MapVisualizator

Архитектурный документ

Команда: 29

Авторы документа:
Доржиев Донир Саянович (БПИ203) – раздел №4, 6;
Кондаков Семен Васильевич (БПИ201) – раздел №3, 5;
Насыхова Анастасия Артемовна (БПИ201) – раздел №1, 5, 6;
Неймышева Юлия Петровна (БПИ201) – раздел №2, 4.

Учебный ассистент: Щукин Владислав Евгеньевич

Раздел регистрации изменений

Версия документа	**Дата**	**Описание изменения**	**Автор изменения**
1	13.03.2023	Добавление разделов №1 и №2.	Насыхова А. А. Неймышева Ю. П.
2	14.03.2023	Добавление раздела №3.	Кондаков С. В.
3	15.03.2023	Добавление подразделов №4.	Доржиев Д. С.
4	17.03.2023	Добавление подразделов №4.	Доржиев Д. С.
5	18.03.2023	Внесение уточнений в подразделы №4.	Кондаков С. В. Доржиев Д. С. Неймышева Ю. П. Насыхова А. А.
6	19.03.2023	Добавление раздела №5.	Насыхова А. А. Кондаков С. В.
7	20.03.2023	Итоговое оформление документа.	Неймышева Ю. П.
8	12.03.2023	Доработка документа и исправление неточностей	Кондаков С. В. Доржиев Д. С. Неймышева Ю. П. Насыхова А. А.
9	26.04.2023	Корректировка исходного документа в соответствии с комментариями преподавателя, ассистента и рецензирующих команд	Неймышева Ю. П.
10	28.04.2023	Обновление информации во всех разделах в соответствии с перестроенной архитектурой	Насыхова А. А.

11	29.04.2023	Добавление нового логического представления и новых диаграмм последовательности	Кондаков С. В. Доржиев Д. С.
12	09.05.2023	Добавление раздела №6.	Насыхова А. А. Неймышева Ю. П.
13	10.05.2023	Добавление data flow diagram	Неймышева Ю. П.
14	11.05.2023	Добавление deployment diagram Общее редактирование документа на предмет соответствия текущему состоянию	Насыхова А. А.

1. Введение

1.1. Название проекта

MapVisualizator

1.2. Задействованные архитектурные представления

Структура архитектурного документа:

1.3. Контекст задачи и среда функционирования системы

MapVisualizator – инструмент для быстрого отображения карты. Применяется в основном в сфере видеоигр для отрисовки карты игрового процесса с отображением различных ресурсов на ней.

1.4. Рамки и цели проекта

MapVisualizator был разработан в рамках курсовой работы на 2 курсе Кондаковым С. В. Система была одним из компонентов игры, в которой карта отображалась на устройствах пользователей и для каждой игры пересоздавалась заново.

Цель разработки: создать инструмент, с помощью которого можно быстро и качественно отобразить сгенерированную карту игрового процесса.

Важным аспектом при разработке была необходимость в том, чтобы карта отображала местность, максимально приближенную к реальности (с плавными переходами от суши к воде, без резких перепадов высот и т. д.)

В рамках текущего состояния проекта система может отображать только карты, сгенерированные самостоятельно, но в перспективе можно предусмотреть возможность отображать карты пользователей, причем не только игровые.

2. Архитектурные факторы (цели и ограничения)

2.1. Ключевые заинтересованные лица

При анализе области применения системы были выявлены следующие заинтересованные лица:

Действующее лицо	**Заинтересованность в системе**
Разработчик проекта	Создание высококачественного программного обеспечения, которое соответствует требованиям проекта и удовлетворяет потребности пользователей. Создание системы, которая работает быстро и эффективно, не потребляет слишком много ресурсов компьютера и не имеет ошибок или проблем в работе. Простота поддержки и введения новой функциональности.
Пользователь	Простота установки инструмента; Простота использования инструмента; Быстрые запросы генерации с указанными параметрами в инструменте; Просмотр сгенерированной карты.

2.2. Ключевые требования к системе

При анализе заинтересованных лиц были выделены следующие ключевые требования к системе:

- Наличие пользовательского интерфейса программы;
- Генерация карты с учетом указанных параметров карты;
- Сохранение сгенерированной карты в формате изображения;
- Совместимость с операционными системами Windows, macOS и Linux;
- Открытый исходный код;
- Визуализация сгенерированной карты в приложении;
- Возможность перемещения по сгенерированной карте.

