

TRABALHO TICTACTOE

Compreendendo o Projeto

O projeto do Jogo da Velha (Tic-Tac-Toe) é dividido em dois arquivos principais em Python:

- `tictactoe.py` : Este arquivo contém toda a lógica do jogo, incluindo como o tabuleiro é representado, como determinar o próximo jogador, como verificar se há um vencedor e como a IA escolhe o melhor movimento.
- `runner.py` : Este arquivo usa a biblioteca Pygame para criar a interface gráfica do jogo, permitindo que um usuário jogue contra a IA.

O que você precisa implementar:

O seu trabalho é completar as funções no arquivo `tictactoe.py` . Essas funções são:

- `player` : Determina de quem é a vez de jogar.
- `actions` : Lista todos os movimentos possíveis.
- `result` : Atualiza o tabuleiro após um movimento.
- `winner` : Verifica se há um vencedor.
- `terminal` : Verifica se o jogo acabou.
- `utility` : Calcula a pontuação do jogo.
- `minimax` : Implementa o algoritmo Minimax para a IA escolher o melhor movimento.

Passo 1: Configuração

1. **Baixar o arquivo:** Baixe o arquivo `tictactoe.zip` fornecido.
2. **Instalar o Pygame:** No diretório do projeto, execute `pip3 install -r requirements.txt` para instalar a biblioteca Pygame. O arquivo `requirements.txt` contém a lista de pacotes necessários para executar o projeto.

Passo 2: Entendendo o `tictactoe.py`

O arquivo `tictactoe.py` já define algumas coisas importantes:

- **Constantes:** `X` , `O` e `EMPTY` representam os jogadores e uma célula vazia no tabuleiro.

- `initial_state()` : Esta função retorna o estado inicial do tabuleiro, que é uma lista de 3x3 com todas as células vazias.

Passo 3: Implementando as Funções

Agora, vamos começar a implementar cada uma das funções. Vou te dar explicações detalhadas e exemplos de código.

Entendendo a Função `player`

A função `player(board)` precisa determinar de quem é a vez de jogar em um dado estado do tabuleiro.

- No início do jogo, o jogador `x` sempre começa.
- Depois disso, os jogadores alternam os turnos.
- Se o jogo já terminou, a função pode retornar qualquer valor (já que não importa de quem é o turno).

Implementação da Função `player`

Aqui está como você pode implementar a função `player` :

Python

```
def player(board):
    """
    Returns player who has the next turn on a board.
    """
    count_x = 0
    count_o = 0
    for row in board:
        for cell in row:
            if cell == X:
                count_x += 1
            elif cell == O:
                count_o += 1

    if count_x == count_o:
        return X
    else:
        return O
```

Explicação:

1. **Contadores** `count_x` e `count_o`: Inicializamos dois contadores para rastrear quantas vezes `x` e `o` aparecem no tabuleiro.
2. **Iterando pelo tabuleiro**: Usamos dois loops `for` para percorrer cada célula do tabuleiro.
3. **Contando** `x` e `o`: Dentro dos loops, verificamos se a célula contém `x` ou `o` e incrementamos o contador apropriado.
4. **Determinando o próximo jogador**:
 - Se `count_x` for igual a `count_o`, significa que é a vez de `x` (porque `x` sempre começa).
 - Caso contrário, é a vez de `o`.

Exemplo:

Se o tabuleiro for:

```
[['x', 'o', 'x'],  
 ['o', ' ', ' '],  
 [' ', ' ', ' ']]
```

A função `player` retornará `x` porque há dois `x` e dois `o`.

Se o tabuleiro for:

```
[['x', 'o', 'x'],  
 ['o', 'x', ' '],  
 [' ', ' ', 'o']]
```

A função `player` retornará `o` porque há três `x` e três `o`.

Entendendo a Função `actions`


A função `actions(board)` deve retornar um conjunto de todas as ações possíveis que podem ser tomadas em um determinado estado do tabuleiro.

