FUNDAÇÃO EDSON QUEIROZ

UNIVERSIDADE DE FORTALEZA - UNIFOR

CENTRO DE CIÊNCIAS TECNOLÓGICAS

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ATIVIDADE FINAL - Jogo Sokoban em Python

N704 - PROGRAMAÇÃO FUNCIONAL

EQUIPE:

Paulo de Oliveira Valente - 2222854

Pedro Ernesto Figueiredo - 2020733

Igor Oliveira Girão - 2213921

Repositório do Projeto:

https://github.com/GrifoEXE/Sokoban-Game-Programa-ao-Funcional

DOCUMENTO DE REQUISITOS

Papeis dos Integrantes do Grupo:

-Paulo de Oliveira Valente

- Papel: Desenvolvedor
- Responsabilidades:
  - Implementar o código-fonte do jogo Sokoban em Python.
  - Garantir que as funcionalidades do jogo estejam corretas e funcionais.
  - Colaborar com os outros membros do grupo para integração e testes.

-Pedro Ernesto Figueiredo

- Papel: Desenvolvedor e Testador
- Responsabilidades:
  - Implementação do código-fonte do jogo Sokoban em Python.
  - Elaborar casos de teste para verificar o funcionamento correto do jogo.
  - Realizar testes de unidade, integração e aceitação do software.
  - Documentar os resultados dos testes.

-Igor Oliveira Girão

- Papel: Documentador
- Responsabilidades:
  - Elaborar a documentação do projeto, incluindo o documento de requisitos.
  - Atualizar a documentação conforme necessário durante o desenvolvimento.

Requisitos Funcionais

Os requisitos funcionais da aplicação estão descritos a seguir:

RF-01: O sistema deve permitir a movimentação do personagem.

-**Descrição:** O jogador deve ser capaz de mover o personagem pelo cenário utilizando as teclas de direção.

- **Entradas necessárias:** Teclas de direção (direita, esquerda, cima, baixo).
- **Saídas esperadas:** Atualização do cenário com a nova posição do personagem.

-Implementação:

função get_key() captura os inputs e retorna o evento de keydown

```
async def get_key():
    while True:
        event = keyboard.read_event(suppress=True)
        if event.event_type == keyboard.KEY_DOWN:
            return event.name
```

dentro do loop da função sokoban()

```
while True:
    print_level()
    print("Use as setas do teclado para mover o jogador.")
    print("Pressione Q para sair.")

    key = await get_key()

    if key == 'q':
        break

    delta = {'up': (-1, 0), 'left': (0, -1), 'down': (1, 0), 'right': (0, 1)}.get(key)
```

verifica para onde o jogador deve se mover dependendo do valor de delta, também tornando a sua posição antiga em vazia

```
if delta is not None:
    new_player_pos = move(player_pos, delta)

    is_valid_move = is_valid(new_player_pos, level)
    if is_valid_move.success:
        if is_box(level)(new_player_pos):
            new_box_pos = move(new_player_pos, delta)
            is_valid_box_move = is_valid(new_box_pos, level)
            if is_valid_box_move.success and not is_box(level)(new_box_pos):
                boxes.remove(new_player_pos)
                boxes.add(new_box_pos)
                level[new_player_pos[0]][new_player_pos[1]] = ' '
                level[new_box_pos[0]][new_box_pos[1]] = '$'
            else:
                continue

        level[player_pos[0]][player_pos[1]] = ' '
        player_pos = new_player_pos
        level[player_pos[0]][player_pos[1]] = '@'
```

RF-02: O Sistema deve permitir a leitura do cenário a partir de um arquivo.

-Descrição: O sistema deve ser capaz de ler o cenário de cada fase a partir de um arquivo de texto. Saídas esperadas: Carregamento correto do cenário no jogo, permitindo que o jogador interaja com o ambiente conforme especificado no arquivo.

- **Entradas Necessárias:** Arquivo de texto contendo o cenário, onde cada caractere representa um elemento do cenário (parede, jogador, caixa, objetivo, espaço vazio).
- **Saídas Esperadas:** Carregamento correto do cenário no jogo, permitindo que o jogador interaja com o ambiente conforme especificado no arquivo.

-Implementação:

Leitura e impressão do nível

```python
# Função principal do jogo
async def sokoban(file_path):
    # Lê o arquivo do nível
    with open(file_path, 'r') as file:
        level = [list(line.strip()) for line in file]

    # Encontra a posição inicial do jogador
    player_pos = next((i, j) for i, row in enumerate(level) for j, char in enumerate(row) if char == '@')

    # Inicializa as posições das caixas e dos objetivos usando List Comprehensions
    boxes = {(i, j) for i, row in enumerate(level) for j, char in enumerate(row) if char == '$'}
    goals = {(i, j) for i, row in enumerate(level) for j, char in enumerate(row) if char == '.'}

    # Função para imprimir o nível usando List Comprehension
    def print_level():
        print('\n'.join(''.join(row) for row in level))
```

-Montagem do cenário em “mapa.txt”:

```
1       #######
2       #@ $ .#
3       #  $  #
4       #  .  #
5       #######
```

- O caractere '@' representa o Jogador
- O caractere '$' representa a caixa
- O caractere '.' representa os pontos para onde as caixas devem ser empurradas
- O caractere '#' representa as paredes/obstáculos do nível

RF-03: O Sistema deve ter restrição de movimento

-Descrição: O código deve impedir que o jogador e as caixas passem pelas paredes do nível.

