CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

INTELIGENCIA ARTIFICIAL

3° "A"

EXAMEN: RESOLVER PROBLEMAS MEDIANTE BÚSQUEDAS: BÚSQUEDA CON ALGORITMOS A* Y MINIMAX

Profesor: Miguel Ángel Meza de Luna

Alumno: Espinoza Sánchez Joel Alejandro

Fecha de Entrega: Aguascalientes, Ags.**, 25** de noviembre de 2019

Índice

La inteligencia artificial aplicada a la resolución de problemas

Es claro que la implementación de algoritmos es muy útil para conocer el procedimiento para resolver problemas y así solucionar dificultades que se presentan y mediante la mecanización o el mero análisis del problema, poder dar una propuesta para completar dicha adversidad.

La programación dedica sus esfuerzos y alcances teóricos a encontrar cada vez más soluciones a más problemáticas, de modo que el alcance que la misma tiene, como Joyanes (2010) explica, se expande con el paso del tiempo en ramas como la programación estructurada, la programación orientada a objetos, diseño web o la muy reciente incorporación científica de la disciplina llamada inteligencia artificial, la cual busca dar un paso muy grande en la concepción de los problemas que esta rama teórica puede abarcar.

Benítez (2013) menciona que la inteligencia artificial es una rama de la programación que, aunque tiene poco campo de exploración, la misma trata de cumplir una meta muy grande en las expectativas de las personas, pues es esta disciplina de la programación la que busca que los algoritmos se modifiquen por sí mismos, que aprendan y salgan de las órdenes directas que el usuario escribió, es decir, que un avance y evolución existan para la mejoría de procesos ya que la concepción de inteligencia artificial es aquella que “sintetiza y automatiza tareas intelectuales y es, por lo tanto, potencialmente relevante para cualquier ámbito de la actividad intelectual humana. En este sentido, es un campo genuinamente universal.” (Russell, 2008).

No obstante, como inteligencia artificial, Russell destaca que la misma rama se analiza – o de otra forma, estudia no precisamente – “las facultades mentales mediante el uso de modelos computacionales” (Russell, 2008) y al hacer esto, el ser humano realiza una introspección a sí mismo para cuestionarse cómo resuelve los problemas un ser humano.

Aunque el ser humano puede responder la pregunta de forma muy simple en relación al problema, al estipular sus procedimientos, “es necesario tomar en cuenta cada parte del análisis y si bien es cierto, la inteligencia artificial ha desarrollado algoritmos para diferentes modelos de pensamiento y análisis que logran descifrar los distintos tipos de análisis que las personas hacen para la resolución de problemáticas” (Palma, 2008), tal es el caso de algoritmos como el Aprendizaje por Refuerzo, las Redes Neuronales o Bayesianas o los métodos de búsqueda.

Estos últimos algoritmos, según Sánchez (2012) han sido fundamentales para la resolución de problemas con un amplio catálogo de posibilidades, es decir, permiten a un agente – ya sea humano o artificial – el movimiento libre en un medio definido, donde cada decisión que se toma genera un campo totalmente distinto de análisis que sería prácticamente imposible en términos computacionales de analizar. Es así como nacen estos algoritmos de la

inteligencia artificial; especializados en el estudio de los casos en el momento y que permiten realizar tareas que en un principio pueden parecer computacionalmente ineficaces.

Un paradigma muy interesante y también amplio es el estudio de los agentes inteligentes planteados con base en objetivos, también llamados agentes resolventes de problemas. “Los agentes resolventes-problemas deciden qué hacer para encontrar secuencias de acciones que conduzcan a los estados deseables. Comenzamos definiendo con precisión los elementos que constituyen el «problema» y su «solución», y daremos diferentes ejemplos para ilustrar estas definiciones. Entonces, describimos diferentes algoritmos de propósito general que podamos utilizar para resolver estos problemas y así comparar las ventajas de cada algoritmo. Los algoritmos son no informados, en el sentido que no dan información sobre el problema salvo su definición” (Russell, 2008) y posteriores a ellos, los algoritmos de búsqueda informada, los cuales se les suministra de algunas nociones para buscar soluciones

El modo en el que un agente puede plantearse con este modelo de pensamiento es estableciendo un objetivo que tratará de satisfacerlo. Lo siguiente es definir una función objetivo originada de la formulación de un problema y encontrar la secuencia que permita llegar al estado objetivo en el cual el agente consigue el éxito deseado. El proceso de hallar esta secuencia se le conoce como búsqueda. La secuencia de acciones que devuelve se le conoce como solución y la fase en la que el agente lleva a cabo estas acciones se le llama ejecución.

Algoritmos que están compuestos por objetivos, funciones objetivo y estados finales, así como se relacionan con los conceptos de búsqueda, solución y ejecución hay muchos, sin embargo, dos de los más importantes son el algoritmo A* y el algoritmo MinMax. El conocer cómo se aplican, su sustento teórico y una propuesta práctica en el que se muestra su desempeño en una prueba determinada se encuentran a continuación.

El Algoritmo A*

El Algoritmo A*, llamado A Estrella o A Star, “presentado en 1968 por Peter E. Hart, Nils J. Nilsson y Bertram Raphael, es un algoritmo que encuentra, siempre y cuando se cumplan unas determinadas condiciones, el camino de menor coste entre una condición inicial y una condición meta. Este algoritmo encuentra el camino de menor coste entre un nodo origen y un nodo meta, tratando de evadir los caminos de coste alto” (Valverde, 2009).

