Universidad de Granada

decsai.ugr.es

Inteligencia Artificial

Seminario 1

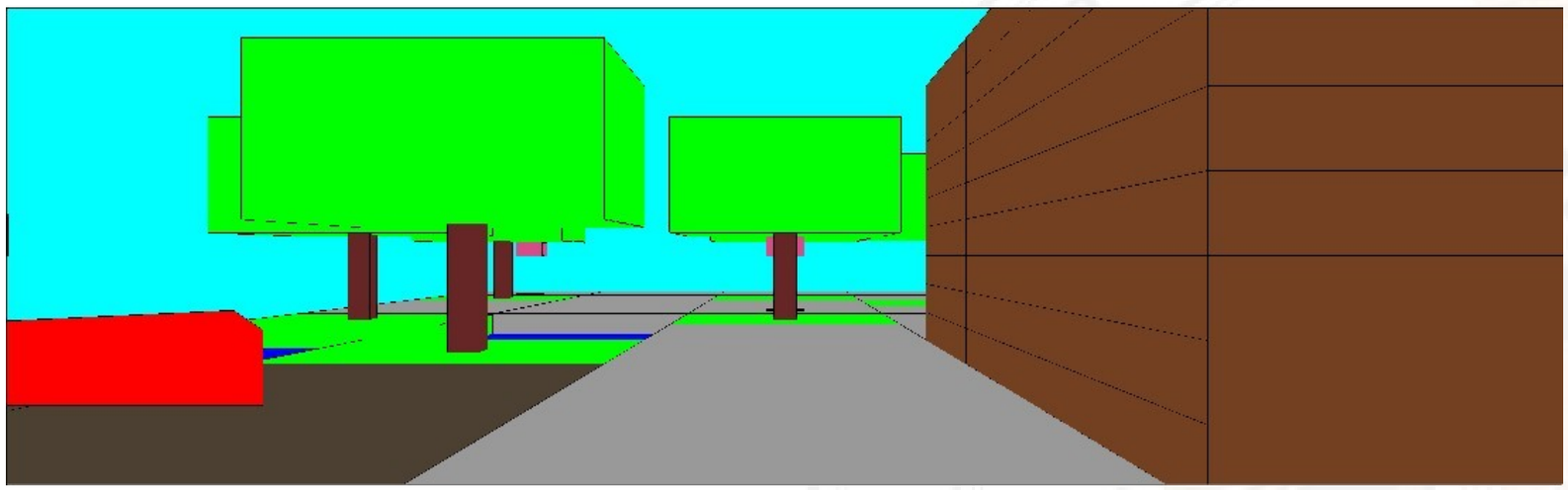
Agentes Reactivos

DECSAI

Departamento de Ciencias de la Computación e Inteligencia Artificial

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

- Diseñar e implementar un **agente reactivo**, capaz de percibir el ambiente y actuar de acuerdo a un comportamiento simple predefinido.