2.3. Ключевые ограничения

При анализе заинтересованных лиц было выделено следующее ключевое ограничение системы:

- Так как визуализатор и генератор изначально разрабатывались как один из сервисов большого проекта-игры, то на этапе разработки должна быть обеспечена интеграция с сервисом, отвечающим за основную механику игрового процесса (перемещение по карте нескольких персонажей, сбор ресурсов и т. д.).

3. Общее архитектурное решение

Состояние ДО

Данная система (MapVisualizator) была спроектирована по принципам монолитной системы, согласно которой приложение работает, как цельный модуль. Это обеспечивает простоту развертывания и разработки, а также устраняет необходимость ISC (межсистемного взаимодействия). Тем не менее, такая система подходит только для небольших однопользовательских приложений.

MapVisualizator изначально разработан в соответствии со стилем main и подпрограммы, когда за обработку всего функционала приложения отвечает один процесс, «жонглирующий» предварительно написанными функциями. Так процесс, соответствующий запуску main.py, вызывает класс ExampleApp, обрабатывающий все действия пользователя, а он в свою очередь вызывает методы генерации и визуализации карты.

Визуализация карты происходит при помощи подгрузки изображений (далее – текстур) из корневой директории (далее – RootDirectory). Сохранение изображений происходит в ту же директорию.

Работа пользовательского интерфейса осуществляется с помощью Tkinter GUI. Сбор всей информации от пользователя происходит с помощью соответствующих объектов интерфейса.

Состояние ПОСЛЕ

В результате работы по улучшению проекта архитектура изменилась следующим образом:

1) main.py был разбит на два разных файла: Visualizator.py и Generator.py, каждый из которых способен работать независимо. И взаимодействовать друг с другом, также - независимо. (Генерация – через вызов скрипта Generator, с передачей туда требуемых параметров, подгрузка карт в Visualizator осуществляется через вспомогательные файлы, сохраняемые генератором в results).
2) Корневая директория RootDirectory была декомпозирована на две папки: ResourcesDirectory (директория, в которой хранятся текстуры) и ResultsDirectory (директория, в которую выводятся результаты работы генератора и визуализатора).

3) В проекте были удалены хардкоды и использование устаревших библиотек.

Таким образом, теперь архитектура системы состоит из двух независимых модулей и основного процесса, в котором они взаимодействуют.

3.1. Принципы проектирования

В изначальном виде разработка приложения не подразумевала принципов проектирования.

На этапе внесения предложенных изменений в код все разработчики придерживались принципа: соблюдение стандарта PEP8 для языка Python.

4. Архитектурные представления

4.1. Представление прецедентов

В данной системе реализуются use cases, изображенные на следующей модели прецедентов (рис. 1).

