Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Inteligência Artificial

Trabalho sobre Métodos de busca

Leonardo V. Eichstaedt Makhles Reuter Lange

5 de Setembro de 2016

1 Estruturas de dados escolhidas

Escolheu-se representar o tabuleiro como uma matrix quadrada de tamanho 15. Sabe-se que no início do jogo e em boa parte deste, a maior parte das casas em que se pode jogar estará vazia e, se nenhum outro tipo de estrutura que determine quais casas devem ser verificadas no cálculo da heurística, há um disperdício de processamento, porém nesta primeira implementação ainda não se está considerando a otimização das computações, pois a heurística ainda está sendo formulada e não há uma implementação efetiva desta. A matriz pode ser vista no construtor da classe `Board`:

```
class Board():
    BoardWidth = 15
    BoardHeight = 15

    def __init__(self):
        self.board = [[0 for i in range(self.BoardWidth)] for ii in range(self.BoardHeight)]
        self.pieces = 0

    ...
```

Uma estrutura a ser considerada é a utilização de um *hash map* que mapeie posições ocupadas às pedras que as ocupam. Nesse caso, tem-se um ganho no consumo de memória utilizada, pois apenas as posições efetivamente ocupadas por pedras brancas ou pretas fazem parte do mapeamento, porém não há nenhum ganho em processamento, visto que uma matriz quadrada de tamanho 15 pode ser armazenada tranquilamente em uma memória cache.

2 Heurística e utilidade

$$h(n) = \sum_{i=1}^{5} 10^{2*i} \times seq_i \tag{1}$$

onde seq_i representa a quantidade de sequências de i pedras consecutivas ou semi-consecutivas, desde que haja a possibilidade de se alcançar 5 pedras consecutivas. Por exemplo, na configuração do tabuleiro da Figura 1, tem-se:

$$
\begin{aligned}
h(n) = & 10^2 \times C_{diagonal2} + E_{diagonal1} + F_{vertical} + F_{diagonal2} + \\
& 10^4 \times BE_{diagonal2} + DE_{horizontal} + \\
& 10^6 \times ABC_{horizontal} + CDF_{diagona1}
\end{aligned} \tag{2}
$$

$$
h(n) = 4 \times 10^2 + 2 \times 10^4 + 2 \times 10^6 \tag{3}
$$

onde *diagonal1* se refere a uma reta de inclinação positiva e *diagonal2* a uma reta de inclinação negativa. Futuramente, pensa-se em incluir também alguma maneira de conferir mais pontos às configurações em que se pode preencher mais de 5 pedras. Por exemplo, utilizando-se a mesma imagem, a sequência `ABC` pode gerar as quíntuplas `xABCx` e `ABCxx`, o que deve conferir mais pontos que a configuração `FxDCx`, que é a única forma de se fazer uma quíntupla com as pedras `C`, `D` e `F`.

