Pontificia Universidad Javeriana

Entrega final

Asignatura

Analisis de Algoritmos

Profesor

Leonardo Florez Valencia

Integrantes

Andrés David Mariño Sánchez
Daniel Esteban Valero Galeano
Diego Alejandro Mateus Cruz
Jonathan Esteban Molina Castañeda

31 de mayo de 2019

1. Introducción

Damas es un juego de mesa para dos contrincantes, en el cual cada jugador posee 12 fichas. El objetivo del juego consiste en derrotar al adversario dejándolo sin piezas o no permitir que haga una jugada.

2. Elementos del juego

El tablero del juego está representado por una matriz de tipo char con dimensiones 8x8, contiene dos tipos de espacios, donde puede estar una ficha y donde no, en los espacios que existe la posibilidad de que se encuentre una ficha tiene opción para que se encuentren 4 diferentes fichas, están son:

- **Fichas blancas normales:** Representan las fichas del primer jugador y solo se pueden mover un espacio en diagonal hacia la parte contraria del tablero de donde empezó a jugar, puede mover más espacios si come (se describe más adelante) y se representan por el carácter 'x'.
- **Fichas negras normales:** Representan las fichas del segundo jugador y solo se pueden mover un espacio en diagonal hacia la parte contraria del tablero de donde empezó a jugar, puede mover más espacios si come (se describe más adelante) y se representa por el carácter 'o'.
- **Fichas blancas coronadas:** Representan las fichas del primer jugador que han llegado al extremo contrario del tablero (el extremo del contrario) por lo cual tienen la posibilidad de mover de forma vertical ya sea hacia arriba o abajo, igualmente pueden comer y son representadas por el carácter 'X'.
- **Fichas negras coronadas:** Representan las fichas del segundo jugador que han llegado al extremo contrario del tablero (el extremo del contrario) por lo cual tienen la posibilidad de mover de forma vertical ya sea hacia arriba o abajo, igualmente pueden comer y son representadas por el carácter 'O'.

En aquellos espacios que no se pueda encontrar una ficha estarán representados por el carácter '-'.

3. Jugadas

Las jugadas identificadas que pueden realizar las fichas, independientemente del tipo que sean, y las cuales se van a implementar son las siguientes:

- **Comer :** Una ficha puede moverse en una diagonal de dos casillas, si la casilla final a la que se mueve esta libre y si en la casilla intermedia se encuentra una ficha del oponente. Si esto ocurre la ficha del oponente se retira del juego. Si existe la posibilidad de realizar esta jugada, deberá ser obligatoria.

- **Asegurar ficha en peligro:** Si en el siguiente turno del oponente se ve la posibilidad de que puedan comer una ficha propia, esta se asegurara con otra ficha, poniéndola en la diagonal en la cual va a terminar la ficha oponente, así evitando que en el siguiente turno del oponente este coma una ficha.

- **Coronar:** Si exite la posibilidad de que una ficha llegue al extremo del contrario, este movimiento se hara para tener una ficha reina.

- **Mover reina sin riesgo:** Dado que la reina se puede mover verticalmente en cualquier sentido es una ficha muy valiosa, por lo cual se pretende moverla la mayor cantidad posible sin que esté en riesgo.

4. Estados del juego

El juego puede tener distintos estados, los cuales son:

- **Jugador oponente sin fichas:** Si a medida que se juegue se comen todas las fichas del jugador oponente, pasara al estado del juego en que el oponente se quedó sin fichas, por lo cual se nos dará la victoria.

- **Jugador sin fichas:** Si el oponente se come todas las fichas del jugador, pasara al estado del juego en que el jugador se quedó sin fichas, por lo cual se dirá que el jugador oponente gano la partida.

- **Oponente sin movimientos posibles:** Dado que el jugador acorrale al jugador oponente dejándolo sin poder realizar movimientos posibles, se dirá que el jugador oponente quedo sin movimientos posibles, dándole la victoria al jugador.

- **Jugador sin movimientos posibles:** Dado que el jugador oponente acorrale al jugador dejándolo sin poder realizar movimientos posibles, se dirá que el jugador quedo sin movimientos posibles, dándole la victoria al jugador oponente.

- **Bucle de 5 jugadas:** Si se repite un mismo ciclo de jugadas más de 5 veces gana el jugador que tenga más fichas. Si ambos jugadores tienen la misma cantidad de fichas, se considerará empate.