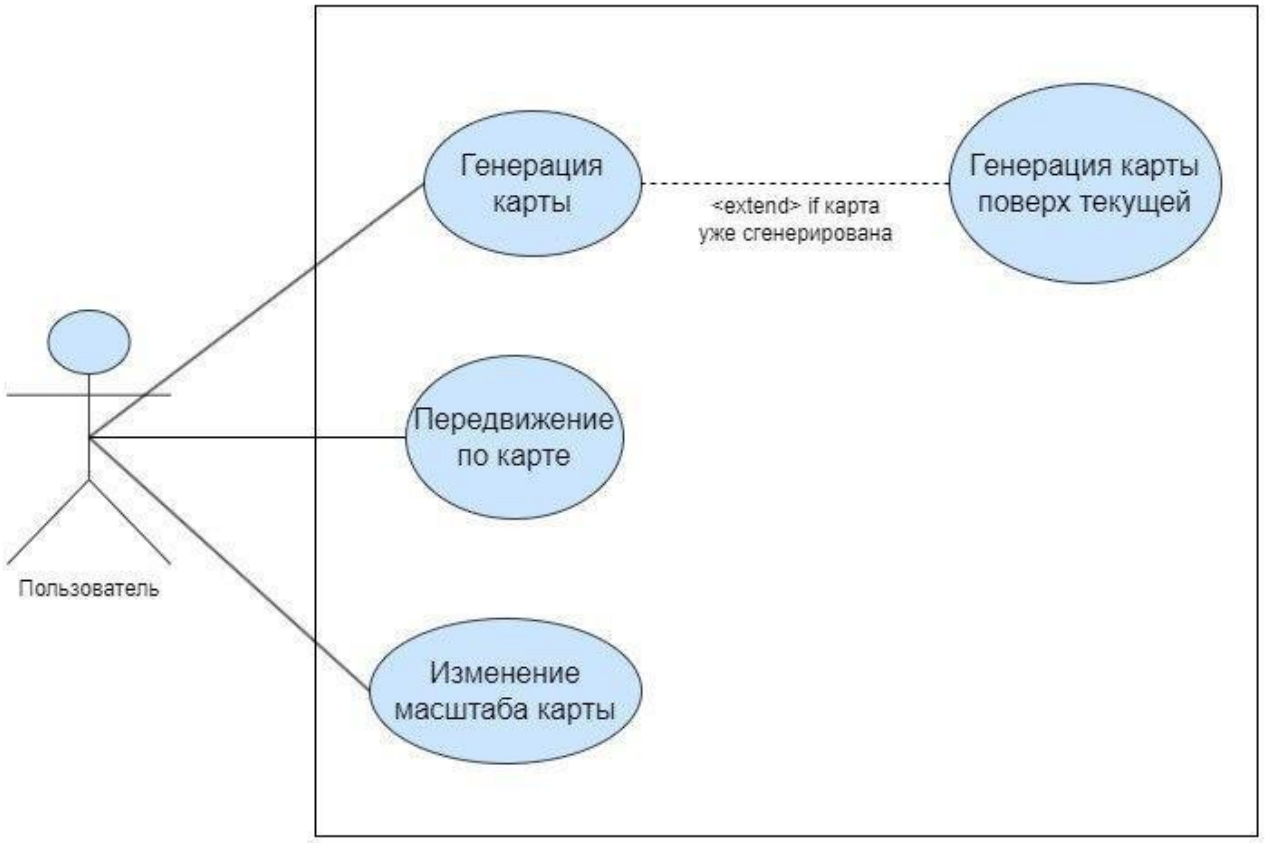

Рис. 1: Модель прецедентов

Внесенные в рамках третьего этапа архитектурные изменения не повлияли на модель прецедентов.

4.2. Логическое представление

Исходное логическое представление архитектуры выглядит следующим образом:

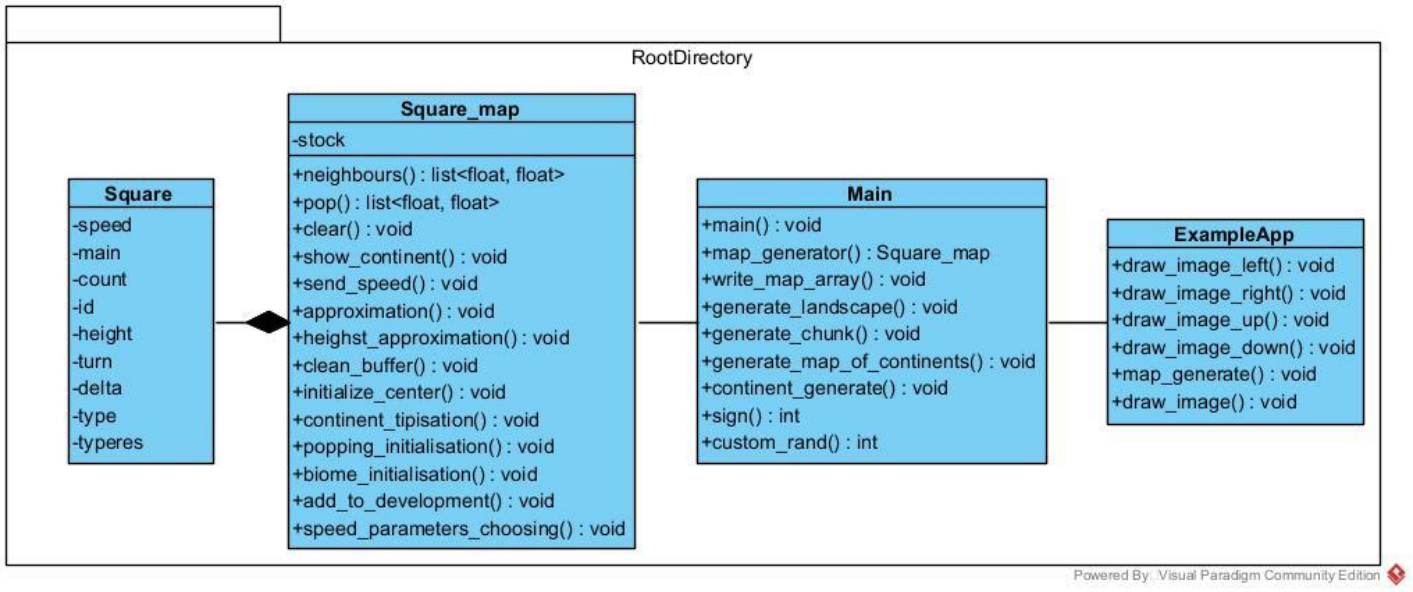

Рис. 2: Диаграмма пакетов до внесения архитектурных изменений

После внесения изменений логическое представление архитектуры MapVisualizator приняло следующий вид:

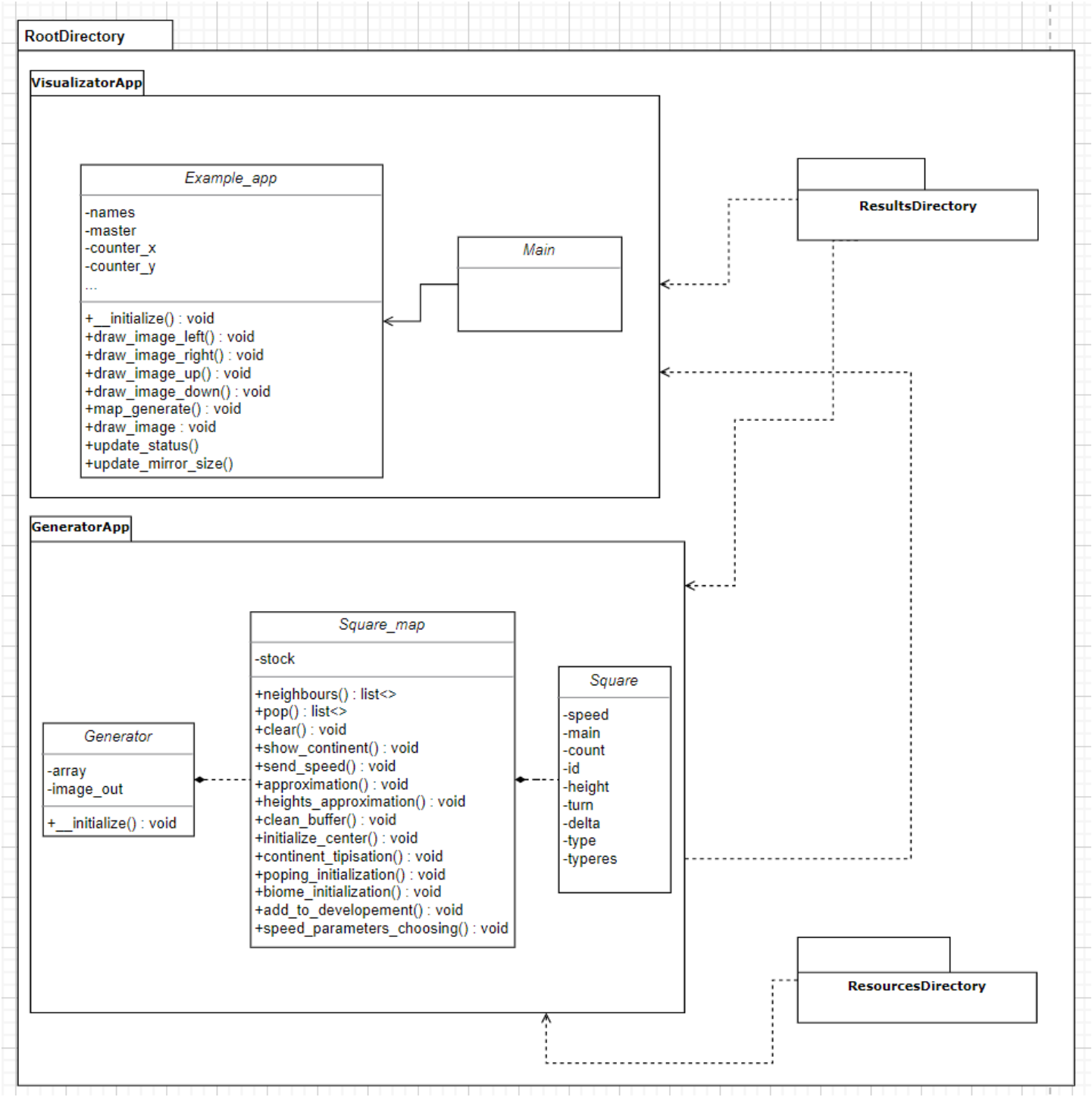

Рис. 3: Диаграмма пакетов после внесения архитектурных изменений

4.3. Представление архитектуры процессов

Изначально был один процесс, который в зависимости от действий пользователя вызывал необходимые собственные методы или методы класса ExampleApp.

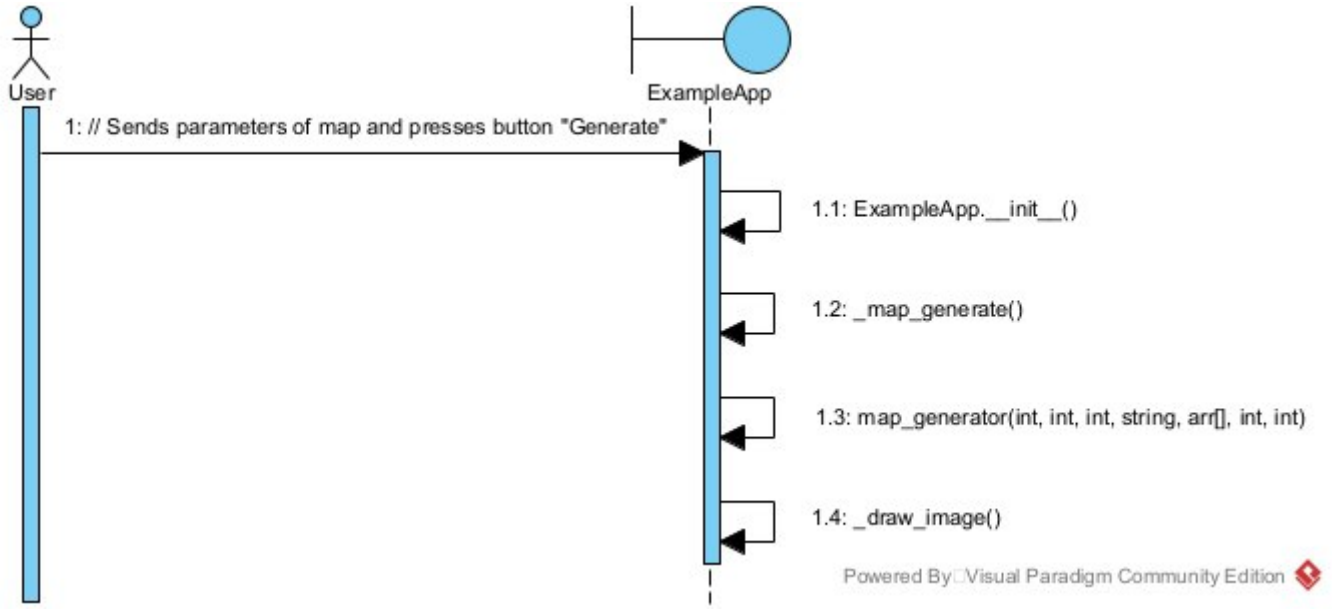

Рис. 4: Диаграмма последовательности для прецедента «Генерация карты»

