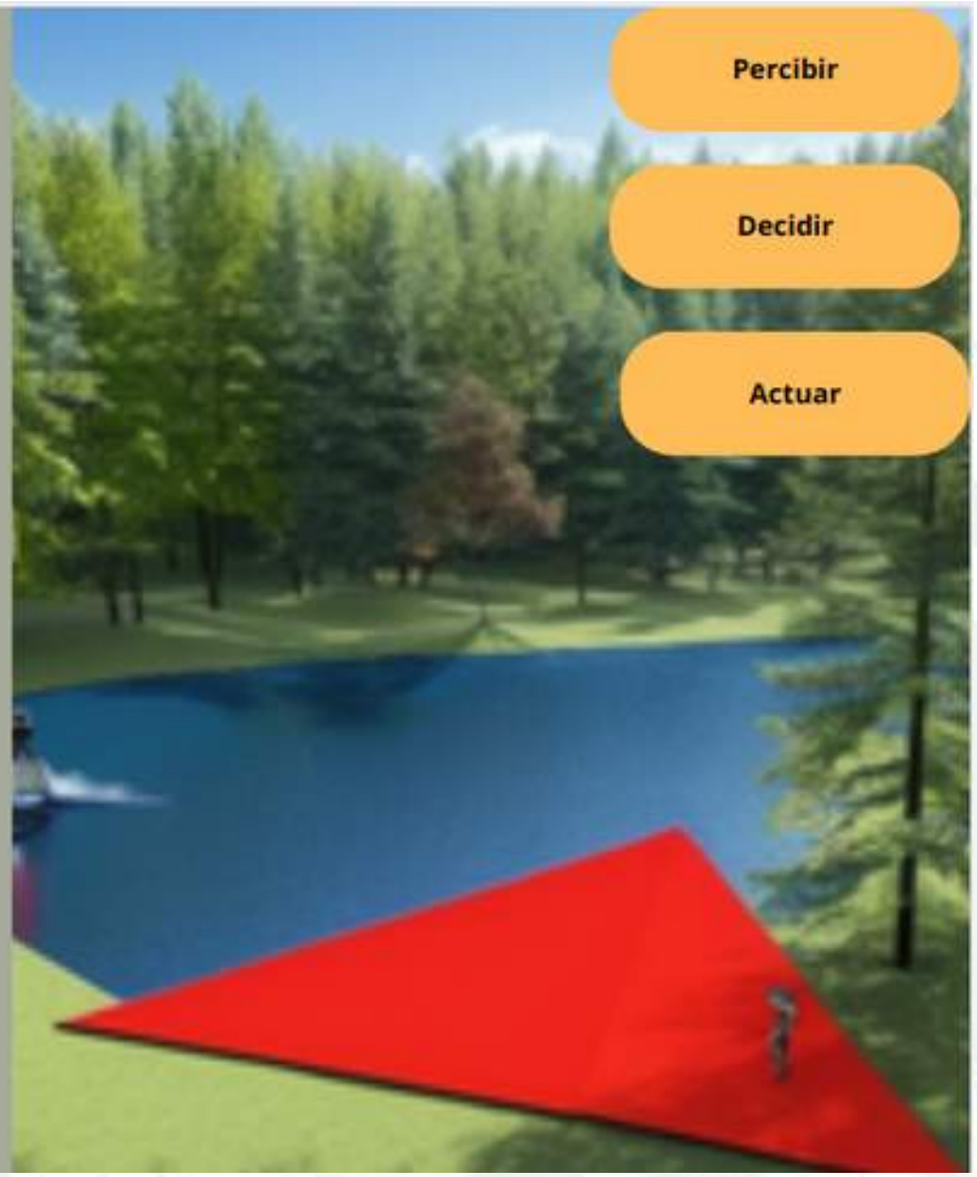
	Planificación de clases de Prácticas/Problemas		Comentarios
Semana	**Prácticas**	**Relaciones de Problemas**	
1	----------	----------	**Sin prácticas**
2	----------	----------	**Sin prácticas**
3	**Seminario 1: Práctica 1 (Agentes Reactivos)**		
4	**Seguimiento/dudas P1**	**RP1**	
5	**Seguimiento/dudas P1**		
	----------	----------	**Semana Santa**
6	**Seguimiento/dudas P1**		
7	**Seminario 2: Práctica 2 (Agentes Deliberativos)**		
8	**Seguimiento/dudas P2**	**RP2**	
9	**Seguimiento/dudas P2**	**RP2**	
10	**Seguimiento/dudas P2**		
11	**Seminario3 / Práctica 3 (Juegos)**		
12		**RP3**	
13	**Seguimiento/dudas P3**		
14	**Seguimiento/dudas P3**		
15	**Seguimiento/dudas P3**		

1. Introducción
2. Presentación del Problema
3. Presentación del Simulador
4. Implementación de un agente
5. Método de evaluación de la práctica

- Un agente reactivo en inteligencia artificial es un proceso que toma decisiones en base a las condiciones actuales del ambiente en el que se encuentra.

Flujo de Control

Tres subtareas se pueden distinguir en un agente reactivo: la de percepción, la de decisión y la de actuación.



