Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет прикладной информатики

Отчет
Тестирование ПО
Лабораторная работа 1

Выполнил:
Савальский М.И.
Проверил:
Кочубеев Н.С.

Санкт-Петербург,
2025

1 Ссылка на репозиторий гитхаб

Ссылка

2 Введение

Для выполнения лабораторной работы 1 по дисциплине Тестирование ПО, проектом для которого разрабатывались юнит тесты использовалась моя практическая работа "Шахматы"по дисциплине Программирование на 2 семестре обучения (подробный отчет прикреплен в репозитории).

Программа представляет из себя файл с расширением .py и позволяет реализовать процесс игры в шахматы (на одном ПК) с графическим интерфейсом(реализованным с помощью библиотеки tkinter), а также возможностью запускаться в терминале (шрифт DejaVu Sans Mono). Кроме обычного игрового процесса шахмат программа ведет учет истории побед и сохраняет их в базу данных в той же директории, что и Game.py .

3 Анализ функциональности

Основные функции и методы классов:

- Основные функции и методы:
- Game.rules() - проверка правил ходов
- Game.possible_moves() - расчет возможных ходов фигур
- Game.is_check() - проверка шаха
- Game.checkmate() - проверка мата
- GameMixin.translate_to_pos() и translate_to_str() - преобразование координат

Критическими участками программы нуждающимеся в тестировании были выделены:

- Валидация ходов (правила шахмат)
- Расчет возможных ходов для разных фигур
- Проверка шаха и мата
- Преобразование координат (алгебраическая нотация <-> матричные индексы)

Ключевые сценарии использования:

- Корректные ходы фигур / Некорректные ходы
- Шах и мат ситуации
- Преобразование координат
- Граничные случаи доски (координат)
- Вызов необходимых методов проверки правил

4 Примеры тестов

В качестве фреймворка для написания юнит тестов использовался pytest с плагином pytest-coverage для сбора метрики покрытия кода. Всего получилось 15 видов юнит тестов (всего тестов 26 из-за data-driven тестов).

Приведу несколько примеров тестов с кодом, например тесты на преобразование координат(Рисунок 1):

```python
class TestChessGame:
    @pytest.fixture
    def game(self):
        game = Game()
        game.board = [['_' for _ in range(8)] for _ in range(8)]
        return game

    @pytest.mark.parametrize("algebraic,expected_matrix", [
        ('a1', [7, 0]),
        ('h1', [7, 7]),
        ('a8', [0, 0]),
        ('h8', [0, 7])
    ])
    def test_translate_coordinates_correct(self, algebraic, expected_matrix):
        # Act
        result = GameMixin.translate_to_pos(algebraic)
        back_result = GameMixin.translate_to_str(expected_matrix)
        # Assert
        assert result == expected_matrix
        assert back_result == algebraic

    @pytest.mark.parametrize("invalid_coord", ['', 'a', 'a9', 'i1', 'a0', 'a10', 'z5'])
    def test_translate_coordinates_invalid(self, invalid_coord):
        # Act and Assert
        with pytest.raises(WrongMove):
            GameMixin.translate_to_pos(invalid_coord)
```

Рис. 1: Тесты на преобразования координат

Анализ возможных ходов различных фигур в разных ситуациях (Рисунок 2)

```python
    def test_pawn_possible_moves_initial_white(self, game):
        # Arrange
        game.board[6][4] = Figure('pawn', 'white')  # e2
        expected_moves = [[5, 4], [4, 4]]  # e3, e4

        # Act
        possible_moves = game.possible_moves([6, 4])

        # Assert
        assert len(possible_moves) == len(expected_moves)
        for move in expected_moves:
            assert move in possible_moves

    def test_rook_edge_of_board_moves(self, game):
        # Arrange
        game.board[0][0] = Figure('rook', 'white')  # a8

        # Act
        possible_moves = game.possible_moves([0, 0])

        # Assert
        for move in possible_moves:
            assert 0 <= move[0] <= 7
            assert 0 <= move[1] <= 7
```

Рис. 2: Тесты на возможные ходы фигур (1)

Тесты на правильную очередь хода и мат (Рисунок 3)

```
102
103     def test_rules_invalid_move_wrong_turn(self, game):
104         """Тест правил для хода не в свою очередь"""
105         # Arrange
106         game.board[6][4] = Figure('pawn', 'white')  # e2
107         game.board[1][4] = Figure('pawn', 'black')  # e7
108         game.move_num = 1  # Ход черных
109 
110         # Act
111         result = game.rules([6, 4], [5, 4])  # Белые пытаются ходить в очередь черных
112 
113         # Assert
114         assert result == 0, "Wrong turn move should return 0"
115 
116     def test_checkmate_detection_simplified(self, game):
117         # Arrange
118         game.board[7][4] = Figure('king', 'white')  # e1
119         game.board[6][0] = Figure('queen', 'black')  # a2
120         game.board[7][1] = Figure('queen', 'black')  # b1
121 
122         # Act
123         is_checkmate = game.checkmate([7, 4])  # Позиция белого короля
124 
125         # Assert
126         assert is_checkmate, "Should detect checkmate"
```

