UNIVERSIDADE FEDERAL DO MARANHÃO

BACHARELADO INTERDISCIPLINAR EM CIÊNCIA E TECNOLOGIA

INTELIGÊNCIA ARTIFICIAL

Juan Pablo Furtado Mondego

João Henrique Calixto Teixeira

João Victor Lima Salomao

São Luís

2024

SUMÁRIO

INTRODUÇÃO

A inteligência artificial (IA) tem desempenhado um papel crucial no desenvolvimento de sistemas capazes de simular habilidades humanas, como a tomada de decisão em jogos. Neste relatório, apresentamos a implementação de uma IA para o jogo da velha, utilizando o algoritmo de Busca em Largura (BFS). O objetivo é demonstrar como técnicas de busca podem ser aplicadas para resolver problemas de decisão em cenários controlados.

O jogo da velha é um exemplo clássico de problema resolvível por algoritmos de busca devido ao espaço limitado de estados possíveis. A IA desenvolvida explora exaustivamente todos os estados do tabuleiro para encontrar a melhor jogada disponível, garantindo competitividade contra o jogador humano. Além disso, o código implementa uma interface gráfica interativa usando a biblioteca Pygame, proporcionando uma experiência intuitiva e visualmente clara.

Este relatório detalha o funcionamento do algoritmo e discute as principais limitações e potenciais melhorias para tornar a solução mais robusta e eficiente em cenários mais complexos.

DESENVOLVIMENTO

O código desenvolvido neste trabalho implementa uma busca por largura (BFS) para encontrar a melhor opção de movimento em um jogo da velha, ele explora todos os possíveis estados do tabuleiro a partir de uma posição inicial e verifica qual deles resulta em uma vitória para o jogador atual.

No código existem várias funções que formam a lógica do programa e que implementam conceitos de IA para tornar a execução eficiente, Podemos dissecar partes do código que são necessárias para entender toda a lógica por trás do programa.

bfs_best_move(board, player)

Essa função utiliza a busca em largura (BFS - Breadth-First Search) para determinar o melhor movimento do computador. Ela explora todos os estados possíveis do tabuleiro começando a partir do estado atual e procurando a melhor jogada que leva à vitória.

Este é um exemplo de algoritmo de busca em grafos, onde cada nó representa um estado do tabuleiro e as arestas representam as jogadas possíveis. Na IA de jogos, determinar o melhor movimento é essencial. A BFS aqui é usada para avaliar todas as possíveis jogadas de forma exaustiva até encontrar a melhor.

Função 1

```
# Função BFS para encontrar o melhor movimento

def bfs_best_move(board, player):

    start = (board, [])

    queue = deque([start])

    visited = set()



    while queue:

        current_board, path = queue.popleft()

        board_tuple = tuple(map(tuple, current_board))

        if board_tuple in visited:

            continue
```

```
        visited.add(board_tuple)


        # Verifica se é o melhor estado
        if check_winner(current_board, player):
            return path[1] if path else None


        # Gera próximos movimentos
        for row in range(3):
            for col in range(3):
                if current_board[row][col] == '':
                    new_board = [r[:] for r in current_board]
                    new_board[row][col] = player
                    queue.append((new_board, path + [(row, col)]))


    return None
```

---

check_winner(board, player)

Essa função verifica se o jogador especificado venceu o jogo. Ela checa todas as linhas, colunas e diagonais do tabuleiro para ver se todas as células são ocupadas pelo jogador atual.

Essa função realiza uma tarefa simples e inteligente de reconhecimento de padrões ao verificar se há três símbolos iguais seguidos em uma linha, coluna ou diagonal, o que caracteriza uma vitória no jogo da velha.

Função 2

```
# Verifica o vencedor

def check_winner(board, player):

    for row in board:

        if all(cell == player for cell in row):

            return True

    for col in range(3):

        if all(board[row][col] == player for row in range(3)):

            return True

    if all(board[i][i] == player for i in range(3)) or all(board[i][2-i] ==
player for i in range(3)):

        return True

    return False
```

---

draw_board() e draw_symbols(board)

Estas funções são responsáveis pela interface gráfica do jogo. draw_board() desenha o tabuleiro e draw_symbols(board) desenha os símbolos 'X' e 'O' nas posições apropriadas. A apresentação visual clara e interativa do jogo é fundamental para a experiência do usuário.

