Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Relatório (Jogo do Conecta-4)

Aluno(s): Daniel Rodrigues Martins(19.1.8147)
Stephane Matos Oliveira(19.2.8983)
Vinicius Gabriel Albuquerque(19.1.8070)

Julho
2023

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Relatório

Relatório do Trabalho 1 da disciplina CSI 457 do Curso de Engenharia da Computação da Universidade Federal de Ouro Preto.

Alunos:Daniel Rodrigues Martins
Stephane Matos Oliveira
Vinicius Gabriel Albuquerque

Professor:Talles

Julho
2023

Conteúdo

1 Resumo

O objetivo deste trabalho prático é desenvolver e aprimorar o entendimento sobre a busca competitiva em jogos de adversários, com foco específico no jogo Conecta-4. O jogo Conecta-4 é um jogo de tabuleiro no qual dois jogadores se alternam colocando peças em uma grade vertical de 6 linhas por 7 colunas, com o objetivo de formar uma linha contínua de quatro peças da mesma cor, seja na vertical, horizontal ou diagonal. O objetivo geral é aprimorar a compreensão e aplicação de técnicas de busca competitiva em jogos de adversários, por meio do desenvolvimento e avaliação de um agente de IA para o jogo Conecta-4. O trabalho envolverá a implementação de uma função de avaliação heurística, a otimização do algoritmo Minimax com a poda alfa-beta.

Os principais objetivos do trabalho são os seguintes:

- Implementar uma função de avaliação heurística para tomada de decisões em tempo real. Como o tamanho do tabuleiro pode ser grande, é importante que a heurística seja capaz de tomar boas decisões mesmo sem explorar todo o tabuleiro.

- Implementar a poda alfa-beta para otimizar o algoritmo Minimax. A utilização da poda alfa-beta permitirá que o agente explore mais profundamente a árvore de jogo em um mesmo período de tempo, melhorando a eficiência do algoritmo.

- o desempenho de cada nova versão do agente utilizando um tabuleiro maior, por exemplo, com 15 linhas e 16 colunas. Será necessário contar o número total de nós visitados durante a busca para medir o desempenho do agente.

2 Apresentação

2.1 Heuristica

Para realização deste trabalho foi desenvolvida a função de avaliação heuristica. A heurística passa uma informação específica do domínio que pode ser usada para guiar o processo de busca. Em muitos casos uma heurística envolve a aplicação de uma função que avalia um nó particular e prediz a qualidade dos seus nós sucessores.

Uma forma de uso da informação heurística sobre um problema consiste em computar estimativas numéricas para os nós no espaço de estados; Uma estimativa indica o quanto um nó é promissor com relação ao alcance de um nó-objetivo; A idéia é continuar a busca sempre a partir do nó mais promissor no conjunto de candidatos. Uma heurística é apenas uma conjectura informada sobre o próximo passo a ser tomado na solução de um problema, é baseada na experiência e na intuição e pode levar um algoritmo de busca a uma solução subótima ou, inclusive, levá-lo a não conseguir encontrar uma solução.

Neste trabalho o tabuleiro por ser muito grande deixa o jogo pesado e lento, para melhorar isso a avaliação heuristica deve escolher a melhor opção sem percorrer todo o tabuleiro.

2.2 Poda Alfa-Beta

A poda alfa-beta é um algoritmo utilizado na busca de jogos e na inteligência artificial para reduzir o número de nós explorados em uma árvore de busca, melhorando assim a eficiência do processo de tomada de decisões.

Nos jogos de estratégia, como xadrez ou go , é utilizado uma árvore de busca para analisar as possíveis jogadas e avaliar a qualidade de cada uma delas. No entanto, explorar toda a árvore de busca é computacionalmente custoso e leva muito tempo. A poda alfa-beta busca evitar explorar certos caminhos que não levam a uma solução ótima, reduzindo assim a quantidade de nós que devem ser examinados.

O algoritmo utiliza dois valores, alfa e beta, que representam os limites inferiores e superiores, respectivamente, da pontuação de um jogador no jogo. Ao realizar uma busca na árvore, os valores de alfa e beta são atualizados à medida que os nós são explorados. A ideia-chave é que, se for encontrado um nó que leva a uma solução pior para o jogador em turno (beta) ou a uma solução melhor para o oponente (alfa), não é necessário explorar além desse nó, pois a estratégia ótima não passará por esse caminho.

Para este trabalho e necessario implementar a poda alfa-beta para otimizar o algoritmo Minimax, permitindo ao seu agente explorar mais a fundo na arvore de jogo e mum mesmo periodo de tempo.

2.3 Algoritmo Minimax

Em 1944 John von Neumann propõe um método de busca (Minimax) para jogos de soma zero que maximiza a sua posição enquanto minimiza a de seu oponente. Para implementar esse método necessitamos medir, de alguma maneira, o quanto boa a nossa posição é.Usamos para isso a função de utilidade; O algoritmo minimax calcula a decisão minimax a partir do estado corrente, ele utiliza a computação recursiva dos valores minimax de cada estado sucessor. Percorre inicialmente todo o caminho até as folhas e depois propaga os valores minimax de volta pela árvore, à medida que a recursão retorna.