- **Entradas Necessárias:** Verificação da posição do jogador e das caixas em relação às paredes do cenário.
- **Saídas Esperadas:** Bloqueio do movimento do jogador e das caixas quando encontrarem uma parede, mantendo-os em suas posições atuais

Implementação:

```python
# Função de continuação para verificar se uma posição é válida
def is_valid(pos, level):
    if pos[0] < 0 or pos[0] >= len(level) or pos[1] < 0 or pos[1] >= len(level[0]):
        return Result(False, error="Out of bounds")
    return Result(level[pos[0]][pos[1]] != '#')

# Função de alta ordem para verificar se uma posição é um ponto de destino
def is_goal(level):
    return lambda pos: level[pos[0]][pos[1]] == '.'

# Closure para verificar se uma posição contém uma caixa
def is_box(level):
    return lambda pos: level[pos[0]][pos[1]] == '$'

# Closure para verificar se uma posição está livre
def is_free(level):
    return lambda pos: level[pos[0]][pos[1]] in [' ', '.']
```

verificação usando is_valid() na movimentação do jogador e das caixas

```python
is_valid_move = is_valid(new_player_pos, level)
if is_valid_move.success:
    if is_box(level)(new_player_pos):
        new_box_pos = move(new_player_pos, delta)
        is_valid_box_move = is_valid(new_box_pos, level)
        if is_valid_box_move.success and not is_box(level)(new_box_pos):
            boxes.remove(new_player_pos)
            boxes.add(new_box_pos)
            level[new_player_pos[0]][new_player_pos[1]] = ' '
            level[new_box_pos[0]][new_box_pos[1]] = '$'
        else:
            continue

    level[player_pos[0]][player_pos[1]] = ' '
    player_pos = new_player_pos
    level[player_pos[0]][player_pos[1]] = '@'
```

- A restrição de movimento é implementada no código que calcula a próxima posição do jogador e das caixas. Antes de mover o jogador ou as caixas para uma nova posição, é verificado se essa posição é uma parede ('#'). Se for, o movimento é bloqueado e o jogador ou a caixa permanece em sua posição atual.

RF-04: O Sistema deve permitir o movimento de empurrar as caixas em espaços vazios

-Descrição: Se o jogador estiver adjacente a alguma caixa e ele se mover para o lado oposto, a caixa também deverá se mover.

- **Entradas Necessárias:** Verificação da posição do jogador e das caixas em relação um ao outro.
- **Saídas Esperadas:** Movimento das caixas junto com o jogador quando ele as empurra para o lado oposto.

Implementação:

```
# Função de continuação para verificar se uma posição é válida
def is_valid(pos, level):
    if pos[0] < 0 or pos[0] >= len(level) or pos[1] < 0 or pos[1] >= len(level[0]):
        return Result(False, error="Out of bounds")
    return Result(level[pos[0]][pos[1]] != '#')

# Função de alta ordem para verificar se uma posição é um ponto de destino
def is_goal(level):
    return lambda pos: level[pos[0]][pos[1]] == '.'

# Closure para verificar se uma posição contém uma caixa
def is_box(level):
    return lambda pos: level[pos[0]][pos[1]] == '$'

# Closure para verificar se uma posição está livre
def is_free(level):
    return lambda pos: level[pos[0]][pos[1]] in [' ', '.']
```

```
# Função lambda para mover o jogador ou caixa
move = lambda pos, delta: (pos[0] + delta[0], pos[1] + delta[1])
```

```
if delta is not None:
    new_player_pos = move(player_pos, delta)

    is_valid_move = is_valid(new_player_pos, level)
    if is_valid_move.success:
        if is_box(level)(new_player_pos):
            new_box_pos = move(new_player_pos, delta)
            is_valid_box_move = is_valid(new_box_pos, level)
            if is_valid_box_move.success and not is_box(level)(new_box_pos):
                boxes.remove(new_player_pos)
                boxes.add(new_box_pos)
                level[new_player_pos[0]][new_player_pos[1]] = ' '
                level[new_box_pos[0]][new_box_pos[1]] = '$'
            else:
                continue
```

- O movimento de empurrar das caixas é implementado dentro do código que calcula a próxima posição do jogador. Se o jogador estiver adjacente a uma caixa e se mover para uma posição vazia adjacente à caixa, a caixa é movida para a posição anteriormente ocupada pelo jogador.

RF-05: O Sistema deve concluir o jogo ao jogador empurrar todas as caixas para os objetivos.

-Descrição: Quando o jogador empurra todas as caixas '$' para a posição dos pontos '.', o jogo acaba e o jogador vence.

- **Entradas Necessárias:** Verificação da posição das caixas e dos pontos no cenário.
- **Saídas Esperadas:** Impressão da mensagem de vitória, caso não há mais '.' no mapa, em relação às caixas '$' e paragem da execução código

Implementação:

```
if not boxes.symmetric_difference(goals):
    print_level()
    print("Parabéns, você venceu!")
    break
```

- A Condição de vitória é realizada dentro do código que calcula o movimento do jogador. Se todas as caixas são empurradas para posições que correspondem a um ponto, o jogador vence o jogo.

RF-06: O Sistema deve esperar o input do jogador para imprimir a próxima frame do jogo

-Descrição: Após a impressão do cenário inicial, o jogo imprimirá a próxima frame do jogo apenas após o input do jogador.

