1. RESOLUCIÓN Y ALGORITMOS JUEGO DE DAMAS

- Ubicación de las fichas en el tablero
- Turno del jugador
- Numero de fichas disponibles en el tablero, por cada jugador

Variables:

tablero;
-1 nulo;
0 espacioVacio;
1 fichaBlanca;
2 fichaNegra;
3 reinaBlanca;
4 reinaNegra

fichasB: Número de fichas blancas
fichasN: Número de fichas negras
reinasB: Número de reinas blancas
reinasN: Número de reinas negras

• T: tablero
T ({-1,0,1,2,3,4} 8×8
Donde:
-1 : espacio no accesible
0 : espacio vacío
1 : ficha Blanca
2 : ficha Negra
3 : reina Blanca
4 : reina Negra

• t : turno
t ({1,2}
Donde:
1 : rojo
2 : negro

5. Estado inicial

2	-1	2	-1	2	-1	2	-1
-1	2	-1	2	-1	2	-1	2
2	-1	2	-1	2	-1	2	-1
-1	0	-1	0	-1	0	-1	0
0	-1	0	-1	0	-1	0	-1
-1	1	-1	1	-1	1	-1	1
1	-1	1	-1	1	-1	1	-1
-1	1	-1	1	-1	1	-1	1

X

Y 8+8

Estado

Condiciones de estado meta:

- Si reinasN =2 entonces gana negro
- Si reinasB= 2 entonces gana blanca
- Empate:
- Si : fichasB + reinasB = 1 y fichasN:+reinasN = 1

Reglas

Mover1(x1,y1,x2,y2): Mover ficha blanca de la posición inicial (x1, y1) a la posición destino (x2, y2)

Mover2(x1,y1,x2,y2): Mover ficha negra de la posición inicial (x1, y1) a la posición destino (x2, y2)

Mover3(x1,y1,x2,y2): Mover reina blancas de la posición inicial (x1, y1) a la posición destino (x2, y2)

Mover4(x1,y1,x2,y2): Mover reina negra de la posición inicial (x1, y1) a la posición destino (x2, y2)

Comer1(x1,y1,x2,y2): Ficha blanca come de la posición inicial (x1,y1) a ficha negra o reina negra en la posición (x2,y2)

Comer2(x1,y1,x2,y2): Ficha negra come de la posición inicial (x1,y1) a ficha blanca o reina blanca en la posición (x2,y2)

Comer3(x1,y1,x2,y2): Reina blanca come de la posición inicial (x1,y1) a ficha negra o reina negra en la posición (x2,y2)

Comer4(x1,y1,x2,y2): Reina negra come de la posición inicial (x1,y1) a ficha blanca o reina blanca en la posición (x2,y2).

ESTADO DE LOS MOVIMIENTOS

Estado inicial

Vacio

Movimiento

Movimieno(x1,y1,x2,y2)

Condicion

T=1 ------ T(x1,y1)=1 ----------T(x2,y2)=0

0 menor o igual a x2= x1 + 1 menor o igual a 7

0 menor o igual a y2=y1+1 menor o igual a 7

Nuevo estado

Si x2= 7 entonces fichasB= -1

fichareinaB=1

Estado Inicial

Vacio

Movimiento

Movimiento (x1,y1,x2,y2)

Condicion

T=2

T(x1,y1)=2 ----------T(x2,y2)=0

0 menor o igual a x2= x1 + 1 menor o igual a 7

0 menor o igual a y2=y1+1 menor o igual a 7

Nuevo estado

Si x2=0

Entonces fichas negras =-1

Fichas negras reina =1

Estado Inicial

Vacio

Moviento

Movimiento (x1,y1,x2,y2)

Condicion

T(x1,y1)=3 ----------T(x2,y2)=0

0 menor o igual a x2= x1 + 1 menor o igual a 7

0 menor o igual a y2=y1+1 menor o igual a 7

Nuevo Estado

Reinas Blanco

T(x1,y1)=2 ----------T(x2,y2)=0

Estado Inicial

Vacio

Moviento

Movimiento (x1,y1,x2,y2)

Condicion

T(x1,y1)=3 ----------T(x2,y2)=0

0 menor o igual a x2= x1 + 1 menor o igual a 7

0 menor o igual a y2=y1+1 menor o igual a 7

Nuevo Estado

Reinas Blanco

T(x1,y1)=0 ----------T(x2,y2)=4

Estado de la comida o eliminacion de fichas

Estado inicial

Vacio

Comida

Comida (x1,y1,x2,y2)

Condicion

T=1 ------ T(x1,y1)=1 ----------T(x2,y2)=0

0 menor o igual a x2= x1 + 1 menor o igual a 6

0 menor o igual a y2=y1+1 menor o igual a 6

Nuevo estado

T(x2,y2)=3

Si x1= 6

entonces Entonces fichas negras =-1

Fichas negras reina =1

Estado inicial

Vacio

Comida

Comida (x1,y1,x2,y2)

