PROYECTO 1

Algoritmos de búsqueda en ambientes discretos

Andrés Felipe Arana Lozano
Jassy Maribel Jaramillo Suarez
Natalia Rubiano Silva

INGENIERIA DE SISTEMAS

Introducción a la inteligencia artificial

Docente: Joshua David Triana Madrid

Universidad del Valle

ALGORITMOS DE BÚSQUEDA EN AMBIENTES DISCRETOS

ANDRÉS FELIPE ARANA LOZANO - 1455881
JASSY MARIBEL JARAMILLO SUÁREZ - 1455708
NATALIA RUBIANO SILVA - 1455962

UNIVERSIDAD DEL VALLE SEDE TULUÁ
ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN VII SEMESTRE
ASIGNATURA: INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL
TULUÁ-VALLE
2017

ALGORITMOS DE BÚSQUEDA EN AMBIENTES DISCRETOS

ANDRÉS FELIPE ARANA LOZANO - 1455881
JASSY MARIBEL JARAMILLO SUÁREZ - 1455708
NATALIA RUBIANO SILVA - 1455962

DOCENTE

Joshua David Triana Madrid, Ingeniero de Sistemas

UNIVERSIDAD DEL VALLE SEDE TULUÁ
ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN VII SEMESTRE
ASIGNATURA: INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL
TULUÁ-VALLE
2017

CONTENIDO

INTRODUCCIÓN

La búsqueda es un tema central en la Inteligencia Artificial, debido que para resolver problemas o realizar alguna función mecanizada se necesita buscar en un ambiente de estados. Generalmente para dar solución a estos problemas se necesita establecer una serie de acciones o decisiones, la cual es ejecutada por un agente que tiene como finalidad alcanzar un objetivo a partir de un estado inicial. Por tal razón existen algoritmos de búsqueda que le permiten a este agente alcanzar su objetivo dado un ambiente determinado.

Estos ambientes pueden ser discretos o continuos, el primero se refiere cuando los posibles estados del ambiente son limitados distintos y claramente definidos. En estos ambientes algunos algoritmos de búsqueda son: Wall Tracing y WayPoint Navigation, estos tratan de obtener respuestas convenientes para el agente cuando se termina la búsqueda.

Por lo tanto en este trabajo se dará a conocer la descripción de cada uno de estos dos algoritmos nombrados, la implementación de cada uno de ellos, algunos ejemplos de su funcionamiento y la comparación entre ambos.

ALGORITMOS DE BÚSQUEDA EN AMBIENTES DISCRETOS

El primer proyecto del curso *Introducción a la Inteligencia Artificia*l consiste en implementar dos algoritmos de búsqueda para ambientes discretos con celdas: Wall tracing y Waypoint Navigation.

ALGORITMO WALL TRACING

Consiste en recorrer las paredes de calabozos, siguiendo un flujo desde el punto de vista del agente, como se muestra en la figura, intenta ir primero hacia su izquierda, frente, derecha finalmente si ninguna opción es válida intenta retroceder.

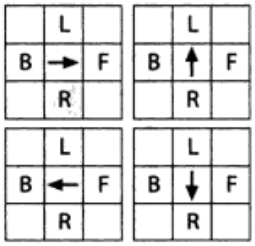


Figura 1: Flujo del Wall Tracing

Pseudocódigo

```
r = entityList[i].row;
c = entityList[i].col;
if (entityList[i].direction == 4) {
   if (terrain[r-1][c]  ==  1){
      entityList[i].row--;
      entityList[i].direction = 2;}
   else if (terrain[r][c+1] == 1){
      entityList[i].col++;
      entityList[i].direction = 4;}
   else if (terrain[r+1][c] == 1){
      entityList[i].row++;
      entityList[i].direction = 6; }
   else if (terrain[r][c-1] == 1){
      entityList[i].col--;
      entityList[i].direction = 8; }
}
else if (entityList[i].direction == 6){
   if (terrain[r][c+1] == 1){
      entityList[i].col++;
      entityList[i].direction = 4; }
   else if (terrain[r+1][c] == 1){
      entityList[i].row++;
      entityList[i].direction = 6; }
   else if (terrain[r][c-1] == 1){
      entityList[i].col--;
      entityList[i].direction = 8;}
   else if (terrain[r-1][c] == 1){
      entityList[i].row--;
      entityList[i].direction = 2;}
}
else if (entityList[i].direction == 8){
   if (terrain[r+1][c] == 1){
      entityList[i].row++;
      entityList[i].direction = 6;}
   else if (terrain[r][c-1] == 1){
      entityList[i].col--;
      entityList[i].direction = 8;}
   else if (terrain[r-1][c] == 1){
      entityList[i].row--;
      entityList[i].direction = 2; }
   else if (terrain[r][c+1] == 1){
      entityList[i].col++;
      entityList[i].direction = 4;}
}
else if (entityList[i].direction == 2){
   if (terrain[r][c-1] == 1){
      entityList[i].col--;
      entityList[i].direction = 8;}
   else if (terrain[r-1][c] == 1){
      entityList[i].row--;
      entityList[i].direction = 2;}
   else if (terrain[r][c+1] == 1){
      entityList[i].col++;
      entityList[i].direction = 4;}
   else if (terrain[r+1][c] == 1){
      entityList[i].row++;
      entityList[i].direction = 6;}
}
```

Figura 2: Pseudocódigo Wall Tracing. Libro AI For Game Developers.

Implementación Wall Tracing

- La aplicación lee un archivo de texto plano, el cual contiene un matriz de enteros de tamaño 20x20, donde “1” son paredes, “0” son espacios en blanco, “2” es el agente y “5” es la meta.

- Se usó dos matrices, una matriz de Jlabel que representa gráficamente el ambiente y una matriz lógica en la cual se realizan los movimientos del agente.

- El agente está representado por un dibujo animado que cambia su posición de vista de acuerdo hacia donde se realice su movimiento. El agente siempre inicia mirando hacia la derecha del jugador.

