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 2
Интеграционное тестирование

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

Санкт-Петербург,
2025

1 Ссылка на репозиторий гитхаб

Ссылка

2 Введение

Для выполнения лабораторной работы 2 по дисциплине Тестирование ПО, проектом для которого разрабатывались интеграционные 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 Chess.py .

3 Анализ взаимодействий

В качестве ключевых точек интеграции были выделены взаимодействия следующих модулей:

1. Модуль выполнение хода в ситуации с последующим матом и модуль завершения игры;
2. Модули проверки правил и выполнение хода (успешные сценарии);
3. Модули проверки правил и выполнение хода (ошибочные сценарии);
4. Модули завершения игры и записи результата в БД;
5. Модуль завершения игры и модуль записи результата в историю

Обоснования выбора соответствующих интеграций:

1. Проверяет, что ход, приводящий к мату, заканчивают игру с правильным результатом (кто победил);
2. Проверяет основную игровую механику - корректность валидации и выполнения ходов (ход отражается на доске) для валидного хода игрока;
3. Проверяет основную игровую механику - корректность валидации и выполнения ходов (ход отражается на доске) для невалидного хода игрока;
4. Проверяет сохранение игровых данных в БД по завершению игры;
5. Проверяет корректный учет данных игры и их запись в историю по завершению;

4 Примеры тестов

В качестве фреймворка для написания интеграционных тестов использовался pytes. Всего получилось 5 видов интеграционных тестов (всего тестов 14 из-за data-driven тестов).

Чтобы обеспечить изоляцию и независимость тестов от внешней среды, были созданы две pytest.fixture, чтобы подготовить game instance и в временную БД для тестов (Рисунок 1).

```
6
7   class TestChessIntegration:
8       @pytest.fixture
9       def temp_db(self):
10          name_db='chess.db'
11          if os.path.exists(name_db): os.remove(name_db)
12          temp_conn=sqlite3.connect(name_db)
13          yield temp_conn
14          temp_conn.close()
15          os.remove(name_db)
16
17      @pytest.fixture
18      def game_setup(self):
19          from Chess import Game, Player, Figure
20          game = Game(side='w', player=Player('TestPlayer'))
21          game.set_all_figures(reverse=True)
22          return game
23
```

Рис. 1: pytest.fixtures

Код интеграционного теста выполнения хода и модуля выявления мата (Рисунок 2). Выстраиваем ситуации с матом в следующем ходу и выполняя этот ход, проверяем, что мат был детектирован.

```python
    @pytest.mark.parametrize("case",[1,2])
    def test_integration_move_and_checkmate(self,game_setup,temp_db,case):
        """
        Тест 1: Интеграция выполнения хода с последующим матом и окончанием игры
        Проверяет, что ход, приводящий к мату, заканчивают игру с правильным результатом (кто победил)
        """
        game = game_setup
        game.board = [['_'] * 8 for _ in range(8)] # пустая доска
        if case == 1: # Мат белому королю двумя ладьями:
            game.move_num=1 # ход черных (нечет)
            game.set_figure(type='rook',color='black',pos=[1,0]) # черная ладья на a7
            game.set_figure(type='king',color='white',pos=[0,4]) # белый король на e8
            game.set_figure(type='rook',color='black',pos=[7,1]) # черная ладья на b1
            game.set_figure(type='king',color='black',pos=[7,7]) # черный король на h1

            game.move([7,1],[0,1]) # b1 -> b8 черной ладьей

            player,result,_,_ = game.g_history
            assert player == game.player.get_name()
            assert result == 'black'

        else: # Мат черному королю двумя конями:
            game.move_num=0 # ход белых (чет)
            game.set_figure(type='knight',color='white',pos=[4,0]) # белый конь на a4
            game.set_figure(type='king',color='black',pos=[0,0]) # черный король на a8
            game.set_figure(type='knight',color='white',pos=[2,2]) # белый конь на c6
            game.set_figure(type='king',color='white',pos=[1,2]) # белый король на c7

            game.move([4,0],[2,1]) # a4 -> b6 белым конем

            player,result,_,_ = game.g_history
            assert player == game.player.get_name()
            assert result == 'white'
```