Este algoritmo se analiza porque “se caracteriza por usar heurísticas admisibles, es decir, heurísticas que nunca sobrepasan el costo real para alcanzar la meta. Además de esto cumple con la propiedad de completitud, es decir, siempre encontrará la solución si es que existe” (Reina, 2011).

Según Palma (2008), existe una función que describe al algoritmo A* la cual es la siguiente:

$$f(n) = g(n) + h(n)$$

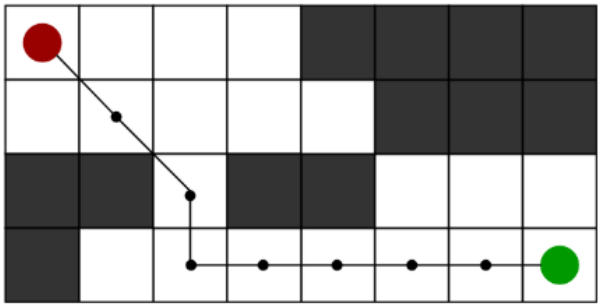
donde

- $f(n)$ es el costo del camino del nodo inicial al nodo n.
- $g(n)$ es el costo estimado del nodo n a la meta.
- $h(n)$ es el costo de recorrido para alcanzar la meta.

Descripción

La motivación del algoritmo de búsqueda A* descrita por Benítez (2013) es dar a conocer cómo aproximar los caminos en la vida real, es decir, el objetivo principal del algoritmo es aproximar el recorrido más corto en situaciones de la vida real. Mapas y juegos pueden ser un buen terreno de prueba para este algoritmo.

Se puede considerar un tablero de dos dimensiones con ciertos obstáculos, donde se empieza de un punto de partida dado (especificado con rojo a continuación) y se llegará a una meta (especificado con un punto verde a continuación).



Se considera el anterior como un tablero cuadriculado que tiene algunos obstáculos y se determina una celda de inicio y de fin. Se desea obtener el camino a la celda objetivo (si existe) de la forma más rápida posible.

Lo que el Algoritmo A* de Búsqueda hace es que, a cada paso busca y toma un nodo de acuerdo con valor obtenido de la función f que es un parámetro equivalente a la suma de los otros dos parámetros g y h. A cada paso se tomará el nodo con el menor valor de f y lo procesará para seguir en su búsqueda por ese camino.

Ahora, para poder realizar la evaluación de costes es necesario tener estos valores para cada nodo o acción, por lo tanto, el problema real se lleva a cabo al momento de la asignación de pesos a los nodos. En términos de la función f establecida, es posible y relativamente sencillo calcular g, pero el problema es el cálculo de h.

Para el cálculo de h puede hacerse mediante dos opciones:

Heurísticas Exactas

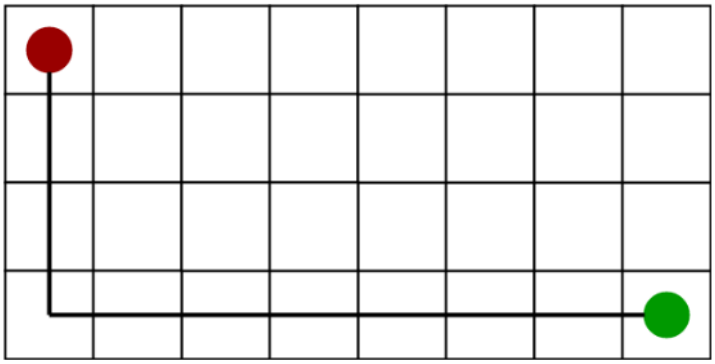
Pueden hallarse los valores exactos de h pero Benítez (2013) advierte desde un inicio que éste generalmente consume mucho tiempo, sin embargo, para hacer este proceso tenemos dos métodos:

- Calcular la distancia entre cada par de nodos antes de ejecutar el Algoritmo A*
- Si no hay nodos bloqueados, entonces fácilmente puede hallarse el valor exacto de h usando la fórmula de distancia entre dos puntos o distancia euclidiana (véase anexo 1.01).

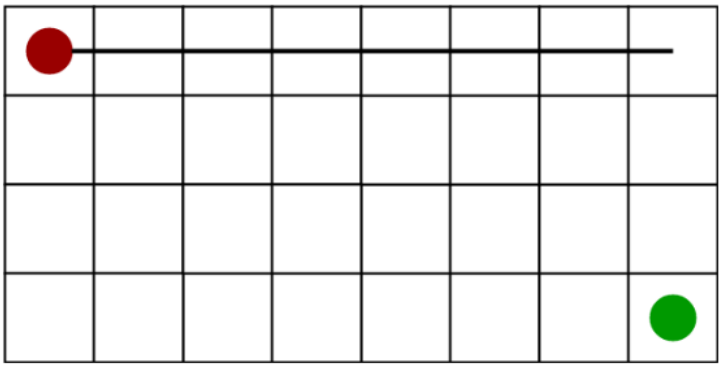
Heurísticas Aproximadas

A modo de generalidad, existen tres métodos de aproximación heurística para calcular h:

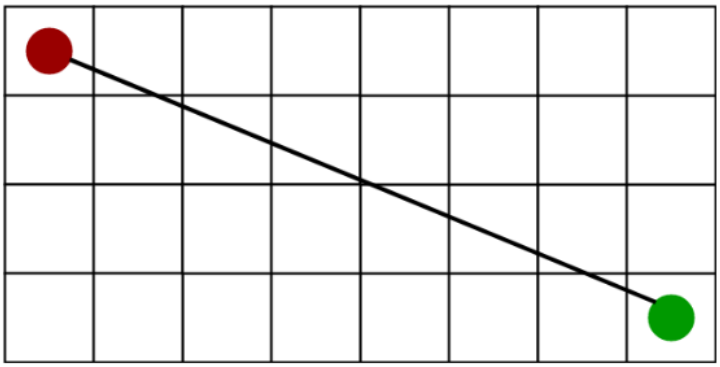
- **Distancia Manhattan:** Se trata de la suma de los valores absolutos de las diferencias de las coordenadas de un punto y otro.



