Aplicação de métodos de busca no Jogo da Velha

Laura N. de Andrade
PUC Minas
Belo Horizonte, Brasil
landrade@sga.pucminas.br

Lucca A. M. Santos
PUC Minas
Belo Horizonte, Brasil
lucca.santos@sga.pucminas.br

Richard V. R. Mariano
PUC Minas
Belo Horizonte, Brasil
richard.mariano@sga.pucminas.br

KEYWORDS

Métodos de Busca, A estrela, A*, MINIMAX

ACM Reference Format:
Laura N. de Andrade, Lucca A. M. Santos, and Richard V. R. Mariano. 2020. Aplicação de métodos de busca no Jogo da Velha. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

Existem jogos que exigem uma estratégia de buscar a melhor jogada possível dado um certo cenário. Tais jogos podem ser estudados muito bem com o uso de inteligências artificiais que trabalham com métodos de busca, como o algoritmo A* e o MINIMAX. Neste trabalho estudamos o clássico jogo da velha aplicando os algoritmos MINIMAX e A*. O programa foi implementado em *ansi C* e roda ambos os algoritmos em sequência. As estruturas de dados utilizadas foram: uma matriz 3x3 para simular o tabuleiro do jogo e uma lista encadeada simples, onde cada nó da lista guarda a posição da jogada, i e j, e o resultado das funções de custo e heurística da jogada.

O programa consiste em 5 arquivos, sendo eles: main.c, responsável pelo fluxo de execução do programa, iniciando os tabuleiros e algoritmos; lista.h, contendo a implementação da lista; tabuleiro.h, contendo macros e funções para manipular o tabuleiro, além de funções para avaliar o estado do jogo; minimax.h, contendo a implementação do algoritmo MINIMAX e finalmente aestrela.h contendo a implementação do algoritmo A*. Este artigo está estruturado da seguinte forma: a Seção 2 explica a estrutura da lista, Seção 3 explicando os métodos e funções de manipulação do tabuleiro, Seção 4 explicando a implementação do MINIMAX e finalmente Seção 5 explicando a implementação do A*.

2 LISTA

A lista foi implementada com duas estruturas de dados, uma estrutura chamada Lista, que contém a estrutura Célula e um ponteiro para o próximo item da lista. A estrutura Célula por sua vez contém informações da possível jogada, ou seja, a linha e coluna da jogada e as funções de custo G(x,y) e H(x,y). Para detalhes das funções de custo veja a Seção A*. A lista possui os métodos inserir e deletar para sua manipulação.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3 TABULEIRO

O tabuleiro foi implementado como uma matriz de inteiros, onde um jogador é representado pelo valor 1 (X) e o oponente pelo valor -1 (O), posições vazias são representadas pelo valor 0. Para manipular o tabuleiro existem duas macros, uma para zerar todo o tabuleiro e uma para realizar uma jogada aleatoriamente. Além disso, existem as seguintes funções:

- mostra_tabuleiro(): imprime na saída padrão o tabuleiro com X e O para indicar os jogadores e zero para as posições vazias.
- tem_posicoes(): retorna 1 caso ainda exista alguma posição vazia no tabuleiro, 0 caso contrário.
- acabou(): retorna 1 caso o jogo tenha acabado e 0 caso contrário.
- valor(): lê linhas, colunas e diagonais do tabuleiro e retorna 10 caso X tenha ganho, -10 caso O tenha ganho e zero em caso de empate.

Todas as funções do tabuleiro têm complexidade O(1), pois o tabuleiro sempre tem 9 posições e cada posição é checada uma vez.

4 MINIMAX

O algoritmo MINIMAX funciona avaliando jogadas de 2 jogadores, MIN e MAX. MIN procura minimizar o valor, ou seja, é o O já que valor() retorna -10 para vitória de O. MAX procura maximizar o valor, ou seja, é X já que valor() retorna 10 para vitória de X. O algoritmo recebe o tabuleiro em seu estado atual e quem está jogando (MIN ou MAX) como entrada e retorna a melhor jogada possível para aquele cenário para aquele jogador. A melhor jogada é determinada recursivamente, para cada jogada possível no tabuleiro ocorre uma chamada recursiva passando o tabuleiro com a possível jogada feita e o oponente como parâmetros. A condição de parada é a vitória de um dos lados ou o empate. Retornando da recursão o algoritmo realiza a jogada que teve melhor resultado. Em seguida é a vez do oponente, que realiza o mesmo processo.