Рис. 2: Тест 1: Интеграция выполнения хода с последующим матом и окончанием игры

Код интеграционного теста модуля проверки валидности хода и модуля выполнения этого хода для успешных сценариев (Рисунок 3). Пытаемся сделать валидные ходы и проверяем изменение состояние игровой доски (ход должен быть выполнен).

```python

    @pytest.mark.parametrize("initial_pos,target_pos,_type,color",[
        ([6,0],[5,0],'pawn','white'), # a2 -> a3 белой пешкой
        ([7,1],[5,0],'knight','white'), # b1 -> a3 белым конем
        ([6,3],[4,3],'pawn','white')  # d2 -> d4 белой пешкой
    ])
    def test_integration_move_validation_and_execution_valid(self, game_setup,temp_db, initial_pos, target_pos,_type,color):
        """
        Тест 2: Связь правил игры с выполнением хода (УСПЕШНЫЕ СЦЕНАРИИ);
        Проверяет основную игровую механику - корректность валидации и выполнения ходов (ход отражается на доске)
        + функция get_figure(position)
        """
        game = game_setup
        # Выполняем ход и проверяем его валидность (Функция проверки валидности хода, это декоратор над move)
        game.move(initial_pos, target_pos)
        # Проверяем, что фигура переместилась и значит ход был валиден
        assert game.get_figure(target_pos) != '_'
        assert game.get_figure(target_pos).type == _type
        assert game.get_figure(target_pos).color == color
        assert game.get_figure(initial_pos) == '_'

```

Рис. 3: Тест 2: Связь правил игры с выполнением хода (успешные сценарии)

Код интеграционного теста модуля проверки валидности хода и модуля выполнения этого хода для неуспешных сценариев (Рисунок 4). Пытаемся сделать невалидные ходы

и проверяем изменение состояние игровой доски (ход не должен быть сделан).

```python

@pytest.mark.parametrize("initial_pos,target_pos,_type,color,movenum",[
    ([1,0],[2,0],'pawn','black',0), # a7 -> a6 валидный ход черной пешкой, но ход белых
    ([1,0],[6,0],'pawn','black',1), # a7 -> a2 нелегальный ход черной пешкой, ход черных
    ([6,3],[2,3],'pawn','white',1)  # d2 -> d6 нелегальных ход белой пешкой и ход черных
])
def test_integration_move_validation_and_execution_invalid(self, game_setup,temp_db, initial_pos, target_pos,_type,color,movenum):
    """
    Тест 3: Связь правил игры с выполнением хода (ОШИБОЧНЫЕ СЦЕНАРИИ);
    Проверяет основную игровую механику - корректность валидации и выполнения ходов (ход отражается на доске)
    + функция get_figure(position)
    """
    game = game_setup
    game.move_num=movenum
    # Выполняем ход и проверяем его валидность (Функция проверки валидности хода, это декоратор над move)
    game.move(initial_pos, target_pos)
    # Проверяем, что фигура НЕ переместилась и значит ход был НЕ валиден
    assert game.get_figure(initial_pos).type == _type
    assert game.get_figure(initial_pos).color == color

```

Рис. 4: Тест 3: Связь правил игры с выполнением хода (ошибочные сценарии)

Код теста для модуля завершения игры и модуля записи результата в БД (Рисунок 5). Чтобы проверить несколько вариантов истории, меняем начальное время игры, тем самым симулируем разную продолжительность игр.

```python

@pytest.mark.parametrize("time_passed",[30,60,90])
def test_integration_game_result_in_database(self,game_setup,temp_db,time_passed):
    """
    Тест 4: Взаимодействие с базой данных;
    Проверяет сохранение игровых данных в БД по завершению игры
    """
    game = game_setup
    game.time_start = time.time() - time_passed
    game.move_num=1 # сейчас ход черных (нечет номер), но игра кончается значит победили белые(в моей имплементации)
    game.finish_game() # Завершаем игру (функция записи в историю и БД это декоратор над finish_game())

    conn = temp_db
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM Games")
    results = cursor.fetchall()

    assert len(results) == 1
    game_record = results[0]
    assert game_record[1] == 'TestPlayer'
    assert game_record[2] == 'white'
    assert game_record[3] == str(time_passed)
```