- Uma "ação" é definida como uma tupla `(i, j)`, onde `i` é o índice da linha e `j` é o índice da coluna da célula onde um jogador pode fazer um movimento.

- As ações possíveis são todas as células que estão vazias (ou seja, contêm `EMPTY`).
- Se o jogo terminou, não há mais ações possíveis, mas a função pode retornar qualquer valor.

Implementação da Função `actions`

Aqui está como você pode implementar a função `actions`:

```
Python 
```

```
def actions(board):  
    """  
    Returns set of all possible actions (i, j) available on the board.  
    """  
    possible_actions = set()  
    for i in range(3):  
        for j in range(3):  
            if board[i][j] == EMPTY:  
                possible_actions.add((i, j))  
    return possible_actions
```

Explicação:

1. `possible_actions = set()`: Inicializamos um conjunto vazio para armazenar as ações possíveis. Usamos um conjunto porque ele garante que não haverá ações duplicadas.
2. Iterando pelo tabuleiro: Usamos dois loops `for` para percorrer cada célula do tabuleiro.
3. Verificando células vazias: Dentro dos loops, verificamos se o valor da célula atual `board[i][j]` é igual a `EMPTY`. Se for, isso significa que a célula está vazia e, portanto, é uma ação possível.
4. Adicionando a ação às ações possíveis: Se a célula estiver vazia, adicionamos a tupla `(i, j)` ao conjunto `possible_actions`.
5. Retornando as ações possíveis: Após percorrer todo o tabuleiro, retornamos o conjunto `possible_actions`, que contém todas as ações possíveis.

Onde inserir o código:

Assim como na função `player`, você deve abrir o arquivo `tictactoe.py` em um editor de texto, encontrar a função `actions` e substituir a linha `raise NotImplementedError` pelo código acima.

Exemplo:

Se o tabuleiro for:

```
[['X', 'O', 'X'],  
 ['O', ' ', ' '],  
 [' ', ' ', ' ']]
```

A função `actions` retornará:

```
{(1, 1), (1, 2), (2, 0), (2, 1), (2, 2)}
```

Essas tuplas representam as posições das células vazias no tabuleiro.

Entendendo a Função `result`

A função `result(board, action)` recebe um estado do tabuleiro e uma ação, e retorna o novo estado do tabuleiro que resulta da aplicação dessa ação.

- A função não deve modificar o tabuleiro original. Em vez disso, deve retornar um novo tabuleiro que representa o estado após a ação. Isso é crucial porque o algoritmo Minimax precisa explorar muitos estados diferentes sem alterar o estado original.
- A ação é uma tupla `(i, j)` que indica a linha e a coluna onde o jogador atual fará o movimento.
- A função deve lançar uma exceção se a ação não for válida (ou seja, se a célula já estiver ocupada).

Implementação da Função `result`

Aqui está como você pode implementar a função `result`:



```
import copy

def result(board, action):
    """
    Returns the board that results from making move (i, j) on the board.
    """
    i, j = action
    if board[i][j] != EMPTY:
        raise Exception("Invalid action")

    new_board = copy.deepcopy(board)
    new_board[i][j] = player(board)
    return new_board
```

Explicação:

1. `import copy` : Importamos o módulo `copy` para usar a função `deepcopy` , que cria uma cópia independente do tabuleiro original.
2. `i, j = action` : Descompactamos a tupla `action` em variáveis `i` (linha) e `j` (coluna) para facilitar o uso.
3. Verificação de ação válida:
 - Verificamos se a célula `board[i][j]` já está ocupada (ou seja, se não é `EMPTY`).
 - Se a célula não estiver vazia, lançamos uma exceção com a mensagem "Invalid action" para indicar que a ação não é permitida.
4. Criando uma cópia do tabuleiro:
 - Usamos `new_board = copy.deepcopy(board)` para criar uma cópia profunda do tabuleiro original. Uma cópia profunda cria um novo objeto composto e, recursivamente, cria cópias de todos os objetos encontrados no original. Isso garante que modificar `new_board` não afetará `board` .
5. Fazendo o movimento:
 - Determinamos qual jogador deve fazer o movimento chamando a função `player(board)` . Isso nos diz se é a vez de 'X' ou 'O'.
 - Atualizamos a célula correspondente em `new_board` com o símbolo do jogador.