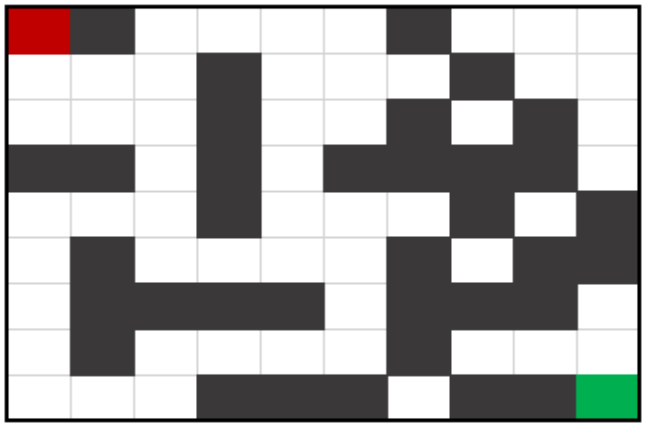
- **Distancia Diagonal:** Es el máximo de los valores absolutos de las diferencias de las coordenadas de un punto y otro. Cabe aclarar que su aproximación no es muy precisa.



- **Distancia Euclidiana:** La distancia real entre la coordenada de salida y coordenada de llegada.



Para la aplicación del algoritmo, se propondrá el siguiente tablero:



Donde las casillas blancas son casillas accesibles, las casillas grises son casillas bloqueadas, la de color rojo es la casilla de salida y la verde es la casilla objetivo.

La codificación del programa comienza con definir el tablero, hecho de la siguiente manera:

```
//Definición del tablero----------------------------
/*
1 = La casilla no está bloqueada
0 = La casilla está bloqueada
*/
int grid[ROW][COL]=
{
    { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 0, 1, 0, 1 },
    { 0, 0, 1, 0, 1, 0, 0, 0, 0, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },
    { 1, 0, 1, 1, 1, 1, 0, 1, 0, 0 },
    { 1, 0, 0, 0, 0, 1, 0, 0, 0, 1 },
    { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
    { 1, 1, 1, 0, 0, 0, 1, 0, 0, 1 }
};
```

Y una vez que se definen las casillas de salida y de meta, el algoritmo entra a la función `aStarSearch` la cual se encarga de realizar el proceso ya explicado anteriormente. Esta, apoyada de otras funciones, en un principio valida casos muy simples que hacen que el algoritmo cierre su procedimiento rápidamente (véase anexo 1.02) tales como valores no válidos, bloqueo de casillas o que la casilla de salida y la casilla objetivo sean la misma.

Posteriormente, el algoritmo inicializa dos listas, las cuales son una lista abierta y una lista cerrada. En un principio, la lista abierta se llena de los datos a analizar del tablero, mientras que la lista cerrada permanece vacía y al comenzar el recorrido de datos, el algoritmo analiza el dato actual de la lista abierta, el cual, cuando es analizado se pasa a la lista cerrada.

Sin embargo, el algoritmo necesita recorrer el tablero y para hacerlo, se pensó que el algoritmo tuviera el conocimiento de las casillas adyacentes a la actual se planteó la siguiente idea:

Es decir, tomar en cuenta las ocho casillas adyacentes de la actual. Después, el algoritmo analiza una por una de estas ocho. De ser la que se busca, se asigna una relación a la casilla en cuestión y se actualizan los valores de la función matemática para la casilla, así como se dedica un bloque de código a la posibilidad de encontrar la casilla que se busca en el siguiente movimiento.

```
//Prueba 1: Norte-----------------------------------------------------------------------------------
if(isValid(i-1, j) == true) //Verifica si la casilla norte es válida para la posición actual
{
    if(isDestination(i-1, j, dest) == true)
    {
        //Asigna la casilla encontrada a las variables de relación de la casilla anterior
        cellDetails[i-1][j].parent_i = i;
        cellDetails[i-1][j].parent_j = j;
        printf("Punto de llegada encontrado\n");
        tracePath (cellDetails, dest);
        foundDest = true;
        return;
    }
    else //Si la casilla ya está en la lista cerrada o está bloqueada, ignora y realiza lo siguiente
    {
        if(closedList[i-1][j] == false && isUnBlocked(grid, i-1, j) == true)
        {
            gNew = cellDetails[i][j].g + 1.0;
            hNew = calculateHValue (i-1, j, dest);
            fNew = gNew + hNew;

            if(cellDetails[i-1][j].f == FLT_MAX || cellDetails[i-1][j].f > fNew)
            {
                openList.insert( make_pair(fNew, make_pair(i-1, j)));

                //Se actualizan los detalles de la casilla
                cellDetails[i-1][j].f = fNew;
                cellDetails[i-1][j].g = gNew;
                cellDetails[i-1][j].h = hNew;
                cellDetails[i-1][j].parent_i = i;
                cellDetails[i-1][j].parent_j = j;
            }
        }
    }
}
```

Esto se revisa con las ocho casillas adyacentes hasta que se encuentre el destino. Si no se encuentra, el ciclo finaliza, accediendo al último condicional de la función, que es el siguiente

```
//Validación final: No es posible generar un camino-------------------------------------------
//Si no se encontró la casilla objetivo y la lista abierta está vacía, se concluye que se falló
if(foundDest == false)
{
    printf("Ocurrió un error al tratar de encontrar el punto de llegada\n");
}
return;
```

