



MARATON DE ROBOTS
Proyecto Practico De Inteligencia Artificial

Mario German Baquero Cedeño
201667471-3743
mario.baquero@correounivalle.edu.co

Andrés Felipe González Rojas
201759599
aadres.gonzalez.rojas@correounivalle.edu.co

Andrés Felipe Medina Tascón
201667602-3743
andres.medina@correounivalle.edu.co

Oscar Alexander Ruiz Palacio
201667600-3743
ruiz.oscar@correounivalle.edu.co

Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación
Programa Académico de Ingeniería de Sistemas
Tuluá, Abril 23 del 2018

Resumen. Este documento presenta el análisis y diseño del primer proyecto práctico del curso de Introducción a la Inteligencia Artificial, contiene la definición del problema y abordaje del mismo, donde se presenta como se implementó el algoritmo de búsqueda informada, se analiza la heurística implementada, se muestran pruebas realizadas a la implementación y el análisis de los resultados, finalizando con las conclusiones del proyecto.

Palabras clave. Búsqueda informada, Heurística, Inteligencia Artificial.

A. Introducción

En esta maratón de robots nos encontramos con 4 prototipos de robots que son la piedra, papel, tijera y pistola. El ambiente de la maratón se representa con una matriz de enteros de tamaño $4 \times n$, donde $n \in [1, 50]$, se presenta un cuadro con el elemento, el valor en la matriz y la imagen que lo representara en la siguiente tabla:

Elemento	Representación	Imagen
Espacio	0	
Piedra	1	
Papel	2	
Tijera	3	
Pistola	4	
Vegetal	5	
Ladrón	6	
Hilarante	7	
Blanco	8	

Cada robot es un prototipo y siempre se mueven de izquierda a derecha en una unidad de movimiento por cada unidad de tiempo.

Cuando los robots se encuentran frente a un obstáculo, se debe decidir cuál robot se quedara enfrentándolos mientras los demás robots saltan el obstáculo.

Cada robot tiene un enemigo contra el que tomara venganza si logra llegar hasta donde se encuentre este. Esto es:

- Si la piedra llega a una posición donde el papel se encuentra peleando, entonces la piedra perjudica al papel.
- Si el papel llega a una posición donde la tijera se encuentra peleando, entonces el papel perjudica a la tijera.
- Si la tijera llega a una posición donde la piedra se encuentra peleando, entonces la tijera perjudica a la piedra,
- Si la pistola llega a una posición donde se encuentra la piedra, el papel o la tijera peleando, entonces la pistola perjudica a cualquiera de estos.



En caso contrario, se ayudan.

También, los robots tienen un costo de pelea contra cada rival que se representa en la siguiente tabla:

Robot	Costo	Contra	Costo	Contra
	1		2	
	1		2	
	1		2	
	1		2	

La aplicación debe leer un archivo de entrada con el siguiente formato que representa el tablero.

Ejemplo:

Datos en el archivo:

10050060070080

20050060070080

30050060070080

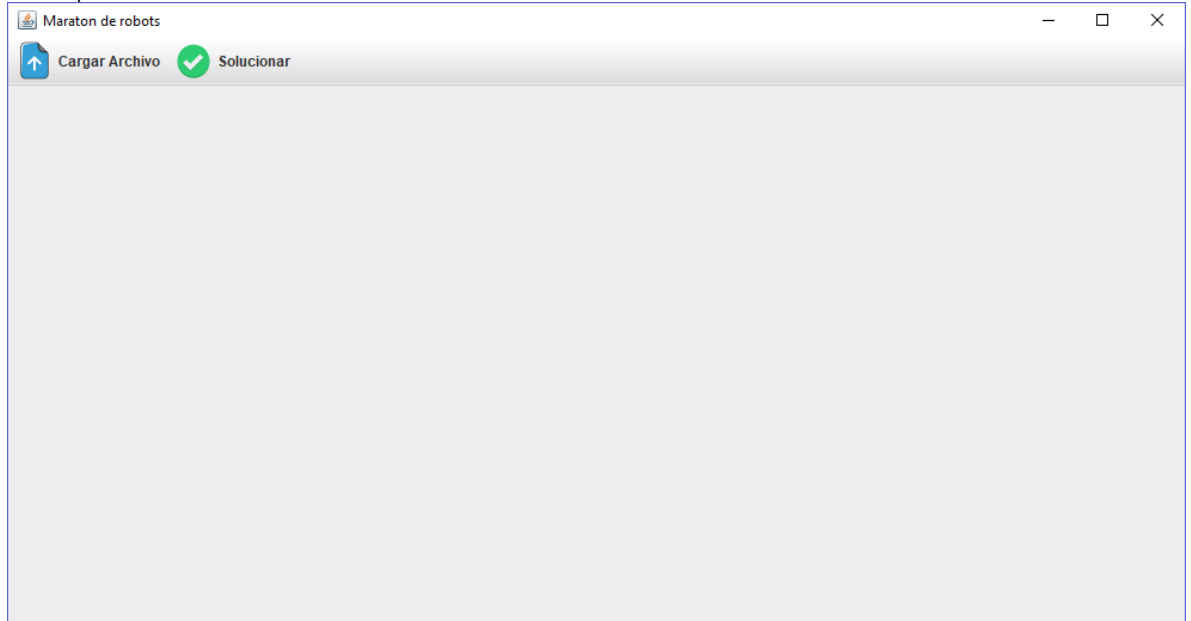
40050060070080

B. Abordaje del problema

Se abordara el problema utilizando el tipo de búsqueda informada, implementando la búsqueda A* con una heurística que se explicara más adelante.

1. Descripción de la Interfaz

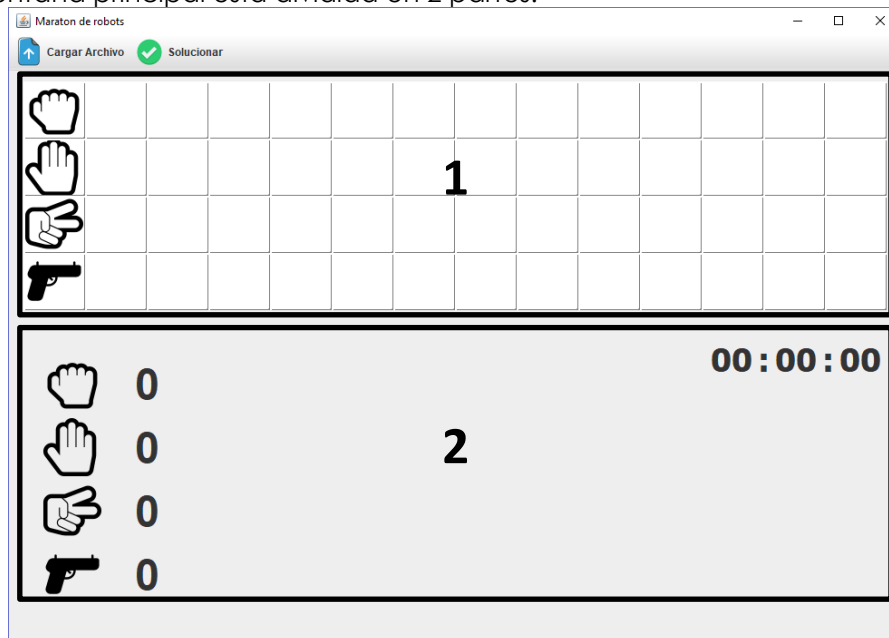
La implementación se inicial con una ventana inicial:



Esta tiene una barra de herramientas en donde se puede elegir el archivo que generara el ambiente de la maratón.

El botón de dar solución al problema se habilitara cuando se haya cargado un archivo.

