Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Петрозаводский государственный университет»
Институт математики и информационных технологий
Кафедра прикладной математики и кибернетики

Направление подготовки магистратуры
09.04.02 - Информационные системы и технологии

Магистерская программа
“Управление данными”

Отчет по учебному курсу «Верификация программного обеспечения»

Выполнил
студент группы 22605
Д.Д. Ларионов

Преподаватель:
к.ф.–м.н., доцент
К. А. Кулаков

Петрозаводск — 2020

Contents

1 Объект тестирования

1.1 Описание приложения

Приложение для управления различными портами видео-игр на игровом движке Doom engine, таких как Doom, Doom 2, Heretic, Hexen и других. Функционал программы следующий:

- Ведение пользователем списка исполняемых файлов с возможностью добавления и удаления файлов из него.
- Фильтрации файлов в формате IWAD, PWAD и PK3 (раздельно) в указанных пользователем директориях.
- Предоставление пользователю возможности выбрать исполняемый файл, IWAD, PWAD и PK3 из списка для запуска.
- Раздельное автоматическое хранение файлов сохранение для каждой уникальной комбинации запуска исполняемого файла, IWAD, PWAD и PK3.
- Автоматизированная загрузка файла конфигурации для каждой уникальной комбинации запуска исполняемого файла, IWAD, PWAD и PK3. Пользователь сам должен создать этот файл заранее и поместить его в нужную директорию. В противном случае исполняемый файл должен запуститься со своими настройками.

2 Стратегия тестирования

2.1 Описание модулей

Программа состоит из ряда модулей:

- config.py – обрабатывает конфигурацию самого приложения.
- dooms_handler.py – обрабатывает список исполняемых файлов.
- files_handler.py – обрабатывает PWAD и IWAD файлы, возможна обработка и других файлов, например PK3.

- saves_handler.py – обрабатывает файлы сохранения.
- configs_handler.py – обрабатывает конфигурационные файлы для игр.
- pathlib_json.py – содержит в себе вспомогательные функции для кодирования путей в файловой системе, обрабатывающихся в библиотеке pathlib, в JSON формат.
- launcher.py – выполняет запуск выбранной игры с заданными параметрами.
- ui_qt – пользовательский интерфейс.

Основным модулем в программе является launcher.py, который получает информацию из других модулей. На основе этой информации он должен создать текстовую строку, содержащую команду, запускающую выбранную пользователем игру с заданными параметрами.

Рисунок 1 содержит диаграмму, в которой показаны все связи между модулями программы.

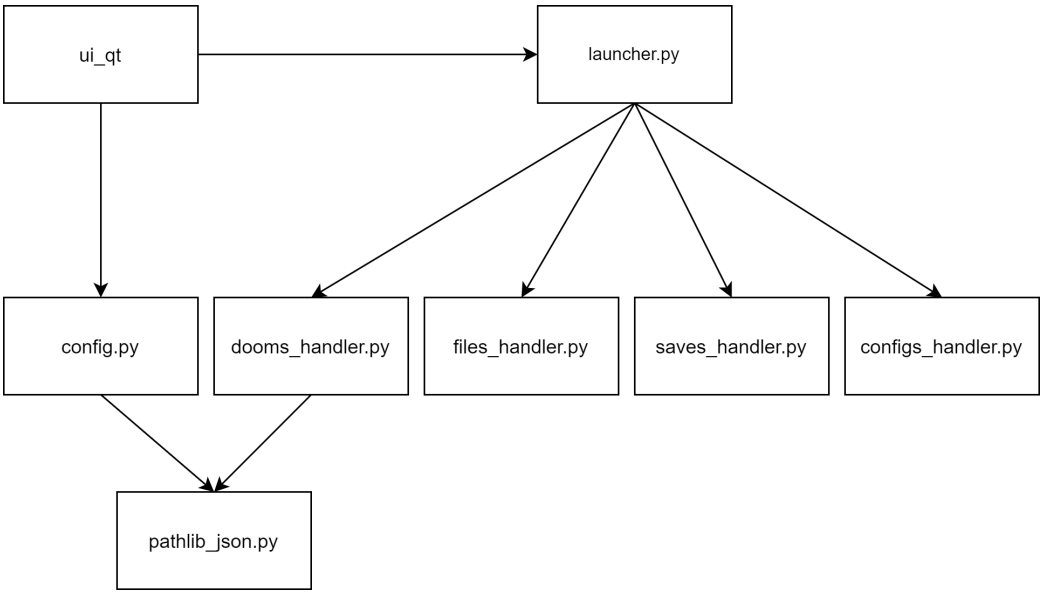

Рисунок 1: Архитектура приложения

2.1.1 pathlib_json.py

Модуль содержит 2 класса PathJSONEncoder и PathJSONDecoder, которые наследуют классы JSONEncoder и JSONDecoder из библиотеки json и переопределяет их методы для кодирования и декодирования JSON. Первый класс отвечает за кодировку объектов библиотеки pathlib в JSON, а второй аз их декодировку из JSON в объекты pathlib.

В классе PathJSONEncoder переопределены следующие методы:

- default(self, o).
  Аргументы:

  1. o – кодируемый объект.

  Возвращаемое значение: строкове значение, полученное из объекта o.
  Описание: если o – экземпляр класса pathlib.WindowsPath или pathlib.PosixPath, то он конвертируется в строку и возвращается эта строка, иначе – вызывается стандартный метод для кодировки json.JSONEncoder.default(self, o) и возвращается его возвращаемое значение.

В классе PathJSONDecoder переопределены следующие методы:

- __object_hook(self, o).
  Аргументы:

  1. o – декодируемый объект (словарь).

  Возвращаемое значение: декодированный словарь.
  Описание: заменяет каждое значение (строку) в словаре на объект pathlib.Path из этой строки и возвращает полученный словарь.

- __init__(self, *args, **kwargs).
  Аргументы:

  1. *args и **kwargs – прочие стандартный аргументы конструктора JSONDecoder.

Возвращаемое значение: None.
Описание: вызывает стандартный конструктор JSONDecoder, которому в качестве аргумента object_hook подается ссылка на метод __object_hook.

Тестирование этого модуля как самостоятельной программы невозможно, так как указанный классы лишь подаются на вход методам библиотеки json как средство описания методов кодирования и декодирования. По этой причине тестирование по сути происходит для используемых в программе методов библиотеки json, но с использованием описанных выше классов.

2.1.2 files_handler.py

Модуль содержит 1 класс – FilesHandler. В этом классе реализована логика работы с различными файлами для Doom engine. Это может IWAD, PWAD, PK3, а также файл с любым другим произвольным заголовком.

При создании объекта этого класса также указывается словарь возможных заголовков, где ключом является расширение искомого файла, а значением – массив его возможных заголовков. Такая методика позволяет гибко настраивать этот класс для чтения любых файлов, используемых в Doom engine.

Получить доступ к составленному словарю файлов можно через свойство files_dict.

Для вызова извне доступны следующие методы:

- __init__(self, filter_config)
  Аргументы:

  1. filter_config – словарь, где ключом является строка с расширением файла, например 'wad', а ключом массив байтовых строк с заголовками искомого файла, например, [b'IWAD'].

  Возвращаемое значение: None.
  Описание: конструктор класса, инициализирует внутренние переменные и задает указанную в filter_config конфигурацию.

- read_files_dict(self, path)
  Аргументы:

  1. path – путь к директории с обрабатываемыми файлами.

  Возвращаемое значение: None.
  Описание: результатом выполнения метода должно быть составление словаря файлов, хранимых в указанной директории и соответствующих описанным при создании экземпляра класса критериям поиска.

2.1.3 dooms_handler.py

Модуль содержит 1 класс – DoomsHandler. В этом модуле реализована логика работы с исполняемыми фалйами игр. Файлы хранятся как словарь, где ключом является имя файла, а значением – путь к нему. Доступ к словарю файлов осуществляется через свойство dooms_dict.

Для вызова извне доступны следующие методы:

- __init__(self)
  Возвращаемое значение: None.
  Описание: конструктор класса.

- read_dooms(self, path)
  Аргументы:

  1. path – пусть к JSON файлу, хранящему словарь, где ключом является имя исполняемого файла, а значением – путь к нему.

  Возвращаемое значение: None.
  Описание: читает по указанному пути JSON-файл, содержащий словарь, где ключом является имя файла, а значением – путь к этому файлу, использует PathJSONDecoder для формирования словаря, где пути в виде строки из файла заменяются на объекты путей из библиотеки pathlib.

- add_doom(self, path, name: str)
  Аргументы:

  1. path – путь к исполняемому файлу,

2. name – имя испольняемого файла.

Возвращаемое значение: None.
Описание: добавляет в словарь испольняемых файлов значение path с ключом name.

- delete_doom(self, name: str)
Аргументы:

  1. name – имя испольняемого файла.

  Возвращаемое значение: None.
  Описание: удаляет из словаря испольняемых файлов значение с ключом name.

- write_dooms(self, path)
Аргументы:

  1. path – пусть к JSON файлу, хранящему словарь, где ключом является имя исполняемого файла, а значением – путь к нему.

  Возвращаемое значение: None.
  Описание: записывает словарь исполняемых файлов в JSON-файл по указанному пути, используя для кодировки PathJSONEncoder.

