Trabalho 1 - parte 2
INE5430 - Inteligência Artificial

Bruno Marques do Nascimento* Johann Westphall †

Florianópolis, 01 de Setembro de 2017

Introdução

Este relatório possui o objetivo de explicar o que foi desenvolvido para a segunda entrega do Trabalho 1 da disiciplina de Inteligência Artifical, ministrada pelo professor Elder Rizzon Santos na Universidade Federal de Santa Catarina(UFSC), campus Florianópolis. O propósito do trabalho é implementar o algoritmo de busca adversária MiniMax com podas α e β. A implementação será testada através do jogo 5 em linha (Gomoku), com tabuleiro tamanho 15x15.

Nesta segunda etapa, serão abordadas novamente as definições matemáticas da função utilidade e heurística, e a implementação completa da função heurística. Além disso, será apresentado o algoritmo de busca adversária *MiniMax* com podas α e β.

A linguagem de programação *C++* foi a escolhida para a implementação do trabalho e para o desenvolvimento da interface gráfica a API gtkmm-3.0.

1 Definições matemáticas

1.1 Utilidade e Heurística

Com relação à heurística a fórmula se manteve. Entretanto foi percebido que o importante são quantas aberturas a sequência possuí e não o simples fato da sequência existir, com exceção da quintupla, por se tratar do fim do jogo.

Os números abaixo foram gerados a partir de certos preenchimentos do tabuleiro:

```
n_max_unidades = 113
n_max_aprox_duplas = 100
n_max_aprox_triplas = 96
n_max_aprox_quadruplas = 70
```

Para o número máximo de unidades, foi suposto um tabuleiro com jogadas intercaladas. Para o número máximo de duplas, triplas e quádruplas foram supostos quadrados

*brunomn95@gmail.com
†johannwestphall@gmail.com

de tamanho respectivamente 2, 3 e 4, a Figura 1 mostra essa estimação para o número máximo de duplas. Foram contados o número de sequências formadas e os valores obtidos foram incrementados por uma certa margem de garantia com o intuito de evitar que uma disposição otimizada das peças no tabuleiro faça com que uma jogada de nível superior tenha menos peso que o máximo de jogadas de um nível inferior, por exemplo, uma dupla contará mais que o máximo de jogadas unitárias.

Figura 1 – Exemplo estimação número máximo de jogadas.


```
nota =
   (n_aberturas_unidade +
    n_max_unidades * n_aberturas_dupla +
    n_max_unidades * n_max_aprox_duplas * n_aberturas_tripla +
    n_max_unidades * n_max_aprox_duplas * n_max_aprox_triplas *
                                          n_aberturas_quadrupla +
    n_max_unidades * n_max_aprox_duplas * n_max_aprox_triplas *
                        n_max_aprox_quadruplas * n_quintuplas) -
   (n_aberturas_unidade_adversario +
    n_max_unidades * n_aberturas_dupla_adversario +
    n_max_unidades * n_max_aprox_duplas *
                                n_aberturas_tripla_adversario +
    n_max_unidades * n_max_aprox_duplas * n_max_aprox_triplas *
                               n_aberturas_quadrupla_adversario +
    n_max_unidades * n_max_aprox_duplas * n_max_aprox_triplas *
              n_max_aprox_quadruplas * n_quintuplas_adversario)

UTILIDADE e HEURÍSTICA = nota
```

2 Detecções

Todas as detecções considerando aberturas e possibilidades de sequências são previstas pelo *MiniMax*, sendo assim, nessa seção apenas apresentaremos a detecção de fim de jogo.

2.1 Fim de jogo

Considerando que uma sequência de vitória é formada por 5 símbolos consecutivos em qualquer direção, após cada jogada realizada por qualquer jogador é verificado se o jogo acabou através da varredura do grid inteiro buscando a vitória de um dos jogadores ou o empate.

É passado por parâmetro o jogador para o qual se deseja verificar a condição de vitória. Com o jogador selecionado é montada uma sequência de vitória (*win_sequence*), se o jogador selecionado for representado pelo número "1", a sequência de bits montada será '11111', caso contrário '00000'. Após possuir a sequência de vitória montada o algoritmo percorre o tabuleiro em todas as direções possíveis e caso encontre uma sequência igual a sequência vencedora, informará que tem um vencedor. Caso nenhum dos dois jogadores for vencedor, é realizada uma terceira verificação para buscar por espaços livres no tabuleiro, e caso não exista é considerado empate.