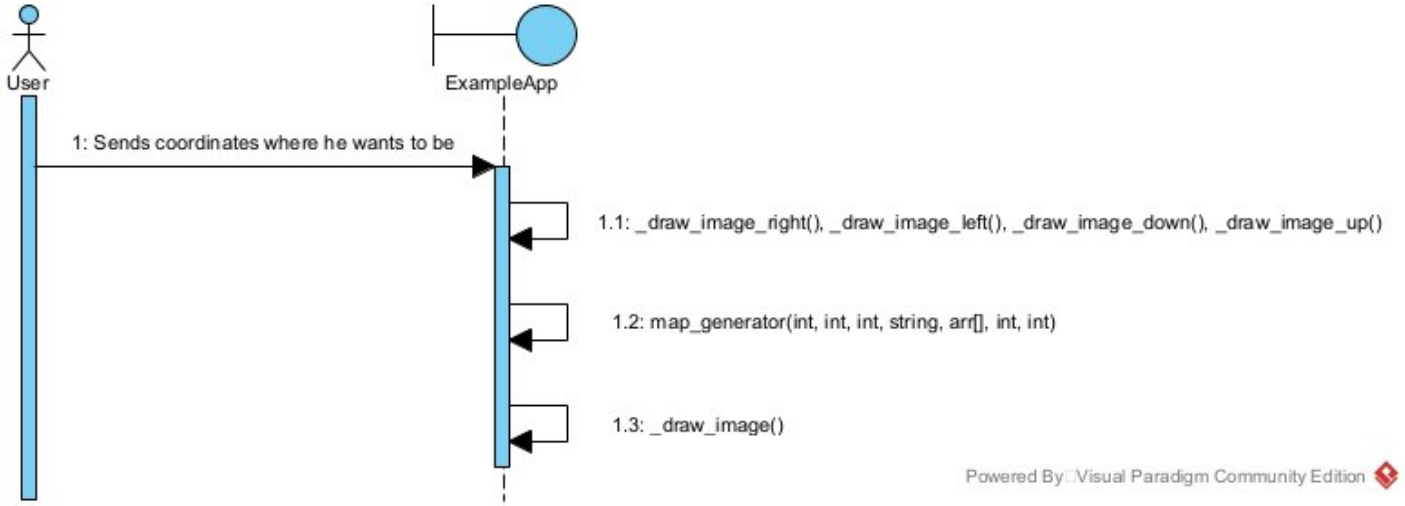

Рис. 5: Диаграмма последовательности для прецедента «Перемещение по карте»