Flujo de Control del Agente

Sistema Sensorial

Visión

Ubicación

Colisión

Reinicio

Bateria

PERCIBIR

DECIDIR

Actuadores

Avanzar

Correr

Girar Izquierda 90º

Girar Derecha 45º

No hacer nada

ACTUAR

Objetivo

Sistema Sensorial

Visión

Ubicación

Colisión

Reinicio

Bateria

PERCIBIR

Comportamiento

Diseñar e implementar un modelo de decisión para un agente reactivo con el objetivo de recolectar la mayor información posible sobre como es el mundo que le rodea.

DECIDIR

Actuadores

Avanzar

Correr

Girar Izquierda 90º

Girar Derecha 45º

No hacer nada

ACTUAR

2. Presentación del Juego

NORTE

OESTE

ESTE

SUR

Tipos de Casillas

	‘B’	Árboles
	‘A’	Agua
	‘P’	Precipicios
	‘S’	Suelo pedregoso
	‘T’	Suelo arenoso
	‘M’	Muros
	‘K’	Bikini
	‘D’	Zapatillas
	‘X’	Recarga
	‘G’	Posicionamiento
	‘?’	Casilla desconocida

- Tablero siempre cuadrado con tamaño máximo de 100x100.
- Tablero cerrado. Siempre 3 filas de precipicios al norte, sur, este y oeste.
- Poblado por el jugador (triángulo rojo), aldeanos (cuadrados naranjas) y lobos (hexágonos rosas).