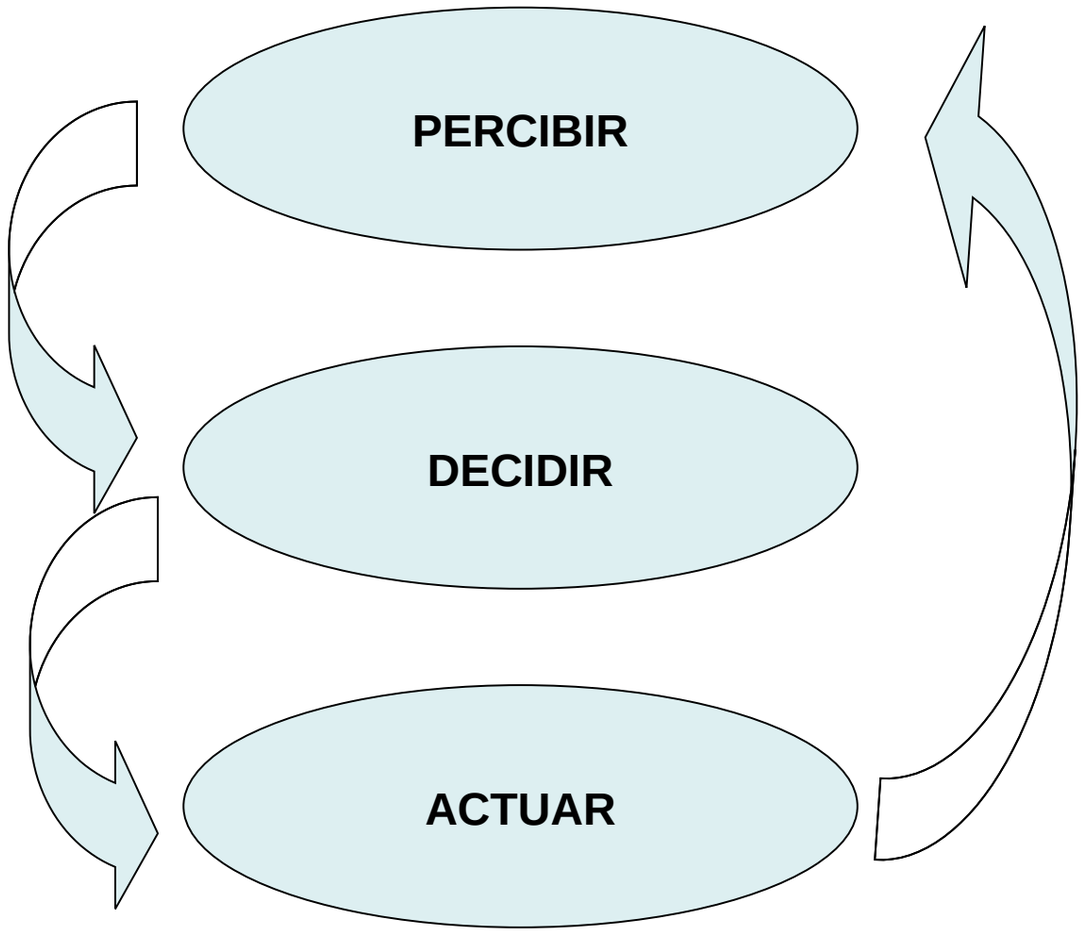


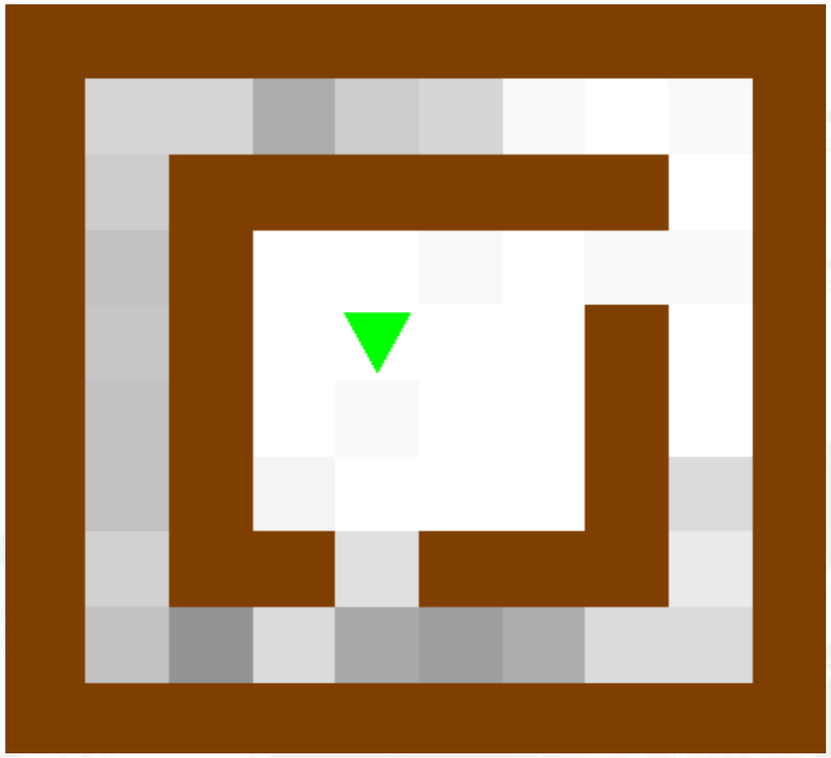
- En esta práctica se diseñará e implementará un agente reactivo y deliberativo basado en los ejemplos del libro *Stuart Russell, Peter Norvig, "Inteligencia Artificial: Un enfoque Moderno", Prentice Hall, Segunda Edición, 2004*.
- El simulador que utilizaremos fue inicialmente desarrollado por el profesor Tsung-Che Chiang de la NTNU (National Taiwan Normal University of Science and Technology), pero la versión sobre la que se va a trabajar ha sido desarrollada por los profesores de la asignatura.

- Originalmente, el simulador estaba orientado a experimentar con comportamientos en aspiradoras inteligentes.

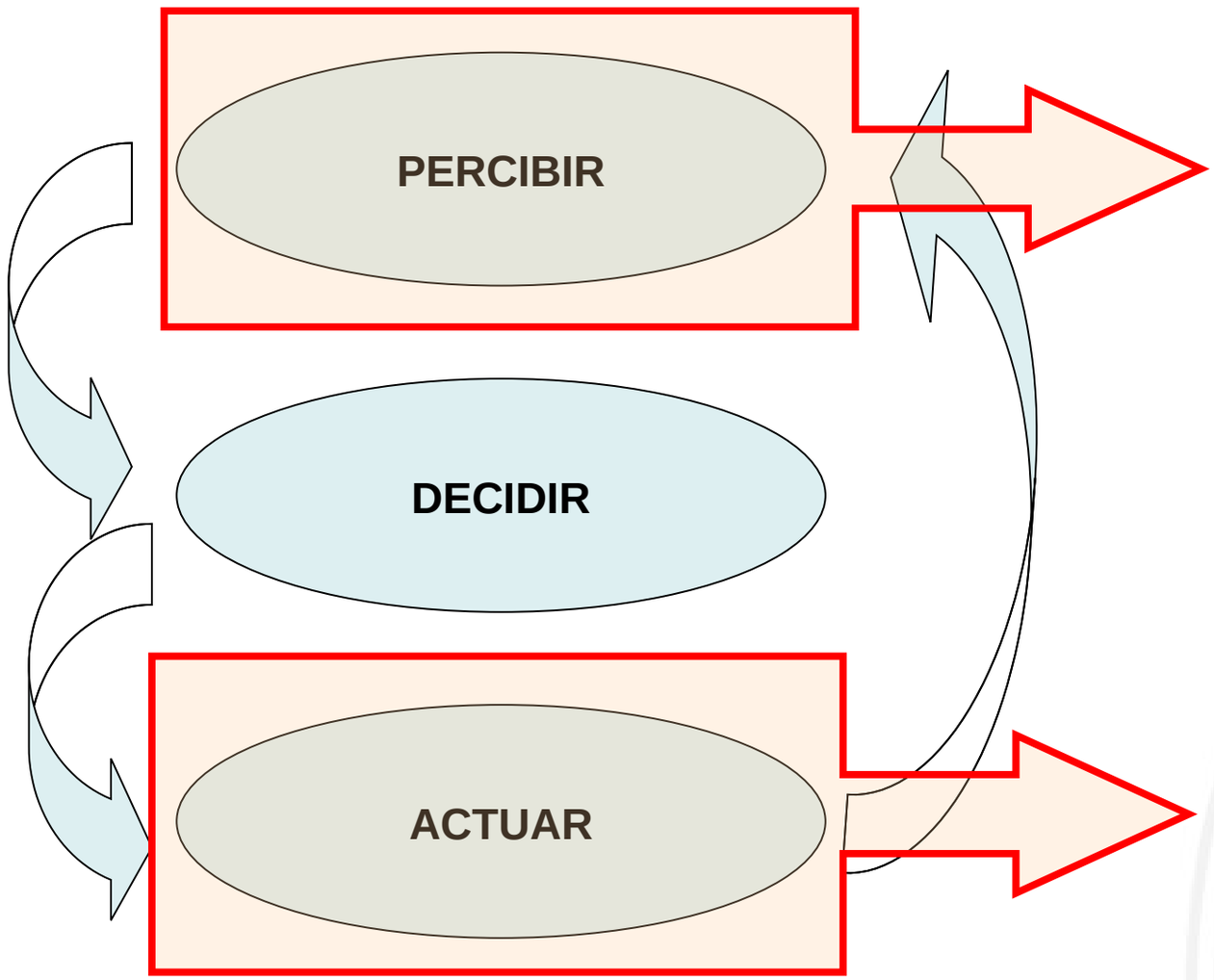
En su versión más simple, una aspiradora inteligente presenta un comportamiento ***reactivo*** puro: busca suciedad, la limpia, se mueve, detecta suciedad, la limpia, se mueve, y continúa con este ciclo hasta que se cumple alguna condición de parada.