2.1.4 config.py

Модуль содержит класс Config, который обрабатывает конфигурационный файл приложения. При создании экземпляра этого класса необходимо задать путь к конфигурационному файлу. По умолчанию это ./config.json. Конфигурация приложения хранится как словарь, доступ к которому осуществляется по свойству config_dict. Ключом в этом словаре является имя конкретного параметра конфигурации, а значением непосредственно значение этого параметра. В текущей версии программы предусмотрены следующие параметры для настройки:

- iwads_path – путь к директории с файлами IWAD.
- pwads_path – путь к директории с файлами PWAD.

- pk3s_path – путь к директории с файлами PK3.
- saves_path – путь к директории сохранений.
- configs_path – путь к директории конфигурационных файлов игр.
- dooms_path – пусть к файлу dooms.json, содержащему словарь исполняемых файлов.

Для вызова извне доступны следующие методы:

- __init__(self, path=Path('./config.json'))
  Аргументы:

  1. path – путь к файлу конфигурации.

  Возвращаемое значение: None.
  Описание: конструктор класса, задает внутренние переменные: путь к конфигурационному файлу path, а также стандартный набор параметров конфигурации.

- init_default_config(self)
  Аргументы: None.
  Возвращаемое значение: None.
  Описание: инициализирует конфигурацию по умолчанию – все директории находятся в той же директории, что и приложение.

- set_field(self, name, value)
  Аргументы:

  1. name – изменяемый параметр,
  2. value – новое значение параметра.

  Возвращаемое значение: None.
  Описание: задает новое значение value параметра name, если такой параметр существует.

- write_config(self)
  Аргументы: None.
  Возвращаемое значение: None.
  Описание: записывает текущий словарь в файл config.json по заданному при создании экземпляра класса пути, используя PathJSONEncoder.

- read_config(self)
  Аргументы: None.
  Возвращаемое значение: None.
  Описание: читает словарь из файла config.json по заданному при создании экземпляра класса пути, используя PathJSONDecoder, а также выполняет проверку, что прочиталось нужное число свойств, а также, что эти свосйства корректны. В противном случае происходит исключение KeyError.

2.1.5 configs_handler.py

Модуль содержит 1 класс – ConfigsHandler и отвечает за управление конфигурационными файлами игры.

Для вызова извне доступны следующие методы:

- __init__(self)
  Возвращаемое значение: None.
  Описание: конструктор класса.

- read_configs_dict(self, path)
  Аргументы:

  1. path – путь к директории с файлами конфигурации.

  Возвращаемое значение: None.
  Описание: заполняет словарь конфигурационных файлов теми файлами, которые находятся по пути path и имеют расширение .ini.

- get_config(self, name)
  Аргументы:

  1. name – имя директории.

  Возвращаемое значение: путь к файлу конфигурации name или None, если такого файла нет.
  Описание: если файл существует, метод просто возвращает путь к нему, если же ее нет, то возвращается None.

2.1.6 saves_handler.py

Модуль содержит 1 класс – SavesHandler и отвечает за управление файлами сохранений. Для каждой комбинации исполняемого файла IWAD, PWAD и PK3 модуль создает директорию, где будут храниться файлы сохранения. Список директорий хранится как словарь, где ключом является имя директории, а значением – путь к ней.

Для вызова извне доступны следующие методы:

- __init__(self)
  Возвращаемое значение: None.
  Описание: конструктор класса.

- read_saves_dict(self, path)
  Аргументы:

  1. path – путь к директории с файлами сохранения.

  Возвращаемое значение: None.
  Описание: заполняет словарь директорий теми директориями, которые находятся по пути path.

- get_saves_dir(self, name)
  Аргументы:

  1. name – имя директории.

  Возвращаемое значение: путь к директории name.
  Описание: если директория существует, метод просто возвращает путь к ней, если же ее нет, то эта директория предварительно создается.

2.1.7 launcher.py

Модуль содержит 1 класс - Launcher. При создании экземпляра класса необходимо задать как минимум пути к исполняемому файлу и IWAD-файлу, так как это обязательный минимум для запуска игры. Остальные, описанные в предыдущих разделах файлы, опциональны.

Для вызова извне доступны следующие методы:

- __init__(self, dooms_handler: DoomsHandler, iwads_handler: FilesHandler, pwads_handler: FilesHandler = None, pk3s_handler: FilesHandler = None, saves_handler: SavesHandler = None, configs_handler: ConfigsHandler = None)
  Аргументы:
  - dooms_handler – ссылка на экземпляр класса DoomsHandler.
  - iwads_handler – ссылка на экземпляр класса FilesHandler (IWAD).
  - pwads_handler – ссылка на экземпляр класса FilesHandler (PWAD).
  - pk3s_handler – ссылка на экземпляр класса FilesHandler (PK3).
  - saves_handler – ссылка на экземпляр класса SavesHandler.
  - configs_handler – ссылка на экземпляр класса ConfigsHandler.

  Возвращаемое значение: None.
  Описание: конструктор класса, сохраняющий ссылки на заданные в аргументах обработчики.

- create_command(self, doom_name, iwad_name, pwad_names=None, pk3_names=None)
  Аргументы:
  - doom_name – имя исполняемого файла.
  - iwad_name – имя файла IWAD.
  - pwad_names – список имен файлов PWAD.
  - pk3_names – список имен файлов PK3.

  Возвращаемое значение: массив с аргументами команды запуска исполняемого файла.
  Описание: формирует команду для запуска исполняемого файла, получая по заданным из аргументов параметрам и заранее заданным обработчикам в конструкторе пути к файлам. В команду добавляется путь к директории сохранения, если она существует. Если директория не существует, ее создаст saves_handler. Также добавляется путь к файлу конфигурации, если он ранее был создан пользователем.

- launch(self, command)
  Аргументы:

  – command – массив строк, которые формируют аргументы команды для запуска исполняемого файла.

  Возвращаемое значение: None.
  Описание: запускает исполняемый файл, формируя соответствующую команду из заданного массив.

2.1.8 ui_qt

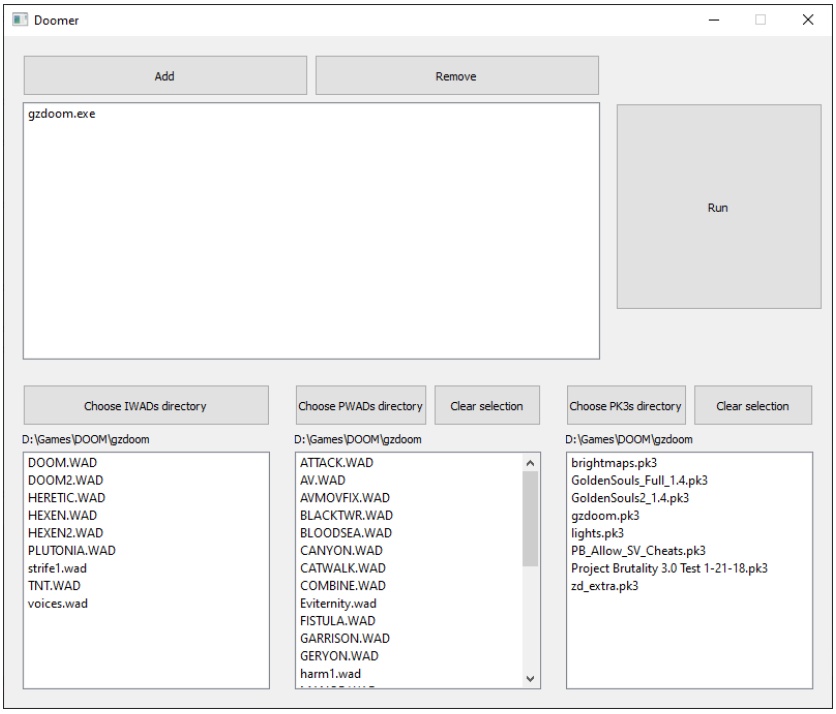

Рисунок 2: Прототип пользовательского интерфейса

Модуль ui_qt является прототипом пользовательского интерфейса приложения. Модуль состоит из нескольких файлов: dooms_widget.py, files_widget.py и ui_widget.py. В первых двух определены виджеты из библиотеки PyQt5, которые предоставляют доступ к различным файлам, которые используют пользователь. Последний файл отвечает за общую

структуру интерфейса. На рисунке 2 можно увидеть единственную форму приложения.'

В левой верхней части формы находится список исполняемых файлов, а также 2 кнопки, позволяющие добавить (Add) новый файл и удалить выбранный (Remove). В списке возможен выбор только одного исполняемого файла.

В нижней части формы находятся 3 списка различных видов файлов: IWAD, PWAD и PK3. Для каждого списка есть кнопка для выбора директории, откуда нужно читать файлы. Так как PWAD и PK3 файлы могут быть не использованы, использованы по одному или по несколько за раз, была добавлена кнопка "Clear selection" для каждого списка. IWAD можно выбрать только один.

В правой верхней части расположена большая кнопка Run, которая запускает выбранный исполняемый файл с заданными параметрами.

Тестирование работы пользовательского интерфейса выполняться на этапе аттестационного тестирование, так как интерфейс не является функциональным без работы всех остальных модулей системы. Его тестирование возможно лишь в конечном приложении.

2.2 Стратегия блочного тестирования

В качестве основных инструментов тестирования будут использоваться среда разработки PyCharm и библиотека unittest. Тестироваться будут только те методы, которые были перечислены выше. Тестирование производится методом черного ящика – при вызове метода с определенными параметрами ожидается определенный результат его работы: возвращаемое значение или изменения полей экземпляра класса.