6. **Retornando o novo tabuleiro:** Retornamos o `new_board`, que representa o estado do tabuleiro após o movimento.

Onde inserir o código:

Abra o arquivo `tictactoe.py` em um editor de texto, encontre a função `result` e substitua a linha `raise NotImplementedError` pelo código acima. Certifique-se de adicionar a linha `import copy` no início do arquivo, se ainda não estiver lá.

Exemplo:

Se o tabuleiro for:

```
[['X', 'O', 'X'],  
 ['O', ' ', ' '],  
 [' ', ' ', ' ']]
```

e a ação for `(1, 1)`, a função `result` retornará:

```
[['X', 'O', 'X'],  
 ['O', 'X', ' '],  
 [' ', ' ', ' ']]
```

porque é a vez de 'X' e 'X' foi colocado na célula `(1, 1)`.

Entendendo a Função `winner`

A função `winner(board)` deve determinar se há um vencedor no estado atual do tabuleiro.

- O vencedor pode ser 'X' ou 'O'.
- Um jogador vence se tiver três de seus símbolos em linha reta: horizontal, vertical ou diagonal.
- Se não houver vencedor, a função deve retornar `None`.
- Você pode assumir que nunca haverá um tabuleiro onde ambos os jogadores tenham três em linha.

Implementação da Função `winner`

Aqui está como você pode implementar a função `winner`:

```
def winner(board):  
    """  
    Returns the winner of the game, if there is one.  
    """  
    # Check rows  
    for row in board:  
        if row[0] == row[1] == row[2] and row[0] != EMPTY:  
            return row[0]  
  
    # Check columns  
    for col in range(3):  
        if board[0][col] == board[1][col] == board[2][col] and board[0][col] != EM  
            return board[0][col]  
  
    # Check diagonals  
    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != EMPTY:  
        return board[0][0]  
    if board[0][2] == board[1][1] == board[2][0] and board[0][2] != EMPTY:  
        return board[0][2]  
  
    return None
```

Explicação:

1. Verificando linhas:

- Iteramos por cada linha do tabuleiro.
- Para cada linha, verificamos se todas as três células são iguais e diferentes de `EMPTY`. Se forem, retornamos o símbolo dessa célula (que será 'X' ou 'O').

2. Verificando colunas:

- Iteramos por cada coluna do tabuleiro.
- Para cada coluna, verificamos se todas as três células são iguais e diferentes de `EMPTY`. Se forem, retornamos o símbolo dessa célula.

3. Verificando diagonais:

- Verificamos as duas diagonais possíveis.
- Se todas as células de uma diagonal forem iguais e diferentes de `EMPTY`, retornamos o símbolo dessa célula.

4. **Retornando `None`**: Se nenhuma das condições acima for satisfeita, significa que não há vencedor, então retornamos `None`.

Onde inserir o código:

Abra o arquivo `tictactoe.py` em um editor de texto, encontre a função `winner` e substitua a linha `raise NotImplementedError` pelo código acima.

Exemplo:

Se o tabuleiro for:

```
[['X', 'O', 'X'],  
 ['O', 'X', ' '],  
 ['O', ' ', 'X']]
```

A função `winner` retornará `'X'` porque há uma diagonal com 'X'.

Se o tabuleiro for:

```
[['X', 'O', 'O'],  
 ['O', 'X', ' '],  
 ['X', 'O', 'O']]
```

A função `winner` retornará `'O'` porque há uma linha com 'O'.

Se o tabuleiro for:

```
[['X', 'O', 'X'],  
 ['O', 'O', ' '],  
 ['X', 'X', 'O']]
```

A função `winner` retornará `None` porque não há três em linha.

