UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO

FACULTAD DE INGENIERIA Y CIENCIAS DE LA COMPUTACION Y TELECOMUNICACIONES

Inteligencia Artificial I

CONTENIDO:

LAB-4. MOVIMIENTO DE LA TORRE.

PORCENTAJE TERMINADO: 100%

GRUPO: 11

INTEGRANTES	DT	HG	HI	EVAL
Flores Veizaga Eudenia Gandira	1	1	0	100
Garcia Taboada Brayan Albaro	1	1	1	100

Fecha de presentación : Jueves, 16 de Mayo de 2024

Fecha Presentada : Jueves, 16 de Mayo de 2024

Días de Atraso : 0

EL PROBLEMA DEL MOVIMIENTO DE LA TORRE.

Comentarios: Con base en estos códigos, logramos comprender y darnos cuenta de mejor manera el manejo de lo que son las listas de acuerdo al movimiento que se solicite realizar.

1. Dado un Tablero de n x m casillas. Se tiene una posición inicial y final, encontrar todos los caminos posibles del estado inicial al estado final (extremo superior izquierdo al extremo inferior derecho) con los movimientos de la TORRE. Implementar este algoritmo, p*ara movimientos en Sentido HORARIO*.

```
public static LinkedList<Regla> reglasAplicables(int m[][], int i, int j) {

    LinkedList<Regla> L1 = new LinkedList();

    int j1=j-1;

    while(posValida(m,i,j1)){

        L1.add(new Regla(i,j1));

        j1--;

    }

    int i1=i-1;

    while(posValida(m,i1,j)){

        L1.add(new Regla(i1,j));

        i1--;

    }

    j1=j+1;

    while(posValida(m,i,j1)){

        L1.add(new Regla(i,j1));

        j1++;

    }

    i1=i+1;
```

```
        while(posValida(m,i1,j)){
            L1.add(new Regla(i1,j));
            i1++;
        }
        return L1;
    }
```

2. Implementar el Ejercicio 1, con una pieza de ALFIL.

```
public static LinkedList<Regla> reglasAplicables(int m[][], int i, int j) {
        LinkedList<Regla> L1 = new LinkedList();
        int i1 = i - 1;
        int j1 = j - 1;

        while (posValida(m, i1, j1)) {
            L1.add(new Regla(i1, j1));
            i1--;
            j1--;
        }
        i1 = i - 1;
        j1 = j + 1;
        while (posValida(m, i1, j1)) {
            L1.add(new Regla(i1, j1));
            i1--;
            j1++;
        }
```

```
        i1 = i + 1;
        j1 = j + 1;
        while (posValida(m, i1, j1)) {
            L1.add(new Regla(i1, j1));
            i1++;
            j1++;
        }
        i1 = i + 1;
        j1 = j - 1;
        while (posValida(m, i1, j1)) {
            L1.add(new Regla(i1, j1));
            i1++;
            j1--;
        }
        return L1;
    }
```

Ejecutar Los Algoritmos 1 y 2, para estados Sin Atajos. Analizar las salidas posibles (todos los caminos del estado inicial al final), mostrar todos los caminos y la cantidad de soluciones posibles.

Al ser un estado sin atajos la cantidad de movimientos viene a ser mas libre por lo cual se obtiene soluciones mas optimas.