Projeto Final - Batalha Naval

Desenvolvido por Ahmed Hachem e João Siqueira
Professor: Guido Pantuza

Índice

Problema Proposto

O projeto foi desenvolvido como resposta a um trabalho acadêmico com os seguintes requisitos específicos:

Requisitos do Jogo

1. **Estrutura Básica**
    - Jogo no formato jogador contra computador
    - Dois tabuleiros de 10 x 10 posições (um para cada participante)
    - 5 embarcações por jogador, cada uma ocupando uma única posição
2. **Mecânicas de Jogo**
    - Posicionamento inicial das embarcações por ambos os participantes
    - Visualização do tabuleiro adversário durante a partida
    - Indicação clara de posições já utilizadas:
        - Água (tiro errado)
        - Embarcação afundada (tiro certeiro)
    - Sistema de turnos alternados
    - Jogador ganha tiro extra ao acertar uma embarcação
3. **Condições de Vitória**
    - Vence quem afundar primeiro todas as embarcações do adversário
4. **Funcionalidades Especiais**
    - Comando especial para visualização do tabuleiro do computador (modo teste)

Solução Implementada

O projeto atende a todos os requisitos propostos através de:

- Interface em linha de comando com visualização clara dos dois tabuleiros

- Sistema de coordenadas intuitivo para posicionamento e tiros
- Feedback visual com cores diferentes para água e embarcações atingidas
- Comando `ver` para revelar o tabuleiro do computador
- Lógica de turno extra implementada ao acertar uma embarcação
- Sistema de validação para garantir regras do jogo

Visão Geral

O projeto implementa uma versão simplificada do clássico jogo Batalha Naval, onde o jogador compete contra o computador. Cada jogador possui 5 navios de tamanho único dispostos em um tabuleiro 10x10, e o objetivo é afundar todos os navios do oponente antes que ele afunde os seus.

Características Principais

- Interface em linha de comando com cores
- Tabuleiro 10x10 com visualização lado a lado
- 5 navios por jogador
- Modo de visualização para debugging
- Sistema de coordenadas baseado em linha e coluna
- IA simples para o computador

Estrutura do Projeto

O projeto é organizado em classes e funções principais:

```
batalha_naval/
│
├── Classes
│   ├── Navio
│   ├── TabuleiroNaval
│   └── JogoBatalhaNaval
│
└── Funções
    ├── menu()
    ├── mostrar_instrucoes()
    └── mostrar_creditos()
```

Classes

Classe Navio

```
class Navio:
    def __init__(self, posicao: Tuple[int, int])
    def esta_afundado(self) -> bool
```

Atributos

- `posicao`: Tupla (x, y) indicando a posição do navio no tabuleiro
- `afundado`: Booleano indicando se o navio foi atingido

Métodos

- `esta_afundado()`: Retorna o estado atual do navio

Classe TabuleiroNaval

```
class TabuleiroNaval:
    def __init__(self)
    def is_valid_posicao(self, x: int, y: int) -> bool
    def is_cell_free(self, x: int, y: int) -> bool
    def posicionar_navio(self, x: int, y: int) -> Tuple[bool, str]
    def receber_tiro(self, x: int, y: int) -> Tuple[Optional[bool], bool, str]
    def cell_content(self, i: int, j: int, mostrar_navios: bool) -> str
    def mostrar(self, mostrar_navios: bool = False) -> list
    def navios_restantes(self) -> int
```

Atributos

- `Navios` : Lista de objetos Navio
- `tiros` : Conjunto de coordenadas (x, y) onde já foram realizados tiros
- `tamanho` : Tamanho do tabuleiro (10x10)

Métodos

- `is_valid_posicao()` : Verifica se uma posição está dentro dos limites do tabuleiro
- `is_cell_free()` : Verifica se uma célula está livre para posicionar um navio
- `posicionar_navio()` : Tenta posicionar um navio na posição especificada
- `receber_tiro()` : Processa um tiro recebido e retorna o resultado
- `cell_content()` : Retorna o conteúdo de uma célula para exibição
- `mostrar()` : Gera a representação visual do tabuleiro
- `navios_restantes()` : Retorna a quantidade de navios não afundados

Classe JogoBatalhaNaval

```
class JogoBatalhaNaval:
    def __init__(self)
    def limpar_tela(self)
    def ler_coordenadas(self, mensagem: str) -> Tuple[Optional[int], Optional[int],
Optional[str]]
    def configurar_jogo(self)
    def jogada_computador(self) -> bool
    def mostrar_status_jogo(self)
    def mostrar_tabuleiros(self, mostrar_navios_computador: bool = False)
    def jogar(self)
```

Atributos

- `tabuleiro_jogador` : Instância de TabuleiroNaval para o jogador
- `tabuleiro_computador` : Instância de TabuleiroNaval para o computador

Métodos

- `limpar_tela()` : Limpa o terminal
- `ler_coordenadas()` : Lê e valida as coordenadas inseridas pelo usuário
- `configurar_jogo()` : Inicializa o jogo posicionando os navios
- `jogada_computador()` : Processa a jogada do computador
- `mostrar_status_jogo()` : Exibe o status atual do jogo
- `mostrar_tabuleiros()` : Exibe os tabuleiros lado a lado
- `jogar()` : Controla o fluxo principal do jogo

Funções Principais

menu()

Função principal que exibe o menu do jogo e gerencia as opções do usuário:

- Novo Jogo
- Instruções
- Créditos
- Sair

mostrar_instrucoes()

Exibe as regras e instruções do jogo, incluindo:

- Número de navios
- Como posicionar navios
- Como realizar tiros
- Significado dos símbolos no tabuleiro
- Comandos especiais

mostrar_creditos()

Exibe os créditos do jogo e informações sobre os desenvolvedores.

