Instituto Tecnológico de Costa Rica - ITCR

Sede de Alajuela

Curso:
Lenguajes de programación IC4700

Profesora:
Samantha Ramijan Carmiol

I Semestre del 2021

Integrantes del grupo:

Jose Alexander Artavia Quesada	2015098028
Bryan Andrey Díaz Barrientos	2019264426
Josué Gerardo Gutiérrez Mora	2018300436

Diseño de la solución

Datos

A lo largo del desarrollo del proyecto se fueron usando muchas variables que cambiaban conforme el código iba cumpliendo su cometido, sin embargo, existe un conjunto de constantes que son un pilar dentro de este trabajo y fluyen a lo largo del código para hacer de este más legible y dinámico para todos los involucrados en su elaboración.

Constantes

```
#define N 7
#define VALOR_VICTORIA 1024
#define FICHA_CPU 'X'
#define FICHA_JUGADOR 'O'
```

N 7: Se usa para definir a lo largo del código el tamaño de la matriz que en este caso es 7x7, lo cual es inmutable dadas las instrucciones del proyecto.

Valor_VICTORIA 1024: Es la puntuación que se le da a un movimiento que parece prometedor para la victoria o prometedor para evitar que el contrincante gane. La elección del número 1024 es simplemente por ser un número un tanto alto y a su vez convención para este tipo de casos.

Ficha_CPU ‘X’: Es la ficha que va a dibujar el CPU en sus movidas, ya sea en el tablero de juego o en el tablero teórico. Se elige la X por ser convención en esta clase de juegos.

Ficha_JUGADOR ‘O’: Es la ficha que va a dibujar el jugador en sus movidas. Se elige la O por ser convención en esta clase de juegos.

Estructura de tipo nodo

```
typedef struct nodo{
      struct nodo **hijos;
      int *movimientos;
      int n_hijos;
      char tablero[N][N];
      double valor;
      int nivel;
} Nodo;
```

Esta estructura se encuentra conformada por los siguientes elementos:

- **Struct nodo **hijos:** Se encarga de los hijos de un nodo.

- ***movimientos:** Indica las columnas donde es posible realizar movimientos.

- **n_hijos:** Dice la cantidad de hijos que acepta la jugada (eso depende de la cantidad de columnas donde se puedan hacer jugadas).

- **tablero[N][N]:** Aquí se guarda una copia del tablero que va a ser el futuro "tablero hipotético" que ayudará a ver qué jugadas son buenas para las hojas.

- **valor:** Es el valor que se le da al nodo dependiendo de si este registra un movimiento que sirva para acercar a la victoria (o evitar que el contrincante se acerque a esta).

- **nivel:** Es el nivel en que se encuentra dentro del árbol.

Interacción con la persona usuaria

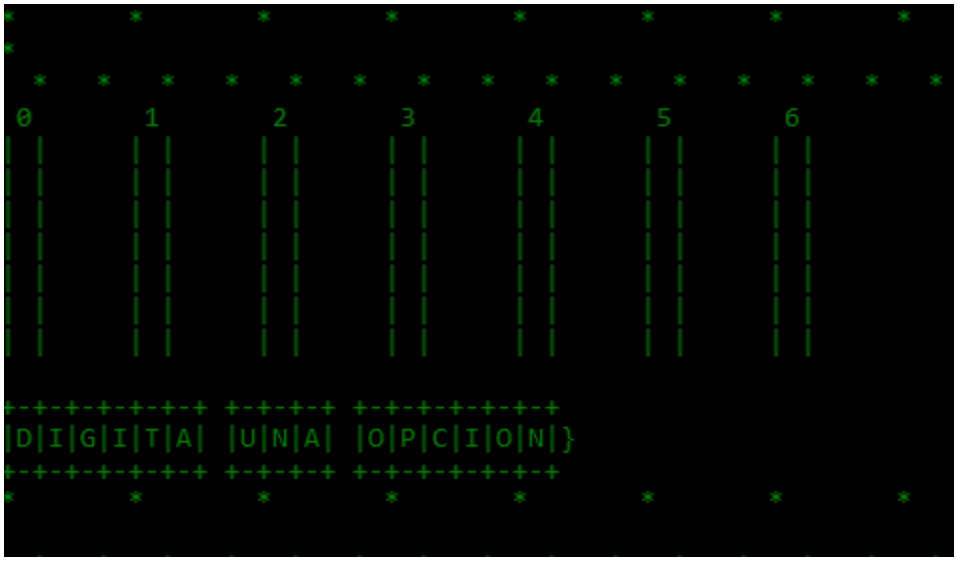
En este apartado se procede a mostrar el cómo son las entradas y salidas con las que interactúa la persona usuaria:

Primera pantalla

Al correr el programa saltará esta primera pantalla, la cual representa las opciones que se tiene para interactuar:

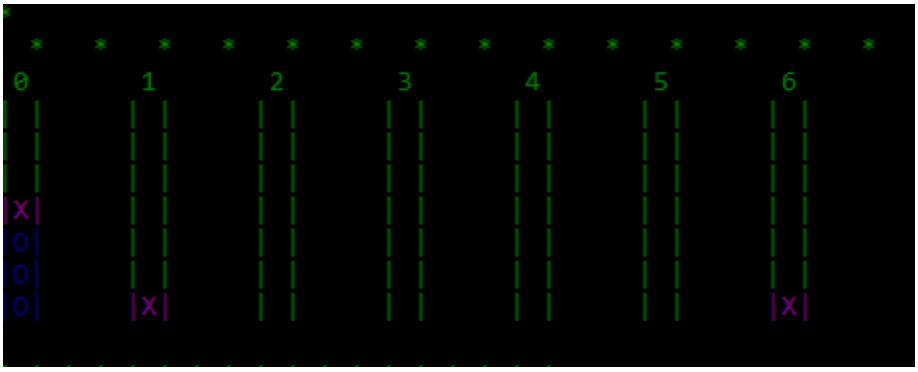
1. Al escribir el número 1 y presionar ENTER se accederá al tablero de juego.
2. Al escribir el número 2 y presionar ENTER se accede a una ventana que muestra el nombre de los programadores del juego.
3. Al escribir el número 3 y presionar ENTER se sale del juego.

Pantalla de juego



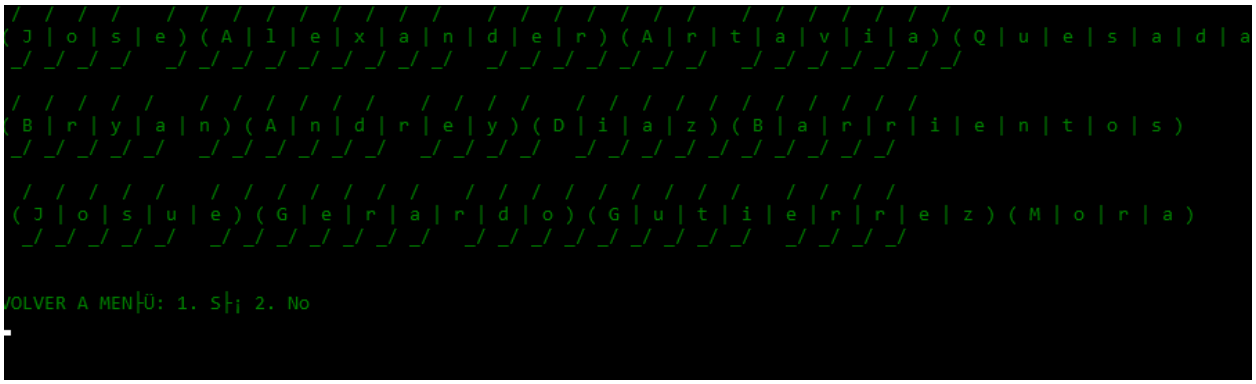
Al haber ingresado en la pantalla de juego se puede apreciar un tablero de 7 columnas, que va de la columna 0 a la columna 6, estas son las opciones que tiene el usuario para introducir cada vez que tiene el turno de la partida: **entradas de números enteros entre 0 y 6**. En caso de ingresar un valor diferente se contabilizará cómo si hubiera elegido el número 6.

Conforme avanza el juego, puede suceder que ciertas entradas entre el 0 y 6 ya no sean aceptadas debido a que la columna se haya llenado.



En esta imagen se puede ver cómo se va viendo el tablero conforme el jugador selecciona columnas donde quiere depositar su ficha.

Nombres de los programadores



Al haber seleccionado esta opción se mostrarán los nombres de los programadores de la forma en que se ve en pantalla. Para regresar basta con ingresar **1 + ENTER** y de este modo se vuelve a la pantalla de inicio o bien, si quiere salir puede presionar **2 + ENTER**.