Função 3

```
# Função para desenhar o tabuleiro

def draw_board():

    screen.fill(BG_COLOR)

    for i in range(1, 3):

        # Linhas verticais
```

```
        pygame.draw.line(screen, LINE_COLOR, (i * CELL_SIZE, 0), (i *
CELL_SIZE, HEIGHT), LINE_WIDTH)

        # Linhas horizontais

        pygame.draw.line(screen, LINE_COLOR, (0, i * CELL_SIZE), (WIDTH, i *
CELL_SIZE), LINE_WIDTH)



# Função para desenhar X e O no tabuleiro

def draw_symbols(board):

    for row in range(3):

        for col in range(3):

            if board[row][col] == 'X':

                pygame.draw.line(screen, X_COLOR,

                                 (col * CELL_SIZE + 20, row * CELL_SIZE +
20),

                                 ((col+1) * CELL_SIZE - 20, (row+1) *
CELL_SIZE - 20), LINE_WIDTH)

                pygame.draw.line(screen, X_COLOR,

                                 (col * CELL_SIZE + 20, (row+1) * CELL_SIZE -
20),

                                 ((col+1) * CELL_SIZE - 20, row * CELL_SIZE +
20), LINE_WIDTH)

            elif board[row][col] == 'O':

                pygame.draw.circle(screen, O_COLOR,

                                   (col * CELL_SIZE + CELL_SIZE//2, row *
CELL_SIZE + CELL_SIZE//2),

                   CELL_SIZE//3, LINE_WIDTH)
```

---

main()

Essa é a função principal que controla o fluxo do jogo, alternando entre o jogador humano e o computador. Ela lida com eventos, chama as funções de desenho, verifica vitórias e empates, e chama a função de IA (bfs_best_move).

Função 4

```
def main():
    board = [['' for _ in range(3)] for _ in range(3)]
    player_turn = True  # True para jogador (X), False para computador (O)

    while True:
        draw_board()
        draw_symbols(board)
        pygame.display.flip()

        # Verifica eventos
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

            # Jogador faz um movimento
            if player_turn and event.type == pygame.MOUSEBUTTONDOWN:
                x, y = event.pos
                col = x // CELL_SIZE
                row = y // CELL_SIZE
                if board[row][col] == '':
                    board[row][col] = 'X'
```

```
            if check_winner(board, 'X'):
                draw_board()
                draw_symbols(board)
                pygame.display.flip()
                show_message("Você ganhou!")
                pygame.quit()
                sys.exit()
            player_turn = False

# Movimento do computador usando BFS
if not player_turn:
    move = bfs_best_move(board, 'O')
    if move:
        row, col = move
        board[row][col] = 'O'
        if check_winner(board, 'O'):
            draw_board()
            draw_symbols(board)
            pygame.display.flip()
            show_message("Você perdeu!")
            pygame.quit()
            sys.exit()
    player_turn = True

# Verifica empate
if all(all(cell != '' for cell in row) for row in board):
    draw_board()
    draw_symbols(board)
```

```
            pygame.display.flip()
            show_message("Empate!")
            pygame.quit()
            sys.exit()

        clock.tick(FPS)

# Executa o jogo
if __name__ == "__main__":
    main()
```

RESULTADOS

A implementação realizada foi capaz de simular partidas do jogo da velha de forma eficiente, com o computador utilizando o algoritmo BFS para identificar o melhor movimento em cada turno. Os testes demonstraram que a IA consegue:

1. Garantir uma exploração completa de todas as possibilidades no tabuleiro, identificando movimentos que levam à vitória ou evitando derrotas iminentes.
2. Manter o jogo competitivo contra um jogador humano, exibindo comportamentos que imitam estratégias básicas de tomada de decisão.
3. Exibir mensagens visuais claras para vitória, empate ou derrota, contribuindo para uma experiência de usuário satisfatória.