Блочное тестирование проводиться для следующих методов:

- pathlib_json.py:
  - json.load при cls=PathJSONDecoder.
  - json.dump при cls=PathJSONEncoder.
- files_handler.py:
  - read_files_dict.

- dooms_handler.py:
  - read_dooms.
  - add_doom.
  - delete_doom.
- config.py:
  - read_config.
  - set_field.
  - write_config.
- configs_handler.py:
  - read_configs_dict.
- saves_handler.py:
  - read_saves_dict.
  - get_saves_dir.

Тестовыми данными послужат заранее подготовленные директории с различными файлами: IWAD, PWAD, PK3, INI и ZDS (файл сохранения). Тесты будут проводиться с различными комбинациями файлов разных типов для каждого модуля, чтобы покрыть все возможные сценарии использования.

Отдельно стоит отметить стратегию тестирования модуля pathlib_json.py. Тестирование этого модуля возможно только в совокупности в методами библиотеки json – dump и load. По сути будут тестироваться эти 2 метода, в качестве параметра которым будут подаваться классы PathJSONEncoder и PathJSONDecoder.

При тестировании использовалась методика DDT (Data-Driven Testing). То есть вместо описания нескольких однотипных тестов описывался всего один тест, его входные данные и ожидаемые результаты. При описании самих тестов в подробном плане тестирования используется схожее описание: описание теста и описание различных тестовых данных.

2.2.1 Критерий корректности

Работа объекта тестирования считается корректной, если ожидаемый результат полностью соответствует полученному.

2.3 Стратегия интеграционного тестирования

Простота связей приложения позволяет проводить интеграционный тесты также с помощью библиотеки unittest и стандартных средств PyCharm.

Автоматическое тестирование возможно для связей с модулем launcher.py. Тут важно отметить, что для запуска исполняемого файла необходимо задать следующие параметры:

- путь к исполняемому файлу,
- путь к IWAD файлу.

Остальные параметры при работе этого модуля опциональны. То есть любая комбинация связей будет обязательно содержать описанные выше 2 параметра. Работа с файлами сохранения также всегда происходит автоматически без участия пользователя. То есть любой запуск исполняемого файла подразумевает автоматическое создание директории с файлами сохранения или чтение уже существующей. Похожим образом происходит работа с конфигурационными файлами.

Тестируемые связи:

- Минимальный необходимый набор связей для запуска исполняемого файла – launcher.py + dooms_handler.py + files_handler.py (IWAD) + configs_handler.py + saves_handler.py.
- Опциональные связи: launcher.py + dooms_handler.py + files_handler.py (IWAD) + следующие комбинации связей:

  files_handler.py (PWAD).

  files_handler.py (PK3).

  files_handler.py (PWAD) + files_handler.py (PK3).

Следующие связи не будут тестироваться:

- ui_qt.py + config.py и qt_ui.py + launcher.py – эти связь по сути обеспечивают конечный функционал, который проявит себя лишь в работе конечного цельного приложения. Тестирование этих связей не имеет смысл, так как достаточными будут лишь аттестационные тесты конечных функций приложения. Тестирование этих связей на более раних этапах по сути дублировало бы аттестационные тесты.

- config.py + pathlin_json.py и dooms_handler.py + pathlin_json.py – тестирование этих связей не имеет смысла, так как config.py и dooms_handler.py не могут функционировать без функций json.dump и json.load, которые используют модуль pathlin_json.py, в котором описан алгоритм конвертации данных в формат JSON и алгоритм его декодирования. Блочные тесты модуля pathlin_json.py и модулей config.py и dooms_handler.py обеспечат тестирование этой связи. Дополнительные тесты на этапе интеграционного тестирования лишь дублировали бы блочные.

Интеграционные тесты связей pathlib_json.py с config.py и dooms_handler.py не имеют смысла, так как последние 2 модуля не могут работать без pathlib_json.py самостоятельно. pathlib_json.py это просто надстройка над стандартными методами библиотеки json, расширяющая ее возможности для задач приложения. Блочных тестов config.py и dooms_handler.py, а также отедльных блочных тестов для pathlib_json.py достаточно, чтобы оценить корректность работы этих модулей.

2.3.1 Критерий корректности

Работа объекта тестирования считается корректной, если ожидаемый результат полностью соответствует полученному.

2.4 Стратегия аттестационного тестирования

Аттестационное тестирование проводится вручную через взаимодействие с пользовательским интерфейсом. На этом этапе тестирования будут проводиться следующие виды тестов:

- корректность отображения данных из модулей в полях интерфейса,

- возможность изменения конфигурации программы и реакции программы на эти изменения,
- корректность запуска исполняемого файла с заданными параметрами.

2.4.1 Критерий корректности

Работа объекта тестирования считается корректной, если ожидаемый результат полностью соответствует полученному.

2.5 Стратегия нагрузочного тестирования

Использование приложения не подразумевает запуска большого числа исполняемых файлов, а также одновременного использования большого числа файлов IWAD, PWAD и т. д. Как правило реальный пользователь запускает один исполняемый файл с одним IWAD, одим или несколькими PWAD файлами и одним или несколькими PK3 файлами.

Узким в плане производительности местом является запуск и изменение конфигурации самого приложения. При каждом запуске или изменении конфигурации происходит чтение указанных пользователем директорий и проверка файлов в них. Пользователь может иметь достаточно большую коллекцию IWAD, PWAD и PK3 файлов – несколько десятков каждого вида. Нагрузочное тестирование должно происходить именно на этапе запуска или изменения конфигурации программы.

2.5.1 Критерий корректности

Критерием прохождения нагрузочного теста является успешная обработка программой указанных директорий, то есть вывод в соответствующем списке файлов нужного формата (PWAD, IWAD, PK3).

3 Детальный план тестов

3.1 Блочное тестирование

3.1.1 pathlib_json.py

3.1.1.1 Тесты PathJSONEncoder

Объект тестирования: json.dump.

Цель теста: проверка корректности конвертации словаря Python в JSON формат с использованием класса PathJSONEncoder в качестве значение параметра cls в методе json.dump.

Данные для тестирования:

- **Тест Б1**
  Пустой словарь. Тип теста: общий.
  Входные данные: словарь

```
{}
```

  Ожидаемый результат: создан файл в формате JSON

```
{}
```

- **Тест Б2**
  Словарь с одной парой ключ-значение. Тип теста: общий.
  Входные данные: словарь

```
{
    'key': Path('./test_path/')
}
```

  Ожидаемый результат: создан файл в формате JSON

```
{'key': 'test_path'}
```

- **Тест Б3**
  Словарь с несколькими парами ключ-значение. Тип теста: общий.
  Входные данные: словарь

```
{
    'key1': Path('./test_path_1/'),
    'key2': Path('./test_path_2/'),
    'key3': Path('./test_path_3/')
}
```

Ожидаемый результат: файл в формате JSON

```
{"key1": "test_path_1", "key2": "test_path_2", "key3": "
   test_path_3"}
```

3.1.1.2 Тесты PathJSONDecoder

Объект тестирования: json.load.

Цель теста: проверка корректности конвертации содержания файла в формате JSON в словарь Python с использованием класса PathJSONDecoder в качестве значение параметра cls в методе json.load.

Данные для тестирования:

- **Тест Б4**
  Файл с пустым объектом JSON. Тип теста: общий.
  Входные данные: путь к файлу в формате JSON содержащим:

```
{}
```

  Ожидаемый результат: словарь

```
{}
```

- **Тест Б5**
  Файл с одной парой ключ-значение. Тип теста: общий.
  Входные данные: путь к файлу в формате JSON содержащим:

```
{"key": "test_path"}
```

  Ожидаемый результат: словарь

```
{
    'key': Path('./test_path/')
}
```

- **Тест Б6**
  Файл с несколькими парами ключ-значение. Тип теста: общий.
  Входные данные: путь к файлу в формате JSON содержащим:

```
{"key1": "test_path_1", "key2": "test_path_2", "key3": "
   test_path_3"}
```

  Ожидаемый результат: словарь

```
{
    'key1': Path('./test_path_1/'),
    'key2': Path('./test_path_2/'),
    'key3': Path('./test_path_3/')
}
```

3.1.2 files_handler.py

3.1.2.1 Тестирование обработки IWAD и PWAD файлов

Объект тестирования: FilesHandler.read_files_dict.
Цель теста: проверка корректности поиска IWAD и PWAD файлов в указанной директории.
Данные для тестирования:

- **Тест Б7**
  Пустая директория. Тип теста: общий.
  Входные данные: путь к пустой директории.
  Ожидаемый результат: словарь

```
{}
```

- **Тест Б8**
  Директория с одним IWAD файлом. Тип теста: общий.
  Входные данные: путь к директории с файлом DOOM.WAD (IWAD).
  Ожидаемый результат: словарь

```
{
    'DOOM.WAD': CASES_PATH / 'one_wad' / 'DOOM.WAD'
}
```

- **Тест Б9**
  Непустая директория с одним неподходящим. Тип теста: негативный.
  Входные данные: путь к директории с файлом 1.txt (не WAD).
  Ожидаемый результат: словарь

```
{}
```