Condicion

T=1 ------ T(x1,y1)=1 ----------T(x2,y2)=0

0 menor o igual a x2= x1 + 1 menor o igual a 6

0 menor o igual a y2=y1+1 menor o igual a 6

Nuevo estado

T(x2,y2)=1

Si x1= 3

Estado inicial

Vacio

Comida

Comida (x1,y1,x2,y2)

Condicion

T=1 ------ T(x1,y1)=4 ----------T(x2,y2)=0

0 menor o igual a x2= x1 + 1 menor o igual a 6

0 menor o igual a y2=y1+1 menor o igual a 6

Nuevo estado

T(x2,y2)=1

Si x1= 4

entonces Entonces fichas negras =-1

Fichas negras reina =1

FUNCIONES

Se ha considerado una función evaluadora f que combina 3 criterios: la distancia, el valor de las fichas y reinas que comen y si se llega al lado extremo del oponente.

Se tiene:

f(x) = 0.25 * F1 + 0.5 * F2 + 0.25 * F3

Donde:

F1: Es la distancia al extremo opuesto del contrincante. Se considera pesos por cada fila, por tanto es la cantidad de filas que faltan para llegar al otro extremo y así convertirse la ficha en reina. La función se basa en la regla de defender los extremos para impedir que el ponente logre el objetivo.

En el caso del movimiento de la máquina

F1 = x1*x1 ; x1: número de fila de la posición inicial

En el caso del movimiento del humano

F1 = (7-x1)*(7-x1) ; x1: número de fila de la posición inicial

F2: Es el valor que se asigna, dependiendo que se coma una ficha o reina.

En el caso de fichas rojas y negras

F2= 200 // 25

En el caso de ser reinas blancas o negras, su valor es mayor en comparación con la de las fichas debido a que ellas cumplen el objetivo.

F2 = 300 // 50

F3: Se considera un puntaje cuando la ficha
llega al extremo opuesto de su competidor y se convierte en
reina.

En el caso de fichas rojas y
negra

F3 = 80 // 100

Resolución Torres de Hanoi

```
torresHanoi( discos,  torre1,  torre2,  torre3){
      // Caso Base
 si (discos==1){
         Escribir ("Mover disco de Torre " + torre1 + " a Torre " +
torre3);
      } sino  {
          Dominio

          Llamamos a la función de tal forma que decrementamos
          el número de discos, y seguimos el algoritmo
    (origen, destino, auxiliar)
         Hanoi(discos-1, torre1, torre3, torre2);
         Escrinbir ("Mover disco de Torre " + torre1 + " a Torre " +
torre3);
          En esta ocasión siguiendo el algoritmo hacemos lo siguiente
          (auxiliar, origen, destino)
          Hanoi(discos-1, torre2, torre1, torre3);
      }

  }
```

Clase Anillo

Publico Anillo()

Generar numero random

//tres colores bases

float r = numero random

float g = numero random

float b = numero random

Color colorAnillo = new Color(r, g, b);

Mandar al panel

Declarar el color del panel

}

ClASE JUEGO

agregarAros(int n) {

cadena de = "aro";

Anillo aro = new Anillo();

Añadir aros

for (int i = 1; i <= n - 1; i++) {

torre1.añade(nuevo Anillo());

}

```
organizar(int n) {
    si (n >= 0) {
      for (int j = 1; j <= n - 1; j++) {

        //panel Anterior
        JPanel anterior = (JPanel) torre1.getComponent(j - 1);
        //posiciones y tamaño del aro anterior
        int x = panel Anterior X();
        int y = panel AnteriorY();
        int w = panel Anterior();
        int h = panel Anterior();

        //Panel que se va a modificar
        aroA = (JPanel) torre1
        aroA.Medidad(x, y - h, w, h);
        anterior.(x - 10, y, w + 20, h);
      }
      organizar(n - 1);
    }

  }

    organizar(n);

    torre1.acftualizar
```

Clase Torre

```
 Constructor de la clase torre
public Torre() {
llama la interfaz
}
Metodo que dibuja la torre en el panel
{
}
```

Clase TorresHanoi

```
String cadena;
ArrayList<String> cadenas;
guarda el numero de pasos completos
int contador;

TorresDeHanoi() {
    cadena = "";
    cadenas= new ArrayList<>();
    contador=0;
}
hanoi( num,  inicio,  auxiliar,  fin) {
    si (num == 1) {
        cadena = "MOVER DE LA TORRE " + inicio + " A LA TORRE " + fin;
        contador++;
        cadenas.add(cadena);
```

```
    } sino {
        hanoi(num - 1, inicio, fin, auxiliar);
        cadena = "MOVER DE LA TORRE " + inicio + " A LA TORRE " + fin;
        contador++;
        cadenas(cadena);

        hanoi(num - 1, auxiliar, inicio, fin);
    }
}

        }
    }

}

 * Meotodo para verificar si ya acabo el juego

verificarFinalJuego(int n, int numeroArosTorre3){

    return n==numeroArosTorre3;
}

/**
 * Metodo que retorna el minimo de movimientos segun el numero de aros
```

```
public int getContador() {
    return contador;
}

 * Metodo para setear el contador
 setContador(int contador) {
    tcontador = contador;
}
```