Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Inteligencia Artificial 1
Ing. Luis Fernando Espino
Tutor Académico: Erick Sandoval

Proyecto Fase 1

Grupo 10

Integrantes

César André Ramírez Dávila	**202010816**
Angel Francisco Sique Santos	**202012039**
José Manuel Ibarra Pirir	**202001800**
Edgardo Andrés Nil Guzmán	**201801119**
Heidy Beatríz Miranda Gámez	**201709051**

Objetivos

Objetivo General

Desarrollar un sistema integrado que combine la función de Tau Prolog para la toma de decisiones estratégicas y Robot.js para la ejecución automatizada de acciones en el navegador, optimizando el desempeño en el juego Battleship Game.

Objetivos Específicos

- Integrar Tau Prolog y Robot.js de manera eficiente, permitiendo la automatización fluida de las decisiones lógicas y su ejecución en el entorno del navegador.
- Desarrollar una automatización de colocación estratégica de piezas, utilizando datos de análisis anteriores para determinar las posiciones con mayor tasa de éxito, basado en estadísticas y patrones de victorias anteriores.
- Implementar un sistema de toma de decisiones optimizado que evalúe dinámicamente las casillas del tablero y seleccione las mejores opciones durante cada turno del juego, incrementando la probabilidad de acertar los barcos del oponente.
- Automatizar las acciones en la interfaz del navegador, asegurando una interacción precisa y eficiente con los elementos del juego, como el reinicio de partidas, la colocación de piezas y la selección de casillas.
- Sincronizar las decisiones de Tau Prolog con las acciones de Robot.js, garantizando una respuesta rápida y coherente entre la lógica del juego y su ejecución en tiempo real.

Uso de tecnologías

Para este proyecto se hace uso de tecnologías como `Robot.js` y `Tau Prolog`

Pasos para el uso de tecnologías

Robot.js

Actualmente si se intenta instalar desde `npm install robotjs` contiene el siguiente error:

```
npm WARN deprecated npmlog@4.1.2: This package is no longer supported.
npm WARN deprecated are-we-there-yet@1.1.7: This package is no longer
supported.
npm WARN deprecated gauge@2.7.4: This package is no longer supported.
```

Por lo que se hace uso de un fork con autor `hurdlegroup`

```
npm install @hurdlegroup/robotjs
```

Un ejemplo básico de uso es el siguiente:

```
const robot = require('@hurdlegroup/robotjs');
robot.moveMouse(100, 200); // Mueve el mouse a la posición X:100, Y:200
robot.mouseClick();         // Hace clic en la posición actual del mouse
```

Tau Prolog

Para este caso si se puede instalar normal desde `npm`

```
npm install tau-prolog
```

Ejemplo básico de una consulta

```
const pl = require("tau-prolog");
const session = pl.create();
session.consult('nombre_archivo.pl', {
```

```
    success: function() {
      session.query('decision_estrategica(X).');
    }
});
```

Uso de las herramientas juntas

Una vez configuradas las herramientas, el sistema puede ejecutarse con los siguientes pasos:

- Iniciar el script principal en robot.js que integra tanto la toma de decisiones con Tau Prolog como la automatización.
- Verificar que las acciones en el navegador son correctas, como la colocación de las piezas y las selecciones durante el juego.

Métodos utilizados

Archivo js

Para el archivo `Battleship.js` los métodos utilizados son los siguientes:

Importaciones

```
const robot = require('@hurdlegroup/robotjs');
const notifier = require('node-notifier');
const fs = require('fs');
const pl = require('tau-prolog');
require('tau-prolog/modules/js');
```

- robot.js: Permite controlar el ratón y el teclado.
- node-notifier: Usado para mostrar notificaciones en el escritorio.
- fs: Módulo de Node.js para manipulación de archivos.
- tau-prolog: Proporciona un entorno Prolog para ejecutar consultas y lógica.

Variables globales

```
let session = pl.create();
let coordenadas = [];
let posicionesMap = {};
const posicionesReset = {};
const prologFilePath = './Juego.pl';
const posicionesFilePath = './posiciones.txt';
const posicionesresetFilePath = './reset_posiciones.txt';
let posicionesDisparadas = [];
let is_target = false;

let contadorBarco = 0;
let pendientesAdyacentes = [];
```

- session: Instancia de Prolog.
- coordenadas: Almacena las coordenadas de los barcos.
- posicionesMap: Mapa de posiciones cargadas desde archivos.
- posicionesReset: Posiciones de barcos a resetear.