- Después de cada movimiento se usa el algoritmo de Bresenham que traza una línea desde el agente hasta donde está la meta, indicando los puntos por donde debe pasar el agente, este recorrido se guarda en un arraylist. Cuando no hay obstáculos en ese camino, el agente sigue ese recorrido, de lo contrario sigue el flujo descrito anteriormente del Wall Tracing.

- Antes de cada movimiento del Wall Tracing se realiza una serie de validaciones, detectando que no haya obstáculos para así poder realizar un movimiento válido.

- Para saber la dirección en la cual se encuentra el agente, se estableció unos puntos de vista del agente, como se ilustra en la *Figura 3*. La flecha indica hacia donde está la vista del agente.

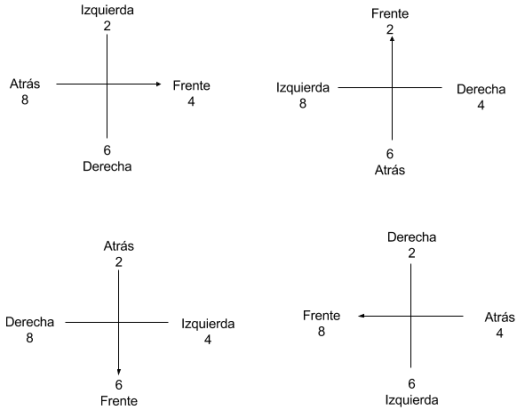


Figura 3: Ejes desde el punto de vista del agente.

- Cuando el agente tiene libre las cuatro direcciones (2, 4, 6, 8), entonces se realizan validaciones en sus diagonales.

Ejemplo Wall Tracing

- Ambiente 1:

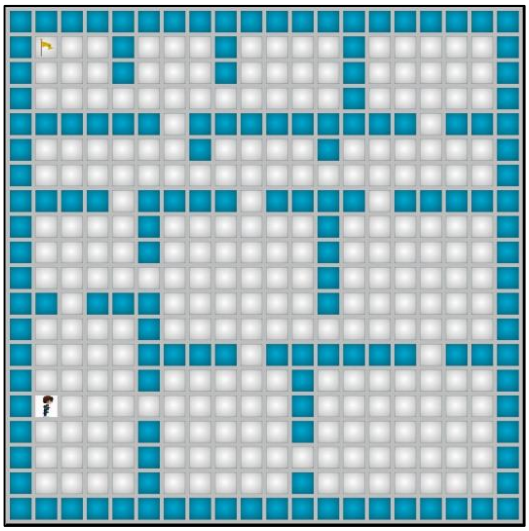


Figura 4: Ambiente 1 - Estado Inicial

En la *Figura 5* se observa el recorrido que realiza el agente hasta llegar a la meta. Las flechas indican hacia donde está mirando el agente, las flechas negras indican que el recorrido ha sido usando el flujo del Wall Tracing y las flechas rojas indican el recorrido del agente después de implementar el algoritmo de Bresenham.

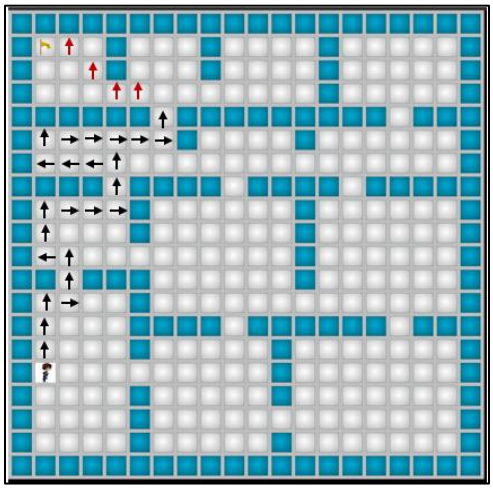


Figura 5: Ambiente 1: Recorrido algoritmo Wall Tracing

- Ambiente 2:

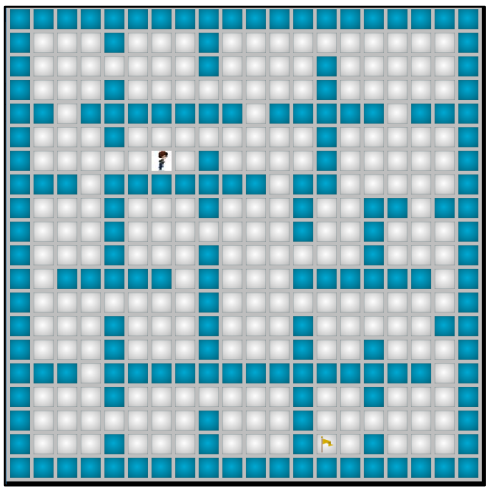


Figura 6: Ambiente 2 - Estado Inicial

En el siguiente ejemplo el agente se queda en un ciclo.

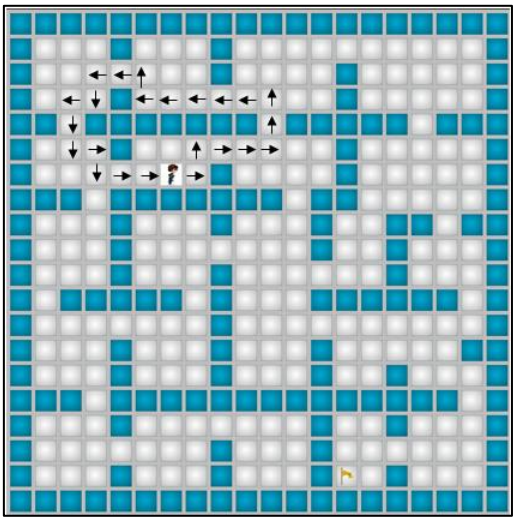


Figura 7: Ambiente 2: Recorrido algoritmo Wall Tracing

- Ambiente 3:

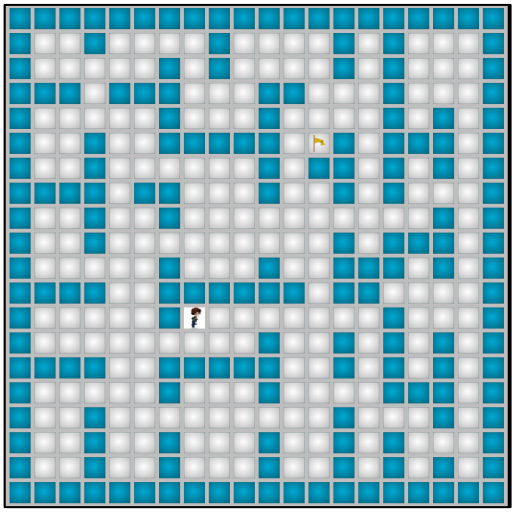


Figura 8: Ambiente 3 - Estado Inicial

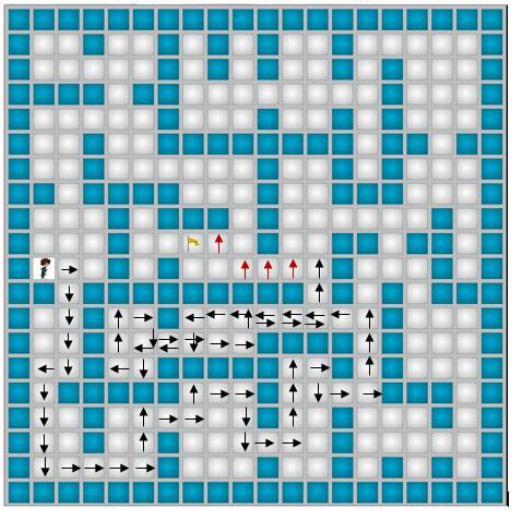


Figura 9: Ambiente 3: Recorrido algoritmo Wall Tracing

ALGORITMO WAYPOINT NAVIGATION

Algoritmo que reduce costos tanto de CPU como de tiempo, debido a que tiene como característica realizar caminos precalculados, basándose en nodos en el entorno del juego.

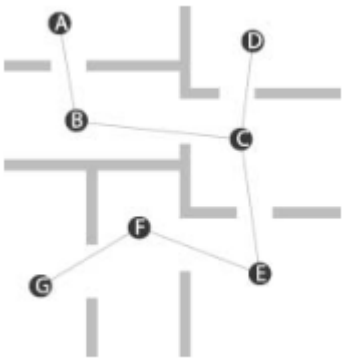


Figura 10: Algoritmo Waypoint

Es de gran importancia saber cómo es la conexión entre cada nodo, por lo tanto se utiliza una matriz de conexión de nodos, que tiene como objetivo establecer las conexiones entre estos.

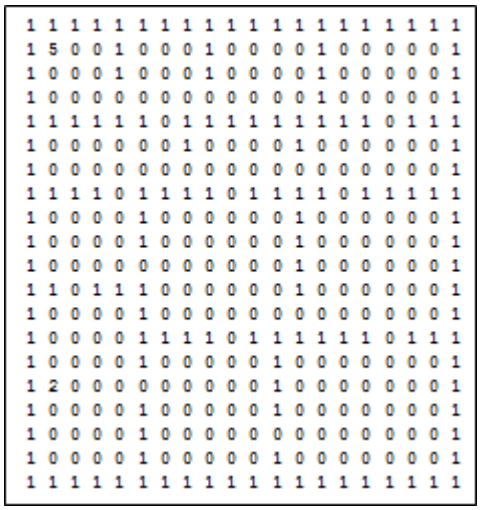
		End						
		A	B	C	D	E	F	G
	A	—	B	B	B	B	B	B
	B	A	—	C	C	C	C	C
	C	B	B	—	D	E	E	E
Start	D	C	C	C	—	C	C	C
	E	C	C	C	C	—	F	F
	F	E	E	E	E	E	—	G
	G	F	F	F	F	F	F	—

Figura 11: Matriz Conexión entre nodos

Esta matriz permite determinar el camino a seguir para llegar al nodo meta.

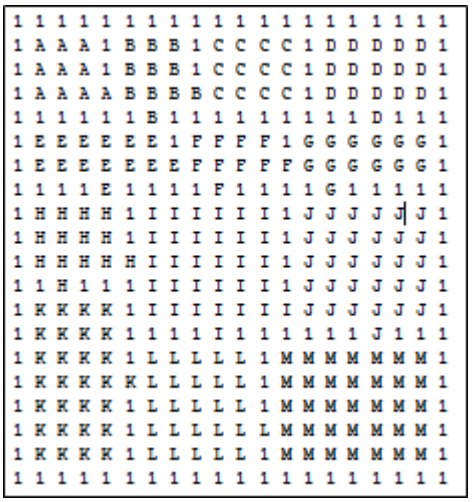
Implementación del Algoritmo

- La aplicación lee un archivo de texto plano, el cual contiene un matriz de enteros de tamaño 20x20, donde “1” son paredes, “0” son espacios en blanco, “2” es el agente y “5” es la meta. También contiene una matriz de String de tamaño 20x20, la cual posee los nodos y asimismo contiene la matriz de conexión entre los nodos.