Interface do Usuário

Elementos Visuais

- Bordas decorativas usando caracteres Unicode
- Cores diferentes para diferentes elementos:
  - **CYAN**: Títulos e bordas
  - **GREEN**: Navios e acertos
  - **RED**: Erros e mensagens de aviso
  - **BLUE**: Água e tiros na água
  - **YELLOW**: Mensagens de sistema

Comandos do Usuário

- **Coordenadas**: `linha coluna` (ex: "3 4")
- **Comandos Especiais**:
  - `ver` : Mostra temporariamente os navios do computador
  - `sair` : Encerra o jogo atual

Mecânicas do Jogo

Posicionamento

- **Jogador**: Manual, através de coordenadas
- **Computador**: Aleatório

Sistema de Turnos

1. Jogador atira.
2. Se acertar, ganha direito a outro tiro.
3. Computador atira (caso o jogador erre).

4. Se o computador acertar, ganha direito a outro tiro.
5. Repete até que um dos jogadores vença.

Condições de Vitória

- Afundar todos os 5 navios do oponente.

Requisitos do Sistema

Dependências

- Python 3.6 ou superior
- Biblioteca **colorama**

Instalação de Dependências

```
pip install colorama
```

Como Executar

1. Instale as dependências.
2. Execute o arquivo principal:

   ```
   python batalha_naval.py
   ```

Fluxo de Jogo

1. Selecione "Novo Jogo" no menu.
2. Posicione seus 5 navios.
3. Alterne turnos com o computador até que haja um vencedor.
4. Retorne ao menu principal.

Observações Importantes

- O jogo valida todas as entradas do usuário.
- Coordenadas inválidas são rejeitadas com mensagens de erro apropriadas.
- O modo de visualização (`ver`) não afeta o estado do jogo.
- O jogo pode ser encerrado a qualquer momento com o comando `sair` .

Casos de Teste

1. Testes de Inicialização do Jogo

1.1 Menu Principal

```python
# Teste do Menu Principal
def test_menu_principal():
    # Entradas esperadas: 1, 2, 3, 4
    assert menu_valida_entrada("1") == True  # Novo Jogo
    assert menu_valida_entrada("2") == True  # Instruções
    assert menu_valida_entrada("3") == True  # Créditos
    assert menu_valida_entrada("4") == True  # Sair
    assert menu_valida_entrada("5") == False # Opção inválida
```

1.2 Posicionamento de Navios

```
# Teste de Posicionamento de Navios
def test_posicionamento_navios():
    tabuleiro = TabuleiroNaval()

    # Teste de posição válida
    assert tabuleiro.posicionar_navio(0, 0)[0] == True

    # Teste de posição fora do tabuleiro
    assert tabuleiro.posicionar_navio(10, 10)[0] == False

    # Teste de posição ocupada
    assert tabuleiro.posicionar_navio(0, 0)[0] == False
```

2. Testes de Mecânicas do Jogo

2.1 Validação de Tiros

```
# Teste de Validação de Tiros
def test_validacao_tiros():
    tabuleiro = TabuleiroNaval()

    # Teste de tiro válido
    hit, game_over, _ = tabuleiro.receber_tiro(5, 5)
    assert hit is not None

    # Teste de tiro repetido
    hit, game_over, _ = tabuleiro.receber_tiro(5, 5)
    assert hit is None

    # Teste de tiro fora do tabuleiro
    hit, game_over, _ = tabuleiro.receber_tiro(10, 10)
    assert hit is None
```

2.2 Condições de Vitória

```
# Teste de Condições de Vitória
def test_condicoes_vitoria():
    jogo = JogoBatalhaNaval()
    tabuleiro = TabuleiroNaval()

    # Posiciona um navio e atira nele
    tabuleiro.posicionar_navio(0, 0)
    hit, game_over, _ = tabuleiro.receber_tiro(0, 0)

    # Verifica se o navio foi atingido
    assert hit == True

    # Verifica se o jogo acabou (com apenas um navio)
    assert game_over == (tabuleiro.navios_restantes() == 0)
```

3. Testes de Interface

3.1 Entrada de Coordenadas

```python
# Teste de Entrada de Coordenadas
def test_entrada_coordenadas():
    jogo = JogoBatalhaNaval()

    # Teste de entrada válida
    x, y, cmd = jogo.ler_coordenadas("0 0")
    assert x == 0 and y == 0

    # Teste de comando 'ver'
    x, y, cmd = jogo.ler_coordenadas("ver")
    assert cmd == 'ver'

    # Teste de comando 'sair'
    x, y, cmd = jogo.ler_coordenadas("sair")
    assert cmd == 'sair'

    # Teste de entrada inválida
    x, y, cmd = jogo.ler_coordenadas("abc")
    assert x is None and y is None
```