Flujo de Control de un Agente Reactivo





Flujo de Control de un Agente Reactivo



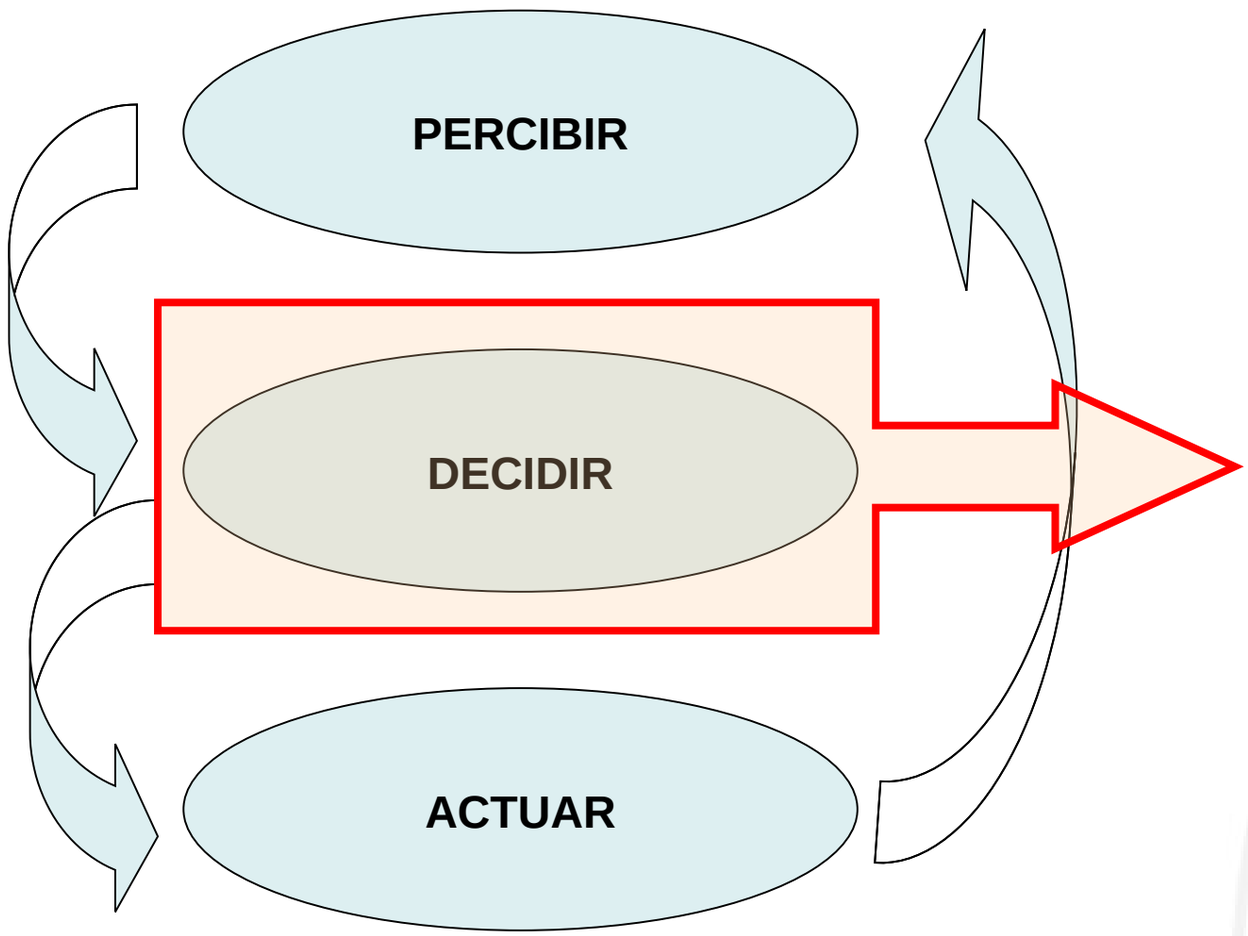
Sensores

1) Visión
2) Posición
3) Orientación
4) Colisión
5) Reinicio
6) Batería
7) Tiempo
8) Nivel

Acciones

1) Avanzar
2) Girar 90 grados Izquierda
3) Girar 90 grados Derecha
4) No hacer nada