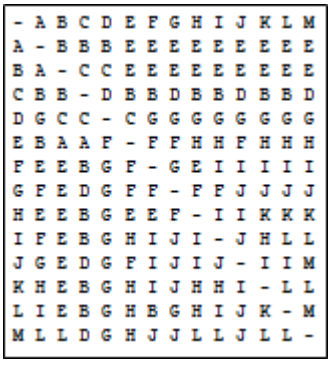
```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 5 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1
1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1
1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1
1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1
1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1
1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1
1 2 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1
1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Figura 12: Ejemplo de una Matriz del Laberinto - .txt



```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 A A A 1 B B B 1 C C C C 1 D D D D D 1
1 A A A 1 B B B 1 C C C C 1 D D D D D 1
1 A A A A B B B B C C C C 1 D D D D D 1
1 1 1 1 1 1 B 1 1 1 1 1 1 1 1 1 D 1 1 1
1 E E E E E E 1 F F F F 1 G G G G G G 1
1 E E E E E E E F F F F G G G G G G 1
1 1 1 1 E 1 1 1 1 F 1 1 1 1 G 1 1 1 1 1
1 H H H H 1 I I I I I I 1 J J J J J 1
1 H H H H 1 I I I I I I 1 J J J J J 1
1 H H H H H I I I I I I 1 J J J J J 1
1 1 H 1 1 1 I I I I I I 1 J J J J J 1
1 K K K K 1 I I I I I I I J J J J J 1
1 K K K K 1 1 1 1 1 I 1 1 1 1 1 J 1 1 1
1 K K K K 1 L L L L L 1 M M M M M M M 1
1 K K K K K L L L L L 1 M M M M M M M 1
1 K K K K 1 L L L L L 1 M M M M M M M 1
1 K K K K 1 L L L L L L M M M M M M M 1
1 K K K K 1 L L L L L 1 M M M M M M M 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Figura 13: Ejemplo de una Matriz de Nodos - .txt



```
- A B C D E F G H I J K L M
A - B B B E E E E E E E E E
B A - C C E E E E E E E E E
C B B - D B B D B B D B B D
D G C C - C G G G G G G G G
E B A A F - F F H H F H H H
F E E B G F - G E I I I I I
G F E D G F F - F F J J J J
H E E B G E E F - I I K K K
I F E B G H I J I - J H L L
J G E D G F I J I J - I I M
K H E B G H I J H H I - L L
L I E B G H B G H I J K - M
M L L D G H J J L L J L L -
```

Figura 14: Ejemplo de una Matriz de Conexión entre Nodos - .txt

- En las dos primeras matrices el tamaño es estático, mientras que la tercera es de tamaño dinámica debido a que el número de nodos por matriz puede variar.

- Para saber el tamaño de esta matriz, se realiza una función que cuenta la cantidad de letras que hay, y se le suma uno con el objetivo de que una columna y una fila sean el símbolo de cada una de ellas.

- Cuando cada matriz está llena se utiliza el algoritmo Bresenham desde el agente hasta la meta para saber por cuales nodos el agente tiene que caminar, este recorrido se guarda en un arraylist, ya que el tamaño es dinámico.

- Después se compara este recorrido con la matriz de conexión entre nodos para saber cuál son todos los nodos que el agente tiene que caminar para evitar obstáculos ya que no todos los nodos están directos uno del otro.
- Para diferenciar cada nodo se realiza una función que retorna las puertas de cada nodo que se va a recorrer.

- Se hacen posibles validaciones de lo que pueda ocurrir si de pronto encuentra un obstáculo (no sirve para todos los ambientes).

- El algoritmo Bresenham permite la unión entre cada puerta y entre el origen y la meta.

Ejemplo del Algortimo

- Ambiente 1:

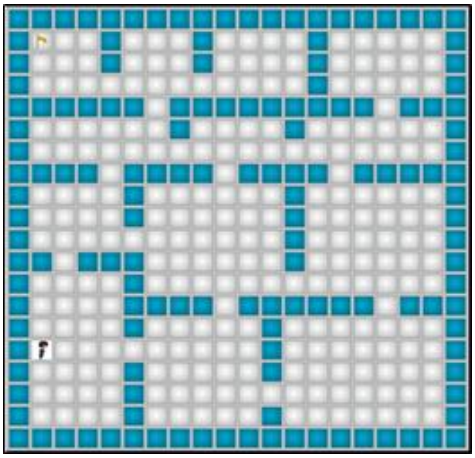


Figura 15: Estado Inicial

Es importante establecer los nodos del ambiente.

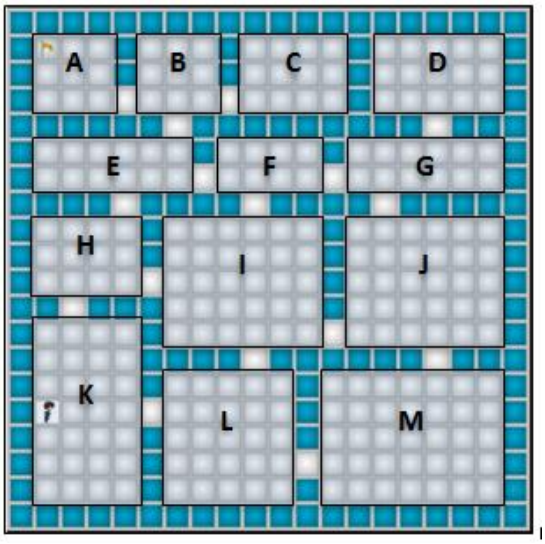


Figura 16: Nodos del Ambiente 1

El agente está en el nodo K y la meta en el nodo A, al aplicar el algoritmo Bresehman se observa que para llegar del nodo K al nodo A hay que pasar por el nodo K, H, E, A. Sin embargo, no todos los nodos están directos uno con el otro, por tal razón con la matriz de conexión entre nodos se halla el recorrido total de nodos.

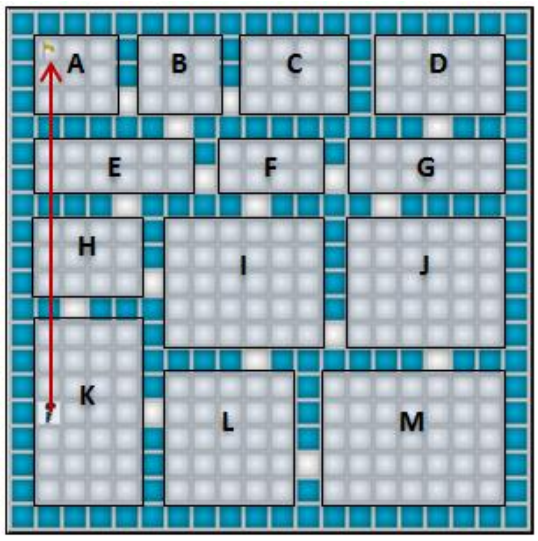


Figura 17: Algoritmo Bresehman

Con la matriz de conexión entre nodos, se halla que para llegar del nodo K al nodo A, hay que recorrer los nodos K, H, E, B, A. Se halla la puerta de cada uno de estos nodos.