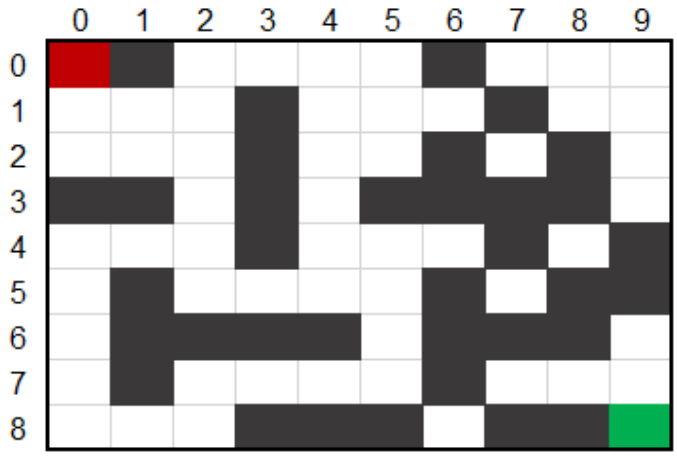
Hasta aquí, la función de A* finaliza y el algoritmo en cuestión termina, pero para terminar el programa, éste imprime el resultado. si encontró un camino, lo imprime, si no lo encontró, imprime cualquiera de las simples validaciones explicadas previamente o el error que tuvo (véase anexo 1.03).

Resultados

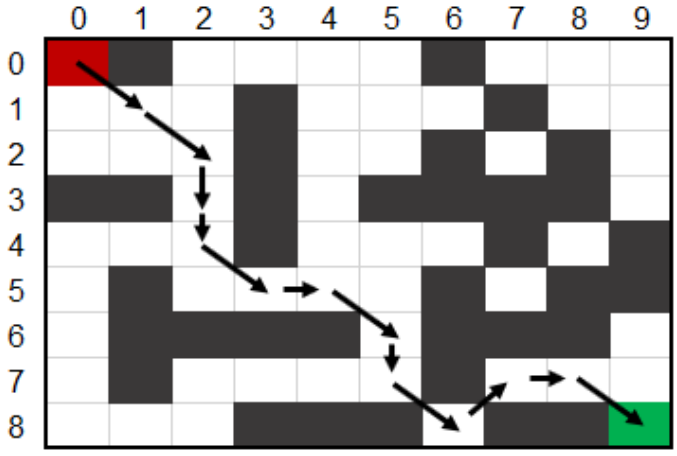
En el caso en el que ninguna de las excepciones del algoritmo se presente, éste arrojaría el camino más corto con base en lo que el algoritmo obtuvo. El resultado concreto del ejemplo propuesto sería el siguiente camino:

```
Punto de llegada encontrado

El camino es -> (0,0) -> (1,1) -> (2,2) -> (3,2) -> (4,2) -> (5,3)
-> (5,4) -> (6,5) -> (7,5) -> (8,6) -> (7,7) -> (7,8) -> (8,9)
--------------------------------
Process exited after 0.3226 seconds with return value 0
Presione una tecla para continuar . . .
```

El algoritmo expresa las coordenadas empezando el conteo desde cero, es decir, en el tablero anterior, puede verse el sistema de coordenadas con la siguiente numeración:



De modo que el camino que el algoritmo A* encontró fue el que dibujado en el tablero se ve de la siguiente manera:



El Algoritmo MiniMax

Tanto Plascencia (2016) como López (2009) mencionan que el algoritmo MiniMax tiene su sustento teórico en las decisiones óptimas en juegos. Para problemas como estos, "consideraremos juegos con dos jugadores, que llamaremos MAX y MIN por motivos que pronto se harán evidentes. MAX mueve primero, y luego mueven por turno hasta que el juego se termina. Al final de juego, se conceden puntos al jugador ganador y penalizaciones al perdedor" (Russell, 2008), es decir que un juego con estas características puede ser catalogado como una clase de problema de búsqueda inteligente, el cual se sabe que tiene los siguientes componentes:

- **El estado inicial:** Incluye la posición de un tablero definido e identifica al jugador que mueve.
- **Una función sucesora:** Devuelve una lista de pares refiriéndose al movimiento y al estado, para indicar que el movimiento se podía llevar a cabo y a su vez, estado que resulta de este movimiento.
- **Un test terminal:** Determina cuándo se termina el juego, pues evalúa estos estados donde el juego se ha terminado, conocidos como estados terminales.
- **Una función utilidad:** Conocida también por el nombre función objetivo o función de rentabilidad. Se encarga de dar un calor numérico a los estados terminales. En el ajedrez, el resultado es un triunfo, pérdida o empate, con valores +1, –1 o 0, pero en otros juegos como el Backgammon, los resultados pueden ir desde –192 hasta +192.

El estado inicial y los movimientos permitidos en el tablero definen una estructura que se le conoce como el árbol de juegos. Este árbol es muy importante en el desarrollo del algoritmo MinMax, pues con él, se irán desglosando los posibles movimientos que se pueden hacer.