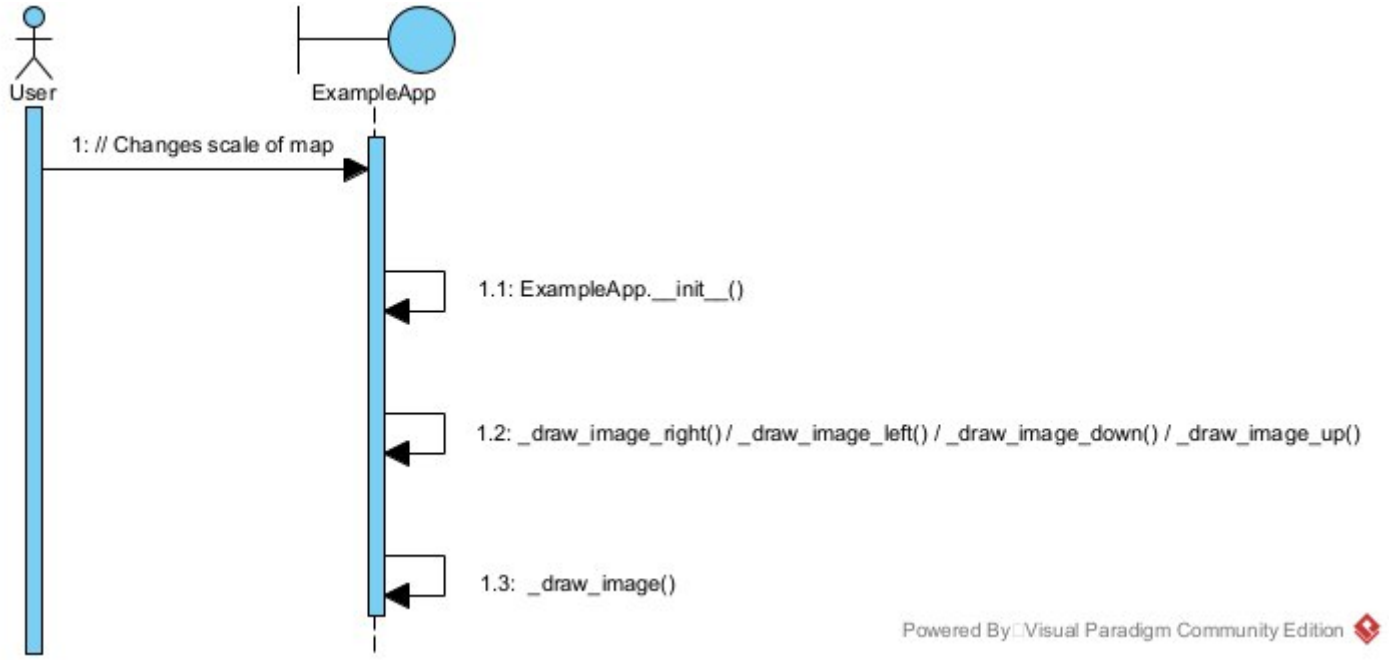

Рис. 6: Диаграмма последовательности для прецедента «Изменение масштаба карты»

После внесения изменений исходный процесс был разбит на три:

1. Исходный процесс, который отслеживает действия пользователя;
2. Процесс, который отвечает за генерацию карты и сохранение ее текстового представления на устройстве пользователя;
3. Процесс, который отвечает за вывод результата (визуализацию в окне пользователя + сохранение png-версии карты на на устройстве пользователя).

Диаграммы последовательности после внесения всех изменений изображены на рисунках 7 - 8.

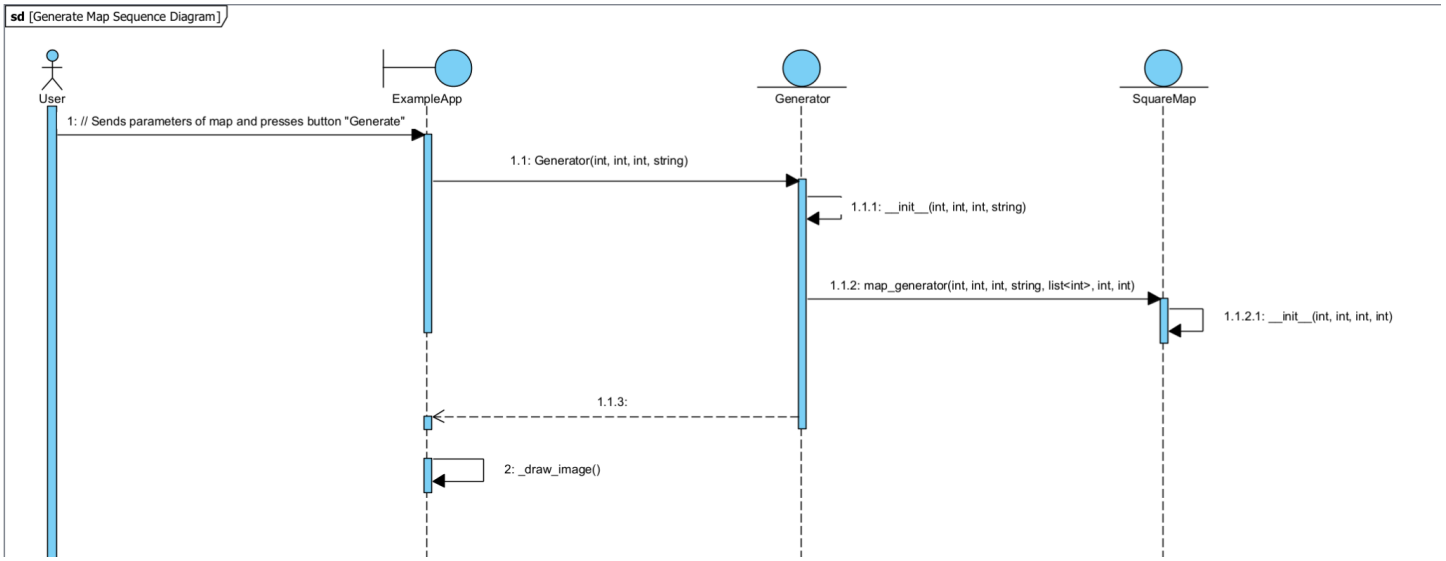

Рис. 7: Диаграмма последовательности для прецедента «Генерация карты»