- **Тест Б10**
  Директория с несколькими IWAD и PWAD файлами. Тип теста: общий.
  Входные данные: путь к директориями с файлами DOOM.WAD (IWAD), DOOM2.WAD (IWAD), SCENARIO 1231.wad (PWAD).
  Ожидаемый результат: словарь

```
{
    'DOOM.WAD': CASES_PATH / 'multiple_wads' / 'DOOM.WAD',
    'DOOM2.WAD': CASES_PATH / 'multiple_wads' / 'DOOM2.WAD
        ',
    'SCENARIO 1231.wad': CASES_PATH / 'multiple_wads' / '
       SCENARIO 1231.wad'
}
```

- **Тест Б11**
  Директория с несколькими IWAD, PWAD файлами и одним неподходящим файлом. Тип теста: негативный.
  Входные данные: файл с файлами DOOM.WAD (IWAD), SCENARIO 1231.wad (PWAD) и 1.txt (WAD).
  Ожидаемый результат: словарь

```
{
    'DOOM2.WAD': CASES_PATH / 'mixed_1' / 'DOOM2.WAD',
    'SCENARIO 1231.wad': CASES_PATH / 'mixed_1' / '
       SCENARIO 1231.wad'
}
```

- **Тест Б12**
  Директория с несколькими IWAD, PWAD файлами и

неподходящими файлами. Тип теста: негативный.
Входные данные: директория с файлами DOOM.WAD (IWAD), DOOM2.WAD (IWAD), MEPHISTO.WAD (PWAD), SCENARIO 1231.wad (PWAD), 1.txt (не WAD), 2.txt (не WAD), brightmaps.pk3 (PK3).
Ожидаемый результат: словарь

```
{
    'DOOM.WAD': CASES_PATH / 'mixed_2' / 'DOOM.WAD',
    'DOOM2.WAD': CASES_PATH / 'mixed_2' / 'DOOM2.WAD',
    'MEPHISTO.WAD': CASES_PATH / 'mixed_2' / 'MEPHISTO.WAD
        ',
    'SCENARIO 1231.wad': CASES_PATH / 'mixed_2' / '
        SCENARIO 1231.wad'
}
```

3.1.2.2 Тестирование обработки PK3 файлов

Объект тестирования: FilesHandler.read_files_dict.
Цель теста: проверка корректности поиска PK3 файлов из указанной директории.
Данные для тестирования:

- **Тест Б13**
  Пустая директория. Тип теста: общий.
  Входные данные: путь к пустой директории.
  Ожидаемый результат: словарь

```
{}
```

- **Тест Б14**
  Директория с одним PK3 файлом. Тип теста: общий.
  Входные данные: путь к директории с файлом brightmaps.pk3 (PK3).
  Ожидаемый результат: словарь

```
{
    'brightmaps.pk3': CASES_PATH / 'one_pk3' / 'brightmaps
        .pk3'
```

```
}
```

- **Тест Б15**
  Директория с несколькими PK3 файлами. Тип теста: общий.
  Входные данные: путь к директории с файлами brightmaps.pk3 (PK3), zd_extra.pk3 (PK3).
  Ожидаемый результат: словарь

```
{
    'brightmaps.pk3': CASES_PATH / 'multiple_pk3s' / '
       brightmaps.pk3',
    'zd_extra.pk3': CASES_PATH / 'multiple_pk3s' / '
       zd_extra.pk3'
}
```

- **Тест Б16**
  Директория с не PK3 файлом. Тип теста: негативный.
  Входные данные: путь к директории с 1.txt (не PK3).
  Ожидаемый результат: словарь

```
{}
```

- **Тест Б17**
  Директория с несколькими PK3 файлами и неподходящими файлами, включая PWAD. Тип теста: негативный.
  Входные данные: путь к директории с файлами 1.txt (не PK3), brightmaps.pk3 (PK3), zd_extra.pk3 (PK3), TEETH.WAD (PWAD).
  Ожидаемый результат: словарь

```
{
    'brightmaps.pk3': CASES_PATH / 'mixed_pk3' / '
       brightmaps.pk3',
    'zd_extra.pk3': CASES_PATH / 'mixed_pk3' / 'zd_extra.
       pk3'
}
```

3.1.3 dooms_handler.py

3.1.3.1 Тестирование чтения списка исполняемых файлов

Объект тестирования: DoomsHandler.read_dooms.

Цель теста: проверка корректности формирования словаря исполняемых файлов из файла в формате JSON.

Данные для тестирования:

- **Тест Б18**
  Пустой файл. Тип теста: общий.
  Входные данные: файл в формате JSON

```
{}
```

  Ожидаемый результат: словарь

```
{}
```

- **Тест Б19**
  Файл с путями к нескольким исполняемым файлам. Тип теста: общий.
  Входные данные: файл в формате JSON

```
{"gzdoom.exe": "D:\\Games\\DOOM\\gzdoom\\gzdoom.exe", "
   zdoom.exe": "D:\\Games\\DOOM\\zdoom\\zdoom.exe"}
```

  Ожидаемый результат: словарь

```
{
    'gzdoom.exe': Path('D:\\Games\\DOOM\\gzdoom\\gzdoom.
       exe'),
    'zdoom.exe': Path('D:\\Games\\DOOM\\zdoom\\zdoom.exe')
}
```

3.1.3.2 Тестирование добавления новых исполняемых файлов

Объект тестирования: DoomsHandler.add_doom.

Цель теста: проверка корректности добавления новых исполняемых файлов в словарь.

Данные для тестирования:

- **Тест Б20**
  Пустой файл с добавлением одного нового. Тип теста: общий.
  Входные данные: файл в формате JSON и список новых исполняемых файлов к добавлению

```
{}
```

```
[
    ('test_name', Path('./test_path'))
]
```

  Ожидаемый результат: словарь

```
{
    'test_name': Path('./test_path')
}
```

- **Тест Б21**
  Файл с путями к нескольким исполняемым файлом с добавлением одного нового. Тип теста: общий.
  Входные данные: файл в формате JSON и список новых исполняемых файлов к добавлению

```
{"gzdoom.exe": "D:\\Games\\DOOM\\gzdoom\\gzdoom.exe", "
   zdoom.exe": "D:\\Games\\DOOM\\zdoom\\zdoom.exe"}
```

```
[
    ('test_name', Path('./test_path'))
]
```

  Ожидаемый результат: словарь

```
{
    'gzdoom.exe': Path('D:\\Games\\DOOM\\gzdoom\\gzdoom.
       exe'),
    'zdoom.exe': Path('D:\\Games\\DOOM\\zdoom\\zdoom.exe')
       ,
    'test_name': Path('./test_path')
}
```

- **Тест Б22**
  Файл с путями к нескольким исполняемым файлом с добавлением нескольких новых. Тип теста: общий.
  Входные данные: файл в формате JSON и список новых исполняемых файлов к добавлению

```
{"gzdoom.exe": "D:\\Games\\DOOM\\gzdoom\\gzdoom.exe", "
   zdoom.exe": "D:\\Games\\DOOM\\zdoom\\zdoom.exe"}
```

```
[
    ('test_name1', Path('./test_path1')),
    ('test_name2', Path('./test_path2')),
]
```

  Ожидаемый результат: словарь

```
{
    'gzdoom.exe': Path('D:\\Games\\DOOM\\gzdoom\\gzdoom.
       exe'),
    'zdoom.exe': Path('D:\\Games\\DOOM\\zdoom\\zdoom.exe')
       ,
    'test_name1': Path('./test_path1'),
    'test_name2': Path('./test_path2')
}
```

3.1.3.3 Тестирование удаления исполняемых файлов

Объект тестирования: DoomsHandler.delete_doom .

Цель теста: проверка корректности удаления значений из словаря исполняемых файлов.

Данные для тестирования:

- **Тест Б23**
  Пустой файл и удаление несуществующего значения. Тип теста: негативный.
  Входные данные: файл в формате JSON и список исполняемых файлов к удалению

```
{}
```

```
[
    'test_name'
]
```

Ожидаемый результат: словарь

```
{}
```

- **Тест Б24**
  Файл с несколькими значениями и удаление одного из них. Тип теста: общий.
  Входные данные: файл в формате JSON и список исполняемых файлов к удалению

```
{"gzdoom.exe": "D:\\Games\\DOOM\\gzdoom\\gzdoom.exe", "
   zdoom.exe": "D:\\Games\\DOOM\\zdoom\\zdoom.exe"}
```

```
[
    'zdoom.exe'
]
```

Ожидаемый результат: словарь

```
{
    'gzdoom.exe': Path('D:\\Games\\DOOM\\gzdoom\\gzdoom.
       exe')
}
```

- **Тест Б25**
  Файл с несколькими значениями и удаление всех значений. Тип теста: общий.
  Входные данные: файл в формате JSON и список исполняемых файлов к удалению

```
{"gzdoom.exe": "D:\\Games\\DOOM\\gzdoom\\gzdoom.exe", "
   zdoom.exe": "D:\\Games\\DOOM\\zdoom\\zdoom.exe"}
```

```
[
    'zdoom.exe',
    'gzdoom.exe'
]
```

Ожидаемый результат: словарь

```
{}
```

3.1.3.4 Тестирование записи списка исполняемых файлов

Объект тестирования: DoomsHandler.write_dooms.

Цель теста: проверка корректности записи словаря файлов в файл в формате JSON.