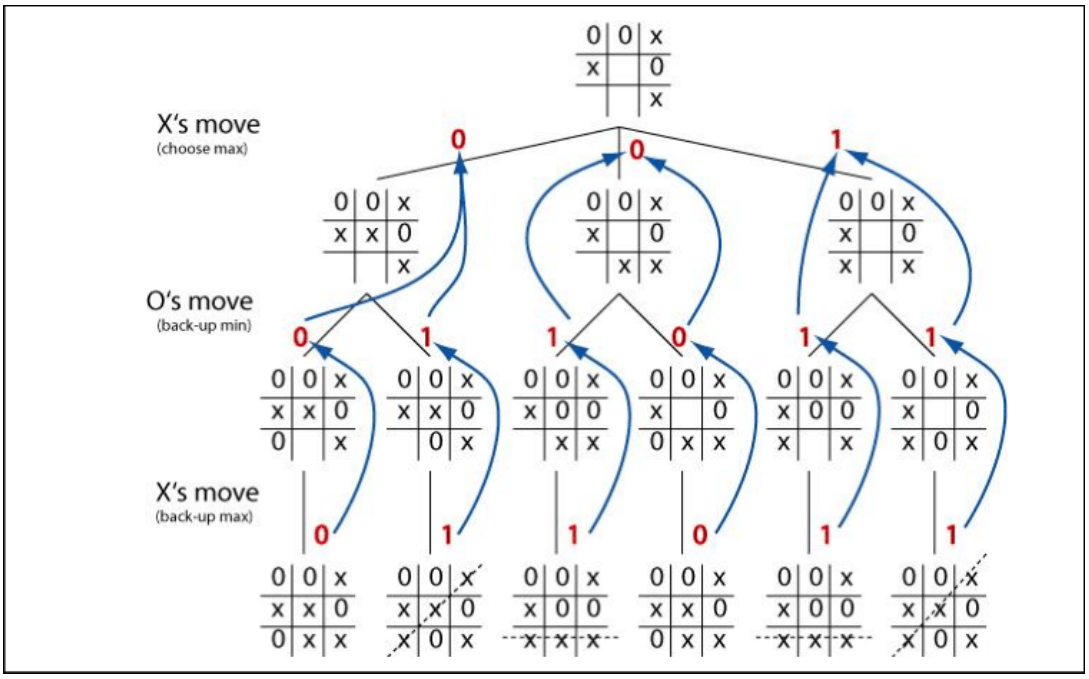
Palma (2008) recalca fuertemente que es necesario recordar que este algoritmo es una competencia entre dos usuarios denominados Min y Max. Supóngase que se trabaja sobre el tablero del gato o Tic Tac Toe (véase anexo 1.04), es decir, con este juego se ejemplificará el algoritmo. Si Max realiza el primer movimiento, éste tiene nueve "movimientos" – o elecciones – posibles. El juego sigue su curso alternando de turno entre Min y Max, con cada vez menos posibles elecciones y todos deben llevar a los estados terminales, que son representados donde alguno de los jugadores tenga un tres en raya o el tablero esté completamente lleno.

Asimismo, hablando de los demás componentes de este problema, "el número sobre cada nodo hoja indica el valor de utilidad del estado terminal desde el punto de vista de Max; se supone que los valores altos son buenos para Max y malos para Min (por eso los nombres de los jugadores). Este trabajo de Max al usar el árbol de búsqueda (en particular la utilidad de estados terminales) determina el mejor movimiento." (Russell, 2008).

Descripción

El Algoritmo MiniMax es un algoritmo muy usado en teoría de juegos. Este proceso consiste en un árbol de decisión para minimizar la pérdida máxima aplicada a juegos de adversarios. “Tiene ciertas características como el ser un algoritmo con información completa, es decir que cada jugador conoce el estado del otro” (Russell, 2008). El concepto de Min Max viene de la elección de los movimientos, pues el algoritmo escogerá el mejor movimiento, suponiendo que el contrincante escogerá la peor.

Tomando como ejemplo el juego del gato o Tic Tac Toe para visualizar un juego de adversarios y aplicar fácilmente el algoritmo, puede observarse rápidamente un árbol de decisión de algunos casos a continuación.



El árbol de decisión está compuesto de una sucesión par e impar de procesos. Las decisiones pares son tomadas por Min, de modo que las decisiones impares las decide Max. Es decir, el jugador Max buscará jugar de modo que se sienta satisfecho y saque el máximo provecho de su jugada, mientras que Min jugará a obstaculizar las jugadas de Max, que consecuentemente le llevarán a Min a ganar el juego.

Cabe resaltar nuevamente que, no se tratan de dos inteligencias artificiales jugando (aunque ese es el objetivo), pues en este caso se plantea el escenario en el que una máquina juega contra un usuario, pero en cualquiera de los escenarios, los jugadores son nombrados Max y Min debido a los roles que toman en el juego de adversarios.

Programación en C

Para la programación y ejemplificación de este algoritmo se tomará el ejemplo ya expuesto, que es el del juego del Gato o también conocido como Tic Tac

Toe. Este ejemplo hará ver mediante un enfrentamiento humano contra máquina el árbol de decisiones que el algoritmo debe tomar según el escenario que se le presente.

El programa comienza con asignaciones iniciales, donde el tablero comienza inicializado en cero; esto significa que no hay movimiento aún en esta casilla. Posteriormente se da la opción de ser el primer o el segundo jugador. Una vez establecidos estos valores, el programa entra en un ciclo de no más de 10 jugadas donde alterna entre el tiro del jugador y el tiro de la computadora.

```
//Las casillas comienzan en 0. 1 = Computadora. -1 = Jugador
int board[9] = {0,0,0,0,0,0,0,0,0};

//Asignaciones iniciales-----------------------------------------
printf("Computadora: O, Tú: X\nJugar\n(1) Primero\n(2) Segundo\n");
int player=0;
scanf("%d",&player);
printf("\n");
unsigned turn;

//Comienza el juego----------------------------------------------
for(turn = 0; turn < 9 && win(board) == 0; ++turn)
{
    if((turn+player) % 2 == 0)
    {
        computerMove(board);
    }
    else
    {
        draw(board);
        playerMove(board);
    }
}
```

Cada tiro lleva a funciones específicas que se componen de lo que un tiro de computadora o de jugador harían (véase anexo 1.05). Lo siguiente fundamental del algoritmo es en la función donde la computadora hace su jugada, pues es esta la que llamará al algoritmo MiniMax para evaluar las posibilidades que tiene de mover y mover de la forma óptima.