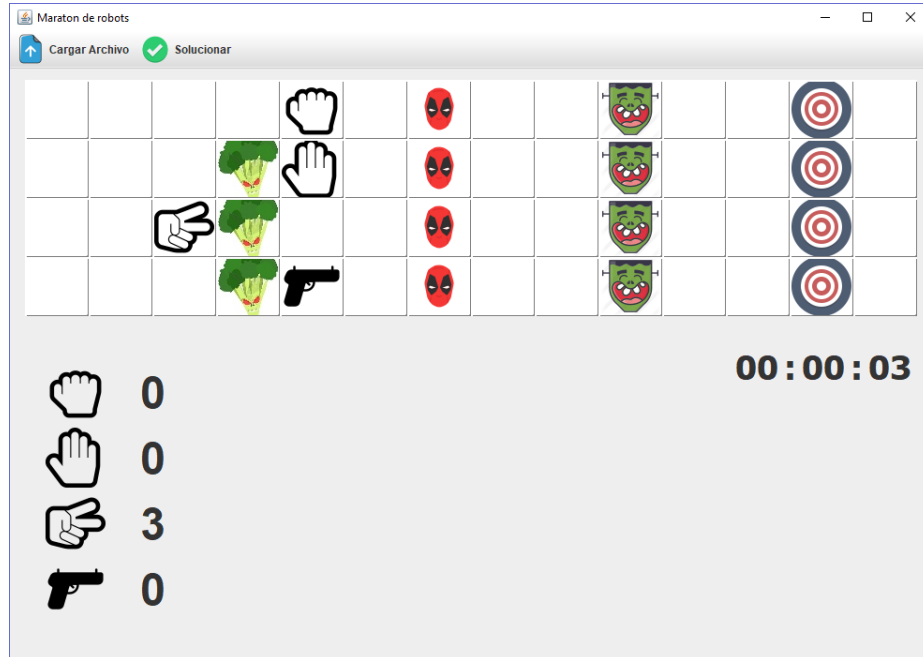
La ventana principal está dividida en 2 partes:



1. **Zona del tablero:** En esta zona se mostrara el tablero de la maratón que se genera según el archivo cargado.

2. **Zona Información:** En esta zona se mostrara el tiempo transcurrido en la maratón y el tiempo que le tardara a cada robot en vencer a sus enemigos.

Así se verá la interfaz cuando hayamos elegido un archivo y un robot este peleando:



2. Estructura de la implementación

El proyecto está dividido en 3 paquetes principales Images, Clases y MaratonIA, a continuación se presenta que contiene cada paquete;

- **Images:** Contiene las imágenes del menú, los robots y los enemigos.
- **Clases:** Este paquete contiene las clases que maneja las cualidades de los robots, sus acciones y la estructura del árbol.
- **MaratonIA:** Este paquete contiene las clases que se encargan de realizar la lectura del archivo de texto plano, la lógica del programa, mostrar la interfaz, y generar los nodos:
 - **MaratonIA:** Esta es la clase principal del programa, desde acá arranca la aplicación.
 - **UIAplicacion:** Esta clase se encarga de crear la interfaz gráfica del programa.
 - **CargarArchivo:** Esta clase se encarga de realizar la carga del archivo de texto plano para que sea visualizado en la interfaz.
 - **AdministrarMaraton:** Esta clase se encarga de realizar toda la logia del desarrollo de la maratón, consta de los siguientes métodos:
 - **AdministrarMaraton:** Es el método constructor de la clase, desde acá se inicializa cada robot con sus atributos de id, posición, costo, fuerza, tiempo que le tarde en derrotar a

todos los enemigos que tenga al frente, $posenemigo$ y $tiempoenemigo$ que es el tiempo que le cuesta en derrotar al enemigo contra el que se esté enfrentando.

También se encarga de inicializar la raíz del árbol el cual contiene un id, un nodo que contiene el tablero actual junto con los robots y los enemigos, un id referente al padre, la heurística, el costo y la suma de estos dos y un atributo booleano para saber si ha sido expandido o no.

Cuenta con un hilo que es el encargado de la ejecución de la maratón, dentro de este hilo se realiza el llamado a los métodos `moverRobot`, `peleaRobot`, `elegirRobot` y `mostrarTablero`, también la creación de nodos cuando lo requiera y la selección del nodo con menor costo más heurística ($g+h$).

