Universidade Federal do Rio Grande do Norte
Departamento de Engenharia de Computação e Automação
Controle Inteligente

Jogo da Velha com Min-Max

Aluno: Daniel Silva de Morais Mat.: 20170010123
Professor: Fábio Meneghetti

9 de maio de 2018
Natal/RN

Conteúdo

1 Introdução **1**

2 Descrição **3**
2.1 Implementação do Min-Max 3
2.2 Estratégia de Nível 4

3 Análise dos Resultados **5**

4 Conclusão **6**

Referências **6**

1 Introdução

Para resolver um problema quando não há computação clara para uma solução válida, nos voltamos para a localização do caminho (Path Finding). Há várias abordagens de busca de caminhos relacionados, como por exemplo árvores de jogos e arvore de buscas. Essas abordagens baseiam-se em uma estrutura comum, ou seja, uma árvore de estados cujo nó raiz representa o estado inicial e as arestas representam movimentos potenciais que transformam o estado em um novo estado. As buscas são desafiadoras porque a estrutura subjacente pode não ser calculada em sua totalidade devido à explosão do número de estados [1].

Assim sendo, há o algoritmo Min-Max, que é o núcleo de várias inteligências artificiais (IA) para construir decisões sobre o melhor movimento. Esse algoritmo encontra a melhor jogada para uma IA em um estado de jogo de dois jogadores, tais como jogos de Damas, Xadrez, Velha e entre outros.

Dada uma posição específica em uma árvore de jogo a partir da perspectiva de um jogador inicial, um programa de busca deve encontrar um movimento que leve à maior chance de vitória (ou pelo menos um empate). Em vez de considerar apenas o estado atual do jogo e os movimentos disponíveis naquele estado, o programa deve considerar quaisquer contra-ataques que seu oponente fará depois de fazer cada movimento. O programa assume que existe uma pontuação de função de avaliação (estado, jogador) que retorna um inteiro representando a pontuação do estado do jogo na perspectiva do jogador; números inteiros inferiores (que podem ser negativos) refletem posições mais fracas.

A Figura 1 representa a estrutura de uma árvore utilizando o algoritmo Min-Max. Ela faz a avaliação de um movimento usando o algoritmo com profundidade 3. A linha inferior da árvore do jogo contém os cinco estados de jogo possíveis, que resultam depois que o jogador faz um movimento, o oponente responde, e então o jogador faz um movimento. Cada um desses estados do jogo é avaliado do ponto de vista do jogador inicial, e a classificação inteira é mostrada em cada nó. A segunda linha MAX da parte inferior contém nós internos cujas pontuações são o máximo de seus respectivos filhos. Do ponto de vista do jogador inicial, estes representam as melhores pontuações que ele pode atingir. No entanto, a terceira linha MIN da parte inferior representa as piores posições que o adversário pode forçar no jogador, portanto, suas pontuações são o mínimo de seus filhos. Cada nível alterna entre selecionar o máximo e o mínimo de seus filhos. A pontuação final demonstra que o jogador original pode forçar o oponente a um estado de jogo que avalia a 3.

Tendo isso em vista, este relatório tem por objetivo descrever o desenvol-

vimento de um programa de Jogo-da-Velha utilizando o algoritmo Min-Max. O utilizador do software joga contra o computador, que utiliza o algoritmo como IA. A variação do nível de dificuldade é feita modificando a profundidade do Min-Max.

