Xadrez de Listas, Filas e Pilhas

Andrey Justen Junior – 1072316310 | Programação, Design

Bárbara Prim de Souza – 10723114770 | Design, Gestão, Apresentação

Leonardo Alves Silva – 10723113466 | Programação, Documentação

Mateus Nunes Lehmkuh – 1072312497 | Design, Documentação

❖ Descrição Geral do Jogo

Este projeto implementa um simulador de xadrez em Python, utilizando a biblioteca Tkinter para a interface gráfica. O tabuleiro padrão (8×8) é renderizado como um Canvas, e cada peça é representada por uma classe específica com sua lógica de movimento.

O usuário interage clicando nas casas: ao selecionar uma peça, são destacados seus movimentos válidos; ao clicar em um destino válido, a peça é animada em um movimento suave para a nova posição. Também é possível desfazer movimentos (undo) usando Ctrl+Z, restaurando o estado anterior do jogo.

❖ Uso de Estruturas de Dados

1. Lista

self.board.grid = [[None]*BOARD_SIZE for _ in range(BOARD_SIZE)]

Tipo: *lista de listas (matriz 8×8)*

Uso: *representa as 64 casas do tabuleiro, cada elemento é None ou um objeto Piece.*

Justificativa: *matrizes em Python são naturalmente implementadas como listas de listas, permitindo acesso direto por índice grid[r][c] e iteração simples nas linhas e colunas.*

PIECE_IMAGES = { ... }

Tipo: *dicionário, mas os valores associados (e.g. chaves listadas) são derivados de literais de lista em código estático (não dinamicamente).*

Uso: *mapeia cada peça ('wp', 'bn' etc.) para o arquivo de imagem correspondente.*

Justificativa: *embora seja um dicionário, ele faz uso de coleções literais para agrupar as referências de imagem — mas não é uma lista dinâmica, apenas armazenagem fixa.*

order = [Rook, Night, Bishop, Queen, King, Bishop, Night, Rook]

Tipo: *lista simples de classes*

Uso: *define a ordem das peças na primeira e última fileiras ao inicializar o tabuleiro.*

Justificativa*: a lista é clara, imutável e sequencial, adequando-se perfeitamente para mapear colunas a classes de peça*.

Night.OFFSETS = [(2,1), (2,-1), ...]

Tipo: *lista de tuplas*

Uso: *deslocamentos relativos que o cavalo pode executar em cada movimento*.

Justificativa: *fácil extensibilidade, iteração e manutenção dos vetores de movimento sem lógica condicional complexa*.

self.possible_moves = []

Tipo: *lista mutável*

Uso: *armazena, a cada seleção de peça, a lista de casas para onde ela pode se mover (tuplas (r,c))*.

Justificativa: *lista é ideal para agrupar um número variável de movimentos possíveis e iterar sobre eles na hora de desenhar os destaques no tabuleiro.*

Listas de direções em _**sliding:**

Uso*: passam como parâmetro a lista de direções (e.g. [(1,1),(1,-1), ...]) para Bispo, Torre e Rainha*.

Justificativa: *separar a lógica genérica de "movimento deslizante" de cada peça, fornecendo apenas um array de vetores a ser percorrido.*

2. Fila

self.turns = ['w', 'b'] self.turns = ['w', 'b']

Tipo: *lista usada como fila circular*

Uso: *controla a ordem de vez dos jogadores: o primeiro elemento (índice 0) é quem joga. Após cada movimento, faz-se self.turns.append(self.turns.pop(0)), deslocando o jogador atual para o fim.*

Justificativa: *usar uma lista como fila FIFO é simples em Python; pop(0) retira o elemento da frente, append(...) rein-sere ao fim. Funciona bem para dois jogadores e escalaria para N.*.

3. Pilha

self.history = []

***Tipo**: lista usada como pilha (LIFO)*

***Uso**: armazena tuplas de "estado antes do movimento" — (r0, c0, piece, r1, c1, captured) — toda vez que um movimento é confirmado.*

***Justificativa**: implementar undo via pilha é padrão: o append() empilha um novo estado, e o pop() no método undo() restaura o último movimento, desfazendo-o na ordem inversa.*

- ## Por que usar essas estruturas?

Listas em Python oferecem acesso por índice em O(1) e são versáteis para coleções de tamanho variável, ideais para tabuleiros, coleções de offsets e movimentos gerados dinamicamente.

Filas modelam perfeitamente turnos em jogos: FIFO garante que a vez "ande" de forma circular entre jogadores.

Pilhas espelham o comportamento de "desfazer" ações, pois a operação mais recente deve ser revertida primeiro (LIFO).

Todas essas escolhas aproveitam diretamente as operações nativas de Python (append, pop, indexação) sem necessidade de estruturas externas, mantendo o código simples, legível e eficiente para o escopo de um jogo de xadrez em Tkinter.

- ❖ **Análise de Complexidade de Tempo**
  - **Lista**

Sequência indexada de elementos, aqui usada para **representar o tabuleiro e coleções de offsets e movimentos**.

Exemplo de operação crítica: acessar e atribuir em grid[r][c]

piece = self.board.grid[r][c]

self.board.grid[r][c] = new_piece

- **Leitura** piece = grid[r][c] → O(1)
- **Escrita** grid[r][c] = new_piece → O(1)

Indexação direta em listas Python tem tempo constante, ideal para acesso rápido ao tabuleiro.

- **Fila**

Alternância de turnos (remoção do primeiro elemento da fila de turnos)

```
jogador = self.turns.pop(0)   # remove da frente
self.turns.append(jogador)   # insere no fim
```

- **pop(0) em lista:** O(n) no pior caso, pois todos os elementos subsequentes devem ser realocados para esquerda.
- **append em lista:** Amortizado O(1).

Logo, a alternância de turnos nessa implementação é **O(n)**, onde *n* é o número de jogadores (aqui fixo em 2, logo é constante na prática). Em um cenário genérico ou com mais participantes, esse custo poderia crescer linearmente.

- **Pilha**

Estrutura LIFO (último a entrar, primeiro a sair), usada para **armazenar histórico de movimentos** e permitir "undo".

```
self.history.append((r0, c0, piece, r1, c1, captured))    # Salvar estado antes do movimento
state = self.history.pop()                                # Desfazer movimento
```

- **append(...)** → O(1) amortizado
- **pop()** → O(1)

Empilhar e desempilhar estados é sempre constante, garantindo undo rápido e eficiente.

- **Instruções para executar o jogo**

  - **Requisitos**

Necessário a instalação da linguagem, **Python**;
Recomendo a adição do Python ao caminho "PATH" durante instalação;
https://www.python.org/downloads/
"Add python.exe to PATH"

  - **Execução**

Baixe o código fonte do código;

Extraia, então abra "Xadrez-Lista-Fila-Pilha-main" onde pode-se ver chess.py;

Abra o terminal (CMD, Command Prompt) neste local (ou execute "cd "local do arquivo"")

Insira o seguinte comando;

"python chess.py"

- **Link para o quadro do Trello do grupo.**

https://trello.com/b/wvLccU9P/a3-estrutura-de-dados-e-an%C3%A1lise-de-algoritimos

---

- **Link para o repositório Git do grupo.**

https://github.com/AndreyJR777/Xadrez-Lista-Fila-Pilha