- **moverRobot:** recibe un parámetro del tipo Robot el cual avanzara.
 - **peleaRobot:** recibe un parámetro del tipo Robot el cual se queda enfrentando a un grupo de enemigos.
 - **elegirRobot:** recibe un array de robots y un array de enemigos. En este método se calcula el menor tiempo en derrotar a los enemigos de los posibles robots candidatos a enfrentarlos, seleccionando a un solo robot para que los enfrente, este robot elegido se envía al método `peleaRobot` y los demás robots van al método `moverRobot`.
 - **mostrarTablero:** Este método actualiza constantemente el tablero que se muestra en pantalla.
 - **animacion:** Este método se encarga de ejecutar el hilo principal del programa.
- **AdministrarArbol:** Esta clase se encarga de realizar toda la logia de la creación de nodos y cálculo de costo y heurística, consta de los siguientes métodos:
- **crearNodo:** Se encarga de crear un nodo, el cual recibe el id que identificara al nodo, el nodo del padre del cual se extiende, un array de enemigos, un array de robots candidatos a enfrentarse a los enemigos, un array con el resto de robots y un id del robot que se quedara peleando en dicho nodo.
 - **moverRobot:** recibe un parámetro del tipo Robot que se moverá en dicho nodo y el tablero del nodo.
 - **peleaRobot:** recibe un parámetro del tipo Robot el cual se queda enfrentando a un grupo de enemigos en dicho nodo, y el tablero del nodo.
 - **calcularCosto:** Calcula el costo del nodo dependiendo del tiempo de pelea de todos los robots.
 - **calcularHeuristica:** Calcula la heurística del nodo dependiendo de la distancia de cada robot a la meta,

La selección del nodo con menor costo+heurística para trabajar sobre el cómo nodo raíz, se realiza al comienzo de cada ciclo del hilo que se ejecuta en la clase `AdministrarMaraton` del paquete `MaratonIA`.

3. Algoritmo de Búsqueda Informada

El algoritmo de búsqueda informada que se implementó fue la búsqueda A*, la cual consiste en principio de expandir el nodo con menor costo como la búsqueda por costo uniforme, pero en este caso se tiene en cuenta una heurística, la cual es una función que asigna a cada nodo un valor que corresponde al costo estimado de llegar a la meta estando en dicho nodo, en caso de la búsqueda A* no se usa solo el costo o la heurística, para elegir que nodo expandir se suma la heurística y el costo del nodo, dando como resultado el costo total $f(x) = g(x) + h(x)$.

Heurística: La heurística implementada consiste en el promedio del producto de la distancia de cada robot a la meta, esto es:

$$h = \frac{1}{4} \prod_{i=1}^4 (n - x_i)$$
, donde x_i es la posición actual del robot y n es el tamaño del tablero.

Esta heurística es admisible ya que cuando un robot llegue a la meta, su distancia respecto a la meta es 0, esto hará que el producto de las distancia de cada robot a la meta de como resultado 0 y por lo tanto nuestra heurística será 0.

C. Pruebas y Análisis de Resultados

Para hacer las pruebas se crearon 4 tableros al dado en el documento del proyecto y se analizaron los resultados dados en cada mapa:

1. Tablero 1



Nodos creados: 11

Nodos expandidos: 4

Costo total: 4

















Heurística total: 4

Total: 8

Ruta: 0 → 3 → 5 → 8 → 10

En este mapa no se presenta el caso de volver a un nodo anterior con menor costo+heurística.

2. Tablero 2

Nodos creados: 23

Nodos expandidos: 8

Costo total: 12





























Heurística total: 0

Total: 12

Ruta: 0 → 3 → 6 → 17

En este mapa se devuelve en total 7 veces hacia un nodo con menor costo+heurística.

3. Tablero 3

Nodos creados: 19

Nodos expandidos: 7

Costo total: 6

Heurística total: 0

Total: 6

Ruta: 0 → 3 → 5 → 8 → 10 → 14 → 16

En este mapa se devuelve en total 3 veces hacia un nodo con menor costo+heurística.

D. Conclusiones

El algoritmo A* alcanza una respuesta optima del problema a diferencia de otros tipos de búsqueda, ya que expande y crea menos nodos que las demás búsquedas, pero no tiene en cuenta los obstáculos en el mapa, como lo son los enemigos.

La heurística que tenemos es admisible sin embargo no es la óptima porque no se toma en cuenta los enemigos que pueden haber en el recorrido de cada robot y no se maneja bien para el tema de la ayuda y la venganza.

E. Referencias

S. Inform, "Inteligencia Artificial," *Inform*, 2010.