Данные для тестирования:

- **Тест Б26**
  Пустой словарь. Тип теста: общий.
  Входные данные: список исполняемых файлов к добавлению

```
[]
```

  Ожидаемый результат: файл в формате JSON

```
{}
```

- **Тест Б27**
  В словарь добавлено несколько значений. Тип теста: общий.
  Входные данные: список исполняемых файлов к добавлению

```
[
    ('gzdoom.exe', Path('D:\\Games\\DOOM\\gzdoom\\gzdoom.
       exe')),
    ('zdoom.exe', Path('D:\\Games\\DOOM\\zdoom\\zdoom.exe
       '))
]
```

  Ожидаемый результат: файл в формате JSON

```
{"gzdoom.exe": "D:\\Games\\DOOM\\gzdoom\\gzdoom.exe", "
   zdoom.exe": "D:\\Games\\DOOM\\zdoom\\zdoom.exe"}
```

Ожидаемый результат: успешно сформирован файл в формате JSON из заданного словаря.

3.1.4 config.py

3.1.4.1 Тестирование чтения конфигурационного файла

Объект тестирования: Config.read_config.

Цель теста: проверка корректности чтения корректного конфигурационного файла.

Данные для тестирования:

- **Тест Б28**
  Корректный конфигурационный файл. Тип теста: общий.
  Входные данные: файл в формате JSON

```
{"iwads_path": "iwads", "pwads_path": "pwads", "pk3s_path
   ": "pk3s", "saves_path": "saves", "screenshots_path": "
   screenshots", "configs_path": "configs", "dooms_path":
   "dooms.json"}
```

  Ожидаемый результат: словарь

```
{
    "iwads_path": Path("iwads"),
    "pwads_path": Path("pwads"),
    "pk3s_path": Path("pk3s"),
    "saves_path": Path("saves"),
    "screenshots_path": Path("screenshots"),
    "configs_path": Path("configs"),
    "dooms_path": Path("dooms.json")
}
```

- **Тест Б29**
  Пустой файл. Тип теста: негативный.
  Входные данные: файл в формате JSON

```
{}
```

Ожидаемый результат: исключение KeyError.

- **Тест Б30**
  Файл с некорректным полем. Тип теста: негативный.
  Входные данные: файл в формате JSON

```
{"wrong": "wrong", "pwads_path": "pwads", "pk3s_path": "
   pk3s", "saves_path": "saves", "screenshots_path": "
   screenshots", "configs_path": "configs", "dooms_path":
   "dooms.json"}
```

Ожидаемый результат: исключение KeyError.

3.1.4.2 Тестирование изменения конфигурационного файла

Объект тестирования: Config.set_field.
Цель теста: проверка корректности изменения полей конфигурации.
Данные для тестирования:

- **Тест Б31**
  Изменение существующего поля. Тип теста: общий.
  Входные данные: поле – 'iwads_path', значение – 'test'.
  Ожидаемый результат: файл в формате JSON

```
{"iwads_path": "test", "pwads_path": "pwads", "pk3s_path":
    "pk3s", "saves_path": "saves", "screenshots_path": "
   screenshots", "configs_path": "configs", "dooms_path":
   "dooms.json"}
```

- **Тест Б32**
  Изменение несуществующего поля. Тип теста: негативный.
  Входные данные: поле – 'some_field', значение – 'test'.
  Ожидаемый результат: файл в формате JSON

```
{"iwads_path": "iwads", "pwads_path": "pwads", "pk3s_path
   ": "pk3s", "saves_path": "saves", "screenshots_path": "
   screenshots", "configs_path": "configs", "dooms_path":
   "dooms.json"}
```

Ожидаемый результат: если поле существует, оно корректно изменено.

3.1.4.3 Тестирование записи конфигурационного файла

Объект тестирования: Config.write_config.

Цель теста: проверить корректность формирования конфигурационного файла в формате JSON из словаря.

Данные для тестирования:

- **Тест Б33**
  Конфигурация по умолчанию. Тип теста: общий.
  Входные данные: инициализируется словарь по умолчанию

```
{
    "iwads_path": Path("iwads"),
    "pwads_path": Path("pwads"),
    "pk3s_path": Path("pk3s"),
    "saves_path": Path("saves"),
    "screenshots_path": Path("screenshots"),
    "configs_path": Path("configs"),
    "dooms_path": Path("dooms.json")
}
```

Ожидаемый результат:

```
{"iwads_path": "iwads", "pwads_path": "pwads", "pk3s_path
   ": "pk3s", "saves_path": "saves", "screenshots_path": "
   screenshots", "configs_path": "configs", "dooms_path":
   "dooms.json"}
```

3.1.5 configs_handler.py

3.1.5.1 Тестирование составления словаря конфигурационных файлов по заданной директории

Объект тестирования: ConfigsHandler.read_configs_dict.

Цель теста: проверить корректность формирования словаря конфигурационных файлов.

Данные для тестирования:

- **Тест Б34**
  Пустая директория. Тип теста: общий.
  Входные данные: пустая директория **Ожидаемый результат:** словарь

```
{}
```

- **Тест Б35**
  Директория с одним INI файлов. Тип теста: общий.
  Входные данные: директория с файлом 1.ini (INI). **Ожидаемый результат:** словарь

```
{
    '1.ini': CASES_PATH / 'one_config/1.ini'
}
```

- **Тест Б36**
  Директория с несколькими INI файлами. Тип теста: общий.
  Входные данные: директория с файлами 1.ini (INI) и 2.ini (INI).
  Ожидаемый результат: словарь

```
{
    '1.ini': CASES_PATH / 'multiple_configs/1.ini',
    '2.ini': CASES_PATH / 'multiple_configs/2.ini'
}
```

- **Тест Б37**
  Директория с не INI файлом. Тип теста: негативный.
  Входные данные: директория с файлом non_config.txt (не INI).
  Ожидаемый результат: словарь

```
{}
```

- **Тест Б38**
  Директория с несколькими INI файлами и не INI файлом. Тип теста: негативный.
  Входные данные: директория с файлами 1.ini (INI), 2.ini (INI) и non_config.txt (не INI) **Ожидаемый результат:** словарь

```
{
    '1.ini': CASES_PATH / 'multiple_configs/1.ini',
    '2.ini': CASES_PATH / 'multiple_configs/2.ini'
}
```

3.1.6 saves_handler.py

3.1.6.1 Тестирование составления словаря директорий

Объект тестирования: SavesHandler.read_saves_dict.
Цель теста: проверить корректность составления словаря директорий файлов сохранения.
Данные для тестирования:

- **Тест Б39**
  Пустая директория.
  Входные данные: пусть к пустой директории.
  Ожидаемый результат: словарь

```
{}
```

- **Тест Б40**
  Директория с одной директорий файлов сохранения.
  Входные данные: директория с поддиректорией '1'.
  Ожидаемый результат: словарь

```
{
    '1': CASES_PATH / 'one_save_dir/1'
}
```

- **Тест Б41**
  Директория с несколькими директориями файлов сохранения.
  Входные данные: директория с поддиректориями ’1’, ’2’ и ’3’.
  Ожидаемый результат: словарь

```
{
    '1': CASES_PATH / 'multiple_saves_dir/1',
    '2': CASES_PATH / 'multiple_saves_dir/2',
    '3': CASES_PATH / 'multiple_saves_dir/3'
}
```

3.1.6.2 Тестирования получения пути к директории по запросу

Объект тестирования: SavesHandler.get_saves_dir.

Цель теста: проверить корректность получения и сиоздания директорий файлов сохранения.

Данные для тестирования:

- **Тест Б42**
  Получение существующей директории.
  Входные данные: директория с поддиректориями ’1’, ’2’ и ’3’, имя получаемой директории ’1’.
  Ожидаемый результат: возвращен путь CASES_PATH / ’get_non-existed_dir’ / ’1’, словарь файлов и множество поддиректорий не изменилось.

- **Тест Б43**
  Получение несуществующей директории.
  Входные данные: директория с поддиректориями ’1’, ’2’ и ’3’, имя получаемой директории ’4’.
  Ожидаемый результат: возвращен путь CASES_PATH / ’get_non-existed_dir’ / ’4’, создана директория по этому пути, а также обновлен словарь директорий файлов сохранения.

```
{
    '1': CASES_PATH / 'get_non-existed_dir/1',
    '2': CASES_PATH / 'get_non-existed_dir/2',
    '3': CASES_PATH / 'get_non-existed_dir/3',
    '4': CASES_PATH / 'get_non-existed_dir/4'
```

```
}
```

3.2 Интеграционное тестирование

3.2.1 Тестирование минимального набора связей

3.2.1.1 Запуск IWAD с существующим конфигурационным файлом и существующей директорией файлов сохранения

Тест И1

Объект тестирования: laucnher + dooms_handler + files_handler (IWAD) + saves_handler + configs_handler.

Цель теста: проверка корректности формирования команды для запуска исполняемого файла.

Входные данные:
Создается экземпляр DoomsHandler и читается файл в формате JSON:

```
{"gzdoom.exe": "D:\\Games\\DOOM\\gzdoom\\gzdoom.exe", "zdoom.
   exe": "D:\\Games\\DOOM\\zdoom\\zdoom.exe"}
```

Выбирается исполняемый файл zdoom.exe.

Создается экземпляр FilesHandler, который обрабатывает директорию с файлом DOOM.WAD (IWAD), этот же файл выбирается для дальнейшей работы.

Создается SavesHandler, который обрабатывает директорию, уже уже существует соответствующая поддиректория для файлов сохранения.

Создается ConfigsHandler, который обрабатывает директорию, где уже есть подходящий файл конфигурации.