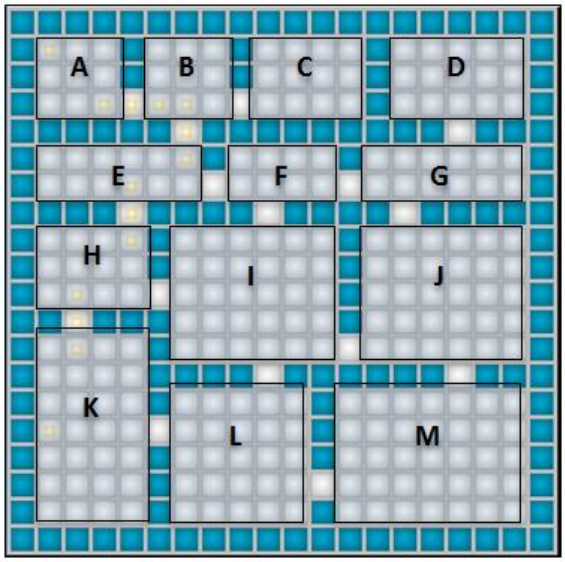


Figura 18: Puertas de los nodos a recorrer

Utilizando Bresenham, se une cada uno de las coordenadas (puertas) halladas.

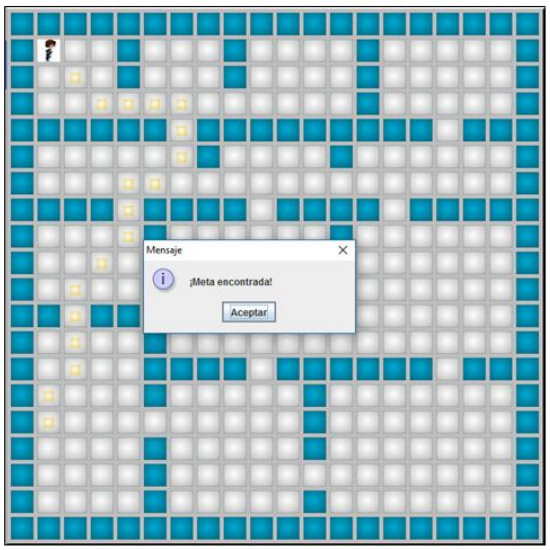


Figura 19: Estado Final - Ambiente 1

- Ambiente 2:

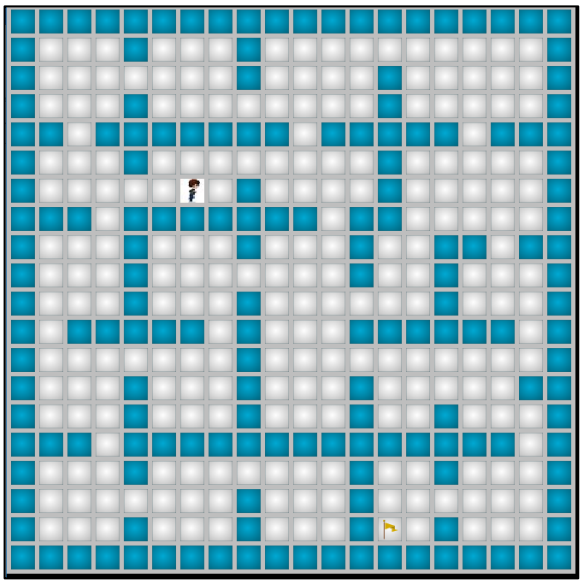


Figura 20: Estado inicial

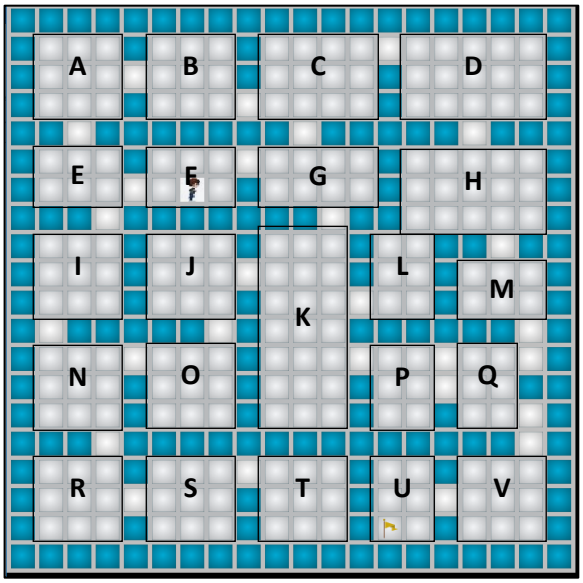


Figura 21: Nodos del Ambiente 2

El agente está en el nodo F y la meta en el nodo U, al aplicar el algoritmo Bresehman se observa que para llegar del nodo K al nodo A hay que pasar por el nodo F, J, K, U.

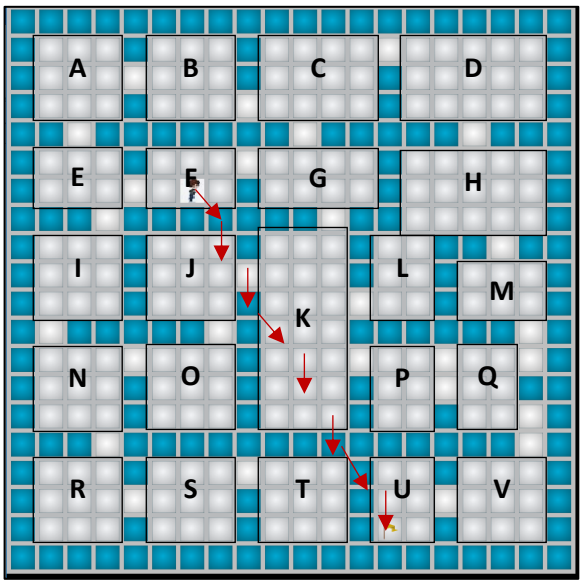


Figura 22: Bresehman

Con la matriz de conexión entre nodos, se halla que para llegar del nodo K al nodo A, hay que recorrer los nodos F, G, K, P, Q, V, U. Se halla la puerta de cada uno de estos nodos.