El algoritmo MiniMax, implementado a este juego del Gato, lo primero que realiza es una validación para revisar que el jugador todavía no haya ganado, en caso de que todavía la victoria por parte del usuario no se haya concretado, entonces procede a realizar el análisis.

Después el algoritmo inicializa dos valores. Uno llamado `move` que es el movimiento “ideal” de la máquina y el otro denominado `score`, el cual es un valor menor, por lo tanto, tiene un costo que penaliza más al algoritmo. A continuación, el algoritmo revisa todas las casillas y a su vez revisa en las que no haya movimientos para realizar el procedimiento en las casillas restantes.

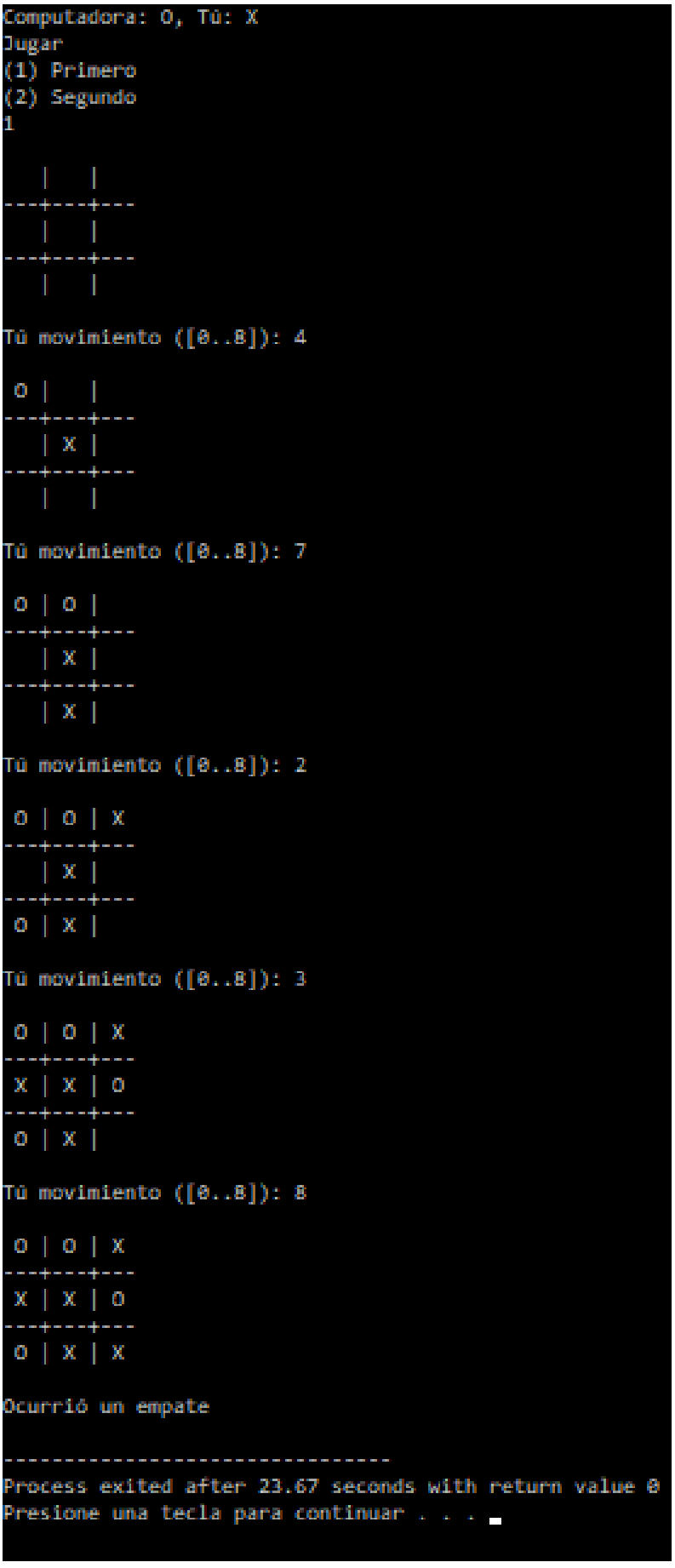
Una vez que encuentra estas casillas restantes para desarrollar el árbol de búsqueda, el programa intenta los movimientos que él desea, guardando en una variable un valor temporal de puntaje y el movimiento que lo orilló a ganar esta recompensa, sin embargo, esto lo realiza con todos los movimientos que se le permite hacer, de modo que el condicional permite acceder sólo al movimiento que mayor puntaje le dio y por lo tanto, también conocer el movimiento que originó este puntaje; siendo estos los valores que retornará a la función de la tirada de la computadora.

```
//El algoritmo MiniMax-----------------------------------------------------------------------------
int minimax(int board[9], int player)
{
    //Revisa si el jugador ya ha ganado
    int winner = win(board);
    if(winner != 0) return winner*player;

    int move = -1;
    int score = -2; //Los movimientos desfavorables son preferibles para la máquina de no hacerlos
    int i;
    for(i=0; i<9; ++i) //Revisa todas las casillas del tablero
    {
        if(board[i] == 0) //Revisa que no haya aún algún movimiento en esa casilla
        {
            board[i] = player; //Intenta un movimiento
            int thisScore = -minimax(board, player*-1);
            if(thisScore > score)
            {
                score = thisScore;
                move = i;
            }
            //Selecciona la que es la peor para el oponente
            board[i] = 0;//Reinicia el tablero
        }
    }

    //Retorna valor la función cuando encuentra el coste mínimo (que es -1)
    if(move == -1)
    {
        return 0;
    }
    //De lo contrario, retorna el coste que tiene en ese momento
    return score;
}
```

Resultados

Este es el procedimiento que sigue el programa para determinar cuál es la decisión que le conllevaría una victoria con mayor probabilidad donde busca y evalúa los escenarios.