Sistema sensorial

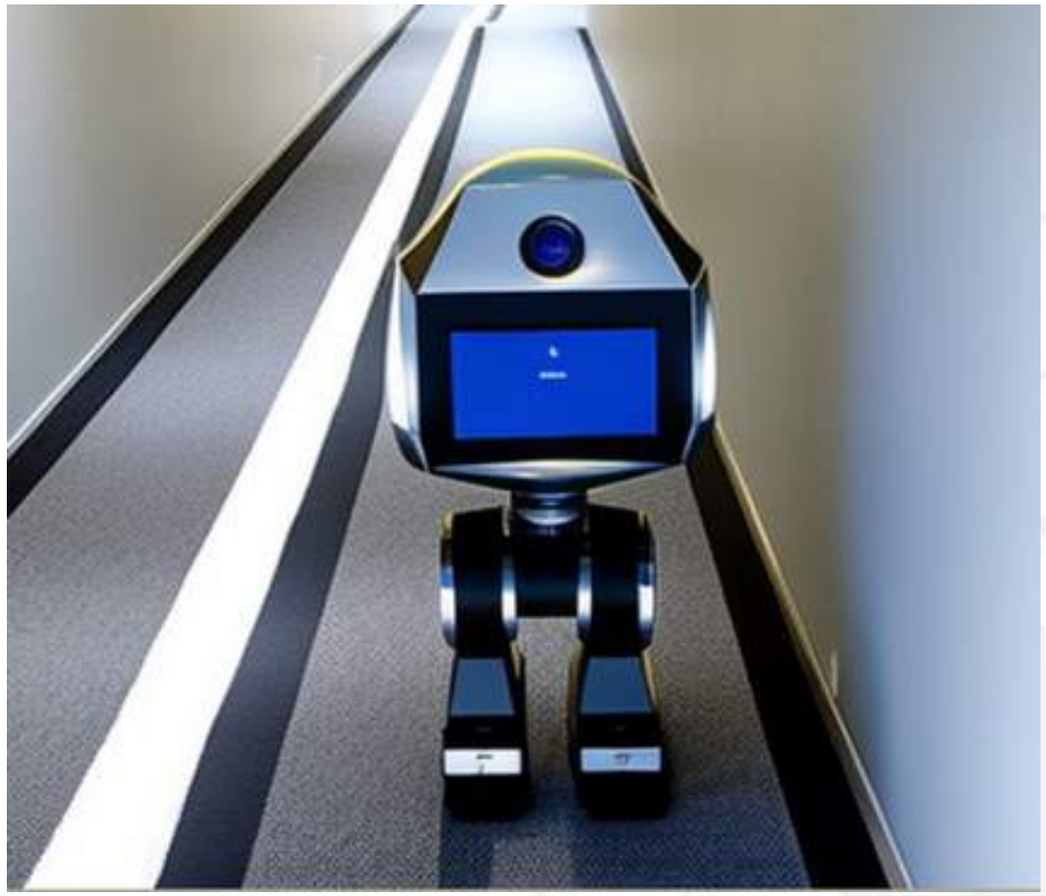
Visión

terreno

agentes

Tiempo transcurrido

tiempo



Sensores de posición

posF

posC

sentido

Nivel del Juego

nivel

Sensor de carga

bateria

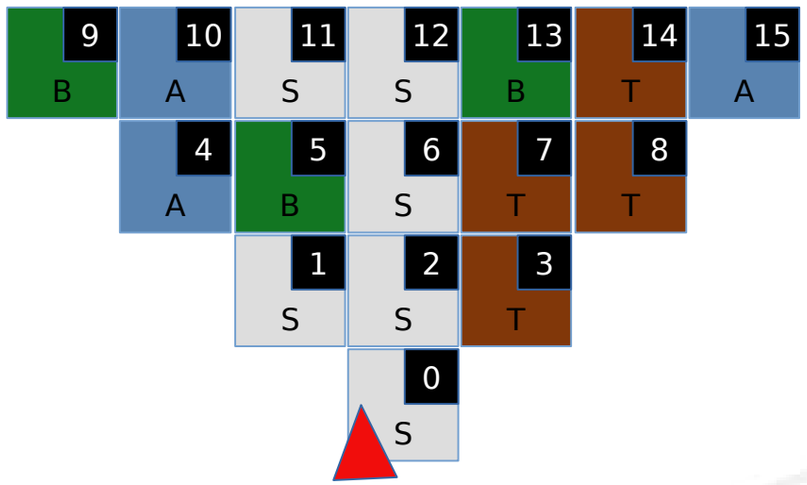
Sensor de choque

colision

Sensor de vida

reset

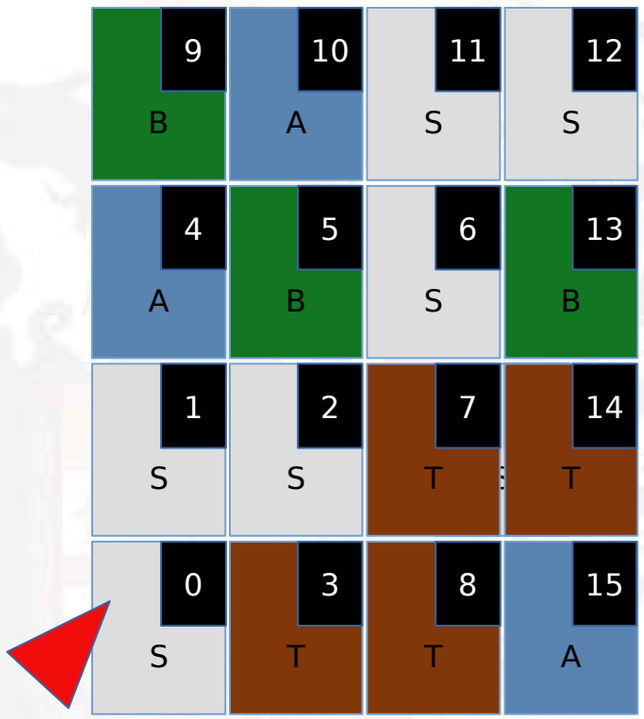
Sistema sensorial (Visión)



Orientaciones:
norte, este, sur, oeste

terreno (A: agua; B: bosque; T: terreno arenoso; S: terreno pedregoso;...)

agentes (a: aldeano; l: lobo; _:desocupado)



Orientaciones:
noreste, sureste, suroeste, noroeste

Acciones

- ***actWALK:*** le permite avanzar a la siguiente casilla del mapa siguiendo su orientación actual. Para que la operación se finalice con éxito es necesario que la casilla de destino sea transitable.

- ***actRUN:*** le permite avanzar dos casillas siguiendo su orientación actual. Para que la operación se finalice con éxito es necesario que la casilla de destino y la casilla intermedia sean transitables.

- ***actTURN_L:*** le permite mantenerse en la misma casilla y girar a la izquierda 90º teniendo en cuenta su orientación.

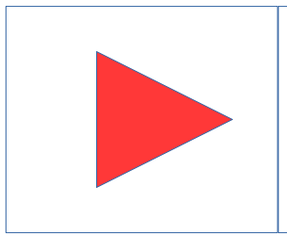
- ***actTURN_SR:*** le permite mantenerse en la misma casilla y girar a la derecha 45º teniendo en cuenta su orientación.

- ***actIDLE:*** no realiza ninguna acción.