Como o algoritmo encontra a melhor jogada para jogador e em seguida para oponente, implementamos de uma forma que o programa simula toda a partida a partir de um ponto inicial. Se esse ponto inicial for o começo do jogo, ou seja, tabuleiro zerado e X realiza o primeiro movimento, sempre ocorrerá empate. Para analisar possíveis jogos aleatórios utilizamos a macro de jogada aleatória, mencionada na Seção Tabuleiro, para realizar algumas jogadas iniciais e em seguida rodamos o algoritmo para ver o desenrolar do jogo a partir daquele ponto. É possível também entrar com um tabuleiro em um estado pré-determinado.

A complexidade do algoritmo MINIMAX implementado é de $O(N\hat{D})$, onde N é o número de jogadas possíveis em um estado do tabuleiro e D a profundidade de busca, que depende de quão avançado o jogo está. No pior dos casos, no início do jogo, D será 9,

pois vão existir no máximo 9 jogadas, ou seja 9 níveis de profundidade, até o fim do jogo. A figura 1 mostra a execução do algoritmo MINIMAX a partir de um jogo aleatório.

5 A*

Diferente do MINIMAX, o algoritmo A* não tem um adversário implementado no algoritmo. Este algoritmo encontra a próxima melhor jogada com base no cenário atual utilizando as funções G(x,y) e H(x,y). A função G(x,y) recebe como parâmetros as coordenadas de uma posição no tabuleiro e retorna quantas jogadas o oponente precisa para vencer se marcar naquela posição. Quanto menor for G(x,y) mais perto o oponente está de ganhar, portanto é uma posição que deve ser ocupada pelo jogador rapidamente.

G(x,y) é uma função que funciona bem para casos de fim de jogo, onde algumas posições já foram ocupadas, para casos de início de jogo G(x,y) tem o mesmo valor para praticamente todas as posições do tabuleiro, impossibilitando uma decisão mais certa. É para suprir essa deficiência que existe a função H(x,y). H(x,y) também recebe como parâmetros as coordenadas de uma posição do tabuleiro mas retorna em quantas linhas, colunas e diagonais o oponente pode vencer utilizando aquela posição. Essa função avalia em quais posições o oponente tem mais possibilidades de vitória. Sabendo essas posições é possível ocupá-las para frustrar o jogo do oponente.

A função de custo final é dada por F(x,y) = G(x,y) - H(x,y), e quanto menor for F, mais importante será ocupar a posição. Dessa forma levamos em conta a urgência de se ocupar uma posição para que o oponente não vença, e qual posição impede mais o oponente de vencer, funcionando para o início e fim de jogo.
O algoritmo A* então insere na fila todas as possíveis jogadas, dado um estado do tabuleiro, e calcula F(x,y) para todas elas. Em seguida faz a jogada na posição onde F(x,y) foi menor. Como A* não possui um adversário, o usuário do programa age como oponente de A*.

Ao iniciar o programa pelo main.c, o usuário insere sua jogada, a jogada é computada no tabuleiro e em seguida A* é chamado e recebe o tabuleiro. A* calcula a melhor jogada e ela é computada no tabuleiro. O fluxo então retorna ao usuário e segue dessa maneira até que o jogo acabe.

A complexidade do A* é O(n) onde n é o número de posições livres no tabuleiro. Para cada posição livre a heurística é calculada em O(1) já que o tabuleiro tem sempre tamanho fixo. A figura 2 mostra a execução do algoritmo A* por 2 jogadas do usuário.

6 RESULTADOS

Os algoritmos foram usados de formas diferentes, o MINIMAX sendo utilizado para calcular um jogo ótimo a partir de certo momento e o A* para calcular a jogada ótima dado um certo estado. O MINIMAX obteve o melhor desempenho sempre. Para jogos começando do zero, sempre houve empate, mostrando que ambos os lados fizeram as melhores jogadas sempre. Para jogos aleatórios o resultado geralmente foi de vitória para um dos lados, o favorecido pelas jogadas aleatórias iniciais, normalmente.

Já o A* não obteve resultados tão bons, para muitos jogos o resultado foi de desempenhos equivalente do algoritmo e do usuário, resultando em empate, mas para outros jogos, a heurística não foi tão boa e resultou em vitória para o usuário. Isso se deve à particularidades de cada jogo que exploraram casos onde a heurística se mostrou não eficiente. Um padrão entre esses casos foram jogadas nas diagonais por parte do usuário, mostrando um ponto fraco da heurística para as diagonais.