Estructuras de datos y algoritmos implementados

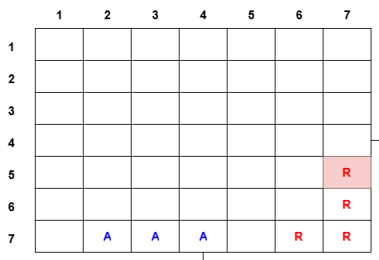
En primera instancia se debe hacer introducción al algoritmo en que se basó este proyecto y que a su vez es la “columna vertebral” que sostiene el proyecto el cual es el algoritmo Minmax, de este modo se puede proceder a entender y describir mejor más adelante el cómo es que este fue implementado en la solución del trabajo.

Algoritmo minimax: Este algoritmo recursivo, trabaja con un árbol de juego o decisiones, su objetivo es buscar la mejor jugada disponible para un jugador, suponiendo que el otro participante también realice la mejor jugada posible. Se le asigna a cada jugador un nombre, Mín y Máx. En cada nodo que se pueda terminar el juego, se calculará un valor negativo o positivo, según corresponda el turno(Min o Max) y su profundidad, se elegirá el mejor valor posible. Por ejemplo si es turno del Max, se elegirá el valor más grande disponible de los nodos.

En la siguiente imagen se presenta un caso hipotético, dado el turno de Max, este deberá elegir la mejor opción, es decir el nodo con el valor más alto, para ello se presentan dos posibles movimientos, uno le permite ganar en el siguiente turno, por lo cual sería la opción con un número más alto. Aun así, este ejemplo solo abarca un pequeño rango de posibilidades, esta idea se debe implementar de manera recursiva para que recorra todos las hojas del árbol. En resumen, el algoritmo visualiza todos los posibles movimientos hasta alcanzar un ganador o empate y selecciona la mejor opción para ganar .

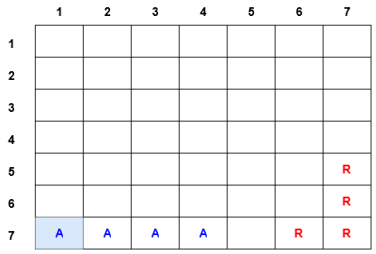
A continuación se muestran dos imágenes que hacen hincapié en cómo es que este algoritmo funciona:

0 Turno: Max

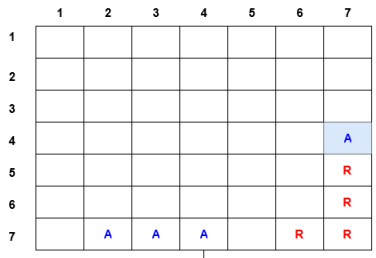


99

1 Turno: Min

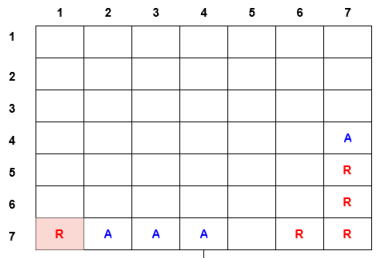


100 - 1 = 99



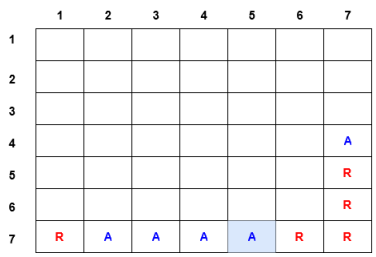
97

2 Turno: Max



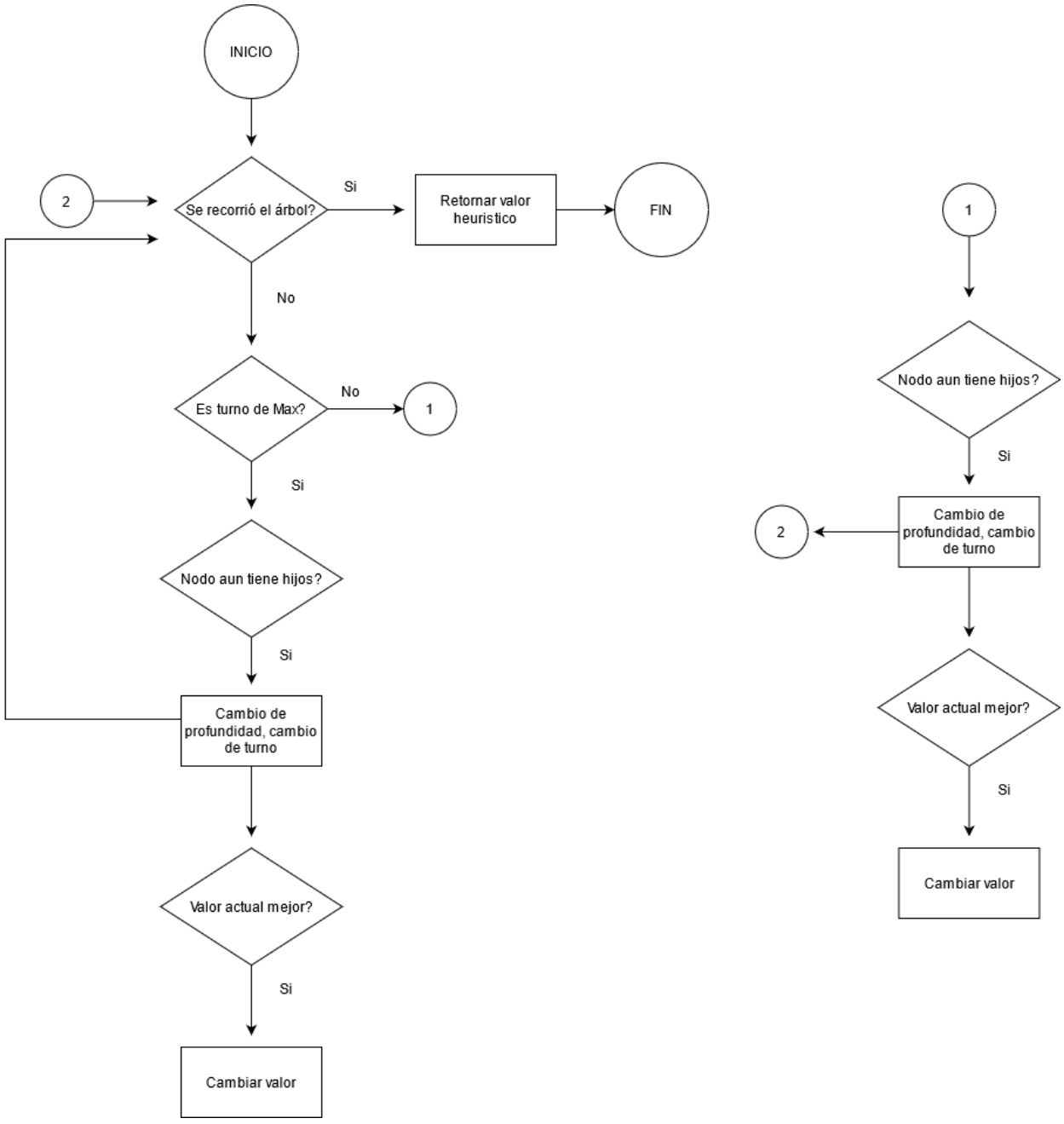
97

3 Turno: Min



100 - 3 = 97

El siguiente diagrama de flujo presenta la idea principal del algoritmo



Ahora bien, se procede a describir brevemente la estructura de datos que conforman al Minimax implementado en el trabajo:

juegaPC: Es la función que da inicio a todo el proceso de "decisión" que toma la computadora para hacer una jugada, se le carga el tablero de juego (una matriz 7x7) y un valor entero que hace referencia a la profundidad de pensamiento del árbol. De esta función se cargan ciertos elementos en el siguiente orden para poder ejecutar el árbol:

- **crearNodo:** crea un nodo raíz que será el inicio del árbol n-ario.
- **crearArbol:** Se encarga de crear el árbol a partir de un nodo raíz (se le debe de inidicar la profundidad de este, de lo contrario la memoria colapsa).
- **valorarHojas:** Evalúa las hojas del árbol creado dándoles así un valor dependiendo de su probabilidad de ganar según se mire en un "tablero hipotético" que se generó para observar las mejores combinaciones.
- **minimax:** Es la función final que le da vida al árbol, se le envía el nodo raíz de este y esta función se encarga mediante de backtracking de ir viendo la calificación de las hojas para así guardar su valor en máximos y mínimos para saber cual es la mejor jugada para un jugador y la peor para el otro.

Por último, al final se meten en una lista todas las opciones que son igual de buenas para proceder a seleccionar una de manera aleatoria y así decirle a la computadora que haga su movida.