3.2 Visualização do Tabuleiro

```python
# Teste de Visualização do Tabuleiro
def test_visualizacao_tabuleiro():
    tabuleiro = TabuleiroNaval()

    # Teste de célula vazia
    assert '~' in tabuleiro.cell_content(0, 0, False)

    # Teste de navio visível
    tabuleiro.posicionar_navio(1, 1)
    assert 'N' in tabuleiro.cell_content(1, 1, True)

    # Teste de tiro na água
    tabuleiro.receber_tiro(2, 2)
    assert 'O' in tabuleiro.cell_content(2, 2, False)

    # Teste de navio atingido
    tabuleiro.receber_tiro(1, 1)
    assert 'X' in tabuleiro.cell_content(1, 1, False)
```

4. Testes de Integração

4.1 Fluxo Completo de Jogo

```python
# Teste de Fluxo Completo
def test_fluxo_completo():
    jogo = JogoBatalhaNaval()
```

```
    # Verifica se todos os navios foram posicionados
    assert len(jogo.tabuleiro_jogador.Navios) == 5
    assert len(jogo.tabuleiro_computador.Navios) == 5

    # Verifica se os navios estão em posições válidas
    for navio in jogo.tabuleiro_jogador.Navios:
        x, y = navio.posicao
        assert 0 <= x < 10 and 0 <= y < 10
```

5. Casos de Teste Específicos

5.1 Validações de Regras

- Verificar se o jogador ganha turno extra ao acertar um navio.
- Verificar se o computador ganha turno extra ao acertar um navio.
- Verificar se não é possível sobrepor navios.
- Verificar se o modo de visualização não afeta o estado do jogo.

5.2 Casos de Borda

- Tentar posicionar navios em todas as bordas do tabuleiro.
- Tentar atirar em todas as bordas do tabuleiro.
- Verificar comportamento com entradas limítrofes.

6. Resultados Esperados

Para cada teste:

1. **Funcionamento Normal do Jogo:**
    - O menu deve responder corretamente a todas as opções.
    - O posicionamento de navios deve respeitar as regras.
    - Os tiros devem ser processados corretamente.
    - O sistema de turnos deve funcionar conforme especificado.
2. **Tratamento de Erros:**
    - Entradas inválidas devem ser rejeitadas apropriadamente.
    - Mensagens de erro devem ser claras e informativas.
    - O jogo deve manter um estado consistente após erros.
3. **Interface do Usuário:**
    - Os tabuleiros devem ser exibidos corretamente.
    - O feedback visual deve ser claro e consistente.
    - Os comandos especiais devem funcionar conforme esperado.

Resultados dos Testes

Rodei o arquivo de teste e o resultado foi o seguinte:

```
Testa se o navio não está afundado ao ser criado. ... Iniciando teste: Testa se o
navio não está afundado ao ser criado.
ok
 SUCCESS: Testa se o navio não está afundado ao ser criado.
Testa a verificação de célula livre e o posicionamento de navios. ... Iniciando teste:
Testa a verificação de célula livre e o posicionamento de navios.
```

```
ok
 SUCCESS: Testa a verificação de célula livre e o posicionamento de navios.
Testa se as posições são validadas corretamente. ... Iniciando teste: Testa se as
posições são validadas corretamente.
ok
 SUCCESS: Testa se as posições são validadas corretamente.
Testa a contagem de navios restantes. ... Iniciando teste: Testa a contagem de navios
restantes.
ok
 SUCCESS: Testa a contagem de navios restantes.
Testa os tiros no tabuleiro. ... Iniciando teste: Testa os tiros no tabuleiro.
ok
 SUCCESS: Testa os tiros no tabuleiro.

----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

Explicação

Os testes foram executados com sucesso e validaram diferentes aspectos do funcionamento do jogo. Aqui está o que cada teste verificou:

1. **Testa se o navio não está afundado ao ser criado**: Confirma que um navio recém-criado não está afundado.
2. **Testa a verificação de célula livre e o posicionamento de navios**: Garante que os navios são posicionados corretamente em células livres.
3. **Testa se as posições são validadas corretamente**: Verifica se o jogo aceita apenas posições válidas no tabuleiro.
4. **Testa a contagem de navios restantes**: Checa se a contagem de navios ainda ativos está correta.
5. **Testa os tiros no tabuleiro**: Testa se os tiros estão sendo registrados e processados corretamente.

O resultado `OK` ao final confirma que todos os testes passaram sem erros.

Como rodar os testes

Para executar os testes, utilize o seguinte comando no terminal:

```
python3 -m unittest test_batalha_naval.py
```

Caso deseje ver os testes em modo mais detalhado, utilize:

```
python3 test_batalha_naval.py
```

Isso imprimirá cada teste executado, mostrando claramente os resultados e mensagens de erro, se houverem.