- **Posibilidad de jugar:** Dado que no se cumpla ninguna condición anteriormente mencionadas se podrá jugar con normalidad.

5. Formalizacion

5.1. Análisis del problema

Se busca crear un "bot"que pueda jugar el juego Damas Chinas (también conocido como Checkers) para que pueda jugar contra otro "bot"mediante la comunicación en pipes que provee un árbitro ya implementado.

5.2. Entradas:

La entrada es una matriz de tablero E que contiene las fichas propias y del oponente. Este tablero va cambiando según el transcurso de juego. A su vez se recibe la ficha del jugador que esta se represnte con J.

5.3. Salidas:

Se retornará un arreglo de coordenadas, la primera posición del arreglo será las coordenadas de la ficha que se desea mover, y las siguientes coordenadas serán las posiciones en que se desplazó dicha ficha.

5.4. Tipo de problema:

Al tener que escoger que ficha se tiene que mover y a qué lugar se tiene que mover la misma, se define como un problema de **desicion**.

5.5. Clase de problema:

Dado que el problema es de decisión, el cual se puede comprobar polinomialmente comprobando si la jugada es válida y al ser super-polinomial. Se declara que es un problema **NP-Completo**

6. Heurística

La heurística que se va a usar pretende dejar al jugador oponente sin movimientos, comiendo cada vez que sea obligatorio, por eso se validaran las siguientes jugadas que están ordenadas de una manera de prioridad mayor a menor, por lo cual, si evalúa que se puede ejecutar una lo hará, este orden es:

- **Comer :** Dada que esta jugada es obligatoria será la primera en validar, pero dado el caso que más de una ficha pueda realizar dicho movimiento, se procederá a realizar aquella que al comer no se la coman después, el siguiente criterio será aquella que más coma, y por ultimo aquella que este más a la izquierda.
- **Asegurar ficha próxima a que se la coman:** Se verificará todas nuestras fichas posibles próximas a que se las coman y se intentará salvar las reinas o la que más arriba este y a la izquierda. Primero se intentará salvar poniendo una ficha en sus diagonales, dependiendo de donde se la vayan a comer, y si esto no es posible se evaluará si ella se puede mover para que no se la coman.
- **Coronar:** Si existe una ficha próxima a coronar se hará este movimiento, si existen más de dos fichas próximas coronar, se escogerá la que este más a la izquierda.

- **Mover reina sin riesgo:** Se procederá a mover la reina más cercana a donde se encuentre la mayor concentración de fichas enemigas, se decidirá donde hay más fichas enemigas dividiendo el tablero en 4 partes iguales y dicha reina se moverá a dicha zona, si la reina ya está en esa zona se moverá otra reina, solo se moverá si no se pone en riesgo.
- **Mover la de más arriba:** Se procederá a mover la ficha que esté más cerca al borde del enemigo de forma segura, si hay más de una ficha que tenga esta opción se procederá a mover la que este más a la izquierda.

7. Pseudocodigo

Algorithm 1 Coronar

```
1: procedure CORONAR(E,M,J)
2:     cont = 1
3:     for i ← 0 in range |M| do
4:         if J in J1['Pieces'] then
5:             if M[i][−1][0] == J1['crown'] then
6:                 return M[i]
7:             end if
8:         end if
9:         if J in J2['Pieces'] then
10:            if M[i][−1][0] == J2['crown'] then
11:                return M[i]
12:            end if
13:        end if
14:    end for
15: end procedure
```

