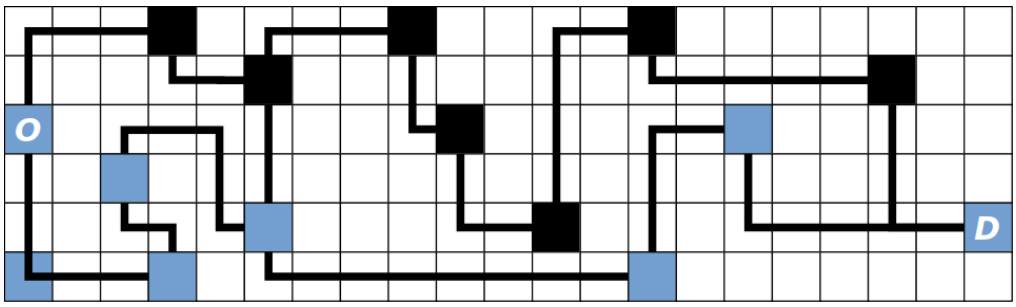
Lista1 MAC0425

Fernanda Itoda 10740825

Exercício 1



f = g + h		
f	g	h
21	0	21
22	2	20
21	2+2	17
24	2+2+2	18
25	2+2+2+5	14
26	2+2+2+5+8	7
29	2+2+2+5+8+4	6
29	2+2+2+5+8+4+6	0

Exercício 2

Com a cobrança de pedágios, a distância de Manhattan deixa de ser uma heurística admissível, ou seja, considerando $f(n) = g(n) + h(n)$, é possível que a função heurística $h(n)$ passe a superestimar o custo para alcançar o objetivo, pois desconsidera o valor atribuído ao pedágio. É possível adaptar a atual heurística admissível se considerarmos a distância de Manhattan somada ao custo de pedágio. Dessa forma, teríamos que $h(n) = P_h(n) + D_h(n)$, e $g(h) = P_g(n) + D_g(n)$, onde $D(n)$ é o custo calculado pela distância de Manhattan, e $P(n)$ é o custo do pedágio no nó n.

Exercício 3

É possível instituir o preço do pedágio por um custo fixo a cada 4 células percorridas, e para a heurística admissível, consideramos o caso mais otimista, onde a cada no máximo 3 células encontra-se a próxima cidade. Ou seja, aplicaríamos o algoritmo A* considerando $f(n) = X(n) + h(n)$, onde, sendo P o custo minimo de pedágio (4 células), D o número de células percorridas, segundo a distância de Manhattan, e X o custo final, definimos:

$$X = \begin{cases} D & \text{se} \quad D < 4 \\ D + \frac{D}{4} \times P & \text{se} \quad D \geq 4 \end{cases} \tag{1}$$

Assim, supondo que P = 10, temos:

f = g + h		
f	g	h
21	0	21
22	2	20
21	2+2	17
24	2+2+2	18
35	2+2+2+(5+10)	14
56	2+2+2+(5+10)+(8+20)	7
69	2+2+2+(5+10)+(8+20)+(4+10)	6
79	2+2+2+(5+10)+(8+20)+(4+10)+(6+10)s	0

Exercício 5

```
função minimax (currentPlayer)
    available = availableMoves ()

    Checa se o tabuleiro atual contém um vencedor

    // houve vencedor
    if (winner == otherPlayer)
        if (otherPlayer == computer)
        // computador vence (retorna pontos: 0 e posição nula)
        return 0, 1 * (nAvailableMoves + 1)
    else
        // computador perde (retorna pontos ¡ 0 e posição nula)
        return 0, (-1) * (nAvailableMoves + 1)

    // se não houver movimentos disponíveis: empate (retorna 0 pontos e posição)
    else if (nAvailableMoves == 0)
        return 0
```

```
// seta valor inicial para comparação
if (currentPlayer == computer)
    bestPoints = -1000
else
    bestPoints = +1000

// testa todas as possibilidades
for (i = 0; i < nAvailableMoves; i++)
    // simula jogada
    board[available[i]] = currentPlayer
    nAvailableMoves–
    // jogada com repetição e sem repetição
    currentPosition, currentPoints = minimax (otherPlayer)
    currentPosition, currentPoints += minimax (currentPlayer)

    // desfaz movimento
    board[available[i]] = ' '
    nAvailableMoves++
    winner = 'N'
    // salva posição que acabou de ser testada
    currentPosition = available[i]

    if (currentPlayer == computer)
        if (currentPoints > bestPoints)
            bestPosition = currentPosition
    else
        if (currentPoints < bestPoints)
            bestPosition = currentPosition

return bestPosition, bestPoints
```