Por último, el programa sólo muestra el tablero y los estados finales, ya sea cuando se gana, se pierde o se empata (véase anexo 1.06). A continuación, puede verse un ejemplo de ejecución del programa.


Computadora: O, Tú: X
Jugar
(1) Primero
(2) Segundo
1
| |
---+---+---
| |
---+---+---
| |
Tú movimiento ([0..8]): 4
O | |
---+---+---
| X |
---+---+---
| |
Tú movimiento ([0..8]): 7
O | O |
---+---+---
| X |
---+---+---
| X |
Tú movimiento ([0..8]): 2
O | O | X
---+---+---
| X |
---+---+---
O | X |
Tú movimiento ([0..8]): 3
O | O | X
---+---+---
X | X | O
---+---+---
O | X |
Tú movimiento ([0..8]): 8
O | O | X
---+---+---
X | X | O
---+---+---
O | X | X
Ocurrió un empate

Process exited after 23.67 seconds with return value 0
Presione una tecla para continuar . . .

Bitácora

Jueves 21 de Noviembre

Tiempo Empleado: 2 horas.

Actividades realizadas: Comencé a leer y buscar artículos que me permitieran saber a profundidad sobre estos algoritmos. Encontré varias páginas de internet y libros que explicaban con lujo de detalle la forma en la que estos algoritmos analizaban el contexto para el que se hicieron, por lo que guardé mis referencias y fuentes consultadas en un documento de Word. También comencé a escribir algunos antecedentes teóricos sobre los algoritmos que en la versión final se convirtió en la introducción del presente trabajo.

Viernes 22 de Noviembre

Tiempo Empleado: 4 horas, 30 minutos.

Actividades realizadas: Dediqué mi día al análisis de los códigos para llevarlos a un lenguaje de programación. Al decidir hacerlos en C, comencé a hacer investigaciones, sacar extractos de internet y otras cosas agregarlas por mi cuenta. Básicamente mi tiempo fue consumido en tener al 100% los códigos en el lenguaje C con una presentación idónea, es decir, aparte de mis "mañas" de programación, también agregué secciones y comentarios que faciliten el entendimiento del programa.

Sábado 23 de Noviembre

Tiempo Empleado: 3 horas, 30 minutos.

Actividades realizadas: Una vez que mis programas estaban preparados para ser enviados y su presentación era la que yo quería, el día sábado fue dedicado a realizar el documento formal en el que se explicaran los algoritmos y se dieran evidencias del proceso y los resultados que los programas comprendían. Investigué de cada algoritmo de las fuentes que ya tenía y ordené el documento junto con las capturas correspondientes. Todo lo fui modificando y llevando con el orden que esperaba que tuviera el producto final. Para el término del día, tanto los programas como el documento se encontraban en un estado de casi el 90% de preparación para ser enviados.

Domingo 24 de Noviembre

Tiempo Empleado: 40 minutos.

Actividades realizadas: Como acto final, revisé la ortografía, la cohesión y la redacción en general del documento, así como las secciones y comentarios de los programas en C. Me aseguré que los anexos y sus nombre tuvieran relación y lo conjunté todo para enviarlo en la noche; acto final que concluía con la realización del presente documento que muestra evidencias de mi examen.

Bibliografía

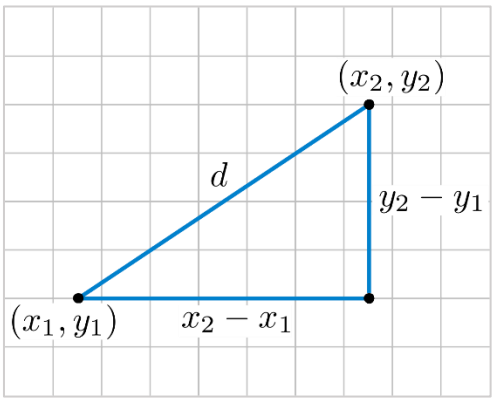
- Benítez, R., Escudero, G., & Kanaan, S. (2013). *Inteligencia artificial avanzada*. Retrieved from https://ebookcentral.proquest.com
- Joyanes, Luis. (2010). *Fundamentos de Programación.* México: McGraw Hill.
- López. T. (2009). *Algoritmo Minimax.* Noviembre 23, 2019, de Instituto Tecnológico de Nuevo Laredo Sitio web: http://www.itnuevolaredo.edu.mx/takeyas/Apuntes/Inteligencia%20Artificial/Apuntes/IA/Minimax.pdf
- Palma, M. J. T., & Marín, M. R. (2008). *Inteligencia artificial: Métodos, técnicas y aplicaciones*. Retrieved from https://ebookcentral.proquest.com
- Plascencia, C. (2016). *El algoritmo Minimax y su aplicación en un juego.* Noviembre 23, 2019, de DevCode Sitio web: https://devcode.la/tutoriales/algoritmo-minimax/
- Reina, M. (2011). *Algoritmo de búsqueda A* (Pathfinding A*).* Noviembre 24, 2019, de Escarbando Código Sitio web: https://escarbandocodigo.wordpress.com/2011/07/11/1051/
- Russell, S. (2008). *Inteligencia Artificial: Un Enfoque Moderno.* México: Pearson.
- Sánchez, S. (2012). *Ingeniería del Software.* México: Alfaomega.
- Valverde, J. (2009). Algoritmo A*. Noviembre 24, 2019, de Blogspot Sitio web: http://jc-info.blogspot.com/2009/05/algoritmo-codigo.html

Anexos

Primera Parte: Anexos Internos al Documento

Anexo 1.01: La distancia euclidiana

Sean $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$ dos puntos dados y d la distancia entre ellos.