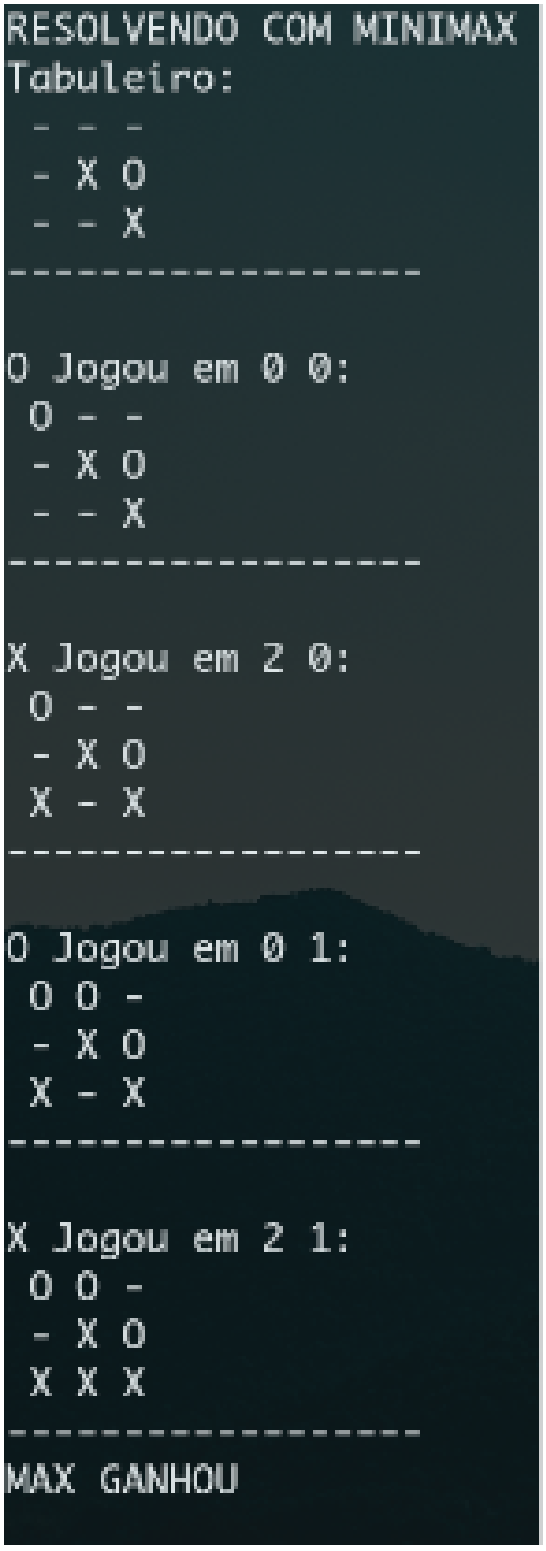

Figure 1: Execução do MINIMAX para um jogo aleatório.

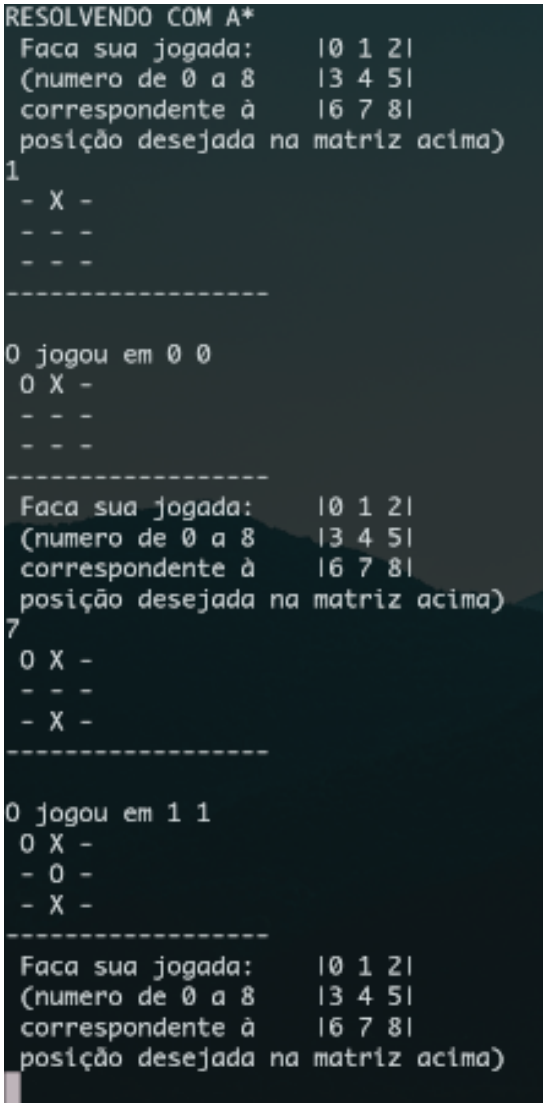

Figure 2: Execução do A* contra o usuário.