- **Entradas Necessárias:** Input do jogador após a impressão do cenário inicial.
- **Saídas Esperadas:** Impressão da próxima frame do jogo somente após o input do jogador, permitindo que ele controle o fluxo do jogo.

Implementação:

```
async def get_key():
    while True:
        event = keyboard.read_event(suppress=True)
        if event.event_type == keyboard.KEY_DOWN:
            return event.name

while True:
    print_level()
    print("Use as setas do teclado para mover o jogador.")
    print("Pressione Q para sair.")

    key = await get_key()
```

```
# Função para imprimir o nível usando List Comprehension
def print_level():
    print('\n'.join(''.join(row) for row in level))
```

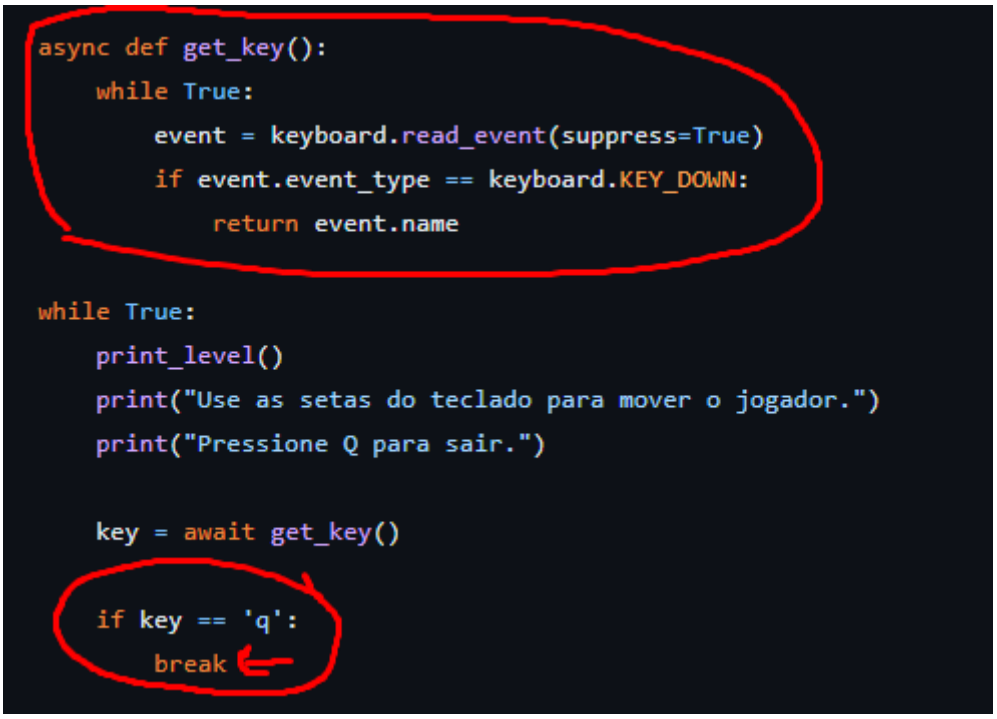
- Após a leitura do cenário inicial do arquivo e sua impressão na tela, o jogo espera pela interação do jogador antes de continuar. Isso significa que o jogo não avançará automaticamente para o próximo passo sem que o jogador realize alguma ação. Essa espera pelo input do jogador é realizada dentro do loop principal do jogo. Após o jogador fornecer uma entrada, como pressionar uma tecla, o jogo calculará a próxima frame do jogo com base nessa entrada e a imprimirá na tela. Esse processo de

espera pelo input do jogador antes de avançar para a próxima frame garante que o jogador tenha controle sobre o fluxo do jogo e possa interagir com ele de acordo com sua vontade.

RF-07: O jogador poderá encerrar o jogo apertando a tecla ‘q’

-Descrição: Após o começo do jogo, o jogador poderá a qualquer momento encerrar a execução do jogo apertando a tecla ‘q’

- **Entradas Necessárias:** Input do jogador na tecla ‘q’ após a impressão do cenário inicial.
- **Saídas Esperadas:** Encerramento da execução do jogo.



```
async def get_key():
    while True:
        event = keyboard.read_event(suppress=True)
        if event.event_type == keyboard.KEY_DOWN:
            return event.name

while True:
    print_level()
    print("Use as setas do teclado para mover o jogador.")
    print("Pressione Q para sair.")

    key = await get_key()

    if key == 'q':
        break
```

- O jogo espera o input do jogador, e se for a tecla ‘q’, o loop acaba e a execução do código é interrompida.

Requisitos Não Funcionais

Os requisitos não funcionais da aplicação estão descritos a seguir:

RNF-01: O Sistema deve oferecer desempenho responsivo

-Descrição: Garantir que o jogo tenha um desempenho responsivo durante a jogabilidade, oferecendo resposta rápida às ações do jogador, sem atrasos perceptíveis.

Implementação:

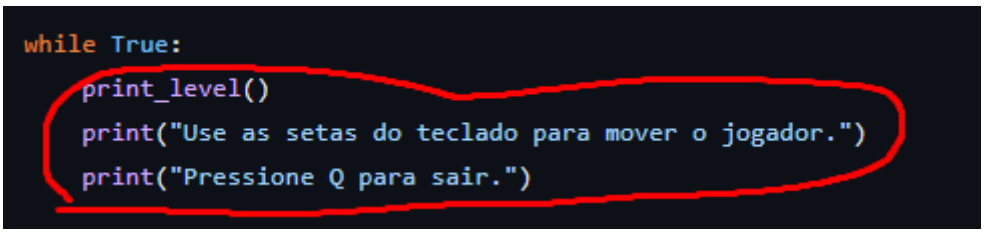
- O jogo deve capturar e processar as entradas do jogador de forma eficiente, sem introduzir atrasos perceptíveis durante a jogabilidade.
- Evitar o uso excessivo de operações de E/S que possam causar bloqueios e prejudicar o desempenho responsivo do jogo

.