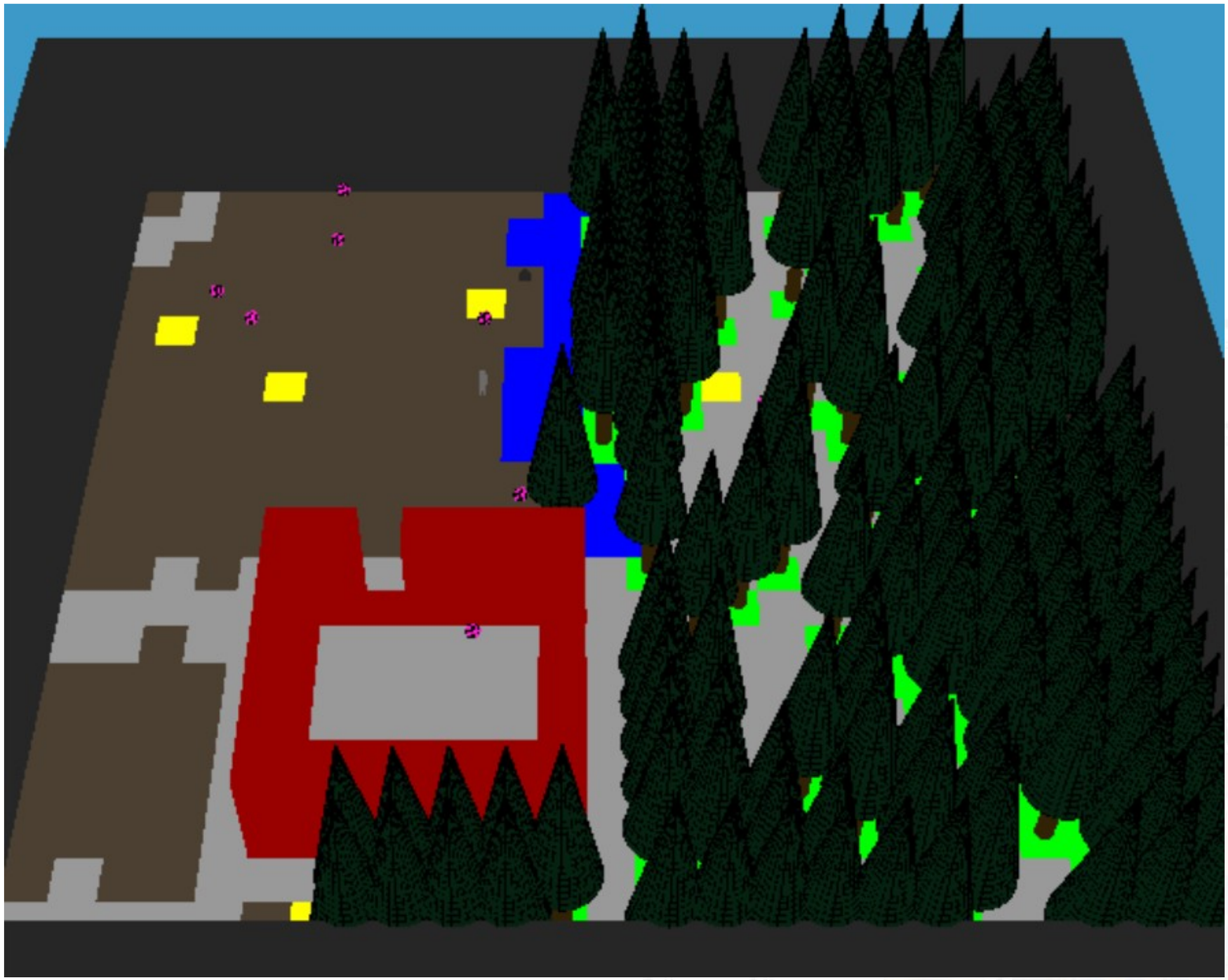
Flujo de Control de un Agente Reactivo

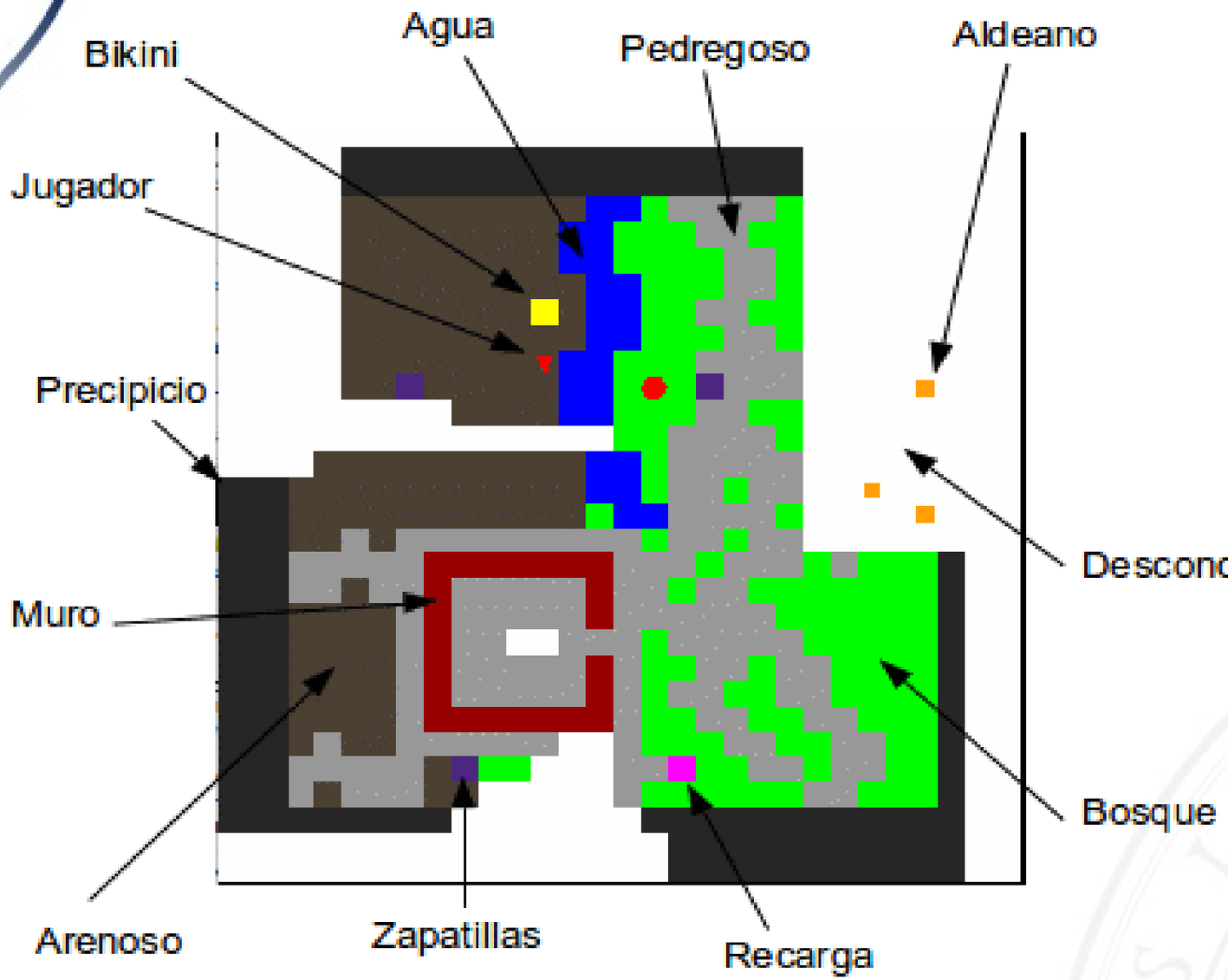


El objetivo de la práctica será:

Diseñar e implementar un modelo de decisión para este agente reactivo.