Desempenho da IA

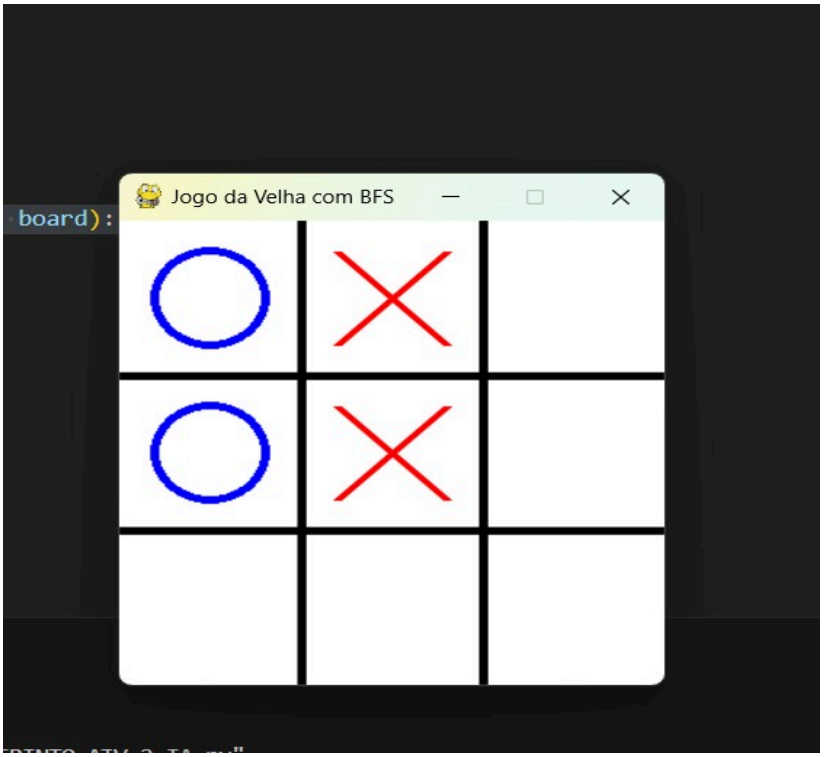
- Tempo de Resposta: O tempo de processamento para encontrar o próximo movimento foi imperceptível ao jogador humano, dada a simplicidade do jogo da velha.
- Precisão: A IA sempre encontrou a melhor jogada disponível no contexto do estado atual do tabuleiro.
- Limitações: A BFS, embora eficaz neste cenário, possui limitações de escalabilidade, não sendo viável para jogos mais complexos como xadrez, devido ao crescimento exponencial do número de estados possíveis.

Com base nesses resultados, podemos concluir que a IA implementada cumpre seu objetivo principal de explorar conceitos básicos de algoritmos de busca em um ambiente controlado. No entanto, aprimoramentos como a implementação do algoritmo Minimax ou a inclusão de aprendizado de máquina poderiam expandir significativamente suas capacidades.

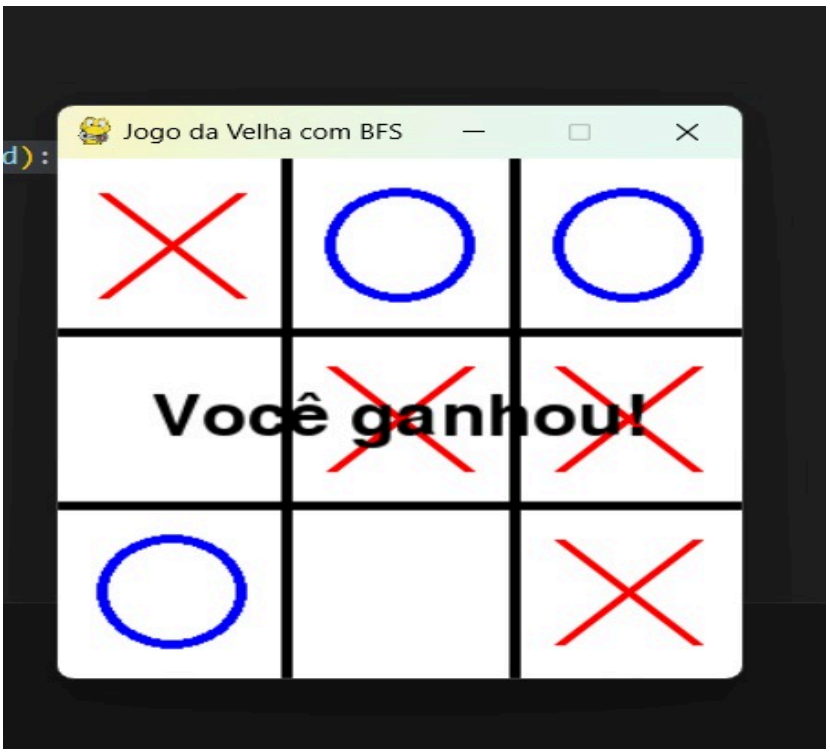
A seguir, apresentamos capturas de tela que ilustram o funcionamento do jogo da velha implementado com a IA baseada em BFS:

1. Estado Inicial do Tabuleiro
   A imagem abaixo apresenta o estado inicial do tabuleiro, antes de qualquer movimento ser realizado. Este estado vazio demonstra a configuração básica da interface gráfica desenvolvida para o jogo.

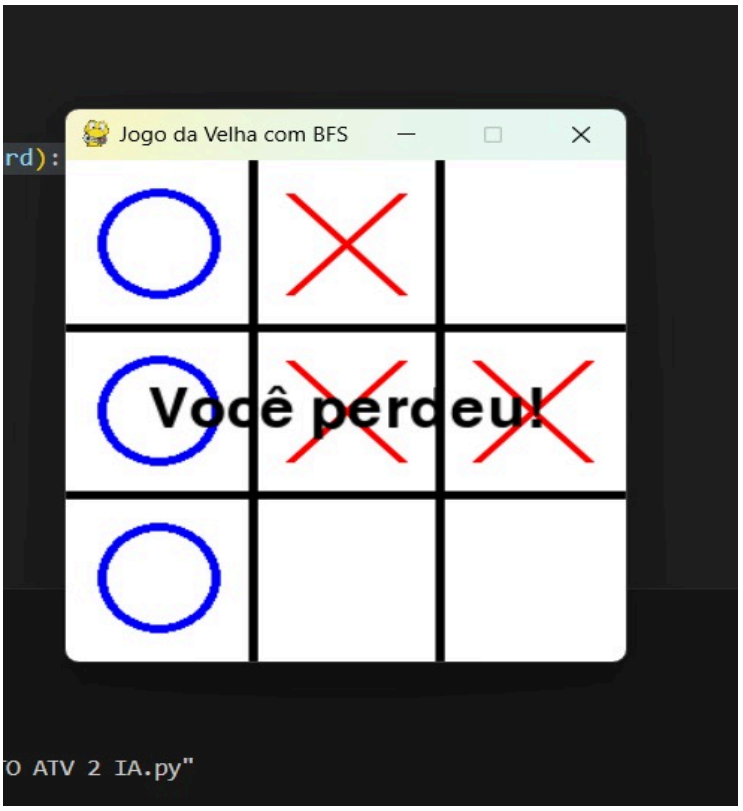
2. Movimentação Durante o Jogo
   A seguinte imagem exibe o estado intermediário de uma partida, com o tabuleiro parcialmente preenchido. Neste ponto, o algoritmo BFS foi executado para determinar a jogada ótima para o computador, que utilizou o símbolo 'O'.



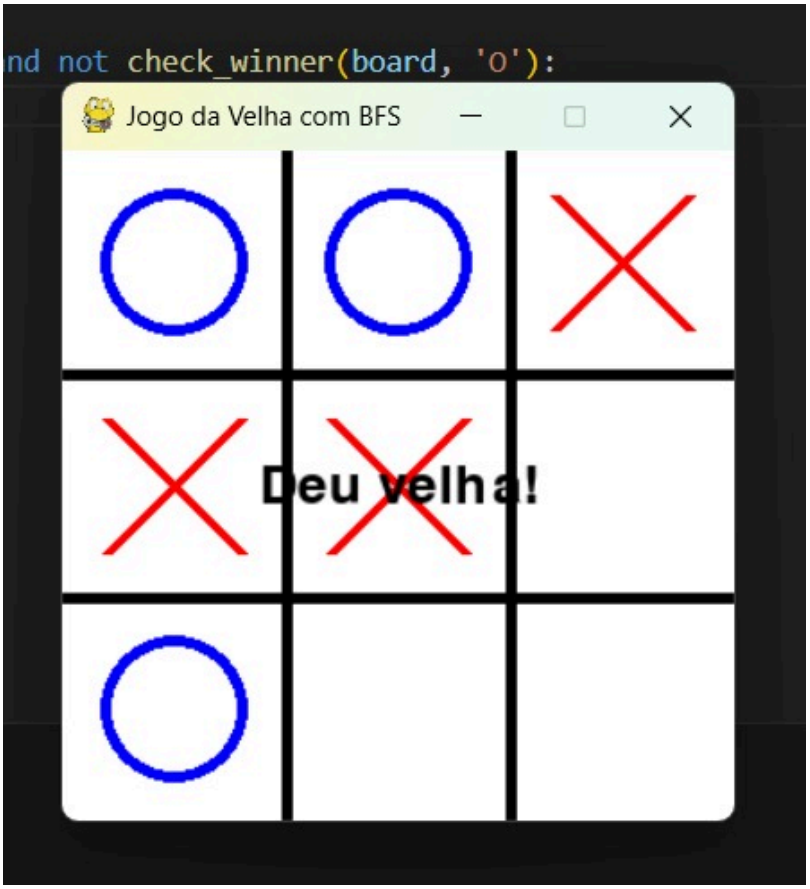
3. Situação de Vitória do Jogador Humano
   Por vez, essa imagem abaixo demonstra o momento em que o jogador humano vence a partida. A mensagem "Você ganhou!" aparece de maneira destacada na interface, indicando o término do jogo.