Algorithm 2 Mover Seguro

procedure MovUp(E,M,J)
  for i in range $|M|$ **do**
    if E $[M[i][0][0]]$ $[M[i][0][1]]$ in $J2$['pieces'][0] **then**
      if $M[i][0][1] > M[i][1][1]$ **then**
        if $(E[(M[i][1][0]) + 1][(M[i][1][1] + 1)]$ not in $J2['adversary'])$ and $E[(M[i][1][0]) + 1][(M[i][1][1]) - 1] == BLANCO$ and $E[(M[i][1][0]) - 1][(M[i][1][1]) + 1] == BLANCO$ **then**
          Return $M[i]$
        else if $E[(M[i][1][0]) + 1][(M[i][1][1]) + 1]$ not in $J2['adversary']$ and $(E[(M[i][1][0]) + 1][(M[i][1][1] - 1)]! = BLANCO$ and $E[(M[i][1][0]) - 1][(M[i][1][1]) + 1]! = BLANCO)$ **then**
          Return $M[i]$
        else if $E[(M[i][1][0]) + 1][(M[i][1][1]) + 1]$ not in $J2['adversary']$ and $(E[(M[i][1][0]) + 1][(M[i][1][1] - 1)]$ not in $J2['adversary'])$ and $E[(M[i][1][0]) - 1][(M[i][1][1]) + 1] == BLANCO$ **then**
          Return $M[i]$
        else if $E[(M[i][1][0]) + 1][(M[i][1][1]) + 1]$ not in $J2['adversary']$ and $E[(M[i][1][0]) + 1][(M[i][1][1] - 1)]! = BLANCO$ and $E[(M[i][1][0]) - 1][(M[i][1][1]) + 1]$ in $J2['adversary'][1]$ **then**
          Return $M[i]$
        else if $E[(M[i][1][0]) + 1][(M[i][1][1]) + 1]$ not in $J2['adversary']$ and $E[(M[i][1][0]) + 1][(M[i][1][1] - 1)] == BLANCO$ and $E[(M[i][1][0]) - 1][(M[i][1][1]) + 1]$ not in $J2['adversary'][1]$ **then**
          Return $M[i]$
        end if
      else:
        Return $M[i]$
      end if
    end if
  end for
end procedure

Algorithm 3 Mover Seguro

if $M[i][0][1] > M[i][1][1]$ **then**
  if $(E[(M[i][1][0]) + 1][(M[i][1][1] - 1)]$ not in $J2['adversary']$) and $E[(M[i][1][0]) + 1][(M[i][1][1]) + 1] == BLANCO$ and $E[(M[i][1][0]) - 1][(M[i][1][1]) - 1] == BLANCO$ **then**
    Return $M[i]$
  else if $E[(M[i][1][0]) + 1][(M[i][1][1]) - 1]$ not in $J2['adversary']$ and $(E[(M[i][1][0]) + 1][(M[i][1][1] + 1)]! = BLANCO$ and $E[(M[i][1][0]) - 1][(M[i][1][1]) - 1]! = BLANCO)$ **then**
    Return $M[i]$
  else if $E[(M[i][1][0]) + 1][(M[i][1][1]) - 1]$ not in $J2['adversary']$ and $(E[(M[i][1][0]) + 1][(M[i][1][1] + 1)]$ not in $J2['adversary']$) and $E[(M[i][1][0]) - 1][(M[i][1][1]) - 1] == BLANCO$ **then**
    Return $M[i]$
  else if $E[(M[i][1][0]) + 1][(M[i][1][1]) - 1]$ not in $J2['adversary']$ and $E[(M[i][1][0]) + 1][(M[i][1][1] + 1)]! = BLANCO$ and $E[(M[i][1][0]) - 1][(M[i][1][1]) - 1]$ in $J2['adversary'][1]$ **then**
    Return $M[i]$
  else if $E[(M[i][1][0]) + 1][(M[i][1][1]) - 1]$ not in $J2['adversary']$ and $E[(M[i][1][0]) + 1][(M[i][1][1] + 1)] == BLANCO$ and $E[(M[i][1][0]) - 1][(M[i][1][1]) - 1]$ not in $J2['adversary'][1]$ **then**
    Return $M[i]$
  else:
    Return $M[i]$
  end if
end if
Return $rand.Choice(M)$

Algorithm 4 Asegurar una ficha

procedure ASEGURAR(E,M,J)
  $s = []$
  if J in $J2['pieces']$ **then**
    $s = estaEnPeligro(E, M, J)$
    if $s! = None$ **then**
      for k in range $(len(M))$ **do**
        if $M[k][1][0] == s[0]$ and $M[k][1][1] == s[1]$ **then**
          Return $M[k]$
        end if
      end for
    end if
  end if
  Return coronar(E, M, J)
end procedure