EL JUEGO



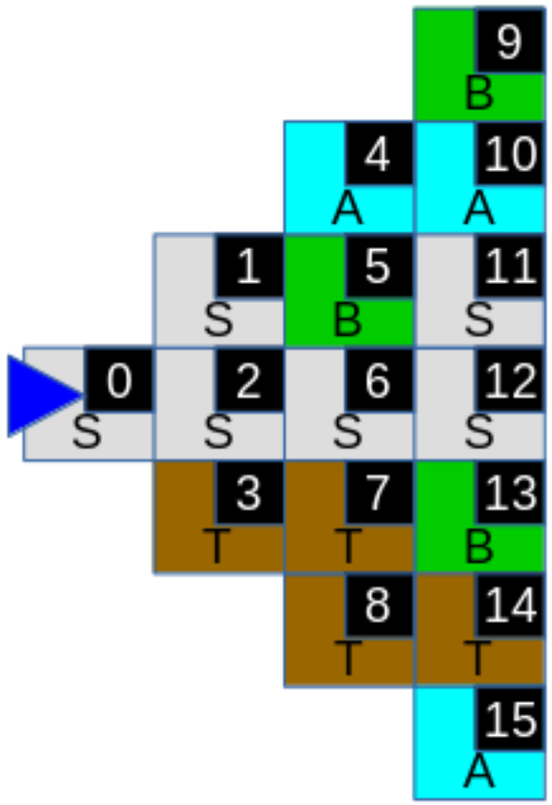


	‘B’	Árboles
	‘A’	Agua
	‘P’	Precipicios
	‘S’	Suelo pedregoso
	‘T’	Suelo arenoso
	‘M’	Muros
	‘K’	Bikini
	‘D’	Zapatillas
	‘X’	Recarga
	‘G’	Posicionamiento
	‘?’	Casillas desconocida

- El agente se reiniciará si cae en una casilla precipicio. Tampoco puede atravesar, muros, ni otros personajes, y por tanto chocará contra ellos. Si choca con un lobo, también se reiniciará.
- El tamaño máximo del mapa es de 100 filas por 100 columnas.

Sistema sensorial

- Sensor de choque (colision)
- Sensor de vida (reset)
- Sensores de posición (posF, posC, sentido)
- Sensor de carga de batería (bateria)
- Sensor de nivel (nivel)
- Sensor de tiempo consumido (tiempo)



Sensor visual: terreno (A: agua; B: bosque; T: terreno arenoso; S: terreno pedregoso;...) y superficie (para ver personajes sobre ese terreno)

Nuestro personaje puede realizar varias **acciones** distintas durante el juego:

- ***actFORWARD:*** 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 para nuestro personaje.
- ***actTURN_L:*** le permite mantenerse en la misma casilla y girar a la izquierda 90º teniendo en cuenta su orientación.
- ***actTURN_R:*** le permite mantenerse en la misma casilla y girar a la derecha 90º teniendo en cuenta su orientación.
- ***actIDLE:*** pues como su nombre indica, no hace nada.

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

- En cuanto al tiempo de simulación, **todas las acciones consumen un instante independientemente de la acción que se realice y del terreno donde se encuentre el jugador**.
  - Cada simulación conlleva 3000 instantes de tiempo.
  - Cada acción consume 1 instante de tiempo.

- En cuanto al **consumo de batería** decir que actIDLE consume 0 de batería y que **el consumo de este recurso** de las acciones actTURN_L, actTURN_R y actFORWARD **depende del tipo de terreno asociado a la casilla donde se inició dicha acción**. En las siguientes tablas se muestran dichos valores de consumo de batería.

actFORWARD		
Casilla	**Gasto Normal**	**Gasto Reducido**
'A'	200	10 (con **Bikini**)
'B'	100	15 (con **Zapatillas**)
'T'	2	2
Resto	1	1

actTURN_L / actTURN_R		
Casilla	**Gasto Normal**	**Gasto Reducido**
'A'	500	5 (con **Bikini**)
'B'	3	1 (con **Zapatillas**)
'T'	2	2
Resto	1	1

Los muros y precipicios no son transitables!

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

- Definir un comportamiento reactivo para nuestro personaje que le permita reconocer el máximo porcentaje posible del mundo y que sepa orientarlo 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 5 niveles de dificultad para realizar ese reconocimiento del mapa que tiene a su alrededor.

Nivel 0: Todo el sistema sensorial funciona correctamente.

Nivel 1: No funciona los sensores de posicionamiento posF y posC (dan siempre el valor -1). En este caso, para poder situarse en el mapa es necesario situarse en una casilla de Posicionamiento que hará que el sensor de los valores correctos en filas y columnas.

Nivel 2: Tampoco funciona el sensor de orientación. Ahora sólo sabemos que en la posición inicial salimos orientados al norte. El sensor de orientación siempre dice que estamos hacía el norte.

Nivel 3: Estamos en las condiciones sensoriales del nivel 2, pero ahora en el mapa puede haber aldeanos.

Nivel 4: Estamos en las condiciones sensoriales del nivel 2, pero ahora puede haber aldeanos y lobos.