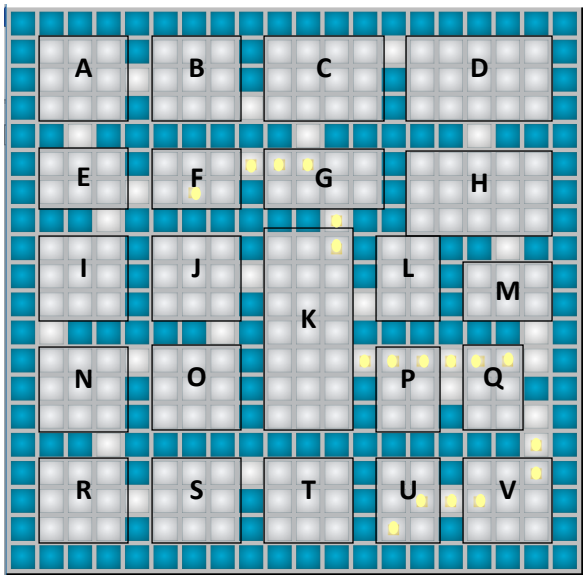


Figura 23: Puertas de los nodos a recorrer

Utilizando Bresenham, se une cada uno de las coordenadas (puertas) halladas.

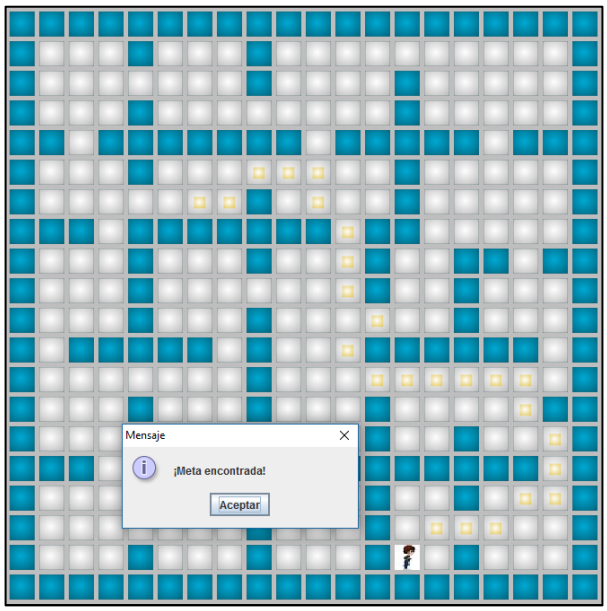


Figura 24: Estado Final - Ambiente 2

- Ambiente 3:

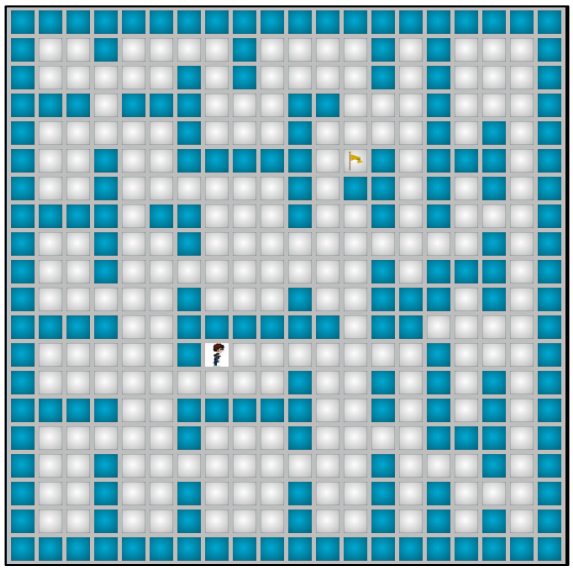


Figura 25: Estado inicial

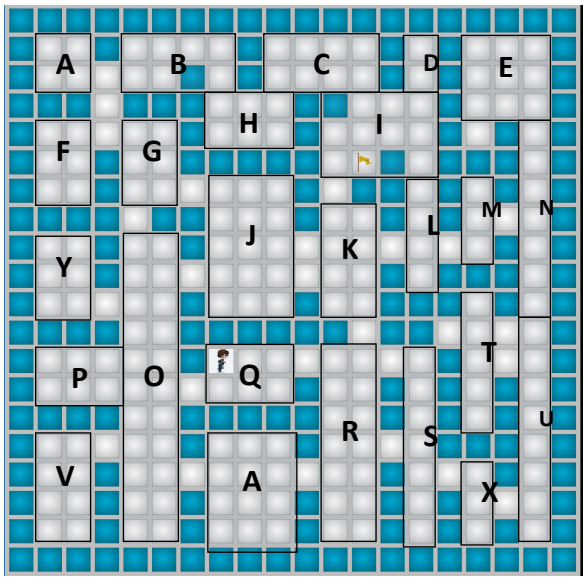


Figura 26: Nodos del Ambiente 3

El agente está en el nodo Q y la meta en el nodo I, al aplicar el algoritmo Bresehman se observa que para llegar del nodo K al nodo A hay que pasar por el nodo Q, J, K, I.