4. Situação de Vitória da IA
   Já essa imagem apresenta uma situação de vitória do computador. A IA utiliza o BFS para encontrar movimentos que resultem em sua vitória, exibindo a mensagem "Você perdeu!" ao final.



5. Situação de Empate
   Como última imagem, temos essa abaixo, que apresenta uma situação em que o jogo termina empatado. A mensagem "Deu velha!" é exibida na interface, indicando que não há mais movimentos disponíveis e nenhum jogador conseguiu atingir a condição de vitória. Este cenário demonstra a capacidade da IA de lidar com todas as possibilidades de jogo, mesmo em casos onde nenhuma vitória é possível.



DISCUSSÃO

Podemos verificar no decorrer das implementações do código várias relações com IA, como:

Verificação de vencedor: A verificação do vencedor é um exemplo de teste de objetivo em um problema de busca, onde o algoritmo precisa avaliar se o estado atual é uma solução válida. A principal limitação é que não considera estratégias de longo prazo, pois apenas verifica o estado atual sem antecipar jogadas futuras.

Interface gráfica: A interface gráfica é um componente fundamental para testar algoritmos de IA em cenários controlados, simulando interações reais com usuários. A clareza do tabuleiro e o feedback visual tornam as jogadas mais intuitivas e as mensagens finais "Você ganhou!"e "Empate!" criam uma experiência interativa e reforçam a mecânica do jogo. Isso facilita a interação e permite que os algoritmos sejam avaliados sob condições próximas a jogos reais.

A **BFS** é um exemplo clássico de busca cega ou não informada. Ela consegue garantir uma exploração completa, por isso ela é uma ótima solução para pequenos problemas como o jogo da velha implementado por nossa equipe.
Ela é capaz de fazer o computador jogar de forma otimizada dentro das possibilidades do tabuleiro, explorando todos os estados possíveis a partir do estado atual.
Ela é especialmente interessante nesse tipo de programa pois todos os movimentos são iguais e ela sempre vai encontrar uma solução(se existir).
Porém ela seria impraticável se escolhêssemos implementá-la no xadrez, por exemplo, já que ela não consegue acompanhar um crescimento exponencial do espaço e de estados, tornando-a inviável para jogos mais complexos.

CONCLUSÃO

O uso da BFS na implementação do código garante que o computador seja competitivo, explorando todos os estados possíveis para encontrar o melhor movimento imediato.

Para melhorar futuras ideias de implementação poderia ser adicionada uma busca informada como Minimax que utiliza funções de avaliação para estimar o valor de estados futuros, também seria interessante adicionar aprendizado de máquina e fazer o computador aprender padrões de jogo ao longo de várias partidas, adaptando-se ao estilo do jogador humano. Modelos como redes neurais ou Q-learning poderiam ser aplicados para treinar o computador a identificar jogadas vantajosas sem uma estratégia fixa.

Os conceitos no código demonstram como diferentes técnicas de IA podem ser aplicadas para resolver problemas de busca, tomada de decisão e interação humano-máquina, destacando os desafios de escalabilidade e eficiência no design de sistemas inteligentes.