A implementação deste algoritmo pode ser encontrada no arquivo *gomoku_core.cc*, no método *game_over()* e *have_winner()*.

3 Classes e métodos principais

- **GomokuCore:**

  Esta classe é responsável pelo maior processamento do jogo, é nela onde estão localizados os algoritmos de detecção de viótria, heurística, utilidade e o MiniMax com podas α e β. O código referente a classe encontra-se nos arquivos *gomoku_core.h* e *gomoku_core.cpp*. Abaixo estão os métodos principais e uma breve descrição de cada um.

  - ***minimax()***:
    Estes métodos são responsáveis pela invocação do *MiniMax*, são quem começa todo o processo para gerar a árvore de mínimos e máximos, na qual serão aplicadas as podas α e β.
  - ***max_search_updown() / max_search_downup()***:
    São responsáveis por extrair os valores dos nodos de minimização e retornar esse valor para o método que os chamou, que realizará o processo de maximização. Além disso, é reponsável pela poda da árvore do *MiniMax*.
  - ***min_search_updown() / min_search_downup()***:
    São responsáveis por extrair os valores dos nodos de maximização e retornar esse valor para o método que os chamou, que realizará o processo de minimização. Além disso, é reponsável pela poda da árvore do *MiniMax*.
  - ***evaluate() / evaluate_incremental()***:
    Juntos do *minimax()* são os mais importantes métodos. O *evaluate()* é responsável por dar uma nota para o tabuleiro tomando como referencia um jogador

escolhido. Enquanto o *evaluate_incremental()* é responsável por também dar uma nota ao tabuleiro, porém tomando como referência uma nota já calculada anteriormente.

  - ***game_over()***:
    Nele é rezaliado a verificação para saber se algum jogador ganhou o jogo ou se o jogo terminou em empate.
  - ***have_winner()***:
    Método que irá verificar se um determinado jogador venceu o jogo.

- **MainWindow**:

  Nesta classe é realizada toda a criação da interface de usuário para que seja possível uma interação mais dinâmica com a implementação desenvolvida. Os métodos desta classe são responsáveis pela montagem da interface, e os métodos que são disparados com o clique dos botões utilizam uma instância do *GomokuCore* para realizar o controle do jogo.

  A label 'iteracoes:' apresenta o número de iterações executadas pela inteligência artificial ao gerar a árvore do *MiniMax*, ao lado esquerdo dela temos uma label que informa o turno do jogador e caso tenha um vencedor ela indica quem venceu. Para a inteligência artificial realizar sua jogada é necessário clicar no botão 'Next Play', em qualquer modo que a envolva.

4 Estrutura de dados

Neste trabalho duas estruturas foram o ponto chave do desenvolvimento, primeiramente um vetor bidimensional de inteiros responsável por representar o estado do tabuleiro, que foi escolhido devido a velocidade durante a busca e comparação de valores que ele guarda, em relação à um vetor bidimensional de *caracteres*, este vetor é representado pelo atributo privado *_grid[ ][ ]* na classe *GomokuCore*. Todas operações de avaliação e busca no tabuleiro são realizadas através dele.

A outra, porém não menos importante, 'estrutura' utilizada foi a pilha, que foi simulada através de um algoritmo recursivo para a criação da árvore do algoritmo *MiniMax*, diminuindo o uso de memória do programa, pois a árvore de pronfundidade definida não precisou estar totalmente construida e alocada em memória para que o algoritmo funcionasse.

