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

Trabalho sobre Métodos de busca

Leonardo V. Eichstaedt Makhles Reuter Lange

6 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.

Pensa-se ainda em criar quatro listas referentes às direções de percorrimento do tabuleiro a partir de qualquer pedra: horizontal, vertical e diagonais. Essas listas podem conter as pedras que devem ser percorridas em suas respectivas direções para contabilizar os pontos. Nesse caso, se em uma dada configuração do tabuleiro foi determinado que uma pedra não irá pontuar na direção horizontal (por não haver a possibilidade de se formar uma quíntupla), então essa pedra é removida da lista horizontal. Assim, nas aplicações seguintes da heurística, não há a necessidade de se percorrer horizontalmente a estrutura de dados a partir daquela pedra.

2 Heurística e utilidade

A função heurística $h(n)$, onde n se refere à configuração do tabuleiro, pode ser composta de diversas outras definições. A principal heurística a ser utilizada nesse trabalho é dada pela Equação 1.

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

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 n do tabuleiro da Figura 1, tem-se:

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

$$h(n) = 4 \times 10^2 + 2 \times 10^4 + 2 \times 10^6$$

onde *diagonal1* se refere a uma reta de inclinação positiva e *diagonal2* a uma reta de inclinação negativa.

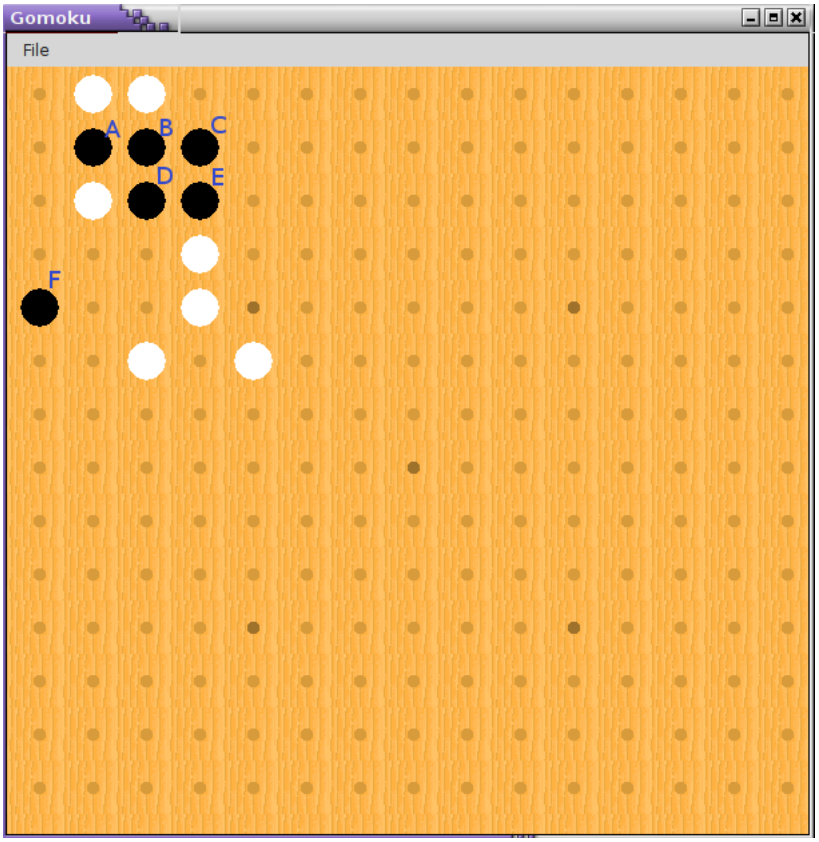

Figura 1: Execução do próprio jogo que está sendo desenvolvido.

A implementação da heurística no código ainda está em fase de desenvolvimento. Não se está utilizando os pesos definidos na Equação 1 ainda, apenas soma-se um valor de 10 para cada peça em uma sequência, desde que se possa chegar a uma quíntupla.

Futuramente, pensa-se em incluir também alguma maneira de conferir mais pontos às configurações em que se pode preencher 5 pedras com mais de uma forma. 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`.

Tem-se ainda a possibilidade de incluir heurísticas referentes a determinadas configurações de pedras que levarão a uma vitória independentemente da jogada do humano.

Uma outra abordagem a ser considerada é como fazer a pontuação nas situações em que a configuração do tabuleiro está favorável para a AI, porém muito mais favorável para o jogador humano.

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
```

Conta-se o número de pedras conectadas à pedra recém-jogada e retorna-se um dos 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.