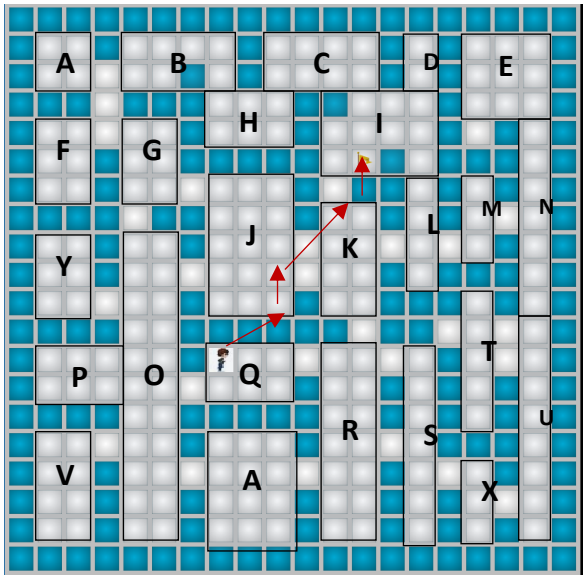


Figura 27: Bresehman

Con la matriz de conexión entre nodos, se halla que para llegar del nodo K al nodo A, hay que recorrer los nodos Q, R, K, I. Se halla la puerta de cada uno de estos nodos.

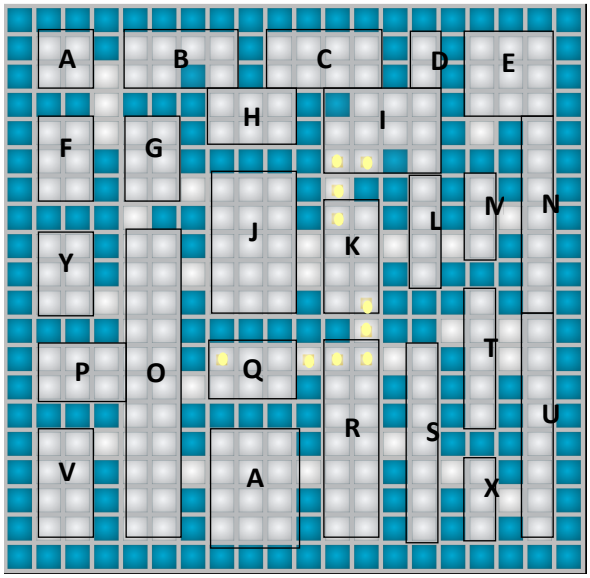


Figura 28: Puertas de los nodos a recorrer

Utilizando Bresenham, se une cada uno de las coordenadas (puertas) halladas.

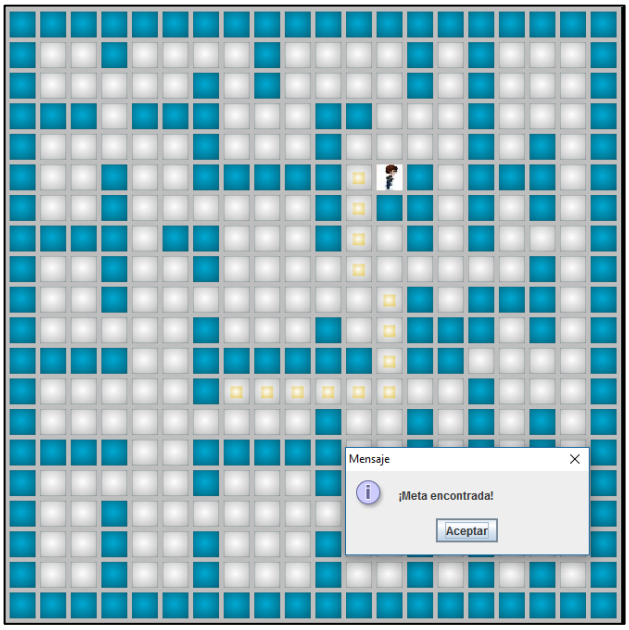


Figura 29: Estado Final - Ambiente 2

ALGORITMO DE BRESENHAM

La función de este algoritmo es calcular la meta trazando el camino desde donde se encuentra el agente hasta donde está ubicada la meta.

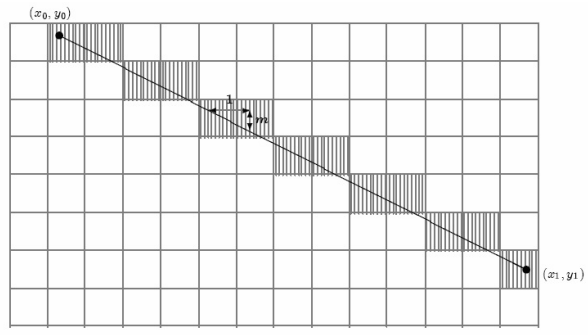


Figura 30: Ejemplo: Algoritmo Bresenhman

Fuente: *Campuzano, S. A. (23 de Abril de 2017). Mi Blog Universitario. Obtenido de Algoritmos Gráficos: http://sergioabrilcampuzano.blogspot.com.co/2012/09/algoritmos-graficos.html*

```
bresenhman() {
    int nextCol = columnaA; //inicioColumna;
    int nextRow = filaA; //inicioFila;
    int deltaRow = filaM - filaA; //finFila - inicioFila;
    int deltaCol = columnaM - columnaA; //finColumna - inicioColumna;
    int stepCol, stepRow;
    int fraction;
    if (deltaRow < 0) {
       stepRow = -1;
    } else {
       stepRow = 1;
    }
    if (deltaCol < 0) {
       stepCol = -1;
    } else {
       stepCol = 1;
    }
    deltaRow =  Math.abs(deltaRow * 2);
    deltaCol =  Math.abs(deltaCol * 2);
    System.out.println(" Meta: " + columnaM + "," + filaM + "  Agente: "  + columnaA + "," +
filaA +".");
    if (deltaCol > deltaRow) {
       fraction = deltaRow * 2 - deltaCol;
       while (nextCol != columnaM) { // finColumna
          if (fraction >= 0) {
             nextRow = nextRow + stepRow;
             fraction = fraction - deltaCol;
          }
```

```
            nextCol = nextCol + stepCol;
            fraction = fraction + deltaRow;
            puntosMetaX.add(nextCol);
                puntosMetaY.add(nextRow);
            System.out.println("Punto: " + nextCol + "," + nextRow);
            if (tablaMatriz[nextRow][nextCol] == 0) {
                                    tablaMatriz[nextRow][nextCol] = 3;
                                matrizJ[nextRow][nextCol].setIcon(iconoCamino);
                            }
        }
    } else {
        fraction = deltaCol * 2 - deltaRow;
        while (nextRow != filaM) { // finFila
            if (fraction >= 0) {
                nextCol = nextCol + stepCol;
                fraction = fraction - deltaRow;
            }
            nextRow = nextRow + stepRow;
            fraction = fraction + deltaCol;
            puntosMetaX.add(nextCol);
                puntosMetaY.add(nextRow);
            System.out.println("Punto: " + nextCol + "," + nextRow);
            if (tablaMatriz[nextRow][nextCol] == 0) {
                                    tablaMatriz[nextRow][nextCol] = 3;
                                matrizJ[nextRow][nextCol].setIcon(iconoCamino);
                            }
        }
    }
}
```