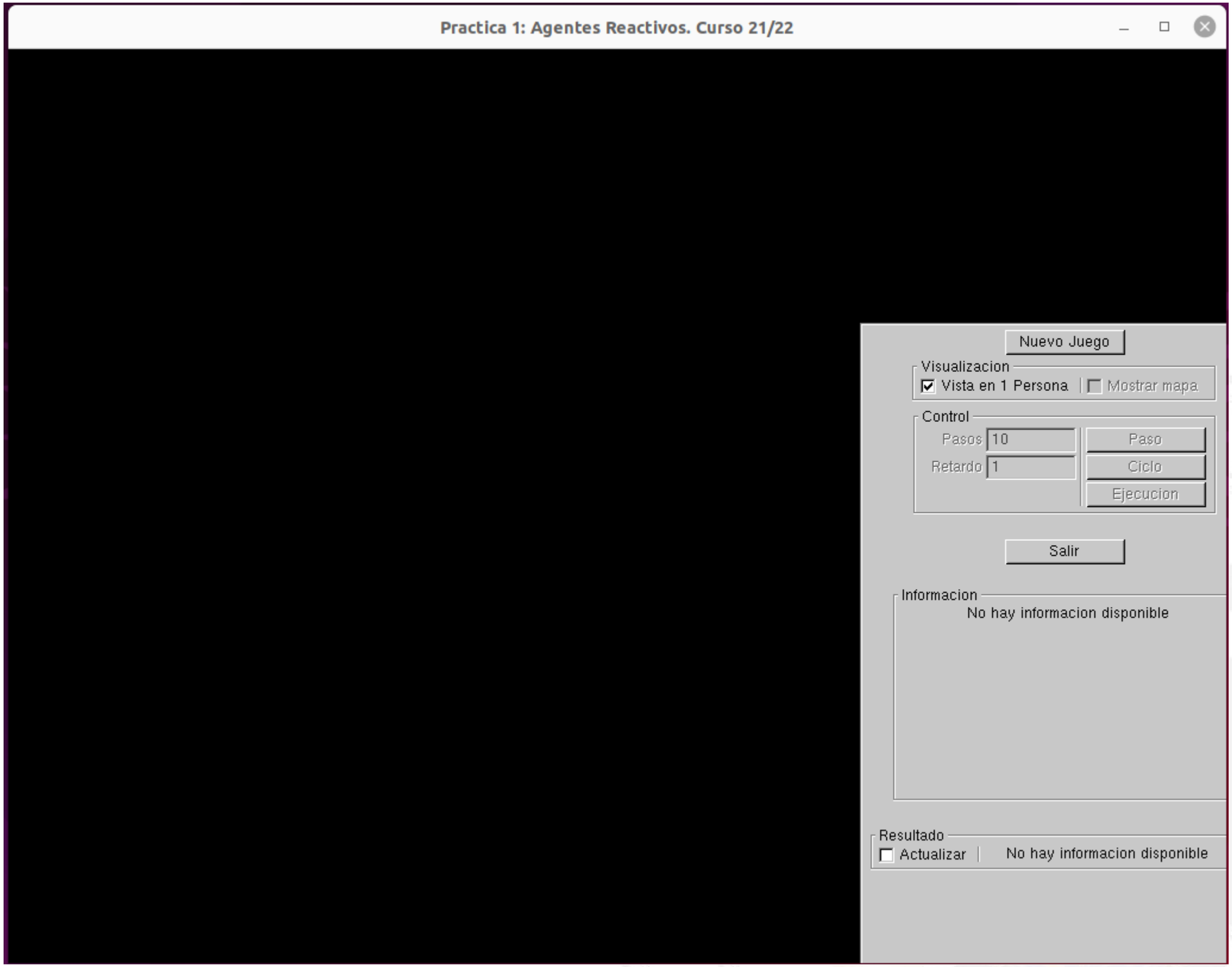
Se proporciona el software para el sistema operativo Linux en el repositorio de GitHub https://github.com/ugr-ccia-IA/practica1

- 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 detalles las variables necesarias para su desarrollo se encuentra en el guion de prácticas asociado a esta presentación y en el repositorio de GitHub (en la parte de instalación).