Más sobre la acción actWALK

- **Casilla Intransitable**: casilla en la que nunca puede estar situado el agente jugador.
  - Casilla intransitable permanente: los muros ‘M’ y los precipicios ‘P’.
  - Casilla intransitable temporal: ocupada por un aldeano ‘a’ o por un lobo ‘l’.

actWALK

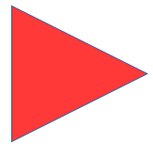
Casilla Inicial	Casilla Destino
	

Casilla Destino	Efecto
‘M’	Choque
‘a’	Choque
‘P’	Reinicio
‘l’	Reinicio

- Choque: el agente se queda en la casilla inicial y se activa el sensor de choque.
- Reinicio: el agente aparece en un nuevo punto del mapa aleatorio orientado hacia el norte.

Más sobre la acción actRUN

actRUN

Casilla Inicial	Casilla Intermedia	Casilla Destino
		

- IMPORTANTE:

1. Para conseguir lo que se obtiene al estar en una casilla del tipo zapatillas, bikini, recarga o posicionamiento, es necesario que esta casilla coincida con la **Casilla Destino** de la acción.

2. **Choque** deja siempre en **Casilla Inicial,** aunque la casilla intransitable sea **Casilla Destino**.

Casilla Intermedia	Casilla Destino	Efecto
‘M’	?	Choque
‘a’	?	Choque
‘P’	?	Reinicio
‘l’	?	Reinicio
Transitable	‘M’	Choque
Transitable	‘a’	Choque
Transitable	‘P’	Reinicio
Transitable	‘l’	Reinicio

Cada acción realizada por el agente tiene un **coste** en **tiempo de simulación** y en **consumo de batería**.