RNF-02: O Sistema deve ter usabilidade intuitiva

-Descrição: Tornar a jogabilidade do jogo intuitiva e fácil de entender para os jogadores, oferecendo controles claros e simples e feedback adequado para as ações do jogador.

Implementação:



- Mensagens claras e instrutivas devem ser apresentadas ao jogador para orientá-lo durante a jogabilidade, como instruções de movimentação e feedback sobre o estado do jogo.
- Os controles do jogo devem ser simples e intuitivos, permitindo que os jogadores entendam facilmente como interagir com o jogo sem a necessidade de instruções complexas.

RNF-03: O Sistema deve ter fácil legibilidade e manutenibilidade do código

-Descrição: O código-fonte do jogo deve estar organizado de forma clara e legível, facilitando a compreensão e a manutenção por parte dos desenvolvedores.

Implementação:

- O código deve ser estruturado de maneira modular, com funções separadas para diferentes funcionalidades e uma nomenclatura significativa para variáveis e funções.
- Comentários adequados devem ser incluídos no código para explicar a lógica complexa ou detalhes de implementação, tornando mais fácil para outros desenvolvedores entenderem o funcionamento do jogo.

***Observações sobre a implementação do código:**

O ChatGPT foi utilizado para otimizar o código e sugerir APIs para melhorar inputs e renderização de frames do jogo, juntamente com dar algumas sugestões sobre como implementar List Comprehensions e Closures no código, sem afetar o funcionamento do jogo.

A API sugerida pelo chatbot foi a “asyncio” que possibilitou a leitura e registro assíncrono dos inputs do jogador, IMPEDINDO que:

- 1: as frames do jogo fossem constantemente impressas
- 2: o input do jogador fosse detectado duas vezes

Mapeamento de Programação Funcional

-Função Lambda

A função lambda está presente no código para calcular a nova posição do jogador ou da caixa com base na posição atual e no deslocamento (delta) fornecido:

```
# Função lambda para mover o jogador ou caixa
move = lambda pos, delta: (pos[0] + delta[0], pos[1] + delta[1])
```

-List Comprehension

As funções de List comprehensions são utilizadas para inicializar as posições das caixas e dos objetivos:

```
# Inicializa as posições das caixas e dos objetivos usando List Comprehensions
boxes = {(i, j) for i, row in enumerate(level) for j, char in enumerate(row) if char == '$'}
goals = {(i, j) for i, row in enumerate(level) for j, char in enumerate(row) if char == '.'}
```

-Função de Continuação

A classe Result e seu método bind são utilizados para encadear operações com objetos Result, o que pode ser considerado uma forma de continuação funcional:

```
class Result:
    def __init__(self, success, value=None, error=None):
        self.success = success
        self.value = value
        self.error = error

    def bind(self, func):
        if self.success:
            return func(self.value)
        else:
            return Result(False, error=self.error)
```

-**Closure**

As funções is_box e is_free são closures que recebem o nível do jogo como argumento e retornam funções lambda que verificam se uma determinada posição contém uma caixa ou está livre, respectivamente:

```
# Closure para verificar se uma posição contém uma caixa
def is_box(level):
    return lambda pos: level[pos[0]][pos[1]] == '$'

# Closure para verificar se uma posição está livre
def is_free(level):
    return lambda pos: level[pos[0]][pos[1]] in [' ', '.']
```

-Função de Alta Ordem

A função is_valid é uma função de alta ordem que recebe uma posição e o nível do jogo como entrada e retorna um objeto Result que indica se a posição é válida ou não:

A função is_goal é uma função de alta ordem que recebe um level e verifica se uma posição é um ponto de destino.

```
# Função de continuação para verificar se uma posição é válida
def is_valid(pos, level):
    if pos[0] < 0 or pos[0] >= len(level) or pos[1] < 0 or pos[1] >= len(level[0]):
        return Result(False, error="Out of bounds")
    return Result(level[pos[0]][pos[1]] != '#')

# Função de alta ordem para verificar se uma posição é um ponto de destino
def is_goal(level):
    return lambda pos: level[pos[0]][pos[1]] == '.'
```

CASOS DE TESTE

Caso de Teste 1: Movimento para a direita

-Cenário de Teste:

- Ações: Pressionar a tecla 'right' (seta para a direita).
- Resultado Esperado:

-O jogador '@' deve se mover para a direita em um espaço vazio.

```
Use as setas do teclado para mover o jogador.
Pressione Q para sair.
#######
#@ $ .#
#  $  #
#  .  #
#######
Use as setas do teclado para mover o jogador.
Pressione Q para sair.
```

-O mapa resultante deve ser exibido com o jogador '@' movido para a direita:

```
Use as setas do teclado para mover o jogador.
Pressione Q para sair.
#######
# @$ .#
#  $  #
#  .  #
#######
Use as setas do teclado para mover o jogador.
Pressione Q para sair.
```