O algoritmo minimax executa uma exploração completa da árvore de jogo fazendo uma busca em profundidade. Algoritmo mais usado em jogos com dois jogadores, chamados MAX e MIN. O princípio do Minimax é descer os 'nós' da Game Tree até chegar no término do jogo, identificando se o jogador perdeu, empatou ou ganhou.

3 Descrição de atividades

Para iniciarmos o trabalho pensamso em algumas ideias para heuristicas como por exemplo dividir a matriz em submatrizes e, pontuar para cada peça do jogador uma determinada quantidade de pontos e a somatoria dos pontos nessa jogada seria o peso dessa escolha. Para otimizarmos essas submatrizes criamso um algoritmo para verificar se a linha estava zerada e, neste caso, poderiamos pular a avaliação de pontos e assim reduzir as opções de escolha mesmo, o codigo ficando mais pesado ele ficou mais detalhado e assim possibilitanto fazer escolhas melhores mesmo quando observamos menos jogadas a frente.

Para melhorarmos a eficiencia utilizamos a poda alfa-beta no algortimo minimax desta forma, poderemos reduzir a profundidade de busca do algoritmo sem perder resultados positivos e diminuindo o tempo de execução gracas a combinação dessa heuristica e da poda. Mostraremos a seguir a aplicação desses algoritmos.

3.1 Heuristica

```python
def heuristica(board):
    avaliacao = 0
    global global_variable

    for row in range(len(board)):
        for col in range(len(board[row])-profundidade_heuristica):
            window = board[row][col:col+profundidade_heuristica]
            print(window)
            global_variable = global_variable + 1
            if linha_zerada(window):
                break
            avaliacao += submatriz(window)

    for col in range(len(board[0])):
        for row in range(len(board)-profundidade_heuristica):
            window = [board[row+i][col] for i in range(profundidade_heuristica)]
            #print(window)
            global_variable = global_variable + 1
            avaliacao += submatriz(window)
```

Figura 1: Codigo da Heuristica

Esse código implementa uma função heuristica que calcula uma pontuação para um tabuleiro de jogo. O parâmetro board é uma matriz que representa o tabuleiro do jogo. Cada elemento da matriz é um inteiro que representa o estado de uma posição do tabuleiro (por exemplo, 0 para uma posição vazia, 1 para uma peça do jogador 1 e 2 para uma peça do jogador 2). A variável avaliacao é inicializada com zero e é usada para acumular a pontuação final

do tabuleiro. A variável global-variable é usada para contar quantas vezes a função submatriz é chamada.

O primeiro loop percorre todas as linhas do tabuleiro. O loop interno percorre todas as sequências de 4 peças em cada linha. A variável window armazena cada sequência de 4 peças. A função linha-zerada é chamada para verificar se a sequência está vazia. Se a sequência estiver vazia, o loop interno é interrompido usando a instrução break. Caso contrário, a avaliação da sequência é adicionada à variável avaliacao. Os três FOR seguintes fazem a mesma coisa que na função heuristica anterior, mas sem a verificação adicional.

Em resumo, a função implementa uma heurística para avaliar o tabuleiro do jogo, considerando todas as possíveis sequências de 4 peças no tabuleiro, mas evitando o processamento de sequências que já foram avaliadas como vazias.

```python
def submatriz(window):
    avaliacao = 0
    player_pieces = 0
    opponent_pieces = 0
    empty_pieces = 0

    if linha_zerada(window):
        return avaliacao

    for piece in window:
        if piece == 2:
            player_pieces += 1
        elif piece == 1:
            opponent_pieces += 1
        elif piece == 0:
            empty_pieces += 1

    if player_pieces == 4:
        avaliacao += 100
    elif player_pieces == 3 and opponent_pieces == 0:
        avaliacao += 75
    elif player_pieces == 2 and opponent_pieces == 0:
        avaliacao += 50
    elif opponent_pieces == 3 and player_pieces == 0:
        avaliacao -= -75

    return avaliacao
```

Figura 2: Codigo da Heuristica 2

Essa função submatriz é usada para avaliar uma sequência de 4 peças em um tabuleiro de jogo. A função conta quantas peças pertencem ao jogador atual, ao oponente e quantas estão vazias. Em seguida, a função atribui uma pontuação à sequência com base no número de peças do jogador atual e do oponente. O parâmetro window é uma lista que contém 4 peças do tabuleiro. A variável avaliacao é inicializada com zero e é usada para armazenar a pontuação da sequência. As variáveis player-pieces, opponent-pieces e empty-pieces são inicializadas com zero e são usadas para contar o número de peças do jogador atual, do oponente e as peças vazias, respectivamente. A função linha-zerada é chamada para verificar se a sequência está vazia. Se a sequência estiver vazia, a função retorna imediatamente com uma avaliação de zero.