La distancia d o distancia euclidiana en el plano está definida como:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Anexo 1.02: Validaciones de casos simples para A*

```
//Verificaciones iniciales---------------------------------------------------------------------------------------
//Verifica que la casilla de salida sea una casilla dentro del tablero
if(isValid(src.first, src.second) == false)
{
    printf("Punto de salida inválido\n");
    return;
}

//Verifica que la casilla objetivo sea una casilla dentro del tablero
if(isValid(dest.first, dest.second) == false)
{
    printf("Punto de llegada inválido\n");
    return;
}

//Verifica que las casillas de salida y objetivo estén disponibles para acceder (no bloqueadas)
if(isUnBlocked(grid, src.first, src.second)==false || isUnBlocked(grid, dest.first, dest.second)==false)
{
    printf("El punto de salida o de llegada está bloqueado\n");
    return;
}

//Verifica que las casillas de salida y objetivo no sean la misma casilla
if(isDestination(src.first, src.second, dest) == true)
{
    printf("Ya estamos en el punto de llegada\n");
    return;
}
```

Anexo 1.03: Resultados alternativos del algoritmo A*

Cuando el punto de salida no es una coordenada válida del tablero:

```
Punto de salida inválido


--------------------------------
Process exited after 0.6015 seconds with return value 0
Presione una tecla para continuar . . .
```

Cuando el punto de meta no es una coordenada válida del tablero:

```
Punto de llegada inválido

--------------------------------
Process exited after 0.5075 seconds with return value 0
Presione una tecla para continuar . . .
```

Cuando el punto de salida o de meta escogido es una casilla bloqueada:

```
El punto de salida o de llegada está bloqueado

--------------------------------
Process exited after 1.394 seconds with return value 0
Presione una tecla para continuar . . .
```

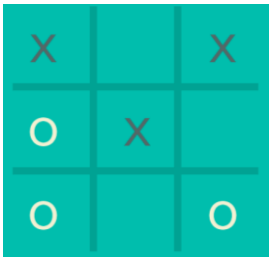
Cuando el punto de salida y de meta son el mismo:

```
Ya estamos en el punto de llegada

--------------------------------
Process exited after 0.3648 seconds with return value 0
Presione una tecla para continuar . . .
```

Cuando el tablero está bloqueado y no existen caminos para que el algoritmo llegue del punto de salida a la meta:

```
Ocurrió un error al tratar de encontrar el punto de llegada

--------------------------------
Process exited after 0.3911 seconds with return value 0
Presione una tecla para continuar . . .
```

Anexo 1.04: El Tablero del Juego del Gato



Anexo 1.05: Las funciones de juego de la computadora y el jugador

La computadora juega

```
//La computadora juega----------
void computerMove(int board[9])
{
    int move = -1;
    int score = -2; //Los movimientos desfavorables son preferibles para la máquina de no hacerlos
    int i;
    for(i=0; i<9; ++i) //Revisa todas las casillas del tablero
    {
        if(board[i] == 0) //Revisa que no haya aún algún movimiento en esa casilla
        {
            board[i] = 1;
            int tempScore = -minimax(board, -1); //Entra a MiniMax después de haber visualizado su ambiente
            board[i] = 0;
            if(tempScore > score) //Al disminuir costes, trata de aumentar ganancias
            {
                score = tempScore; //Se asigna el puntaje más alto hallado y el movimiento que la originó
                move = i;
            }
        }
    }
    //Retorna un puntaje obtenido del MiniMax y del movimiento que le produzco este puntaje
    board[move] = 1;
}
```

El jugador juega

```c
//El jugador juega-----------------------------
void playerMove(int board[9])
{
    int move = 0;
    do
    {
        printf("\nTú movimiento ([0..8]): ");
        scanf("%d", &move);
        printf("\n");
    }
    while (move>=9 || move<0 && board[move]==0);
    board[move] = -1;
}
```

Anexo 1.06: Impresión de los estados finales

```c
//Despliegue de victoria por pantalla-------
switch(win(board))
{
    case 0:
        {
            draw(board);
            printf("\nOcurrió un empate\n");
            break;
        }
    case 1:
        {
            draw(board);
            printf("\nPerdiste\n");
            break;
        }
    case -1:
        {
            printf("\nGanaste\n");
            break;
        }
}
```

Segunda Parte: Anexos Externos al Documento

Esta sección de anexos no pudo ser incluida en el presente, debido a que son los archivos de código de los métodos de búsqueda.

Se anexan en caso de que se quiera realizar una prueba de ejecución o revisar el código de forma completa, puesto que los extractos más importantes del código se encuentran en los anexos internos al documento.

Estos anexos se adjuntaron al mismo documento en su entrega con el mismo nombre que se describen a continuación.

Anexo 2.01: Algoritmo A*

Anexo 2.02: Algoritmo MiniMax