- posicionesDisparadas: Registra las posiciones ya disparadas.
- is_target: Indica si hay un objetivo activo.
- contadorBarco: Contador del tamaño del barco actualmente en juego.
- pendientesAdyacentes: Almacena las celdas adyacentes a disparar.

Función Principal: ColocarBarcos

```
        async function ColocarBarcosFinal() {
    console.log("Iniciando ColocarBarcosFinal...");

    // Mueve el mouse a la posición del botón de reset
    console.log("Esperando 3 segundos antes de mover el mouse...");
    await esperarSegundos(3);
    robot.moveMouse(610, 590);
    console.log("Mouse movido a la posición del botón de reset.");

    // Hace clic para abrir la página
    console.log("Esperando 1 segundo antes de hacer clic...");
    await esperarSegundos(1);
    robot.mouseClick();
    console.log("Clic realizado.");

    // Cargar el archivo Prolog después de 2 segundos
    console.log("Esperando 2 segundos antes de cargar el archivo
Prolog...");
    await esperarSegundos(2);

    cargarArchivoProlog();

    console.log("Archivo Prolog cargado y proceso iniciado.");
}
```

Descripción

Esta función inicia el proceso de colocación de barcos, mueve el ratón a la posición del botón de reinicio en la interfaz del juego, y posteriormente carga el archivo Prolog que contiene la lógica del juego.

Cargar Posiciones desde Archivos

```
function cargarPosiciones() {
    fs.readFile(posicionesFilePath, 'utf8', (err, data) => {
        if (err) {
            console.error("Error al leer el archivo de posiciones:", err);
            return;
        }
        const lineas = data.split('\n');
        lineas.forEach(linea => {
            const match = linea.match(/([A-Z]\d+)\s+\(X:\s*(\d+),\s*Y:\s*
(\d+)\)/);
            if (match) {
                const nombre = match[1];
                const x = parseInt(match[2], 10);
                const y = parseInt(match[3], 10);
                posicionesMap[nombre] = { x, y };
            }
        });
        ColocarBarcosFinal();
    });
}
```

Descripción

Esta función carga las posiciones de los barcos desde un archivo de texto y las almacena en posicionesMap. Al finalizar la carga, se inicia la colocación de barcos.

Mover y Colocar Barcos

```
async function moverBarco(origen, destino, orientacion) {
    const { x: origenX, y: origenY } = origen;
    const { x: destinoX, y: destinoY } = destino;

    robot.moveMouse(origenX, origenY);
    await esperarSegundos(2);

    robot.mouseToggle("down");
    await esperarSegundos(1);
    robot.dragMouse(destinoX, destinoY);
    robot.mouseToggle("up");
    await esperarSegundos(1);

    if (orientacion === 'V') {
```

```
            robot.moveMouse(destinoX, destinoY);
            await esperarSegundos(1);
            robot.mouseClick();
        }

        await esperarSegundos(1);
    }
```

Descripción

Esta función mueve el barco desde su posición de origen a un destino especificado, manejando la orientación del barco (vertical u horizontal).

Lógica del Juego

```
function iniciar() {
    const estado = detectarMensaje();

    if (estado === "Ataque") {
        if (is_target) {
            manejarObjetivo();
        } else {
            dispararSiguienteCoordenada();
        }
        is_target = true;
    }
}
```

La función `iniciar` verifica el estado del juego. Si es el turno de atacar, determina si hay un objetivo activo y maneja la lógica de disparo correspondiente.

Archivo Tau Prolog

Para el archivo `Juego.pl` los métodos utilizados son los siguientes:

Reglas para la colocación de barcos

En el tablero del juego, los barcos están colocados de acuerdo a sus dimensiones (una, dos, tres o cuatro celdas) y su orientación (horizontal o vertical). Las reglas definidas son las siguientes:

- Barcos de una celda: Estos barcos ocupan una sola casilla en el tablero y se definen mediante los predicados barco_una_celda. Las posiciones y orientaciones de estos barcos son las siguientes:
- barco_una_celda('A1', 'H').
- barco_una_celda('B10', 'H').
- barco_una_celda('C2', 'H').
- barco_una_celda('J10', 'H').

- Barcos de dos celdas: Ocupan dos casillas contiguas y se definen con el predicado barco_dos_celdas:
- barco_dos_celdas('E1', 'H').
- barco_dos_celdas('A8', 'H').
- barco_dos_celdas('H2', 'H').

- Barcos de tres celdas: Tienen tres casillas y están definidos por el predicado barco_tres_celdas:
- barco_tres_celdas('D4', 'H').
- barco_tres_celdas('H6', 'V').