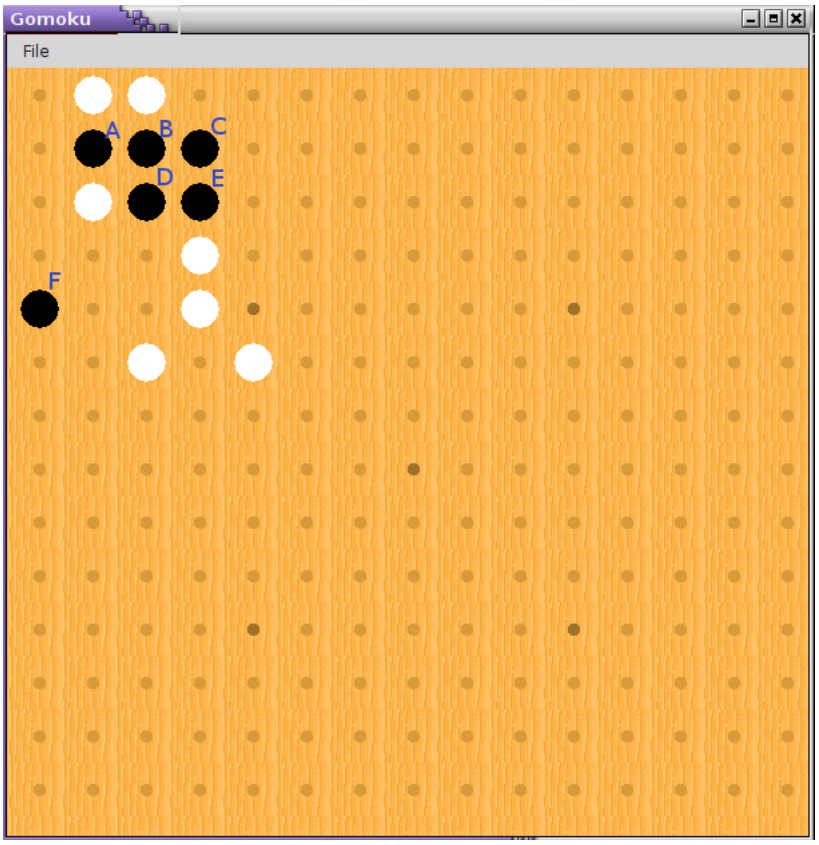

Figura 1: Execução do próprio jogo que está sendo desenvolvido.

3 Detecção do Fim de Jogo

Nessa implementação parcial do jogo, faz-se a detecção do fim do jogo no método `checkWin` da classe `Board`, conforme o trecho de código abaixo apresentado.

```
def checkWin(self, m, n, player):

    count = 1
    self.pieces = self.pieces + 1
    if (self.pieces == 15 * 15):
        return DRAW

    # Vencendo pela diagonal baixo direita
    for i in range(1, 5):
```

```
    if (m - i >= 0 and n - i >= 0 and self.board[m - i][n - i] == player):
        count = count + 1
    else:
        break
if (count == 5):
    return WIN
else:
    count = 1

# Vencendo pela diagonal cima direita
for i in range(1, 5):
    if (m + i <= 14 and n - i >= 0 and self.board[m + i][n - i] == player):
        count = count + 1
    else:
        break
if (count == 5):
    return WIN
else:
    count = 1

# Vencendo pela diagonal baixo esquerda
for i in range(1, 5):
    if (m - i >= 0 and n + i <= 14 and self.board[m - i][n + i] == player):
        count = count + 1
    else:
        break
if (count == 5):
    return WIN
else:
    count = 1

# Vencendo pela diagonal cima esquerda
for i in range(1, 5):
    if (m + i <= 14 and n + i <= 14 and self.board[m + i][n + i] == player):
        count = count + 1
    else:
        break
if (count == 5):
    return WIN
else:
    count = 1

# Vencendo pela esquerda
for i in range(1, 5):
    if (n + i <= 14 and self.board[m][n + i] == player):
        count = count + 1
    else:
        break
if (count == 5):
    return WIN
else:
    count = 1

# Vencendo pela direita
for i in range(1, 5):
    if (n - i >= 0 and self.board[m][n - i] == player):
        count = count + 1
    else:
```

```
            break
    if (count == 5):
        return WIN
    else:
        count = 1

    # Vencendo por Cima
    for i in range(1, 5):
        if (m + i <= 14 and self.board[m + i][n] == player):
            count = count + 1
        else:
            break
    if (count == 5):
        return WIN
    else:
        count = 1

    # Vencendo por Baixo
    for i in range(1, 5):
        if (m - i >= 0 and self.board[m - i][n] == player):
            count = count + 1
        else:
            break
    if (count == 5):
        return WIN
    else:
        count = 1

    return CONTINUE
```

A função pode retornar os seguintes valores:

- **CONTINUE** - o jogo está em andamento (retorno padrão);
- **WIN** - o jogo terminou com a vitória de um dos jogadores;
- **DRAW** - o jogo terminou em um empate.

4 Detecção de Sequências de 4 Peças

5 Referências Bibliográficas