- Tiempo de simulación: **todas las acciones consumen un instante de tiempo de simulación independientemente de la acción que se realice y del terreno donde se encuentre el jugador**.

- Una simulación tiene como máximo 3000 instantes de tiempo.

Consumo de Batería

actWALK		
Tipo de Casilla	**Gasto Normal Batería**	**Gasto Reducido Batería**
'A'	50	5 (con **Bikini**)
'B'	25	5 (con **Zapatillas**)
'T'	2	2
Resto de Casillas	1	1

No hay sensor que indique si disponemos del bikini o de las zapatillas, por lo que debemos crear las variables de estado correspondientes.

actRUN		
Tipo de Casilla	**Gasto Normal Batería**	**Gasto Reducido Batería**
'A'	500	10 (con **Bikini**)
'B'	250	15 (con **Zapatillas**)
'T'	3	3
Resto de Casillas	1	1

Consumo de Batería

actTURN_SR		
Tipo de Casilla	**Gasto Normal Batería**	**Gasto Reducido Batería**
'A'	50	5 (con **Bikini**)
'B'	10	1 (con **Zapatillas**)
'T'	2	2
Resto de Casillas	1	1

No hay sensor que indique si disponemos del bikini o de las zapatillas, por lo que debemos crear las variables de estado correspondientes.

actTURN_L		
Tipo de Casilla	**Gasto Normal Batería**	**Gasto Reducido Batería**
'A'	50	5 (con **Bikini**)
'B'	20	1 (con **Zapatillas**)
'T'	2	2
Resto de Casillas	1	1

Objetivo

Definir un comportamiento reactivo para nuestro personaje que le permita reconocer el máximo porcentaje posible del mundo y que sepa orientarse adecuadamente. Para realizar esta tarea tiene todo el tiempo de simulación y sólo al final es cuando este objetivo es evaluado.

El agente se enfrentará ante 4 niveles de dificultad para realizar ese reconocimiento del mapa que tiene a su alrededor.

Niveles del juego

Nivel 0: Todo el sistema sensorial funciona correctamente y el agente es el único agente que está en la simulación.

Nivel 1: No funciona los sensores de posicionamiento posF, posC y sentido (dan siempre el valor -1 los dos primeros y siempre ***norte*** el último). Inicialmente el agente empieza con orientación ***norte***.
Para poder situarse en el mapa se utilizan las casillas de **Posicionamiento**. En esas casillas, los sensores darán los valores correctos.

Nivel 2: La situación sensorial del nivel anterior, pero ahora en el mapa pueden aparecer otros agentes como los aldeanos y los lobos. En este nivel se deben gestionar las colisiones y los reinicios con estos agentes.

Nivel 3: Las condiciones del nivel 2 pero en este caso, (1) la orientación es desconocida al inicio, aunque se sabe que no es diagonal (es decir, es norte o sur o este u oeste). Además, el sensor de visión del terreno no da información sobre lo que aparece en las casillas 6, 11, 12 y 13.

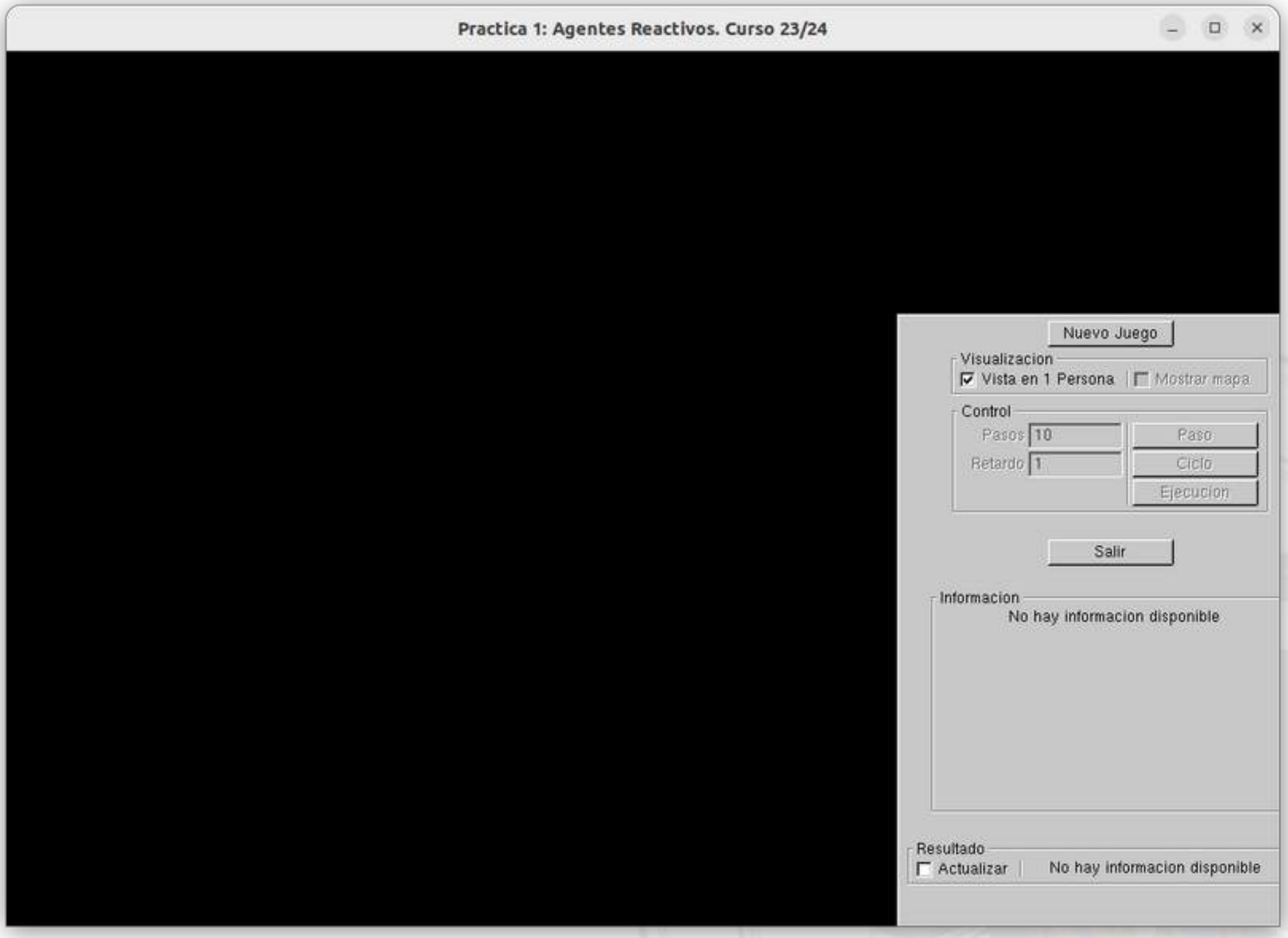
Se proporciona el software para el sistema operativo Linux en el repositorio de GitHub https://github.com/ugr-ccia-IA/practica1