В качестве входных параметров в Launcher.create_command подаются:

1. Выбранный из экземпляра DoomsHander путь к исполняемому файлу.
2. Выбранный из экземпляра FilesHandler путь к файлу IWAD.
3. Экземпляр SavesHandler.
4. Экземпляр ConfigsHandler.

Ожидаемый результат: сформирована команда для запуска исполянемого файла в виде списка:

```
[
    ZDOOM_PATH,
    '-iwad',
    str(CASES_PATH / 'minimal/iwads/DOOM.WAD'),
    '-savedir',
    str(CASES_PATH / 'minimal/conf_save/zdoom.exe_DOOM.WAD'),
    '-config',
    str(CASES_PATH / 'minimal/conf_save/zdoom.exe_DOOM.WAD.ini
        ')
]
```

3.2.1.2 Запуск IWAD без конфигурационного файла, но с существующей директорией файлов сохранения

Тест И2

Объект тестирования: laucnher + dooms_handler + files_handler (IWAD) + saves_handler + configs_handler.

Цель теста: проверка корректности формирования команды для запуска исполняемого файла.

Входные данные:

Создается экземпляр DoomsHandler и читается файл в формате JSON:

```
{"gzdoom.exe": "D:\\Games\\DOOM\\gzdoom\\gzdoom.exe", "zdoom.
    exe": "D:\\Games\\DOOM\\zdoom\\zdoom.exe"}
```

Выбирается исполняемый файл zdoom.exe.

Создается экземпляр FilesHandler, который обрабатывает директорию с файлом DOOM.WAD (IWAD), этот же файл выбирается для дальнейшей работы.

Создается SavesHandler, который обрабатывает директорию, уже уже существует соответствующая поддиректория для файлов сохранения.

Создается ConfigsHandler, который обрабатывает пустую директорию.

В качестве входных параметров в Launcher.create_command подаются:

1. Выбранный из экземпляра DoomsHander путь к исполняемому файлу.

2. Выбранный из экземпляра FilesHandler путь к файлу IWAD.

3. Экземпляр SavesHandler.

4. Экземпляр ConfigsHandler.

Ожидаемый результат: сформирована команда для запуска исполянемого файла в виде списка:

```
[
    ZDOOM_PATH,
    '-iwad',
    str(CASES_PATH / 'minimal/iwads/DOOM.WAD'),
    '-savedir',
    str(CASES_PATH / 'minimal/noconf_save/zdoom.exe_DOOM.WAD')
]
```

3.2.1.3 Запуск IWAD с существующим конфигурационным файлом, но без существующей директории файлов сохранения

Тест И3

Объект тестирования: laucnher + dooms_handler + files_handler (IWAD) + saves_handler + configs_handler.

Цель теста: проверка корректности формирования команды для запуска исполняемого файла.

Входные данные:

Создается экземпляр DoomsHandler и читается файл в формате JSON:

```
{"gzdoom.exe": "D:\\Games\\DOOM\\gzdoom\\gzdoom.exe", "zdoom.
   exe": "D:\\Games\\DOOM\\zdoom\\zdoom.exe"}
```

Выбирается исполняемый файл zdoom.exe.

Создается экземпляр FilesHandler, который обрабатывает директорию с файлом DOOM.WAD (IWAD), этот же файл выбирается для дальнейшей работы.

Создается SavesHandler, который обрабатывает пустую директорию.

Создается ConfigsHandler, который обрабатывает пустую директорию.

В качестве входных параметров в Launcher.create_command подаются:

1. Выбранный из экземпляра DoomsHander путь к исполняемому файлу.
2. Выбранный из экземпляра FilesHandler путь к файлу IWAD.
3. Экземпляр SavesHandler.
4. Экземпляр ConfigsHandler.

Ожидаемый результат: сформирована команда для запуска исполянемого файла в виде списка:

```
[
    ZDOOM_PATH,
    '-iwad',
    str(CASES_PATH / 'minimal/iwads/DOOM.WAD'),
    '-savedir',
    str(CASES_PATH / 'minimal/noconf_nosave/zdoom.exe_DOOM.WAD
       ')
]
```

Создана директория CASES_PATH / 'minimal/noconf_nosave/zdoom.exe_DOOM.WAD'.

3.2.1.4 Запуск IWAD с существующими конфигурационным файлом и без существующей директории файлов сохранения

Тест И4

Объект тестирования: laucnher + dooms_handler + files_handler (IWAD) + saves_handler + configs_handler.

Цель теста: проверка корректности формирования команды для запуска исполняемого файла.

Входные данные:

Создается экземпляр DoomsHandler и читается файл в формате JSON:

```
{"gzdoom.exe": "D:\\Games\\DOOM\\gzdoom\\gzdoom.exe", "zdoom.
   exe": "D:\\Games\\DOOM\\zdoom\\zdoom.exe"}
```

Выбирается исполняемый файл zdoom.exe.

Создается экземпляр FilesHandler, который обрабатывает директорию с файлом DOOM.WAD (IWAD), этот же файл выбирается для дальнейшей работы.

Создается SavesHandler, который обрабатывает пустую директорию.

Создается ConfigsHandler, который обрабатывает директорию, где уже есть подходящий файл конфигурации.

В качестве входных параметров в Launcher.create_command подаются:

1. Выбранный из экземпляра DoomsHander путь к исполняемому файлу.
2. Выбранный из экземпляра FilesHandler путь к файлу IWAD.
3. Экземпляр SavesHandler.
4. Экземпляр ConfigsHandler.

Ожидаемый результат: сформирована команда для запуска исполянемого файла в виде списка:

```
[
    ZDOOM_PATH,
    '-iwad',
    str(CASES_PATH / 'minimal/iwads/DOOM.WAD'),
    '-savedir',
    str(CASES_PATH / 'minimal/conf_nosave/zdoom.exe_DOOM.WAD'),
    '-config',
    str(CASES_PATH / 'minimal/conf_nosave/zdoom.exe_DOOM.WAD.
       ini')
]
```

Создана директория CASES_PATH / 'minimal/no-conf_nosave/zdoom.exe_DOOM.WAD'.

3.2.2 Тестирование опциональных связей

Тесты используют один и тот же набор директорий с файлами, но в различных условиях. Это следующие директории:

- CASES_PATH / 'optional/conf_save' – директория конфигурационных файлов и директорий для файлов сохранения. Все файлы и директории здесь подготовлены заранее и не должны пересоздаваться.
- CASES_PATH / 'optional/iwads' – директория с файлами IWAD: DOOM.WAD.
- CASES_PATH / 'optional/pwads' – директория с файлами PWAD: ATTACK.WAD и BLOODSEA.WAD.
- CASES_PATH / 'optional/pk3s' – директория с файлами PK3: GoldenSouls_Full_1.4.pk3 и PB_Allow_SV_Cheats.pk3.

Также создан следующий файл CASES_PATH / 'optional/-dooms.json':

```
{"gzdoom.exe": "D:\\Games\\DOOM\\gzdoom\\gzdoom.exe", "zdoom.
   exe": "D:\\Games\\DOOM\\zdoom\\zdoom.exe"}
```

Заранее создаются экземпляры следующих классов:

- DoomsHadler.
- FilesHandler (IWAD).
- FilesHandler (PWAD).
- FilesHandler (PK3).
- SavesHandler.
- Configs Handler.

Заранее вызываются соответствующиме методы для чтения данныз из описанных выше директорий и файлов.

Далее создается объект класса Laucnher, конструктор которого принимает как аргументы на описанные выше экземпляры классов-обработчиков.

3.2.2.1 Использование одного PWAD

Тест И5

Объект тестирования: laucnher + dooms_handler + files_handler (IWAD) + files_handler (PWAD) + saves_handler + configs_handler.

Цель теста: проверка корректности формирования команды для запуска исполняемого файла.

Входные данные:

1. 'zddom.exe'.
2. 'DOOM.WAD' (IWAD).
3. ['ATTACK.WAD'] (PWAD).
4. [] (PK3).

Ожидаемый результат: сформирована команда для запуска исполянемого файла в виде списка:

```
[
    ZDOOM_PATH,
    '-iwad',
    str(CASES_PATH / 'optional/iwads/DOOM.WAD'),
    '-file',
    str(CASES_PATH / 'optional/pwads/ATTACK.WAD'),
    '-savedir',
    str(CASES_PATH / 'optional/conf_save/zdoom.exe_DOOM.
       WAD_ATTACK.WAD'),
    '-config',
    str(CASES_PATH / 'optional/conf_save/zdoom.exe_DOOM.
       WAD_ATTACK.WAD.ini')
]
```

3.2.2.2 Использование одного PK3

Тест И6

Объект тестирования: laucnher + dooms_handler + files_handler (IWAD) + files_handler (PK3) + saves_handler + configs_handler.

Цель теста: проверка корректности формирования команды для запуска исполняемого файла.

Входные данные:

1. 'zddom.exe'.
2. 'DOOM.WAD' (IWAD).
3. [] (PWAD).
4. ['PB_Allow_SV_Cheats.pk3'] (PK3).