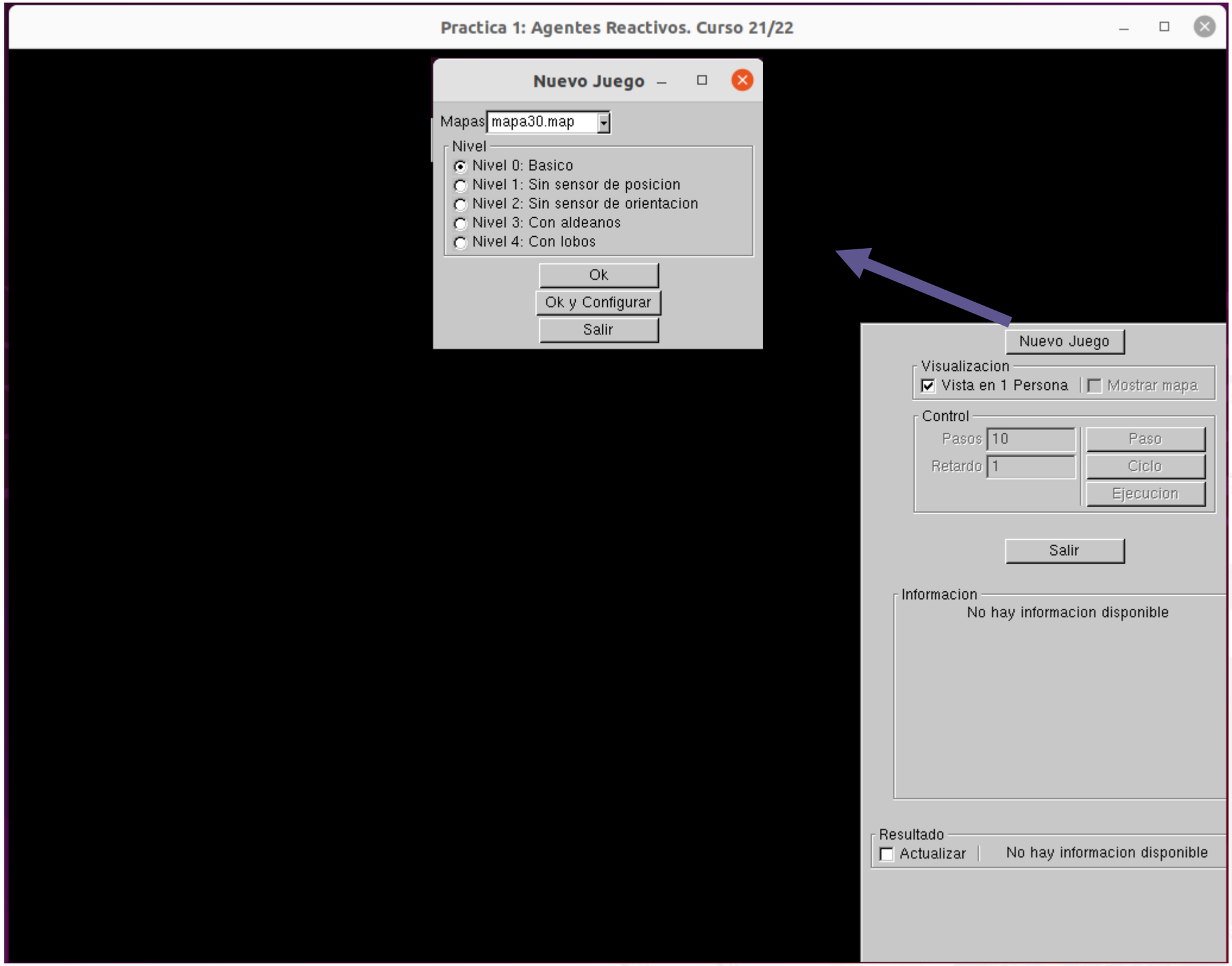
Práctica 1

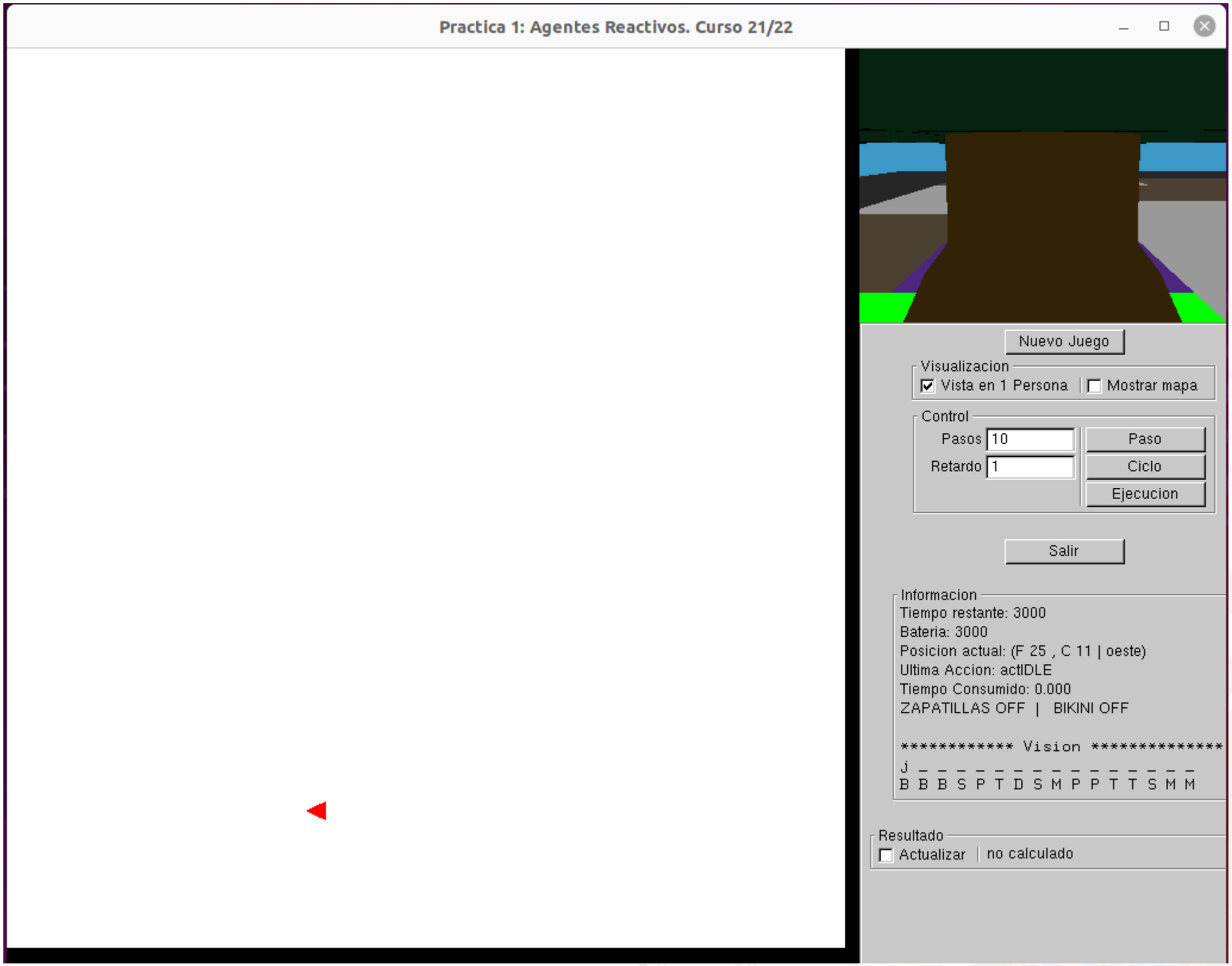
El software: ./practica1



Práctica 1

El software: ./practica1




Practica 1: Agentes Reactivos. Curso 21/22
Nuevo Juego
Visualizacion
Vista en 1 Persona
Mostrar mapa
Control
Pasos 10
Retardo 1
Paso
Ciclo
Ejecucion
Salir
Informacion
Tiempo restante: 3000
Bateria: 3000
Posicion actual: (F 25 , C 11 | oeste)
Ultima Accion: actIDLE
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/mapas30.map 1 0 4 5 1
```

- fichero de mapa
- semilla generador de números aleatorios
- Nivel (0, 1, 2, 3 o 4)
- fila origen
- columna origen
- Orientación (0=norte, 1=este, 2=sur, 3=oeste)

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 o windows. Obviamente, desde el entorno CodeBlocks también se puede hacer esta ejecución usando las ventanas.

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/mapas30.map 1 0 4 5 1
```