-E a caixa simbolizada pelo $ é movida pra direita

```
Use as setas do teclado para mover o jogador.
Pressione Q para sair.
#######
#  @$.#
#  $  #
#  .  #
#######
Use as setas do teclado para mover o jogador.
Pressione Q para sair.
```

Caso de teste 2 : Checagem de validade de posição

-Cenário de Teste:

- Ações: Pressionar a tecla 'left' (seta para a esquerda).
- Resultado Esperado:

-quando pressionamos o botão para ir a esquerda(que na posição inicial não é possível, a sua posição volta à inicial, permitindo apenas pros cantos marcados como ‘ ‘.

```
def test_is_valid():
    level = [
        ['#', '#', '#', '#', '#'],
        ['#', ' ', ' ', ' ', '#'],
        ['#', ' ', '$', ' ', '#'],
        ['#', '#', '#', '#', '#'],
    ]

    assert is_valid((1, 1), level).success == True
    assert is_valid((1, 4), level).success == False
    assert is_valid((2, 2), level).success == True
    assert is_valid((2, -1), level).success == False
    assert is_valid((4, 0), level).success == False
```

-A função test_is_valid() está definida para testar a função is_valid(), que verifica se uma posição dada é válida em um determinado nível do jogo.

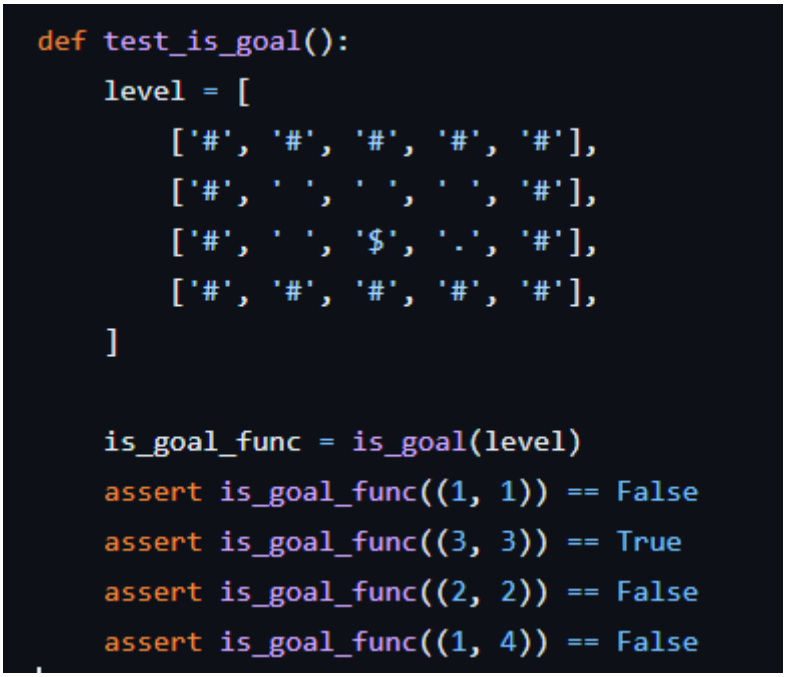
-Portanto, se o jogador tentar mover-se para a esquerda a partir de uma posição inicial onde isso não é possível, sua posição não será alterada e ele permanecerá onde está. Isso é exatamente o que é testado no caso de teste assert is_valid((1, 4), level).success == False, onde a posição (1, 4) está bloqueada por uma parede, então is_valid() retorna False indicando que a posição não é válida para o movimento.

Caso de teste 3 : Define objetivo

-Cenário de Teste:

- Ações: Pressionar uma tecla para chegar ao objetivo do jogo.
- Resultado Esperado:

-No código aparece uma posição verdadeira de onde o objetivo está localizado e se a caixa é entregue na posição, o jogo termina.



```
def test_is_goal():
    level = [
        ['#', '#', '#', '#', '#'],
        ['#', ' ', ' ', ' ', '#'],
        ['#', ' ', '$', '.', '#'],
        ['#', '#', '#', '#', '#'],
    ]

    is_goal_func = is_goal(level)
    assert is_goal_func((1, 1)) == False
    assert is_goal_func((3, 3)) == True
    assert is_goal_func((2, 2)) == False
    assert is_goal_func((1, 4)) == False
```

-O objetivo é determinado baseado na posição configurada no mapa.

-Este teste verifica se todas as posições corretamente identificadas como pontos de destino no mapa são reconhecidas corretamente pela função is_goal().

Caso de teste 4 : Define o que é considerado uma caixa

-Cenário de Teste:

- Ações: Identificar o que é a caixa no jogo.
- '#' representa uma parede, ' ' (espaço) representa um espaço vazio, e '$' representa uma caixa.
- Resultado Esperado:

-O teste verifica se a função `is_box()` identifica corretamente a presença de uma caixa em uma determinada posição no mapa.

```
def test_is_box():
    level = [
        ['#', '#', '#', '#', '#'],
        ['#', ' ', ' ', ' ', '#'],
        ['#', ' ', '$', '.', '#'],
        ['#', '#', '#', '#', '#'],
    ]

    is_box_func = is_box(level)
    assert is_box_func((1, 1)) == False
    assert is_box_func((2, 2)) == True
    assert is_box_func((3, 3)) == False
    assert is_box_func((1, 4)) == False
```

-O teste traz como resultado que a caixa ‘$’ está na posição (2, 2).

Caso de teste 5 : Define se uma parte do mapa está livre para movimentação tanto do player como da caixa.