O FOR percorre cada peça na sequência e conta quantas peças pertencem ao jogador atual, ao oponente e quantas estão vazias. Em seguida, a função atribui uma pontuação à sequência com base no número de peças do jogador atual e do oponente. Se o jogador atual tiver 4 peças na sequência, a avaliação é aumentada em 100 pontos. Se o jogador atual tiver 3 peças e o oponente não tiver nenhuma, a avaliação é aumentada em 75 pontos. Se o jogador atual tiver 2 peças e o oponente não tiver nenhuma, a avaliação é aumentada em 50 pontos. Se o oponente tiver 3 peças e o jogador atual não tiver nenhuma, a avaliação é diminuída em -75 pontos. Por fim, a função retorna a pontuação da sequência.

3.2 Minimax-Poda Alfa-Beta

A poda alfa beta realiza uma busca em profundidade na árvore de jogo, avaliando os nós em uma ordem específica. Conforme se desce pela árvore, os valores de alfa e beta são atualizados e as podas correspondentes são aplicadas. Se em um determinado nó for encontrada uma solução que excede o limite superior (beta) ou uma solução que é pior do que o limite inferior (alfa), é realizada a poda e evita-se explorar os nós descendentes, pois eles não afetarão a escolha da jogada ótima.

A função implementa o algoritmo Minimax com poda alfa-beta para determinar a melhor jogada para um jogador em um determinado estado do tabuleiro.O parâmetro depth é a profundidade atual da busca no espaço de estados. A profundidade começa em um valor máximo e diminui a cada chamada recursiva da função. Os parâmetros alpha e beta são usados para realizar a poda alfa-beta. Eles representam os valores mínimo e máximo que a função pode retornar, respectivamente. A poda alfa-beta é usada para evitar a avaliação de ramos desnecessários da árvore de busca. O parâmetro maximizing-player é um valor booleano que indica se o jogador atual é o

jogador maximizador ou não. O jogador maximizador é aquele que tenta maximizar sua pontuação, enquanto o jogador minimizador tenta minimizar a pontuação do adversário. A função começa verificando se o jogo acabou, isto é, se alguém ganhou ou se o jogo empatou. Se o jogador da IA ganhou, a função retorna uma pontuação de 100. Se o jogador humano ganhou, a função retorna uma pontuação de -100. Se o jogo empatou, a função retorna a avaliação heurística do tabuleiro. Se a profundidade máxima foi atingida, a função retorna a avaliação heurística do tabuleiro.

A primeira linha da função chama a função get-valid-locations para obter uma lista de todas as colunas que ainda têm espaço para uma nova peça. Se o jogador atual é o jogador maximizador, a função inicializa a variável value com um valor negativo infinito e escolhe uma coluna aleatória da lista de colunas válidas. Em seguida, a função percorre cada coluna válida e simula uma jogada nessa coluna, chamando a função minimax recursivamente com um nível de profundidade reduzido. A função minimax retorna uma tupla contendo a coluna escolhida e a pontuação da jogada. Se a pontuação da jogada atual for maior que a pontuação anterior, a função atualiza o valor de value e escolhe a coluna atual como a melhor jogada até agora. A função também atualiza o valor de alpha com o máximo entre alpha e value. Se alpha for maior ou igual a beta, a função interrompe o loop usando a instrução break. Por fim, a função retorna a coluna escolhida e sua pontuação.

Se o jogador atual é o jogador minimizador, a função faz o mesmo que acima, mas inicializa a variável value com um valor positivo infinito e escolhe uma coluna aleatória da lista de colunas válidas. A função também atualiza o valor de beta com o mínimo entre beta e value. Se alpha for maior ou igual a beta, a função interrompe o loop usando a instrução break. Por fim, a função retorna a coluna escolhida e sua pontuação.

4 Análise dos Resultados

Para visualizar os resultados obtidos, executamos o codigo com e sem cada componente adicionada para que podessemos visualizar a melhoria em seu tempo de execução. Inicialmente visualizamos o tempo de execução utilizando apena a heuristica.

Observando o tempo de execução ja e possivel verificar uma melhora consideravel no desempenho do agente.

Realizando agora o teste com a poda alfa-beta:

Com a poda e visivel uma grande melhoria, observando uma redução de 5 vezes no tempo de execução e uma redução no numero de estados explorados. Concluimos que o uso da poda alfa-beta melhora o desempenho do agente e torna a heuristica mais eficiente.

```
tempo: 29.101016521453857
[[2. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Figura 3: Teste do Codigo inicial

```
tempo: 133.83291363716125
Quantidade de estados: 40632320
[[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Figura 4: Teste do Codigo com a Poda alfa-beta

```
Tempo de Execução:  5.0548694133758545
Quantidade de estados visitados: 1882920
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Figura 5: Teste do Codigo com a heuristica e zeros

```
Tempo de Execução:  0.6674032211303711
Quantidade de estados visitados: 270248
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Figura 6: Teste do Codigo completo

Bibliografia

AGUIRRE, L. A. Introdução à Identificação de Sistemas, Técnicas Lineares e Não lineares Aplicadas a Sistemas Reais. Belo Horizonte, Brasil, EDUFMG. 2004.

Algoritmo Minimax - Introdução à Inteligência Artificial. Disponível em: ¡https://www.organicadigital.com/bl minimax-introducao-a-inteligencia-artificial/¿. Acesso em: 13 jul. 2023.