Ожидаемый результат:

```
[
    ZDOOM_PATH,
    '-iwad',
    str(CASES_PATH / 'optional/iwads/DOOM.WAD'),
    '-file',
    str(CASES_PATH / 'optional/pk3s/PB_Allow_SV_Cheats.pk3'),
    '-savedir',
    str(CASES_PATH / 'optional/conf_save/zdoom.exe_DOOM.
       WAD_PB_Allow_SV_Cheats.pk3'),
    '-config',
    str(CASES_PATH / 'optional/conf_save/zdoom.exe_DOOM.
       WAD_PB_Allow_SV_Cheats.pk3.ini')
]
```

3.2.2.3 Использование нескольких PWAD

Тест И7

Объект тестирования: laucnher + dooms_handler + files_handler (IWAD) + files_handler (PWAD) + saves_handler + configs_handler.

Цель теста: проверка корректности формирования команды для запуска исполняемого файла.

Входные данные:

1. 'zddom.exe'.
2. 'DOOM.WAD' (IWAD).

3. ['ATTACK.WAD', 'BLOODSEA.WAD'] (PWAD).

4. [] (PK3).

Ожидаемый результат:

```
[
    ZDOOM_PATH,
    '-iwad',
    str(CASES_PATH / 'optional/iwads/DOOM.WAD'),
    '-file',
    str(CASES_PATH / 'optional/pwads/ATTACK.WAD'),
    str(CASES_PATH / 'optional/pwads/BLOODSEA.WAD'),
    '-savedir',
    str(CASES_PATH / 'optional/conf_save/zdoom.exe_DOOM.
       WAD_ATTACK.WAD_BLOODSEA.WAD'),
    '-config',
    str(CASES_PATH / 'optional/conf_save/zdoom.exe_DOOM.
       WAD_ATTACK.WAD_BLOODSEA.WAD.ini')
]
```

3.2.2.4 Использование нескольких PK3

Тест И8

Объект тестирования: laucnher + dooms_handler + files_handler (IWAD) + files_handler (PWAD) + saves_handler + configs_handler.

Цель теста: проверка корректности формирования команды для запуска исполняемого файла.

Входные данные:

1. 'zddom.exe'.

2. 'DOOM.WAD' (IWAD).

3. [] (PWAD).

4. ['PB_Allow_SV_Cheats.pk3', 'GoldenSouls_Full_1.4.pk3'] (PK3).

Ожидаемый результат:

```
[
    ZDOOM_PATH,
    '-iwad',
    str(CASES_PATH / 'optional/iwads/DOOM.WAD'),
    '-file',
    str(CASES_PATH / 'optional/pk3s/PB_Allow_SV_Cheats.pk3'),
    str(CASES_PATH / 'optional/pk3s/GoldenSouls_Full_1.4.pk3'),
    '-savedir',
    str(CASES_PATH / 'optional/conf_save/zdoom.exe_DOOM.
       WAD_PB_Allow_SV_Cheats.pk3_GoldenSouls_Full_1.4.pk3'),
    '-config',
    str(CASES_PATH / 'optional/conf_save/zdoom.exe_DOOM.
       WAD_PB_Allow_SV_Cheats.pk3_GoldenSouls_Full_1.4.pk3.ini
       ')
]
```

3.2.2.5 Использование нескольких PK3 и PWAD

Тест И9

Объект тестирования: laucnher + dooms_handler + files_handler (IWAD) + files_handler (PWAD) + files_handler (PK3) + saves_handler + configs_handler.

Цель теста: проверка корректности формирования команды для запуска исполняемого файла.

Входные данные:

1. 'zddom.exe'.
2. 'DOOM.WAD' (IWAD).
3. ['ATTACK.WAD', 'BLOODSEA.WAD'] (PWAD).
4. ['PB_Allow_SV_Cheats.pk3', 'GoldenSouls_Full_1.4.pk3'] (PK3).

Ожидаемый результат:

```
[
    ZDOOM_PATH,
    '-iwad',
```

```
    str(CASES_PATH / 'optional/iwads/DOOM.WAD'),
    '-file',
    str(CASES_PATH / 'optional/pwads/ATTACK.WAD'),
    str(CASES_PATH / 'optional/pwads/BLOODSEA.WAD'),
    str(CASES_PATH / 'optional/pk3s/PB_Allow_SV_Cheats.pk3'),
    str(CASES_PATH / 'optional/pk3s/GoldenSouls_Full_1.4.pk3'),
    '-savedir',
    str(CASES_PATH / 'optional/conf_save/zdoom.exe_DOOM.
        WAD_ATTACK.WAD_BLOODSEA.WAD_PB_Allow_SV_Cheats.
        pk3_GoldenSouls_Full_1.4.pk3'),
    '-config',
    str(CASES_PATH / 'optional/conf_save/zdoom.exe_DOOM.
        WAD_ATTACK.WAD_BLOODSEA.WAD_PB_Allow_SV_Cheats.
        pk3_GoldenSouls_Full_1.4.pk3.ini')
]
```

3.3 Аттестационное тестирование

3.3.1 Тестирование функции ведения списка исполняемых файлов

3.3.1.1 Добавление файла

Тест А1

Объект тестирования (функционал): ведение списка исполняемых файлов.

Цель тестирования: проверка корректности добавления исполняемого файла в список.

Порядок действий:

1. Нажать на кнопку "Add" над списком исполняемых файлов.
2. Во всплывающем окне выбрать исполняемый файл в файловой системе.

Ожидаемый результат: выбранный исполняемый файл появился в списке.

3.3.1.2 Удаление файла

Тест А2

Объект тестирования (функционал): ведение списка исполняемых файлов.

Цель тестирования: проверка корректности удаления исполняемого файла из списка.

Порядок действий:

1. Выделить появившейся исполняемый файл в списке.
2. Нажать на кнопку "Remove".

Ожидаемый результат: выбранный файл исчез из списка.

3.3.2 Тестирование функции фильтрации IWAD, PWAD и PK3

3.3.2.1 Фильтрация IWAD

Тест А3

Объект тестирования (функционал): фильтрация IWAD, PWAD, PK3.

Цель тестирования: проверка корректности фильтрации IWAD.

Порядок действий:

1. Нажать на кнопку "Choose IWADs directory" над списком исполняемых файлов.
2. Во всплывающем окне выбрать желаемую директорию.

Ожидаемый результат: в списке появится только IWAD файлы из выбранной директории.

3.3.2.2 Фильтрация PWAD

Тест А4

Объект тестирования (функционал): фильтрация IWAD, PWAD, PK3.

Цель тестирования: проверка корректности фильтрации PWAD.

Порядок действий:

1. Нажать на кнопку "Choose PWADs directory" над списком исполняемых файлов.

2. Во всплывающем окне выбрать желаемую директорию.

Ожидаемый результат: в списке появится только PWAD файлы из выбранной директории.

3.3.2.3 Фильтрация PK3

Тест А5

Объект тестирования (функционал): фильтрация IWAD, PWAD, PK3.

Цель тестирования: проверка корректности фильтрации PK3.

Порядок действий:

1. Нажать на кнопку "Choose PK3s directory" над списком исполняемых файлов.

2. Во всплывающем окне выбрать желаемую директорию.

Ожидаемый результат: в списке появится только PK3 файлы из выбранной директории.

3.3.3 Тестирование запуска исполняемого файла

Тест А6

Объект тестирования (функционал): запуск выбранного исполняемого файла при наличии всех опциональных файлов (PWAD и PK3).

Подготовка:

Для проведения этого теста был выбран и заранее внесен в список gzdoom.exe – один из самых популярных портов Doom. В качестве IWAD был выбран и заранее внесен в указанную программе директорию DOOM2.WAD. В качестве PWAD был выбран AV.WAD (Alien Vendetta), так как он заменяет уровни оригинальной игры, благодаря чему его работоспособность будет легко проверить – игра сразу начнется не в стандартном DOOM2.WAD. Важно отметить, что для работы AV.WAD DOOM2.WAD необходим. В качестве PK3 был выбрал Project Brutality (Project Brutality 3.0 Test 1-21-18.pk3). Этот PK3 значительно изменяет

игровой процесс и интерфейс, из-за чего его корректную загрузку также будет легко проверить.

Порядок действий:

1. Выделить в списке исполняемых файлов gzdoom.exe.
2. Выделить в списке IWAD DOOM2.WAD.
3. Выделить в списке PWAD AV.WAD.
4. Выделить в списке PK3 Project Brutality 3.0 Test 1-21-18.pk3.
5. Нажать кнопку "Run".
6. Следуя указаниям меню игры запустить первый уровень.

Ожидаемый результат: gzdoom.exe корреткно загрузится и при запуске самой игры будет открыт AV.WAD, интерфейс будет отличаться от стандартного интерфейса DOOM2.WAD, а также появится более реалистичное освещение, которое недоступно в оригинальном AV.WAD.

3.3.4 Тестирование автоматического раздельного хранения файлов сохранения

Тест А7

Объект тестирования (функционал): раздельное автоматическое хранение файлов сохранение длякаждой уникальной конбинации запуска исполняемого файла, IWAD, PWAD и PK3.

3.3.4.1 Первый запуск

Цель тестирования: проверка корректности создания новой директории для файлов сохранения.

Порядок действий:

1. Очистить директорию ./saves.
2. Повторить тест запуска исполняемого файла с IWAD, PWAD и PK3.
3. Через интерфейс игры создать файл сохранения.

Ожидаемый результат: в ./saves должна появиться специальная директория, внутри которой будет созданный файл сохранения.

3.3.4.2 Повторные запуски

Тест A8

Цель тестирования: проверка загрузки уже существующей директории файлов сохранения.

Подготовка: провести предыдущий тест.