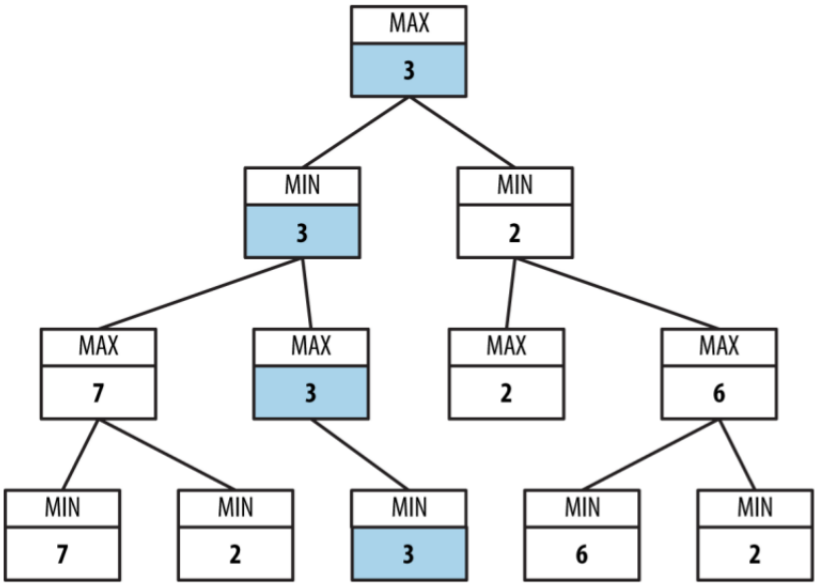

Figura 1: Exemplo de árvore de jogo com Min-Max

2 Descrição

Para o desenvolvimento do programa foi utilizada a linguagem de programação `Python 3` com `PyQt5`. O programa é dividido em 4 arquivos `.py`. Há duas classes, `tabuleiro.py` e `point.py`, que formam a estrutura básica para o jogo, enquanto `jvelha.py` é responsável de fazer as movimentações do jogo e `main.py` possui a interface do Jogo-da-Velha.

2.1 Implementação do Min-Max

No desenvolvimento da função `minimax(tab, prof, player)`, foi utilizada a recurção, de forma que foi possível a criação da "árvore"e obtenção dos valores ótimos sem a necessidade de construir sua estrutura propriamente dita. A função possui 3 parâmetros, `tab`: tabuleiro com o estado atual do jogo, `prof`: profundidade da árvore e `player`: jogador atual do estado.

A seguir, temos implementação da função:

```
def minimax(self, tab, prof, player):
    if player == 1: # 1 PC | -1 HUMANO
        best = [-1, -1, -inf]
    else:
        best = [-1, -1, +inf]
    if prof == 0 or self.isVencedor(tab, 1) or
                                self.isVencedor(tab, -1):
        score = self.heuristica(tab)
        return [-1, -1, score]

    for item in tab.getNone(): #lista de None points
        tab.setLocal(item, player)
        score = self.minimax(tab, prof-1, -player)
        tab.setLocal(item, None)
        score[0], score[1] = item.x, item.y
        if player == 1:
            if score[2] > best[2]:
                best = score  # max value
        else:
            if score[2] < best[2]:
                best = score  # min value

    return best     #retorna x y score
```

Foram utilizadas outras funções para complementar `minimax()`, como a função `heuristica(tab)` e `isVencedor(tab, x)` com `x` assumindo 1 ou -1, que são as representação na tabela para o jogador computador e o jogador humano, respectivamente.

A função `heuristica(tab)` é responsável por calcular o valor associado ao "nó"da árvore de jogo, sendo definida simplesmente como 1 se o computador vencer, -1 se o jogador vencer e 0 em caso de empate. A seguir temos sua definição:

```
    def heuristica(self, tab):
        if self.isVencedor(tab, 1):
            return 1
        elif self.isVencedor(tab, -1):
            return -1
        else:
            return 0
```

As outras funções são bastantes simples e fáceis de entender, então não serão descritas nesse relatório, mas será disponibilizado os códigos.

2.2 Estratégia de Nível

O programa é composto por dois níveis de dificuldade, um *fácil* e um *difícil*. Inicialmente a implementação do Min-Max foi feita para funcionar imbatível, em seguida, foi adequado para facilitar o jogo para o jogador. Isso foi possível modificando apenas a profundidade do Min-Max.

Para o jogo em nível *difícil* a profundidade é máxima, que é a quantidade de espaços vazios no tabuleiro, que vai diminuído a medida que uma nova jogada é feita. Para melhorar no desempenho, caso a célula central do tabuleiro esteja vazia em sua primeira jogada, ela será marcada, pois é sabido que traz resultados melhores.

Para o jogo em nível *fácil* a profundidade é mínima, de forma que o computador tente otimizar apenas a próxima jogada. Desta forma, possibilita que o jogador ganhe.

3 Análise dos Resultados

Para iniciar o programa, é necessário escolher nos *comboBox* quem inicia primeiro e qual o nível. A configuração estará pronta quando apertar o botão *jogar*. Em seguida, é só jogar. Para uma nova partida, deve ser apertado o botão *jogar*, que resetarar o tabuleiro.

As figuras 2, 3 e 4 a seguir apresentam o programa em funcionamento. Como podemos analisar, no nível *fácil* é possível que o jogador ganhe, como já era de esperado. Entretanto, o jogador sempre perderá ou empatará com o computador no nível *difícil*, independente de quem começar.

Figura 2: Jogador ganha em nível fácil.

Figura 3: Empate no nível difícil.

Figura 4: Computador ganha em nível difícil.

4 Conclusão

Desta forma, podemos concluir que o objetivo do trabalho proposto foi cumprido, tendo seus resultados validados com os jogos.

Referências

[1] George T. Heineman Gary Pollice Stanley Selkow. *Algorithms in a Nutshell*. 2016.