- Existen dos versiones del software: practica1 y practica1SG. El primero corresponde al simulador con interfaz gráfica, mientras que el segundo es un simulador en modo *batch* sin interfaz.

- Todos los detalles y explicación de la instalación, uso y descripción de las variables necesarias para su desarrollo se encuentran en el guion de prácticas asociado a esta presentación y en el repositorio de GitHub (en la parte de instalación).

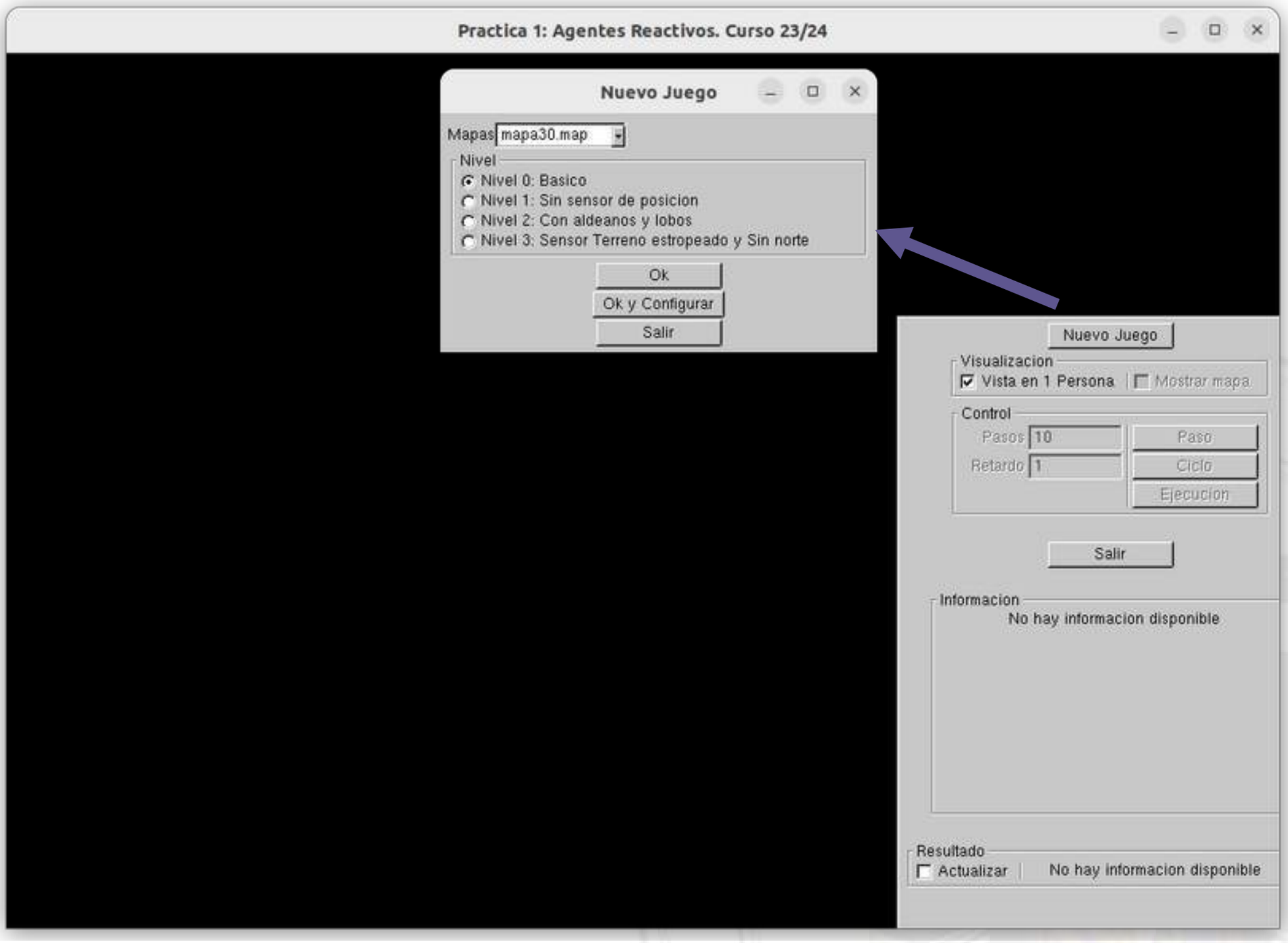
Práctica 1

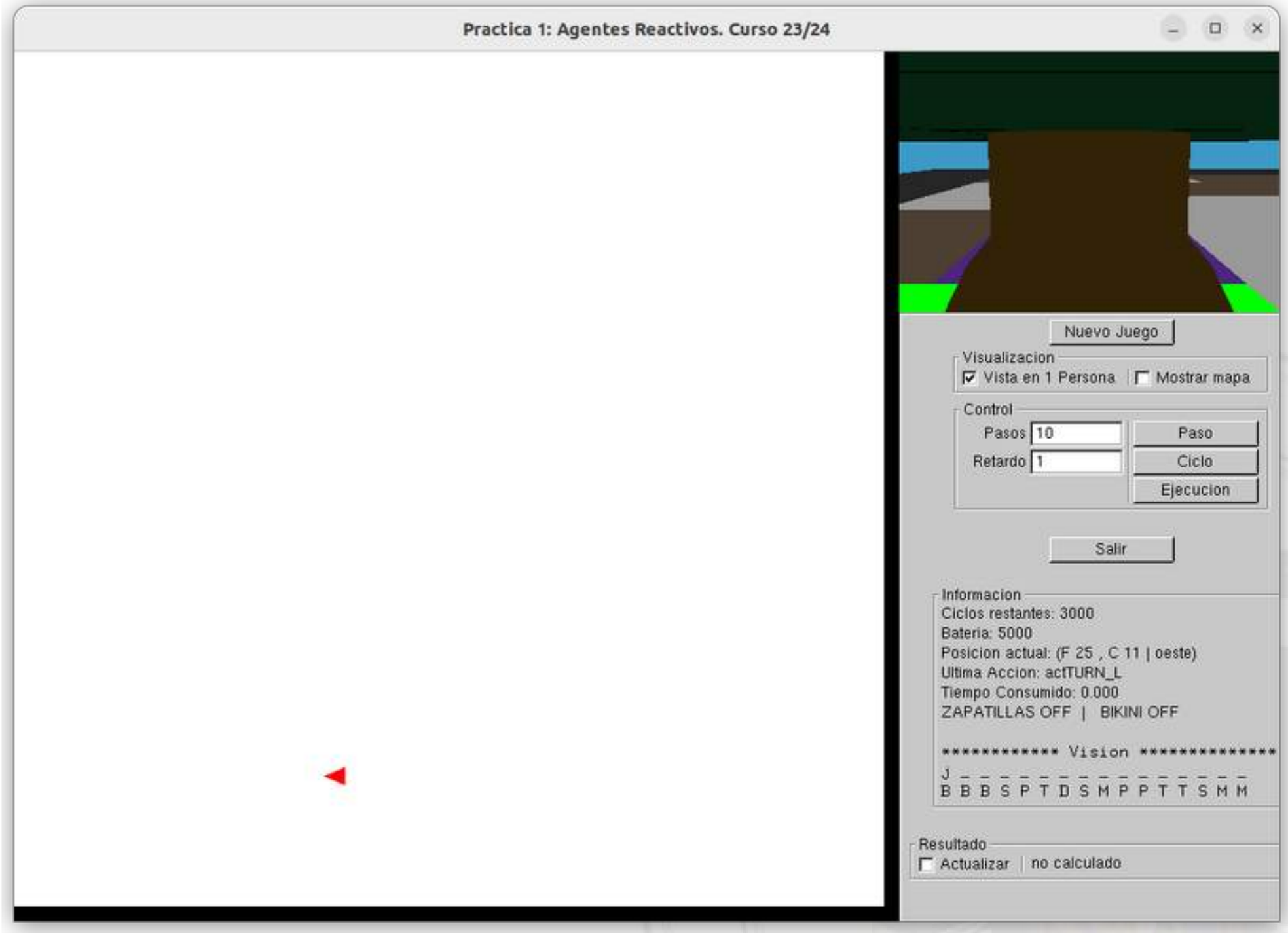
El software: ./practica1



Práctica 1

El software: ./practica1




Practica 1: Agentes Reactivos. Curso 23/24
Nuevo Juego
Visualizacion
Vista en 1 Persona
Mostrar mapa
Control
Pasos 10
Retardo 1
Paso
Ciclo
Ejecucion
Salir
Informacion
Ciclos restantes: 3000
Bateria: 5000
Posicion actual: (F 25 , C 11 | oeste)
Ultima Accion: actTURN_L
Tiempo Consumido: 0.000
ZAPATILLAS OFF | BIKINI OFF
********** Vision **********
J _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
B B B S P T D S M P P T T S M M
Resultado
Actualizar | no calculado

