

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 нуждающимися 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-covrage 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):

```
5 class TestChessGame:
6     @pytest.fixture
7     def game(self):
8         game = Game()
9         game.board = [['_' for _ in range(8)] for _ in range(8)]
10        return game
11
12    @pytest.mark.parametrize("algebraic,expected_matrix", [
13        ('a1', [7, 0]),
14        ('h1', [7, 7]),
15        ('a8', [0, 0]),
16        ('h8', [0, 7])
17    ])
18    def test_translate_coordinates_correct(self, algebraic, expected_matrix):
19        # Act
20        result = GameMixin.translate_to_pos(algebraic)
21        back_result = GameMixin.translate_to_str(expected_matrix)
22        # Assert
23        assert result == expected_matrix
24        assert back_result == algebraic
25
26    @pytest.mark.parametrize("invalid_coord", ['', 'a', 'a9', 'i1', 'a0', 'a10', 'z5'])
27    def test_translate_coordinates_invalid(self, invalid_coord):
28        # Act and Assert
29        with pytest.raises(WrongMove):
30            GameMixin.translate_to_pos(invalid_coord)
```

Рис. 1: Тесты на преобразования координат

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

```
31
32    def test_pawn_possible_moves_initial_white(self, game):
33        # Arrange
34        game.board[6][4] = Figure('pawn', 'white') # e2
35        expected_moves = [[5, 4], [4, 4]] # e3, e4
36
37        # Act
38        possible_moves = game.possible_moves([6, 4])
39
40        # Assert
41        assert len(possible_moves) == len(expected_moves)
42        for move in expected_moves:
43            assert move in possible_moves
44
45    def test_rook_edge_of_board_moves(self, game):
46        # Arrange
47        game.board[0][0] = Figure('rook', 'white') # a8
48
49        # Act
50        possible_moves = game.possible_moves([0, 0])
51
52        # Assert
53        for move in possible_moves:
54            assert 0 <= move[0] <= 7
55            assert 0 <= move[1] <= 7
56
```

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

Тесты на правильную очередь хода и мат (Рисунок 3)

```
102
103
104 def test_rules_invalid_move_wrong_turn(self, game):
105     """Тест правил для хода не в свою очередь"""
106     # Arrange
107     game.board[6][4] = Figure('pawn', 'white') # e2
108     game.board[1][4] = Figure('pawn', 'black') # e7
109     game.move_num = 1 # Ход черных
110
111     # Act
112     result = game.rules([6, 4], [5, 4]) # Белые пытаются ходить в очередь черных
113
114     # Assert
115     assert result == 0, "Wrong turn move should return 0"
116
117 def test_checkmate_detection_simplified(self, game):
118     # Arrange
119     game.board[7][4] = Figure('king', 'white') # e1
120     game.board[6][0] = Figure('queen', 'black') # a2
121     game.board[7][1] = Figure('queen', 'black') # b1
122
123     # Act
124     is_checkmate = game.checkmate([7, 4]) # Позиция белого короля
125
126     # Assert
127     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

```
C:\WINDOWS\system32\cmd.exe
C:\Users\sta30\Downloads\Учеба\PROG\DZ2>pytest -v tests.py
===== test session starts =====
platform win32 -- Python 3.11.1, pytest-8.4.2, pluggy-1.6.0 -- C:\Users\sta30\AppData\Local\Programs\Python\Python311\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\sta30\Downloads\Учеба\PROG\DZ2
plugins: anyio-4.2.0
collected 26 items

tests.py::TestChessGame::test_translate_coordinates_correct[a1-expected_matrix0] PASSED [ 3%]
tests.py::TestChessGame::test_translate_coordinates_correct[h1-expected_matrix1] PASSED [ 7%]
tests.py::TestChessGame::test_translate_coordinates_correct[a8-expected_matrix2] PASSED [ 11%]
tests.py::TestChessGame::test_translate_coordinates_correct[h8-expected_matrix3] PASSED [ 15%]
tests.py::TestChessGame::test_translate_coordinates_invalid[] PASSED [ 19%]
tests.py::TestChessGame::test_translate_coordinates_invalid[a] PASSED [ 23%]
tests.py::TestChessGame::test_translate_coordinates_invalid[a9] PASSED [ 26%]
tests.py::TestChessGame::test_translate_coordinates_invalid[h1] PASSED [ 30%]
tests.py::TestChessGame::test_translate_coordinates_invalid[a0] PASSED [ 34%]
tests.py::TestChessGame::test_translate_coordinates_invalid[a10] PASSED [ 38%]
tests.py::TestChessGame::test_translate_coordinates_invalid[z5] PASSED [ 42%]
tests.py::TestChessGame::test_pawn_possible_moves_initial_white PASSED [ 46%]
tests.py::TestChessGame::test_rook_edge_of_board_moves PASSED [ 50%]
tests.py::TestChessGame::test_capture_moves PASSED [ 53%]
tests.py::TestChessGame::test_empty_square_move_attempt PASSED [ 57%]
tests.py::TestChessGame::test_king_in_check_situation PASSED [ 61%]
tests.py::TestChessGame::test_knight_possible_moves_corner PASSED [ 65%]
tests.py::TestChessGame::test_rules_invalid_move_wrong_turn PASSED [ 69%]
tests.py::TestChessGame::test_checkmate_detection_simplified PASSED [ 73%]
tests.py::TestChessFigure::test_figure_creation[pawn-white] PASSED [ 76%]
tests.py::TestChessFigure::test_figure_creation[king-black] PASSED [ 80%]
tests.py::TestChessFigure::test_figure_creation[queen-white] PASSED [ 84%]
tests.py::TestChessFigure::test_figure_comparison PASSED [ 88%]
tests.py::TestChessFunctions::test_rules_call_when_move PASSED [ 92%]
tests.py::TestChessFunctions::test_is_check_call_when_move PASSED [ 96%]
tests.py::TestChessFunctions::test_is_checkmate_call_when_move PASSED [100%]

===== 26 passed in 0.36s =====
```

Рис. 5: Запуск тестов

```
C:\Users\sta30\Downloads\Учеба\PROG\DZ2>pytest --cov=./ tests.py
===== test session starts =====
platform win32 -- Python 3.11.1, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\sta30\Downloads\Учеба\PROG\DZ2
plugins: anyio-4.2.0, cov-7.0.0
collected 26 items

tests.py ..... [100%]

===== tests coverage =====
coverage: platform win32, python 3.11.1-final-0
Name      Stmts  Miss  Cover
-----
Game.py    636    360    43%
tests.py    113      0   100%
TOTAL      749    360    52%

===== 26 passed in 0.60s =====
```

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