Порядок действий:

1. Запустить тот же исполняемый файл, IWAD, PWAD и PK3, что и в предыдущем тесте.
2. Открыть список файлов сохранения через интерфейс игры.

Ожидаемый результат: в списке файлов сохранения будет виден файл созданный на предыдущем тесте.

3.3.5 Тестирование автоматизации загрузки конфигурационного файла

Тест A9

Объект тестирования (функционал): автоматическая загрузка заранее заданного пользователем файла конфигурации.

3.3.5.1 Ручное добавление файла конфигурации в структуру проекта

Цель тестирования: проверка корректности загрузки конфигурационного файла.

Порядок действий:

- Скопировать в директорию ./configs стандартный конфигурационный файл gzdoom.
- Модифицировать следующие значение в файле:

```
...
vid_scale_customheight=960
vid_scale_customwidth=1280
...
```

- Изменить имя файла на "gzdoom.exe_DOOM2.WAD_AV.WAD_Project Brutality 3.0 Test 1-21-18.pk3".
- Повторить шаги теста запуска исполняемого файла с IWAD, PWAD и PK3.

Ожидаемый результат: игра запустится в разрешении 1280x960.

3.4 Нагрузочное тестирование

Для проведения тестов созданы специальные директории, заполненные файлами IWAD, PWAD и PK3. Тестирование будет проводиться путем запуска программы и ручным указанием соответствующих директорий в интерфейсе приложения.

3.4.1 Тестирование малого числа файлов

Тест H1
Тип теста: общий.
Объект тестирования (функционал): отображение списков файлов при выборе директории с малым числом файлов.
Входные данные: директория, содержащая 5 IWAD, 5 PWAD и 5 PK3 файлов.
Ожидаемый результат: в соответствующих списках будут отображены 5 IWAD, 5 PWAD и 5 PK3 файлов.

3.4.2 Тестирование среднего числа файлов

Тест H2
Тип теста: общий.
Объект тестирования (функционал): отображение списков файлов при выборе директории со среднем числом файлов.
Входные данные: директория, содержащая 10 IWAD, 20 PWAD и 10 PK3 файлов.
Ожидаемый результат: в соответствующих списках будут отображены 10 IWAD, 20 PWAD и 10 PK3 файлов.

3.4.3 Тестирование большого числа файлов

Тест Н3

Тип теста: общий.

Объект тестирования (функционал): отображение списков файлов при выборе директории с большим числом файлов.

Входные данные: директория, содержащая 20 IWAD, 100 PWAD и 50 PK3 файлов.

Ожидаемый результат: в соответствующих списках будут отображены 20 IWAD, 100 PWAD и 50 PK3 файлов.

4 Журнал тестирования

4.1 Журнал блочного тестирования

№ теста	Дата	Результат	Отчет об ошибке
Б1	26.12.20	пройден	
Б2	26.12.20	пройден	
Б3	26.12.20	пройден	
Б4	26.12.20	пройден	
Б5	26.12.20	не пройден	1
Б6	26.12.20	не пройден	1
Б7	26.12.20	пройден	
Б8	26.12.20	пройден	
Б9	26.12.20	пройден	
Б10	26.12.20	не пройден	2
Б11	26.12.20	не пройден	2
Б12	26.12.20	не пройден	2
Б13	26.12.20	пройден	
Б14	26.12.20	пройден	
Б15	26.12.20	пройден	
Б16	26.12.20	пройден	
Б17	26.12.20	пройден	
Б18	26.12.20	пройден	
Б19	26.12.20	пройден	
Б20	26.12.20	пройден	

Б21	26.12.20	пройден	
Б22	26.12.20	пройден	
Б23	26.12.20	пройден	
Б24	26.12.20	пройден	
Б25	26.12.20	пройден	
Б26	26.12.20	пройден	
Б27	26.12.20	пройден	
Б28	26.12.20	пройден	
Б29	26.12.20	пройден	
Б30	26.12.20	не пройден	3
Б31	26.12.20	пройден	
Б31	26.12.20	пройден	
Б33	26.12.20	пройден	
Б34	26.12.20	пройден	
Б35	26.12.20	пройден	
Б36	26.12.20	пройден	
Б37	26.12.20	пройден	
Б38	26.12.20	пройден	
Б39	26.12.20	пройден	
Б40	26.12.20	пройден	
Б41	26.12.20	пройден	
Б42	26.12.20	пройден	
Б43	26.12.20	не пройден	4

4.2 Журнал интеграционного тестирования

№ теста	Дата	Результат	Отчет об ошибке
И1	28.12.20	не пройден	5
И2	28.12.20	не пройден	5
И3	28.12.20	не пройден	5
И4	28.12.20	не пройден	5
И5	28.12.20	пройден	
И6	28.12.20	пройден	
И7	28.12.20	пройден	
И8	28.12.20	пройден	
И9	28.12.20	пройден	

4.3 Журнал аттестационного тестирования

№ теста	Дата	Результат	Отчет об ошибке
A1	29.12.20	пройден	
A2	29.12.20	не пройден	6
A3	29.12.20	пройден	
A4	29.12.20	пройден	
A5	29.12.20	пройден	
A6	29.12.20	пройден	
A7	29.12.20	пройден	
A8	29.12.20	пройден	
A9	29.12.20	пройден	

4.4 Журнал нагрузочного тестирования

№ теста	Дата	Результат	Отчет об ошибке
H1	30.12.20	пройден	
H2	30.12.20	пройден	
H3	30.12.20	пройден	

5 Журнал найденных ошибок

1. **Ошибка:** PathJSONDecoder при чтении пути из файла JSON не создавал объект пути из pathlib.
   Решение: исправлено формирование словаря из файла JSON.

2. **Ошибка:** FileHandler некорректно обрабатывал словарь заголовков файлов, обрабатывался только первый из заголовков.
   Решение: исправлена обработка словаря.

3. **Ошибка:** метод Config.read_config не читает некорректный файл только в случае несовпадения числа полей, но не проверяет сами поля.
   Решение: добавлена проверка всех полей.

4. **Ошибка:** метод SavesHandler.get_saves_dir пытался создать директорию, даже если она существует.

Решение: вызов метода dict.get был заменен на ручную проверку наличия ключа в словаре, так как вызов функции SavesHandler.__init_save_dir в качестве аргумента по умолчанию происходил раньше выполнения функции dict.get.

5. **Ошибка:** метод Launcher create_command добавлял аргумент -file независимо от наличия PWAD или PK3.
   Решение: добавление аргумента -file происходит только после проверки, заданны ли какие-либо подохдящие файлы.

6. **Ошибка:** если удалить из списка исполняемый файл кнопкой "Remove", не выбрав сам файл, программа вылетает.
   Решение: нажатие игнорируется, если пользователь не выбрал файл в списке.

6 Покрытие кода тестами

Расчет процента покрытия кода тестами рассчитывался с помощью среды разработки PyCharm. На рисунках 2 и 4 предоставлены отчеты по покрытию тестами кода основной логике и пользовательского интерфейса соответственно. Эти значения были вычислены исходя из блочных, интеграционных и аттестационных тестов в совокупности.

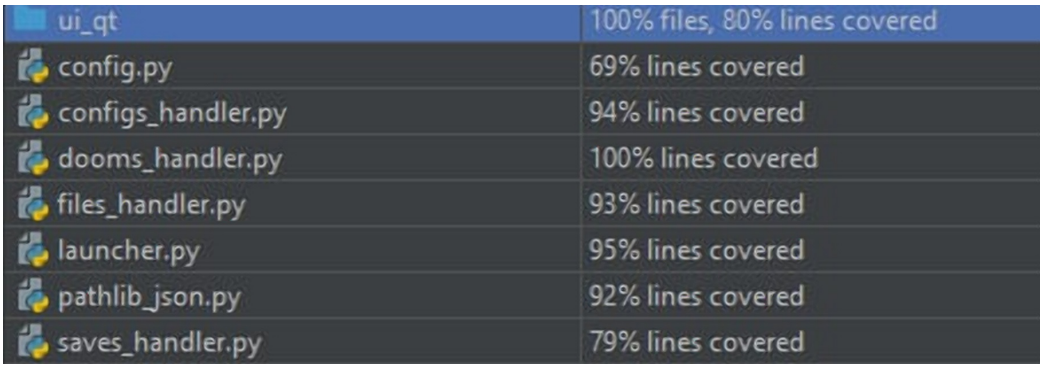

Рисунок 3: Покрытие кода тестами основных модулей программы

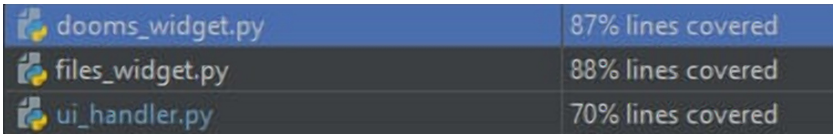

Рисунок 4: Покрытие кода тестами основных модулей программы

7 Заключение

Было проведено 64 вида теста, включая 43 блочных, 9 интеграционных, 9 аттестационных, 3 нагрузочных. ИЗ 64 тестов успешно были пройдены 52. Благодаря не пройденным 12 тестам удалось выявить и исправить 6 ошибок. При составлении тестов удалось добиться проверки программы в различных сценариях ее использования, за счет чего была достигнута высокое покрытие кода тестами – от 69% основной логики программы и от 70% кода пользовательского интерфейса.