- **Sistema *batch:***

  ./practica1SG mapas/mapa30.map 1 0 4 5 1

- fichero de mapa
- semilla generadora de números aleatorios
- Nivel (0, 1, 2 o 3)
- fila origen
- columna origen
- Orientación (0=norte, 1=noreste, 2=este, 3=sureste, 4=sur, 5=suroeste, 6=oeste, 7=noroeste)

Se incluye esta opción para acelerar la ejecución completa de una simulación por un lado y por otro, para que el entorno gráfico no sea un inconveniente a la hora de depurar errores.

Nota: Esta es la forma de invocarlo desde una terminal de Linux.

Al finalizar la ejecución, se nos proporciona el tiempo consumido, la cantidad de batería con la que terminó la simulación, el número de colisiones que hemos sufrido (por obstáculos u otros agentes), la cantidad de reinicios y el porcentaje de mapa descubierto.

```
Instantes de simulacion no consumidos: 0
Tiempo Consumido: 0.049377
Nivel Final de Bateria: 3000
Colisiones: 0
Reinicios: 0
Porcentaje de mapa descubierto: 0
```

- **Sistema *batch y Entorno Gráfico:***

  ./practica1 mapas/mapa30.map 1 0 4 5 1

- **Se puede usar una versión combinada del sistema batch con el entorno gráfico. Tras esta llamada, se levanta el entorno gráfico del simulador y *se detiene tras ejecutar la primera acción*.**

- **La interpretación y orden de los parámetros en la llamada en línea es igual que cuando se ejecuta practica1SG.**

- **Esta sería una manera simple de fijar las condiciones iniciales del simulador sin tener que navegar por el sistema de ventanas.**

- **Los resultados que se obtienen en esta versión paramétrica de practica1 debe coincidir con los que se obtienen con los mismos parámetros en la versión practica1SG.**

- Sólo hay 2 ficheros relevantes para la práctica: “jugador.cpp” y “jugador.hpp”.

- Estos dos archivos se encuentran en la carpeta “Comportamientos_Jugador”

- El resto de los archivos que se adjuntan con la práctica no se pueden modificar y se han incluido para poder hacer la compilación.

- La compilación genera dos ejecutables:
  - 1. practica1 (simulador con entorno gráfico)
  - 2. practica1SG (simulador sin entorno gráfico)

Fichero
jugador.hpp

```
1   #ifndef COMPORTAMIENTOJUGADOR_H
2   #define COMPORTAMIENTOJUGADOR_H
3
4   #include "comportamientos/comportamiento.hpp"
5   using namespace std;
6
7   class ComportamientoJugador : public Comportamiento{
8
9     public:
10      ComportamientoJugador(unsigned int size) : Comportamiento(size){
11        // Constructor de la clase
12        // Dar el valor inicial a las variables de estado
13      }
14
15      ComportamientoJugador(const ComportamientoJugador & comport) : Comportamiento(comport){}
16      ~ComportamientoJugador(){}
17
18      Action think(Sensores sensores);
19      int interact(Action accion, int valor);
20
21    private:
22    // Declarar aquí las variables de estado
23  };
24  #endif
25
```

Constructor de Clase

Fichero
jugador.hpp

```
1   #ifndef COMPORTAMIENTOJUGADOR_H
2   #define COMPORTAMIENTOJUGADOR_H
3
4   #include "comportamientos/comportamiento.hpp"
5   using namespace std;
6
7   class ComportamientoJugador : public Comportamiento{
8
9     public:
10      ComportamientoJugador(unsigned int size) : Comportamiento(size){
11        // Constructor de la clase
12        // Dar el valor inicial a las variables de estado
13      }
14
15      ComportamientoJugador(const ComportamientoJugador & comport) : Comportamiento(comport){}
16      ~ComportamientoJugador(){}
17
18      Action think(Sensores sensores);
19      int interact(Action accion, int valor);
20
21    private:
22    // Declarar aqui las variables de estado
23  };
24  #endif
25
```

Constructor de Copia

Fichero
jugador.hpp

```
1   #ifndef COMPORTAMIENTOJUGADOR_H
2   #define COMPORTAMIENTOJUGADOR_H
3
4   #include "comportamientos/comportamiento.hpp"
5   using namespace std;
6
7   class ComportamientoJugador : public Comportamiento{
8
9     public:
10      ComportamientoJugador(unsigned int size) : Comportamiento(size){
11        // Constructor de la clase
12        // Dar el valor inicial a las variables de estado
13      }
14
15      ComportamientoJugador(const ComportamientoJugador & comport) : Comportamiento(comport){}
16      ~ComportamientoJugador(){}
17
18      Action think(Sensores sensores);
19      int interact(Action accion, int valor);
20
21    private:
22    // Declarar aquí las variables de estado
23  };
24  #endif
25
```