- **Se puede usar una versión combinada del sistema batch con el entorno gráfico.**

- **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.**

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

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

- El restos de 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**

```
#ifndef COMPORTAMIENTOJUGADOR_H
#define COMPORTAMIENTOJUGADOR_H

#include "comportamientos/comportamiento.hpp"
using namespace std;

class ComportamientoJugador : public Comportamiento{

  public:
    ComportamientoJugador(unsigned int size) : Comportamiento(size){
      // Constructor de la clase
      // Dar el valor inicial a las variables de estado
    }

    ComportamientoJugador(const ComportamientoJugador & comport) : Comportamiento(comport){}
    ~ComportamientoJugador(){}

    Action think(Sensores sensores);
    int interact(Action accion, int valor);

  private:

  // Declarar aquí las variables de estado

};
#endif      You, 2 weeks ago • initial commit
```

Constructor de Clase

Fichero **jugador.hpp**

```
#ifndef COMPORTAMIENTOJUGADOR_H
#define COMPORTAMIENTOJUGADOR_H

#include "comportamientos/comportamiento.hpp"
using namespace std;

class ComportamientoJugador : public Comportamiento{

  public:
    ComportamientoJugador(unsigned int size) : Comportamiento(size){
      // Constructor de la clase
      // Dar el valor inicial a las variables de estado
    }

    ComportamientoJugador(const ComportamientoJugador & comport) : Comportamiento(comport){}
    ~ComportamientoJugador(){}

    Action think(Sensores sensores);
    int interact(Action accion, int valor);

  private:

  // Declarar aquí las variables de estado

};
#endif      You, 2 weeks ago • initial commit
```

Constructor de Copia

Fichero **jugador.hpp**

```
#ifndef COMPORTAMIENTOJUGADOR_H
#define COMPORTAMIENTOJUGADOR_H

#include "comportamientos/comportamiento.hpp"
using namespace std;

class ComportamientoJugador : public Comportamiento{

  public:
    ComportamientoJugador(unsigned int size) : Comportamiento(size){
      // Constructor de la clase
      // Dar el valor inicial a las variables de estado
    }

    ComportamientoJugador(const ComportamientoJugador & comport) : Comportamiento(comport){}
    ~ComportamientoJugador(){}

    Action think(Sensores sensores);
    int interact(Action accion, int valor);

  private:

  // Declarar aquí las variables de estado

};
#endif      You, 2 weeks ago • initial commit
```

Destructor de la clase

Fichero **jugador.hpp**

```
#ifndef COMPORTAMIENTOJUGADOR_H
#define COMPORTAMIENTOJUGADOR_H

#include "comportamientos/comportamiento.hpp"
using namespace std;

class ComportamientoJugador : public Comportamiento{

  public:
    ComportamientoJugador(unsigned int size) : Comportamiento(size){
      // Constructor de la clase
      // Dar el valor inicial a las variables de estado
    }

    ComportamientoJugador(const ComportamientoJugador & comport) : Comportamiento(comport){}
    ~ComportamientoJugador(){}

    Action think(Sensores sensores);
    int interact(Action accion, int valor);

  private:

  // Declarar aquí las variables de estado

};
#endif      You, 2 weeks ago • initial commit
```

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

Fichero **jugador.cpp**

```cpp
#include "../Comportamientos_Jugador/jugador.hpp"
#include <iostream>
using namespace std;



Action ComportamientoJugador::think(Sensores sensores){

    Action accion = actIDLE;

    cout << "Posicion: fila " << sensores.posF << " columna " << sensores.posC << " ";
    switch(sensores.sentido){
        case 0: cout << "Norte" << endl; break;
        case 1: cout << "Este" << endl; break;
        case 2: cout << "Sur " << endl; break;
        case 3: cout << "Oeste" << endl; break;
    }
    cout << "Terreno: ";
    for (int i=0; i<sensores.terreno.size(); i++)
        cout << sensores.terreno[i];
    cout << endl;

    cout << "Superficie: ";
    for (int i=0; i<sensores.superficie.size(); i++)
        cout << sensores.superficie[i];
    cout << endl;

    cout << "Colisión: " << sensores.colision << endl;
    cout << "Reset: " << sensores.reset << endl;
    cout << "Vida: " << sensores.vida << endl;
    cout << endl;


    // Determinar el efecto de la ultima accion enviada
    return accion;
}

int ComportamientoJugador::interact(Action accion, int valor){
  return false;
}
```

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.

- 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 entregarse mediante la plataforma web de la asignatura, en un fichero ZIP (de nombre “practica1.zip” que no contenga carpetas ni sub-carpetas).

- El archivo ZIP deberá contener solo 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.

- 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 5 niveles.
  - Sobre cada nivel y mapa se realizará una simulación, y llamaremos ***pd*** al valor devuelto por el software de porcentaje de mapa descubierto.
  - A partir de ***pd***, se obtendrá s_i de valoración del mapa explorado que se define como ***0*** si ***pd*** < 10 y como ***min(100,pd+30)*** en otro caso.

- La nota asociada a cada mapa se calcula como:

$$\textbf{Nota} = s_0\, p_0 + s_1\, p_1 + s_2\, p_2 + s_3\, p_3 + s_4\, p_4$$

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

Nivel	0	1	2	3	4
Puntuación	1	2	3	2	2

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

Fecha límite entrega	Cuestionario de autoevaluación
4 de Abril hasta las 23:00 horas	Disponible desde el 5 de Abril al 7 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.