-Cenário de Teste:

- Ações: Identificar espaços vazios no mapa.
- Resultado Esperado:

-Identificar corretamente se uma posição no mapa está livre ou ocupada, considerando paredes, espaços vazios, caixas e a posição do jogador.

```
def test_is_free():
    level = [
        ['#', '#', '#', '#', '#'],
        ['#', ' ', ' ', ' ', '#'],
        ['#', ' ', '$', '.', '#'],
        ['#', '#', '#', '#', '#'],
    ]

    is_free_func = is_free(level)
    assert is_free_func((1, 1)) == True
    assert is_free_func((2, 2)) == False
    assert is_free_func((3, 3)) == False
    assert is_free_func((1, 4)) == False
```

-Este teste verifica se a função is_free() retorna corretamente se uma posição no mapa está livre, ou seja, se é possível mover-se para essa posição. Ele testa diferentes casos, incluindo posições com parede, espaços vazios, caixas e o jogador.

Caso de teste 6: Define se uma posição está fora dos limites do mapa.

-Cenário de Teste:

- Ações: Testa se a função is_valid() retorna corretamente False e uma mensagem de erro quando uma posição está fora dos limites do mapa.
- Resultado Esperado:

-a função is_valid() retorna corretamente False e a mensagem de erro 'Out of bounds' quando uma posição está fora dos limites do mapa. Ele testa um caso em que posição (-1, 1) está fora dos limites do mapa definido pela matriz level.

```
def test_out_of_bounds(self):
    pos = (-1, 1)
    level = [['#', ' ', '#'],
             ['#', ' ', '#']]
    result = is_valid(pos, level)
    self.assertFalse(result.success)
    self.assertEqual(result.error, 'Out of bounds')
```

Caso de teste 7:

-Cenário de Teste:

- Ações: Testa se a função is_valid() retorna corretamente False quando uma posição colide com uma parede no mapa.
- is_valid(): verifica se uma posição é válida dentro dos limites do mapa.
- Colisão com a parede: Verifica se a função retorna False quando uma posição colide com uma parede no mapa.
- Resultado Esperado:

-Chama a função is_valid() para verificar se a posição colide com uma parede no mapa

- result = is_valid(pos, level)

-Verifica se a função retornou False para a colisão com a parede

- self.assertFalse(result.success)

```
def test_wall_collision(self):
    pos = (0, 0)
    level = [['#', ' ', '#'],
             ['#', ' ', '#']]
    result = is_valid(pos, level)
    self.assertFalse(result.success)
```

Caso de teste 8: Teste de movimento para cima.

-Cenário de Teste:

- Ações: Testa se a função move() move corretamente a posição para cima no mapa.
- Define uma posição inicial (1, 1)
- Define um delta para mover para cima (-1, 0)
- Chama a função move() para mover a posição de acordo com o delta especificado: new_pos = move(pos, delta)
- Resultado Esperado:

self.assertEqual(new_pos, (0, 1))

-Verifica se a nova posição é correta após o movimento para cima.

```
class TestMovement(unittest.TestCase):
    def test_move_up(self):
        pos = (1, 1)
        delta = (-1, 0)
        new_pos = move(pos, delta)
        self.assertEqual(new_pos, second: (0, 1))
```

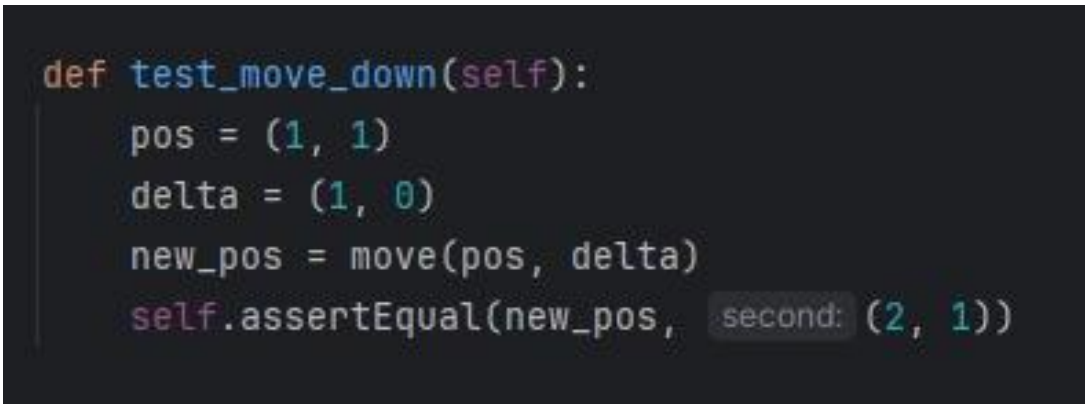
Caso de teste 9: Teste de movimento para baixo.

-Cenário de Teste:

- Ações: Testa se a função move() move corretamente a posição para cima no mapa.
- Define uma posição inicial (1, 1)
- Define um delta para mover para baixo (1, 0)
- Chama a função move() para mover a posição de acordo com o delta especificado: new_pos = move(pos, delta)
- Resultado Esperado:

self.assertEqual(new_pos, (2, 1))

-Verifica se a nova posição está correta após o movimento para baixo.