Lo anteriormente mencionado es un resumen de cómo está estructurado el comportamiento principal de la estructura encargada de llevar a cabo el Minimax. Ahora bien, dicha estructura no funcionaría sin la ayuda de otras funciones externas a este tales cómo:

Heuristica: Es la función que ayuda a valorar las hojas del árbol definiendo en el tablero hipotético cual sería la mejor movida y dándole así un valor a la hoja.

comprobarVictoriaJugador: Es una función sencilla que se encarga de observar un tablero de NxN (en este caso 7x7) y verificar si en alguna parte **las fichas del jugador** se encuentra una línea de cuatro ya sea horizontal, vertical o diagonal. Es principalmente usada por la función

heuristica para verificar las posibles jugadas y elegir la que menos ayude al rival.

comprobarVictoriaPC: Es una función sencilla que se encarga de observar un tablero de NxN (en este caso 7x7) y verificar si en alguna parte **las fichas de la computadora** se encuentra una línea de cuatro ya sea horizontal, vertical o diagonal. Es principalmente usada por la función *heuristica* para verificar las posibles jugadas **que benefician** al ordenador y así tomar una mejor decisión para que este llegue a ganar.

crearNivel: Es la función encarga de ir generando los nodos en cada nivel del árbol.

tableroTeorico: Crea una copia de cómo va el tablero de juego, esto con el fin de ser usado para jugadas teóricas que ayuden a la máquina a ganar.

Posibilidades: Devuelve la cantidad de posibilidades (columnas con posibilidad de insertar ficha) que tiene el CPU para ejecutar una movida.

hacerMovimiento: Cómo su nombre lo indica, se encarga de hacer movimientos en el tablero, esto se limita únicamente al tablero de juego, puede hacer movimientos en el tablero teórico para ver así poder ver los posibles mejores movimientos.

Lecciones aprendidas

Entre las lecciones aprendidas durante el desarrollo de este trabajo podemos encontrar las siguientes:

1. Organización de un trabajo grupal mediante distintos medios (Telegram, Discord y Whatsapp) sin comprometer el flujo del trabajo gracias a que estos permitieron la fluidez de las ideas entre los miembros del grupo.

2. Se aprendió a usar de forma más eficiente la herramienta de Git, dando la oportunidad de que en caso de que un código se estuviera volviendo "código spaghetti" se pudiese volver atrás en la línea del tiempo de Git.

3. Mejor uso de la herramienta de GitHub para poder estar al tanto del código que aportaban los otros compañeros, volviendo más dinámico el intercambio de códigos (sobre todo a la hora de unir las distintas partes).

4. La posibilidad de trabajar un mismo problema y un mismo código desde sistemas operativos distintos, aprendiendo así las distinciones tan mínimas que puede existir entre correr un código en uno o en otro (por ejemplo, para este caso, el único cambio que se encontró al correr el código en Windows o Linux era en la sentencia *system("cls")* con *system("clear")*).

5. El funcionamiento de un algoritmo que simula inteligencia artificial cómo lo es el Minimax, esto permitió obtener una perspectiva más amplia de cómo es que funcionan las IA's y de paso repasar los conceptos de *bactracking* y *estructuras de datos* que se habían visto en cursos pasados.

Bibliografía

1. El algoritmo Minimax y su aplicación en un juego. (2021). Retrieved 14 March 2021, from https://devcode.la/tutoriales/algoritmo-minimax/#:~:text=El%20algoritmo%20de%20minimax%20en,recorre%20todo%20el%20%C3%A1rbol%20de

2. Beginning C Programming - Part 38 - Tic Tac Toe #9 - Proper AI with MinMax. (2021). Retrieved 14 March 2021, from https://www.youtube.com/watch?v=Ht3bghfos_I

3. Minimax Algorithm in Game Theory | Set 1 (Introduction) - GeeksforGeeks. (2021). Retrieved 14 March 2021, from https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

4. The coding train!. (2021). Coding Challenge 154: Tic Tac Toe AI with Minimax Algorithm. Retrieved 15 March 2021, from https://www.youtube.com/watch?v=trKjYdBASyQ

5. Tic Tac Toe: Understanding the Minimax Algorithm — Never Stop Building - Crafting Wood with Japanese Techniques. (2021). Retrieved 15 March 2021, from https://www.neverstopbuilding.com/blog/minimax

6. Padron-McCarthy, T. (2021). MiniMax algorithm tic-tac-toe in C explanation. Retrieved 15 March 2021, from https://stackoverflow.com/questions/32659158/minimax-algorithm-tic-tac-toe-in-c-explanation