Рис. 5: Тест 4: Взаимодействие с базой данных по завершению игры

Код теста для модуля завершения игры и модуля записи результата в историю (Рисунок 6). В данном случае мокаем функцию time.time, чтобы симулировать разное время игры и проверяем их корректную запись в историю.

```python
@pytest.mark.parametrize("time_passed",[10,100,1000])
def test_integration_game_result_in_history(self, game_setup,temp_db,time_passed):
    """
    Тест 5: Интеграция временной системы с завершением игры и записью результата в историю;
    Проверяет корректный учет времени игры и его запись в историю по завершению
    """
    game = game_setup
    initial_time = game.time_start
    # Симулируем небольшое прошедшее время
    with patch('time.time') as mock_time:
        mock_time.return_value = initial_time + time_passed

        game.finish_game()
        _, _, duration, _ = game.g_history
        assert len(game.g_history) == 4
        assert duration == time_passed
```

Рис. 6: Тест 5: Интеграция модуля завершения игры и модуля записью результата в историю

5 Результаты запуска тестов

Конечные результаты запуска тестов представлены на рисунке 7

```
C:\Users\sta30\Downloads\Учеба\Тестирование\Лаба 2>pytest tests.py -v
============================= 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\Тестирование\Лаба 2
plugins: anyio-4.2.0, cov-7.0.0
collected 14 items

tests.py::TestChessIntegration::test_integration_move_and_checkmate[1] PASSED                                        [  7%]
tests.py::TestChessIntegration::test_integration_move_and_checkmate[2] PASSED                                        [ 14%]
tests.py::TestChessIntegration::test_integration_move_validation_and_execution_valid[initial_pos0-target_pos0-pawn-white] PASSED [ 21%]
tests.py::TestChessIntegration::test_integration_move_validation_and_execution_valid[initial_pos1-target_pos1-knight-white] PASSED [ 28%]
tests.py::TestChessIntegration::test_integration_move_validation_and_execution_valid[initial_pos2-target_pos2-pawn-white] PASSED [ 35%]
tests.py::TestChessIntegration::test_integration_move_validation_and_execution_invalid[initial_pos0-target_pos0-pawn-black-0] PASSED [ 42%]
tests.py::TestChessIntegration::test_integration_move_validation_and_execution_invalid[initial_pos1-target_pos1-pawn-black-1] PASSED [ 50%]
tests.py::TestChessIntegration::test_integration_move_validation_and_execution_invalid[initial_pos2-target_pos2-pawn-white-1] PASSED [ 57%]
tests.py::TestChessIntegration::test_integration_game_result_in_database[30] PASSED                                  [ 64%]
tests.py::TestChessIntegration::test_integration_game_result_in_database[60] PASSED                                  [ 71%]
tests.py::TestChessIntegration::test_integration_game_result_in_database[90] PASSED                                  [ 78%]
tests.py::TestChessIntegration::test_integration_game_result_in_history[10] PASSED                                   [ 85%]
tests.py::TestChessIntegration::test_integration_game_result_in_history[100] PASSED                                  [ 92%]
tests.py::TestChessIntegration::test_integration_game_result_in_history[1000] PASSED                                 [100%]

============================== 14 passed in 0.46s ==============================
```

Рис. 7: Запуск тестов

6 Вывод

В процессе написания интеграционных тестов и их запуска была выявлена проблема оставления подключений к БД открытыми, что выдавало ошибку по завершению теста (в pytest.fixture), а также некоторые баги в системе выявления мата; баги были исправлены, можно сказать, что написание интеграционных тестов было продуктивным. Полный код доступен по ссылке.