Destructor de la clase

Fichero
jugador.hpp

```
1  #ifndef COMPORTAMIENTOJUGADOR_H
2  #define COMPORTAMIENTOJUGADOR_H
3
4  #include "comportamientos/comportamiento.hpp"
5  using namespace std;
6
7  class ComportamientoJugador : public Comportamiento{
8
9    public:
10     ComportamientoJugador(unsigned int size) : Comportamiento(size){
11       // Constructor de la clase
12       // Dar el valor inicial a las variables de estado
13     }
14
15     ComportamientoJugador(const ComportamientoJugador & comport) : Comportamiento(comport){}
16     ~ComportamientoJugador(){}
17
18     Action think(Sensores sensores);
19     int interact(Action accion, int valor);
20
21   private:
22   // Declarar aquí las variables de estado
23 };
24 #endif
25
```

Método donde definir el comportamiento del agente
Su implementación se hace en jugador.cpp

Fichero
jugador.cpp

```
1  #include "../Comportamientos_Jugador/jugador.hpp"
2  #include <iostream>
3  using namespace std;
4
5  Action ComportamientoJugador::think(Sensores sensores)
6  {
7
8      Action accion = actIDLE;
9
10     // Mostrar el valor de los sensores
11     cout << "Posicion: fila " << sensores.posF << " columna " << sensores.posC;
12     switch (sensores.sentido)
13     {
14         case norte:    cout << " Norte\n";    break;
15         case noreste:  cout << " Noreste\n";  break;
16         case este:     cout << " Este\n";     break;
17         case sureste:  cout << " Sureste\n";  break;
18         case sur:      cout << " Sur\n";      break;
19         case suroeste:cout << " Suroeste\n";    break;
20         case oeste:    cout << " Oeste\n";    break;
21         case noroeste:cout << " Noroeste\n";    break;
22     }
23     cout << "Terreno: ";
24     for (int i=0; i<sensores.terreno.size(); i++)
25         cout << sensores.terreno[i];
26
27     cout << "  Agentes: ";
28     for (int i=0; i<sensores.agentes.size(); i++)
29         cout << sensores.agentes[i];
30
31     cout << "\nColision: " << sensores.colision;
32     cout << "  Reset: " << sensores.reset;
33     cout << "  Vida: " << sensores.vida << endl<< endl;
34
35     return accion;
36 }
```

***think()* es un método que toma como entrada el estado sensorial del agente y devuelve como salida la siguiente acción a realizar.**

Es la encargada de definir el comportamiento del agente.

En su versión inicial, *think*(), únicamente muestra la forma en la que hay que invocar a los distintos sensores del agente.

Importante indicar que existe una matriz llamada *mapaResultado* donde se ha de colocar lo que se ha descubierto del mapa.

Para empezar a implementar esta parte se recomienda hacer el tutorial que se adjunta como material adicional a esta práctica.

- La forma de evaluación es la siguiente:
  - Se tomarán 3 mapas semejantes a los que se proporcionan como material de la práctica.
  - Sobre cada mapa se aplicará el agente propuesto por el estudiante para los 4 niveles (del 0 al 3).
  - Sobre cada nivel y mapa se realizará una simulación, y llamaremos s_i al valor devuelto por el software de porcentaje de mapa descubierto.
  - Si la simulación no termina correctamente por alguna razón (normalmente suele ser por un error de segmentación y que en general denominaremos "core") el valor que se le dará a s_i será -0.1.

- Se pide desarrollar un programa (modificando el código de los ficheros del simulador ‘jugador.cpp’ y ‘jugador.hpp’) con el comportamiento requerido para el agente.

- Estos dos ficheros deberán subirse a la parte de Prácticas de la asignatura en PRADO, en un fichero ZIP (de nombre “practica1.zip” que no contenga carpetas ni subcarpetas).

- El archivo ZIP deberá contener solo el código fuente de estos dos ficheros con la solución del alumno.

- La nota asociada a cada mapa se calcula como:

$$\mathbf{Nota = s_0\, p_0 + s_1\, p_1 + s_2\, p_2 + s_3\, p_3}$$

siendo p_i la puntuación asociada al nivel i, y que se describe en la siguiente tabla.

Nivel	0	1	2	3
Puntuación	2	3	2	3

y siendo $\mathbf{s_i}$ el grado de descubrimiento del mapa descubierto en el nivel i. y se calcula siguiendo la siguiente fórmula

$$\mathbf{s_i = min(1.0, (aciertos\text{-}errores)/(totaldecasillas * 0.85))}$$

Para el cálculo de $\mathbf{s_i}$ se tiene en cuenta que cada casilla situada en ***mapaResultado*** que no coincidan con el mapa original resta un acierto.

La **nota final** es la **media aritmética** de las obtenidas en los 3 mapas.

Fecha límite entrega	Cuestionario de autoevaluación
7 de Abril hasta las 23:00 horas	Disponible desde el 8 de Abril al 10 de Abril hasta las 23:00 horas

Esta **práctica es INDIVIDUAL**

En el caso de detectar prácticas copiadas, los involucrados (tanto el que se copió como el que se ha dejado copiar) tendrán suspensa la asignatura.