El algoritmo de Bresenham se utiliza en ambos algoritmos. En Wall Tracing se utiliza después de cada movimiento realizado para encontrar la meta. En Waypoint Navigation se utiliza para calcular los nodos por los que tiene que pasar el agente y para unir un nodo con otro.

FUNCIONAMIENTO DEL PROGRAMA

Para el funcionamiento del programa primero se debe dar clic en el botón “Cargar Archivo”, para elegir el ambiente que se desea utilizar. Después se da clic al algoritmo de búsqueda a usar. Y si desea elegir otro ambiente se da clic en el botón “Limpiar”, seguido del botón “Cargar Archivo” y el botón del Algoritmo a usar.

Archivos de Texto

Cada archivo de texto contiene:

- Veinte líneas de texto de 20 enteros cada una, donde cada entero del uno del otro está separado con espacio “ ”.
- Salto de línea
- Veinte líneas de texto de 20 String, donde cada string del uno del otro está separado con espacio “ ”.
- Salto de línea
- n líneas de texto de n String donde cada string del uno del otro está separado con espacio “ ” (matriz dinámica-depende del ambiente elegido).

Lectura de archivos

- El programa un objeto JFileChooser, el cual permite cargar la ventana, cuando se presione el botón “CargarArchivo”.
- Luego se crea un entero “opción” el cual devuelve la opción que va a realizar el usuario, cancelar, aceptar o error.
- Si “opcion” retorna JFileChooser.APPROVE_OPTION, quiere decir que el usuario pincho en aceptar. si es así, entonces se selecciona el archivo con: File fichero = archivo.getSelectedFile().
- Luego se escribe la ruta del fichero en el campo de texto.

El archivo txt posee tres matrices, sin embargo cuando se elige el algoritmo Wall Tracing, el algortimo solo lee la primera y en el algoritmo WayPoint si lee las tres.

El programa realiza esto de la siguiente manera:

- Se crea un objeto scanner el cual contiene el archivo de texto seleccionado
- Se crea una matriz “tablero” de enteros de tamaño 20x20, la cual va a contener primera la matriz que está en el archivo de texto.
- Se recorre la matriz “tablero”, se lee con “sc.nextInt();” los datos del archivo de texto y se asignan a una posición en la matriz “tablero”
- De esta forma en la matriz “tablero” queda guardada la matriz de enteros que se utilizará para dibujar la matriz gráfica y para desarrollar los dos algoritmos.

Si el algoritmo a elegir es Wall Tracing, la función de lectura termina. De lo contrario, si se elige el algoritmo Waypoint, la función realiza:

- Se crea otra matriz de String de tamaño 20x20, la cual va a contener la segunda matriz que está en el archivo de texto.
- Se recorre la matriz, se lee con “sc.nextLine();” los datos del archivo de texto y se asignan a una posición en la matriz. De esta forma en la matriz queda guardada la matriz de String. Esta matriz es la matriz de nodos.

- Después el programa crea un arraylist y realiza un ciclo para determinar si aún existen líneas en el archivo de texto, estas líneas las agrega en un arraylist.
- Este arraylist se agrega a una matriz, el tamaño de esta matriz es dinámica, varía según el ambiente elegido. Esta matriz es la matriz de conexión entre nodos.

Para dibujar la matriz gráfica, la cual será una matriz de JLabel: Se crea una matriz de JLabels llamada “matrizJuego” del mismo tamaño que la matriz de enteros (tablero) creada anteriormente, es decir 20x20, a cada JLabel se le agrega una imagen dependiendo del número que haya en la posición por la que vaya el recorrido del for en la matriz “tablero”, si es “1”, se crea un JLable con la imagen de un bloque azul; si es “0”, se crea un JLabel con la imagen de un bloque blanco, si es “2” se crea un JLabel con la imagen del Agente, y si es es “5”, el JLabel contendrá la imagen de la meta. La posición de cada JLabel en el panel se calcula asi: ubicación en “X” es la posición en “j” multiplicada por el ancho del JLabel, la ubicación en “Y” es posición en “i” por la altura del JLabel.

ANALISIS COMPARATIVO

El algoritmo Waypoint Navigation es más rápido para encontrar la meta (en ambientes no muy complejos) ya que tiene caminos precalculados, además éste recorre el ambiente por nodos, yendo hacia la puerta de cada módulo o cuarto; mientras que el algoritmo Wall Tracing recorre sólo paredes y podrá llegar a la meta sólo cuando no haya obstáculos, por lo cual podría caer en ciclos.

REFERENCIAS BIBLIOGRÁFICAS

David M. Bourg, Glenn Seeman. IA For Game Developers.

Inteligencia Artificial. (24 de Abril de 2017). Obtenido de Agentes Inteligentes: https://sites.google.com/site/intart8vosemestre/agentes-inteligentes

Todo Sobre Inteligencia Artificial. (24 de Abril de 2017). Obtenido de Métodos de Búsqueda y Ejemplos Prácticos : http://inteligenciaartificialgrupo33.blogspot.com.co/p/metodos-de-busqueda-y-ejemplos.html