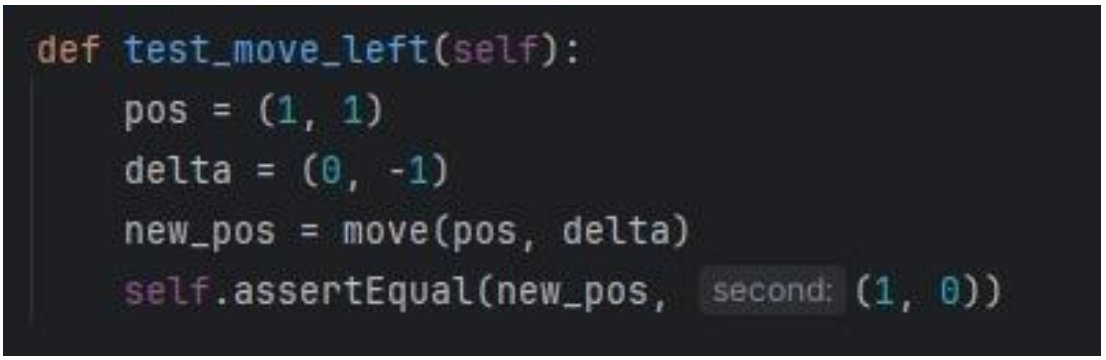
Caso de teste 10: Teste de movimento para a Esquerda.

-Cenário de Teste:

- Ações: Testa se a função move() move corretamente a posição para a esquerda no mapa.
- Define uma posição inicial (1, 1)
- Define um delta para mover para direita (0, 1)
- Chama a função move() para mover a posição de acordo com o delta especificado: new_pos = move(pos, delta)
- Resultado Esperado:

self.assertEqual(new_pos, (1, 0))

-Verifica se a nova posição está correta após o movimento para a esquerda.



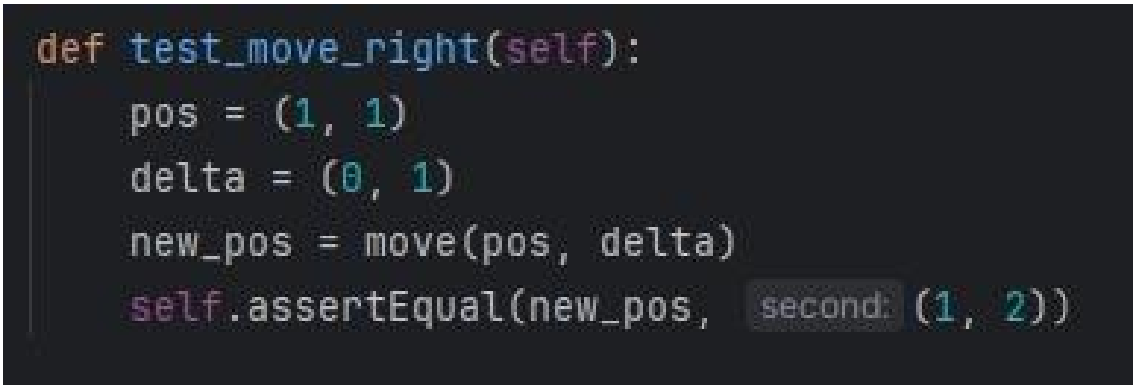
Caso de teste 11: Teste de movimento para a Direita.

-Cenário de Teste:

- Ações: Testa se a função move() move corretamente a posição para a direita no mapa.
- Define uma posição inicial (1, 1)
- Define um delta para mover para esquerda (0, -1)
- Chama a função move() para mover a posição de acordo com o delta especificado: new_pos = move(pos, delta)
- Resultado Esperado:

self.assertEqual(new_pos, (1, 2))

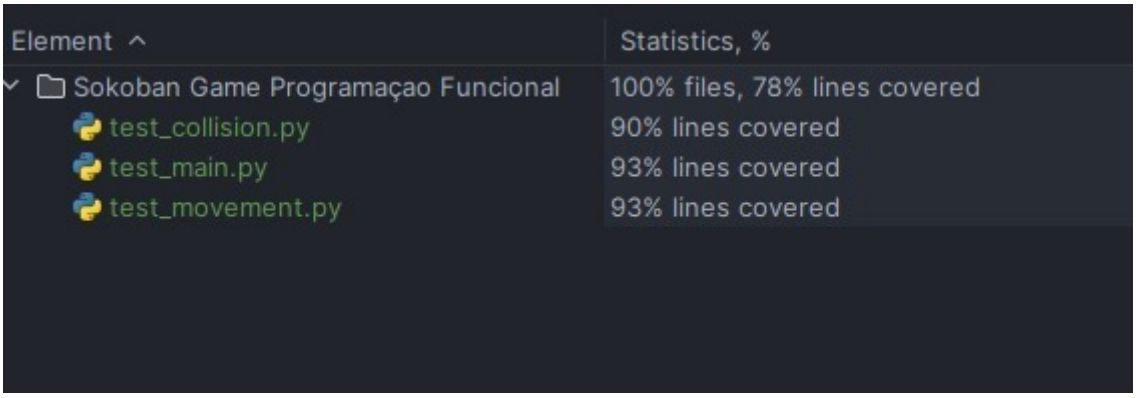
-Verifica se a nova posição é correta após o movimento para a direita