Algorithm 5 Ficha en peligro

```
1: procedure estaEnPeligro(E,M,J)
2:     for i in range (8) do
3:         for j in range (8) do
4:             if j + 1 < 8 and i + 1 < 8 and E[i][j] in J2['pieces'] and E[i − 1][j − 1] in J2['adversary'] then
5:                 if E[i + 1][j + 1] == BLANCO then
6:                     Return i + 1, j + 1
7:                 end if
8:             else if j+1 < 8 and i+1 < 8 and E[i][j] in J2['pieces'] and E[i+1][j−1] == BLANCO then
9:                 if E[i − 1][j + 1] in J2['adversary'] then
10:                    Return i + 1, j − 1
11:                end if
12:            else if j + 1 < 8 and i + 1 < 8 and E[i][j] in J2['pieces'] and E[i + 1][j − 1] in J2['adversary'][1] then
13:                if E[i − 1][j + 1] == BLANCO then
14:                    Return i − 1, j + 1
15:                end if
16:            else if j+1 < 8 and i+1 < 8 and E[i][j] in J2['pieces'] and E[i−1][j−1] == BLANCO then
17:                if E[i + 1][j + 1] in J2['adversary'][1] then
18:                    Return i − 1, j − 1
19:                end if
20:            end if
21:        end for
22:    end for
23:    Return None
24: end procedure
```

Algorithm 6 Realizar posibles movimientos

```
1: procedure POSIBLES(M,a,b)
2:     aux = 0
3:     mAUX
4:     ma[]
5:     if a − 1 <> 0 then
6:         mAUX = M
7:         aux = M[a − 1][b]
8:         mAUX[a − 1][b] = 9
9:         mAUX[a][b] = aux
10:        ma.add(mAUX)
11:    end if
12:    if b − 1 <> 0 then
13:        mAUX = M
14:        aux = M[a][b − 1]
15:        mAUX[a][b − 1] = 9
16:        mAUX[a][b] = aux
17:        ma.add(mAUX)
18:    end if
19:    if a + 1 <> 4 then
20:        mAUX = M
21:        aux = M[a + 1][b]
22:        mAUX[a + 1][b] = 9
23:        mAUX[a][b] = aux
24:        ma.add(mAUX)
25:    end if
26:    if b + 1 <> 4 then
27:        mAUX = M
28:        aux = M[a][b + 1]
29:        mAUX[a][b + 1] = 9
30:        mAUX[a][b] = aux
31:        ma.add(mAUX)
32:    end if
33:    Return ma
34: end procedure
```

8. Invariante

Se recibe el estado actual del tablero y se retorna un movimiento de parte del aplicativo, dicho movimiento tiene que ser valido acorde a las reglas establecidas previamente.

9. Complejidad

El funcionamiento del programa se basa en la utilizacion de una función moves(), la cual utiliza 3 ciclos anidados para sacar las posibles jugadas, debido a esto la complejidad general del algoritmo es

$$O(n^3)$$

Para nuestra heurística utilizamos diferentes niveles de verificacion (if, elseif) y solo se utilizo 2 ciclos anidados en la función estaEnPeligro()

$$O(n^2)$$

10. Pruebas

Para probar el funcinamiento del aplicativo se realizaron pruebas con otro aplicativo que hace movimientos aleatorios, contra distintos humanos y contra si mismo, dicho esto, se miro la cantidad de victorias de 10 partidas por cada contrincante, y se midio el tiempo que tardo nuestro aplicativo y el total de turnos para que se acabara la partida. Los resultados fueron los siguientes:

- **Aplicativo contra aleatorio:**

Victoria	**Turnos**	**Tiempo (s)**
si	18	0.001578
si	27	0.002036
si	18	0.001473
si	26	0.001121
si	39	0.002287
si	33	0.001678
si	28	0.007149
si	103	0.005250
si	27	0.010259
si	28	0.002926

Cuadro 1: Aplicativo vs aleatorio

  El aplicativo gano el 100 porciento de las veces contra un aplicativo que solo realizaba movimientos aleatorios, por lo que se supuso que es mas efectiva nuestra heuristica.

- **Aplicativo contra humanos**

Victoria	Turnos	Tiempo (s)
si	28	0.01415
si	22	0.02686
si	24	0.01285
si	41	0.02124
no	32	0.01450
si	22	0.04943
si	28	0.03185
si	23	0.00774
si	27	0.00616
no	26	0.00487

Cuadro 2: Aplicativo vs humanos

- **Aplicativo contra si mismo**

Turnos	Tiempo (s)
71	0.02279
49	0.01555
62	0.01735
54	0.01998
94	0.02435
167	0.03798
73	0.023555
72	0.023871
117	0.031154

Cuadro 3: Aplicativo vs Aplicativo

Cuando el aplicativo jugaba contra si mismo, se pudo observar que se demoraba mas turnos en acabar la partida con respecto a sus demas contrincantes.