Entendendo a Função `terminal`

A função `terminal(board)` deve verificar se o jogo acabou. Um jogo acaba em duas situações:

- Alguém venceu (ou seja, há três em linha de 'X' ou 'O').
- Todas as células do tabuleiro estão preenchidas e não há vencedor (ou seja, é um empate).

A função deve retornar `True` se o jogo acabou e `False` caso contrário.

Implementação da Função `terminal`

Aqui está como você pode implementar a função `terminal`:

Python

```
def terminal(board):
    """
    Returns True if game is over, False otherwise.
    """
    if winner(board) is not None:
        return True

    for row in board:
        for cell in row:
            if cell == EMPTY:
                return False

    return True
```

Explicação:

1. Verificando se há um vencedor:

- Chamamos a função `winner(board)` para verificar se há um vencedor.
- Se `winner(board)` retornar 'X' ou 'O' (ou seja, se houver um vencedor), a função `terminal` retorna `True` porque o jogo acabou.

2. Verificando se há células vazias:

- Iteramos por todas as células do tabuleiro.
- Se encontrarmos pelo menos uma célula vazia (`EMPTY`), sabemos que o jogo ainda não acabou e retornamos `False` .

3. Retornando `True` se não houver células vazias:

- Se o loop terminar sem encontrar nenhuma célula vazia, significa que todas as células estão preenchidas.
- Como já verificamos que não há vencedor, isso significa que o jogo terminou em empate e retornamos `True` .

Onde inserir o código:

Abra o arquivo `tictactoe.py` em um editor de texto, encontre a função `terminal` e substitua a linha `raise NotImplementedError` pelo código acima.

Exemplo:

Se o tabuleiro for:

```
[['X', 'O', 'X'],  
 ['O', 'X', 'O'],  
 ['O', 'X', 'X']]
```

A função `terminal` retornará `True` porque todas as células estão preenchidas e não há vencedor.

Se o tabuleiro for:

```
[['X', 'O', 'X'],  
 ['O', ' ', 'O'],  
 ['O', 'X', 'X']]
```

A função `terminal` retornará `False` porque ainda há uma célula vazia.

Se o tabuleiro for:

```
[['X', 'O', 'X'],  
 ['O', 'X', 'O'],  
 ['O', ' ', 'X']]
```

A função `terminal` retornará `False` porque ainda há uma célula vazia.

Entendendo a Função `utility`

A função `utility(board)` deve calcular o valor de utilidade de um estado terminal do jogo.

- Em jogos como o Jogo da Velha, a "utilidade" é uma medida do quão bom é um estado final para um dos jogadores.
- No nosso caso:
 - Se 'X' venceu, a utilidade é 1.
 - Se 'O' venceu, a utilidade é -1.
 - Se o jogo terminou em empate, a utilidade é 0.
- A função só será chamada quando o jogo já tiver terminado (ou seja, quando `terminal(board)` for `True`).

Implementação da Função `utility`

Aqui está como você pode implementar a função `utility`:

Python



```
def utility(board):  
    """  
    Returns 1 if X has won the game, -1 if O has won, 0 otherwise.  
    """  
    winner_player = winner(board)  
    if winner_player == 'X':  
        return 1  
    elif winner_player == 'O':  
        return -1  
    else:  
        return 0
```

Explicação:

1. Obtendo o vencedor:

- Chamamos a função `winner(board)` para determinar se há um vencedor.

2. Verificando o vencedor e retornando a utilidade:

- Se o vencedor for 'X', retornamos 1.
- Se o vencedor for 'O', retornamos -1.
- Se não houver vencedor (ou seja, o jogo terminou em empate), retornamos 0.

Onde inserir o código:

Abra o arquivo `tictactoe.py` em um editor de texto, encontre a função `utility` e substitua a linha `raise NotImplementedError` pelo código acima.

Exemplo:

Se o tabuleiro for:

```
[['X', 'O', 'X'],  
 ['O', 'X', 'O'],  
 ['X', 'O', 'X']]
```

A função `utility` retornará 0 porque o jogo terminou em empate.