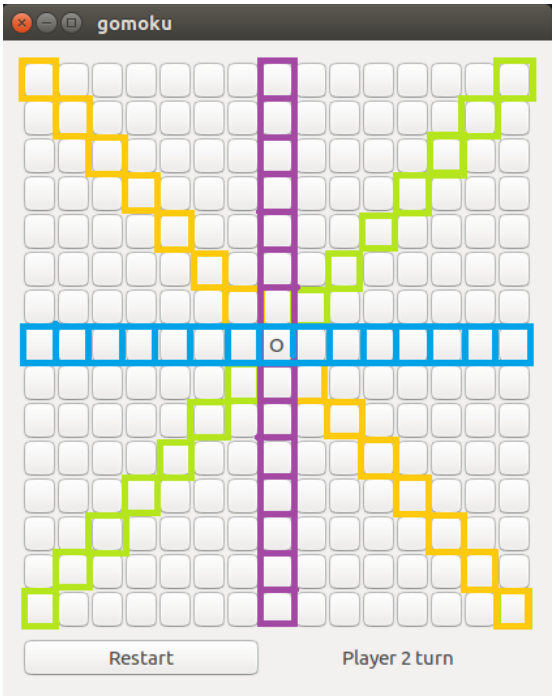
5 Otimizações

Na implementação do *MiniMax* o *_grid[][]* que detém o estado do tabuleiro, é quem é passado por parâmetro, porém essa passagem é realizada por referência desta maneira não é realizada a cópia do vetor bidimensional o que gera um ganho de desempenho, tendo em vista que essa cópia poderia se replicar para todos os níveis da árvore gerada e assim grande parte do tempo de execução ser gastado realizando cópias em memória.

Outra otimização presente no *MiniMax* é a avaliação heurística incremental dos estados do tabuleiro, onde para cada nível da árvore esse valor é cálculado baseando-se em uma avaliação heurística realizada em um nível mais acima, com isso ao invés de só calcular a heurística quando chegar em um nó folha, essa heurística é calculada no percurso de

descida da árvore, tomando como referência a última jogada realizada que será responsável por localizar no tabuleiro todas as sequências por ela afetada. Dessa maneira, ao invés de precisar varrer todas as sequências diagonais, horizontais e verticais quando o algoritmo chegar no nó folha, só serão realizadas as varreduras da sequência horizontal, vertical e das duas sequencias diagonais, afetadas pelas jogadas realizadas na descida da árvore para chegar naquele estado do jogo.

Figura 2 – Cálculo heurístico incremental conforme jogada realizada.



Para acelerar o processo de podas α e β, é realizada uma contagem de quantas jogadas estão na parte superior do tabuleiro e quantas estão na parte inferior. Assim, estas variáveis de controle que irão comandar se a varredura começará pela parte inferior ou superior do tabuleiro. Para isso existem os métodos *max_search_updown() , min_search_updown(), minimax_updown()* e *max_search_downup() , min_search_downup(), minimax_downup()*.

O uso de um vetor bidimensional de inteiros também foi uma otimização, a implementação tornou-se mais complexa, porém o tempo de busca e comparação em um vetor bidimensional de caracteres, mostrou-se mais de 5 vezes mais demorado que a busca e comparação num vetor bidimensional de inteiros.

6 Exemplo de cálculo da heurística

Olhando para a figura Figura 3 e considerando que a inteligência artificial calculará sua jogada, o tabuleiro inicialmente possui a nota -8 para ela, pois seu nó pai possui oito aberturas. Ao descer um nivel na árvore é possivel chegar na configuração apresentada na figura Figura 4. Colocando um "X"ao lado do "O", para a inteligência artificial, uma unidade de abertura do pai é fechada ou seja nota do pai é decrementada em 1, e 7 aberturas de unidade são criadas em volta do "X", somando 7 para sua nota própria. Com isso o

resultado será -7 da nota do pai +7 de sua própria nota, resultando em nota 0 para o tabuleiro.

Supondo mais uma descida na arvore chega-se num nodo MIN. A nota do tabuleiro anterior era 0. Caso o jogador "O", jogue ao lado do "X", a inteligência artificial perderá 1 ponto de sua nota anterior por ter uma abertura fechada pelo oponente e ao mesmo tempo essa jogada proporciona 7 aberturas de unidade para o jogador "O". Logo a nota do tabuleiro que era zero do ponto de vista da inteligência artificial, passa a valer -8 = 0 - 1 - 7.

Figura 3 – Exemplo heurística: nodo pai.

Figura 4 – Exemplo heurística: nodo filho1.

Figura 5 – Exemplo heurística: nodo filho2.