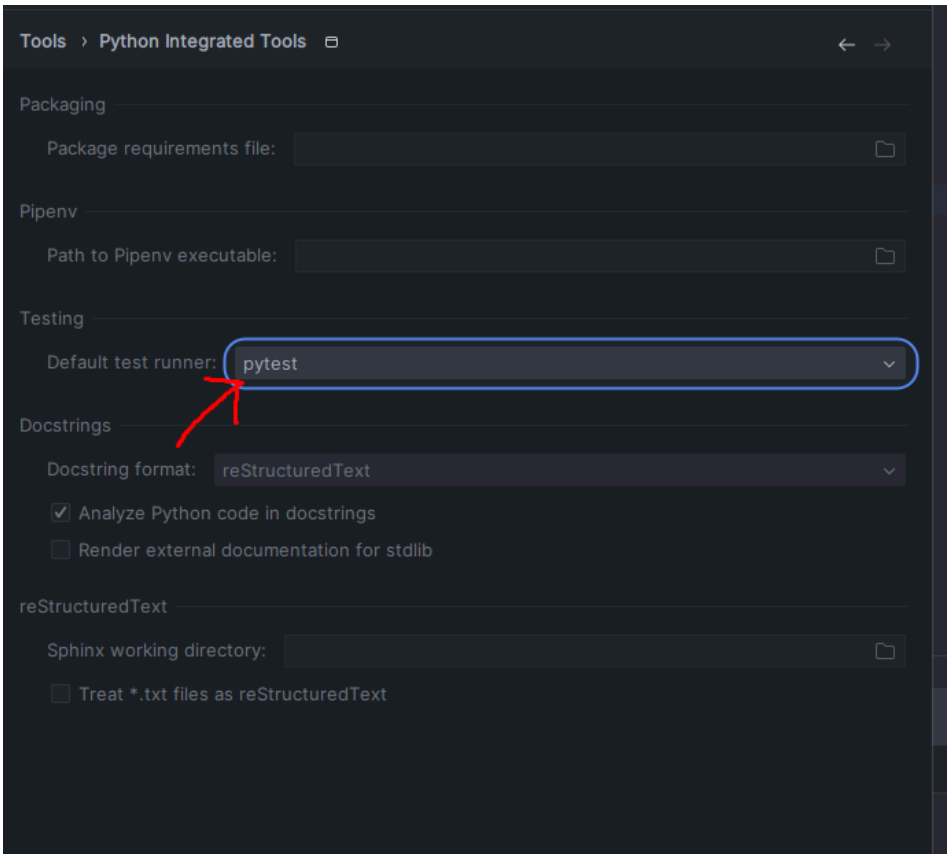
Comprovação de Cobertura mínima:

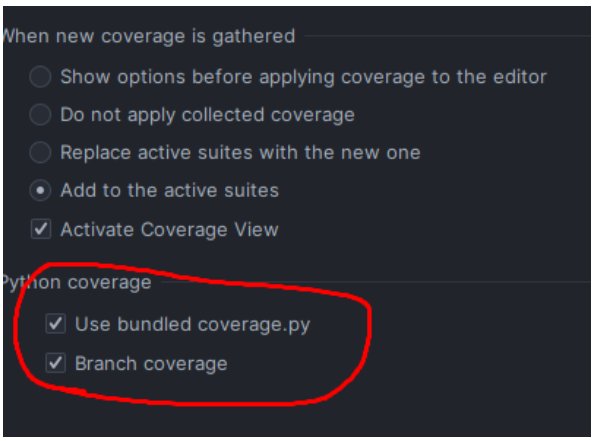
O processo de execução dos testes foi feito usando o **code coverage** da IDE PyCharm Profissional

É uma ferramenta de análise de testes integrada, revelando esses resultados para cada um dos arquivos de teste.



Todos os casos de teste foram incluídos nesses arquivos, e foram rodados utilizando o pacote pytest como test runner.






```
============================= test session starts ==============================
collecting ... collected 4 items

test_movement.py::TestMovement::test_move_down PASSED                    [ 25%]
test_movement.py::TestMovement::test_move_left PASSED                    [ 50%]
test_movement.py::TestMovement::test_move_right PASSED                   [ 75%]
test_movement.py::TestMovement::test_move_up PASSED                      [100%]

============================== 4 passed in 0.10s ===============================
```

```
============================= test session starts ==============================
collecting ... collected 5 items

test_main.py::test_is_valid PASSED                                       [ 20%]
test_main.py::test_is_goal FAILED                                        [ 40%]
test_main.py:16 (test_is_goal)
False != True

Expected :True
Actual   :False
<Click to see difference>

def test_is_goal():
        level = [
            ['#', '#', '#', '#', '#'],
            ['#', ' ', ' ', ' ', '#'],
            ['#', ' ', '$', '.', '#'],
            ['#', '#', '#', '#', '#'],
        ]

        is_goal_func = is_goal(level)
        assert is_goal_func((1, 1)) == False
>       assert is_goal_func((3, 3)) == True
E       assert False == True
E        +  where False = <function is_goal.<locals>.<lambda> at 0x00000209C32B1B80>((3, 3))

test_main.py:27: AssertionError

test_main.py::test_is_box PASSED                                         [ 60%]
test_main.py::test_is_free PASSED                                        [ 80%]
test_main.py::test_move PASSED                                           [100%]
```

```
============================= test session starts ==============================
collecting ... collected 2 items

test_collision.py::TestCollision::test_out_of_bounds PASSED              [ 50%]
test_collision.py::TestCollision::test_wall_collision PASSED             [100%]

============================== 2 passed in 0.08s ===============================
```

```
test_collision.py 90% lines covered
test_main.py 93% lines covered
test_movement.py 93% lines covered
```