Рис. 3: Тесты на очередь хода и мат

Тесты на вызов функции проверки правил, а также проверки шаха и мата (Рисунок 4).

```
157 
158 class TestChessFunctions:
159     @pytest.fixture
160     def game(self):
161         game = MockGame()
162         game.board = [['_' for _ in range(8)] for _ in range(8)]
163         game.board[7][4] = Figure('king', 'white')  # e1
164         game.board[0][4] = Figure('king', 'black')  # e1
165         game.board[6][0] = Figure('queen', 'black')  # a2
166         game.board[0][1] = Figure('queen', 'black')  # b8
167         game.move_num = 1 # ход черных
168         return game
169 
170     def test_rules_call_when_move(self,game):
171         # Arrange
172         def rules(self,*args):
173             self.rules_called = True
174         game.rules = rules.__get__(game)
175         # Act
176         game.move([0,1],[7,1])
177         game.move_num += 1
178         # Assert
179         assert game.rules_called == True
180 
181     def test_is_check_call_when_move(self,game):
182         # Arrange
183         def is_check(self,king_pos):
184             self.is_check_called = True
185         game.is_check = is_check.__get__(game)
186         # Act
187         game.move([0,1],[7,1])
188         # Assert
189         assert game.is_check_called == True
190 
191     def test_is_checkmate_call_when_move(self,game):
192         # Arrange
193         def is_checkmate(self,pos):
194             self.is_checkmate_called = True
195         game.checkmate = is_checkmate.__get__(game)
196         # Act
197         game.move([0,1],[7,1])
198         # Assert
199         assert game.is_checkmate_called == True
```

Рис. 4: Тесты на проверку правил, шах, мат

5 Результаты запуска тестов

Конечные результаты запуска тестов представлены на рисунке 5, а также метрика code coverage 6

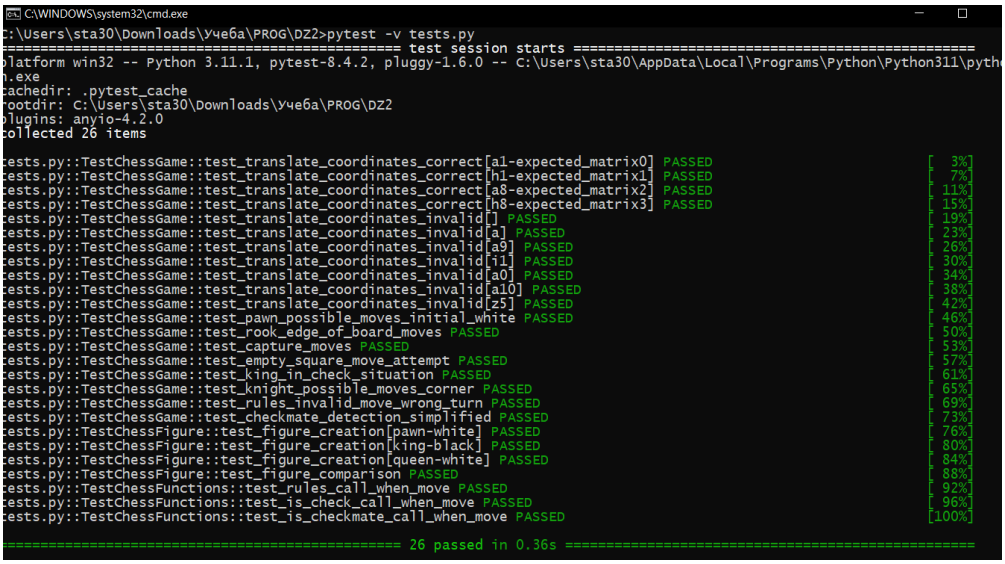

Рис. 5: Запуск тестов

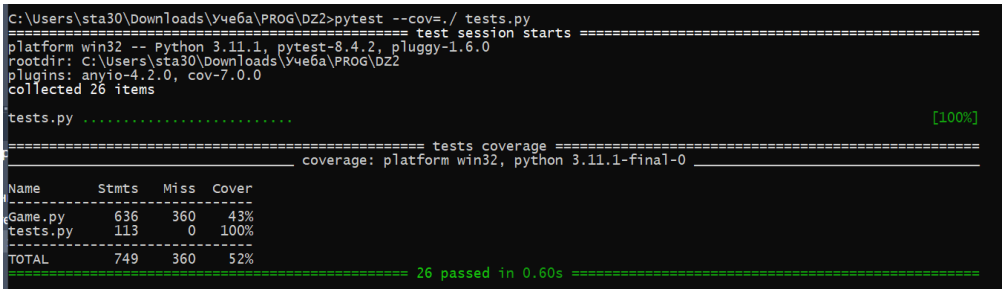

Рис. 6: Code coverage

6 Вывод

В процессе написания юнит тестов и их последовательного запуска были выявлены баги в методах связанных с расчетом возможных ходов, перевод координат из алгебраического вида ('a1','g7') в матричный и незаконными ходами по правилам игры. Поэтому можно сказать, что тестирование было довольно эффективным. Метрика Code coverage составила 43% для основного файла Game.py, который содержит все классы и методы необходимые для работы программы. Полный код доступен по ссылке