- Barcos de cuatro celdas: Son los barcos más grandes y se definen a través del predicado barco_cuatro_celdas:
- barco_cuatro_celdas('F6', 'V').

Reglas para determinar posiciones de alto potencial de ataque

Los siguientes predicados indican las posiciones en el tablero que se consideran más probables de contener un barco enemigo:

- Primer Haunting: Este conjunto de posiciones se representa mediante el predicado primer_haunting, el cual incluye posiciones como:
- primer_haunting('A1').
- primer_haunting('A3').
- primer_haunting('E9').

Estas posiciones son elegidas estratégicamente para maximizar las posibilidades de acertar durante el juego.

- Segundo Haunting: Similar al primero, este conjunto de posiciones es definido por el predicado segundo_haunting y se considera que son posiciones menos probables, pero todavía relevantes para la estrategia de ataque:
- segundo_haunting('A2').
- segundo_haunting('A4').
- segundo_haunting('E10').

Estas reglas permiten a los jugadores formular estrategias de ataque al identificar áreas del tablero con alta probabilidad de contener barcos.

Archivos creados

Los archivos que se usaron para el desarrollo del proyecto son los siguientes:

- Battleship.js
- Juego.pl

1. Archivo de js

- Propósito: Este archivo contiene el código JavaScript responsable de la interacción automatizada con la página web del juego Battleship Game. Utiliza la biblioteca robot.js para simular acciones de usuario, como mover el ratón, hacer clic en celdas específicas y leer la matriz de posiciones en el tablero del juego. También maneja la lógica para identificar cuándo es el turno del jugador en base a los mensajes que aparecen en la interfaz del juego.
- Principales funciones y responsabilidades:
  - Lectura de la matriz de posiciones: El archivo contiene un método para identificar y leer las posiciones actuales de los barcos y las casillas seleccionadas en el juego.
  - Colocación automática de barcos: Utilizando la información de destinos obtenida desde el archivo Prolog (tau-prolog.pl), el script de robot.js mueve el ratón y coloca los barcos en las coordenadas calculadas por el algoritmo en Prolog.
  - Manejo de las decisiones de "haunting": En el juego, el algoritmo Prolog decide qué celdas tienen mayor probabilidad de tener un barco enemigo. robot.js ejecuta esas decisiones moviendo el ratón y haciendo clic en dichas celdas.
  - Gestión de turnos: Se detecta si es el turno del jugador basándose en mensajes visibles en la página web, como "Your turn" o "Opponent's turn", lo que permite al sistema tomar decisiones automáticamente durante el turno del jugador.
  - Interacción con Tau Prolog: robot.js también se encarga de consultar los predicados y reglas definidas en el archivo Prolog para recibir las mejores decisiones posibles durante el juego.

2. Archivo de Tau prolog

- Propósito: Este archivo contiene las reglas y hechos en Prolog que permiten realizar la toma de decisiones lógica del juego. Específicamente, contiene las reglas para la colocación de barcos y para seleccionar las celdas con mayor probabilidad de acierto durante el juego, basándose en la estrategia del "primer haunting" y "segundo haunting".
- Principales funciones y responsabilidades:
  - Colocación de barcos: El archivo define reglas para la colocación de los barcos en el tablero según el tamaño de los mismos. Cada barco está asociado a una celda inicial y una orientación (`H` para horizontal, `V` para vertical), lo que es utilizado por el sistema automatizado para realizar la colocación inicial.

- Decisiones de "haunting: Se implementan reglas que determinan las celdas que tienen mayor probabilidad de contener barcos enemigos. El "primer haunting" define las primeras posiciones estratégicas a atacar, y el "segundo haunting" se activa cuando no se ha acertado un barco durante las primeras fases del juego.
- Interacción con robot.js: Las decisiones lógicas tomadas por Prolog son enviadas a robot.js, que ejecuta las acciones correspondientes en la interfaz gráfica del juego. Por ejemplo, Prolog selecciona una casilla con alta probabilidad de éxito, y robot.js se encarga de mover el ratón y hacer clic en esa casilla.

Relación entre archivos

- robot.js actúa como el ejecutor de las decisiones que tau-prolog.pl toma. Mientras que el archivo Prolog contiene toda la lógica y las estrategias del juego, robot.js se encarga de realizar esas acciones en el juego, como colocar barcos o seleccionar casillas.
- Juntos, estos archivos permiten la automatización completa de la partida en Battleship Game, desde la colocación inicial de los barcos hasta la elección de las celdas a atacar basándose en probabilidades calculadas por el sistema.