Se o tabuleiro for:

```
[['X', 'O', 'X'],  
 ['O', 'X', 'O'],  
 ['X', ' ', 'X']]
```

A função `utility` retornará 1 porque 'X' venceu.

Se o tabuleiro for:

```
[['O', 'X', 'O'],  
 ['O', 'O', 'X'],  
 ['X', 'X', 'O']]
```

A função `utility` retornará -1 porque 'O' venceu.

Entendendo a Função `minimax`

A função `minimax(board)` deve retornar a ação ótima para o jogador atual no tabuleiro.

- O algoritmo Minimax é um algoritmo recursivo usado para escolher o melhor movimento para um jogador, assumindo que o oponente também jogará de forma ótima.
- Ele funciona explorando todos os possíveis movimentos futuros e atribuindo uma pontuação a cada um deles.
- O jogador maximizador (nós) tenta escolher o movimento com a maior pontuação, enquanto o jogador minimizador (o oponente) tenta escolher o movimento com a menor pontuação.
- No Jogo da Velha, com jogo perfeito de ambos os lados, o resultado é sempre um empate.

Implementação da Função `minimax`

Aqui está como você pode implementar a função `minimax`

```
def minimax(board):
    """
    Returns the optimal action for the current player on the board.
    """
    if terminal(board):
        return None

    current_player = player(board)

    def max_value(board):
        if terminal(board):
            return utility(board)
        v = -math.inf
        for action in actions(board):
            v = max(v, min_value(result(board, action)))
        return v

    def min_value(board):
        if terminal(board):
            return utility(board)
        v = math.inf
        for action in actions(board):
            v = min(v, max_value(result(board, action)))
        return v

    best_action = None
    if current_player == X:
        best_value = -math.inf
        for action in actions(board):
            value = min_value(result(board, action))
            if value > best_value:
                best_value = value
                best_action = action
    else:
        best_value = math.inf
        for action in actions(board):
            value = max_value(result(board, action))
            if value < best_value:
                best_value = value
                best_action = action

    return best_action
```

Explicação:

1. Caso base:

- Se o jogo acabou (ou seja, `terminal(board)` é `True`), não há movimento a ser feito, então retornamos `None`.

2. Determinando o jogador atual:

- Usamos `current_player = player(board)` para saber se é a vez de 'X' ou 'O'.

3. Funções `max_value` e `min_value`:

- Estas são funções recursivas que implementam o algoritmo Minimax.
- `max_value(board)`:
 - Se o jogo acabou, retorna a utilidade do tabuleiro.
 - Caso contrário, inicializa `v` com o menor valor possível (`-math.inf`).
 - Itera por todas as ações possíveis e calcula o valor mínimo (`min_value`) do tabuleiro resultante de cada ação.
 - Atualiza `v` com o máximo entre o valor atual de `v` e o valor mínimo calculado.
 - Retorna o valor máximo `v`.
- `min_value(board)`:
 - Se o jogo acabou, retorna a utilidade do tabuleiro.
 - Caso contrário, inicializa `v` com o maior valor possível (`math.inf`).
 - Itera por todas as ações possíveis e calcula o valor máximo (`max_value`) do tabuleiro resultante de cada ação.
 - Atualiza `v` com o mínimo entre o valor atual de `v` e o valor máximo calculado.
 - Retorna o valor mínimo `v`.

4. Encontrando a melhor ação:

- Inicializamos `best_action` como `None` e `best_value` com o menor (para 'X') ou maior (para 'O') valor possível.
- Iteramos por todas as ações possíveis.
- Para cada ação, calculamos o valor usando `min_value` (se for a vez de 'X') ou `max_value` (se for a vez de 'O').
- Se o valor for melhor do que o `best_value` atual, atualizamos `best_value` e `best_action`.

5. Retornando a melhor ação:

- Após verificar todas as ações, retornamos a `best_action`, que é a ação ótima para o jogador atual.