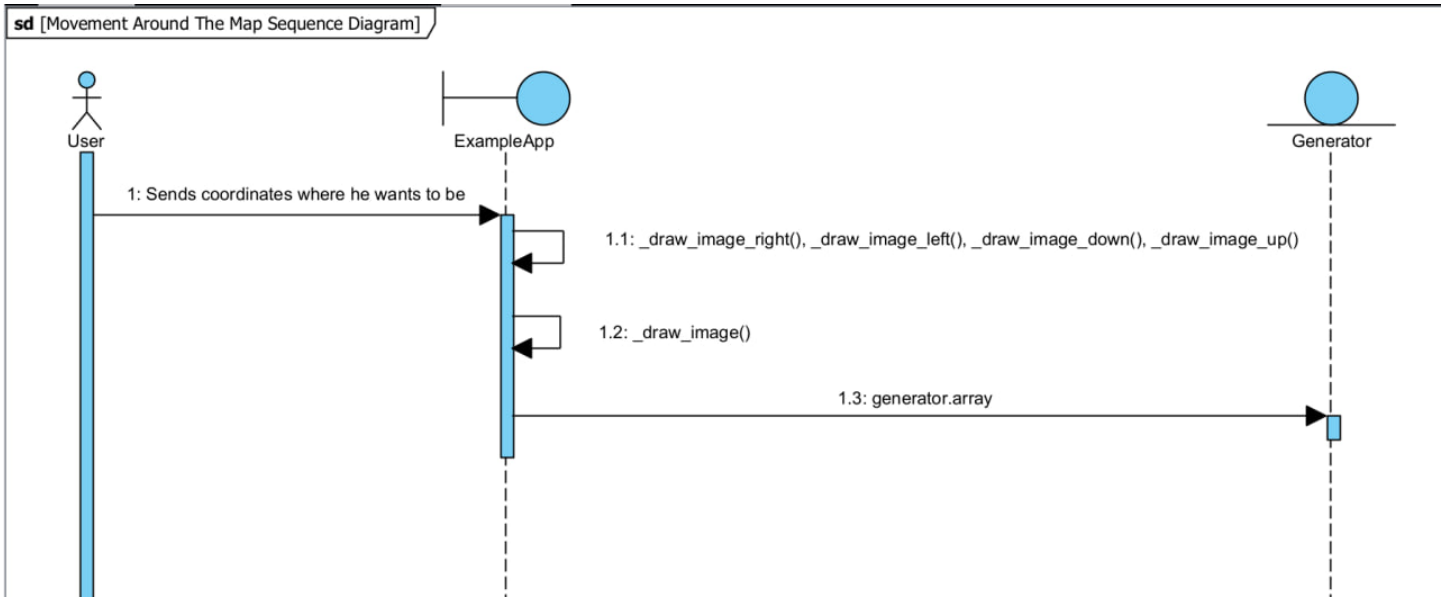

Рис. 8: Диаграмма последовательности для прецедента «Перемещение по карте»

4.4. Физическое представление архитектуры

MapVisualizator – кроссплатформенное desktop-приложение. Оно может быть развернуто на устройствах с одной из следующих операционных систем:

- Windows 7, 8, 8.1, 10
- macOS 10.10.5+
- Ubuntu 18.04+, Debian 10+

Минимальные требования для устройства, на которое может быть установлен MapVisualizator:

- 64-разрядный компьютер с оперативной памятью не менее 2 ГБ;
- Не менее 4 ГБ свободного места на жестком диске.

4.5. Представление развертывания

Система предоставляется в виде набора программных файлов на языке Python.

Существуют различные способы установки MapVisualizator.

- Запуск предварительно скомпилированных файлов;

Программа поставляется со скомпилированным исполняемым файлом для MacOS с архитектурой Intel. Для запуска необходимо дважды кликнуть на исполняемый файл.

- Сборка из исходного кода с использованием среды программирования;

Для запуска программы необходимо предварительно установить среду программирования, поддерживающую язык Python, например, PyCharm.

До внесения изменения программу таким способом можно было запустить с помощью следующих последовательных шагов:

1) Скачать исходный код с Github’а с помощью кнопки «<> Code»;
2) Разархивировать проект;
3) Запустить среду программирования на компьютере;
4) Открыть в ней проект с исходным кодом (Menu => open => “папка с проектом”);
5) Задать конфигурацию запуска (выбрать main.py в качестве параметра Script path и Python 3.10 в качестве Python Interpreter);
6) Запустить код с помощью интерфейса среды (run или сочетание клавиш Shift+F10).

Диаграмма развертывания программы до внесения изменений изображена на рисунке 10.

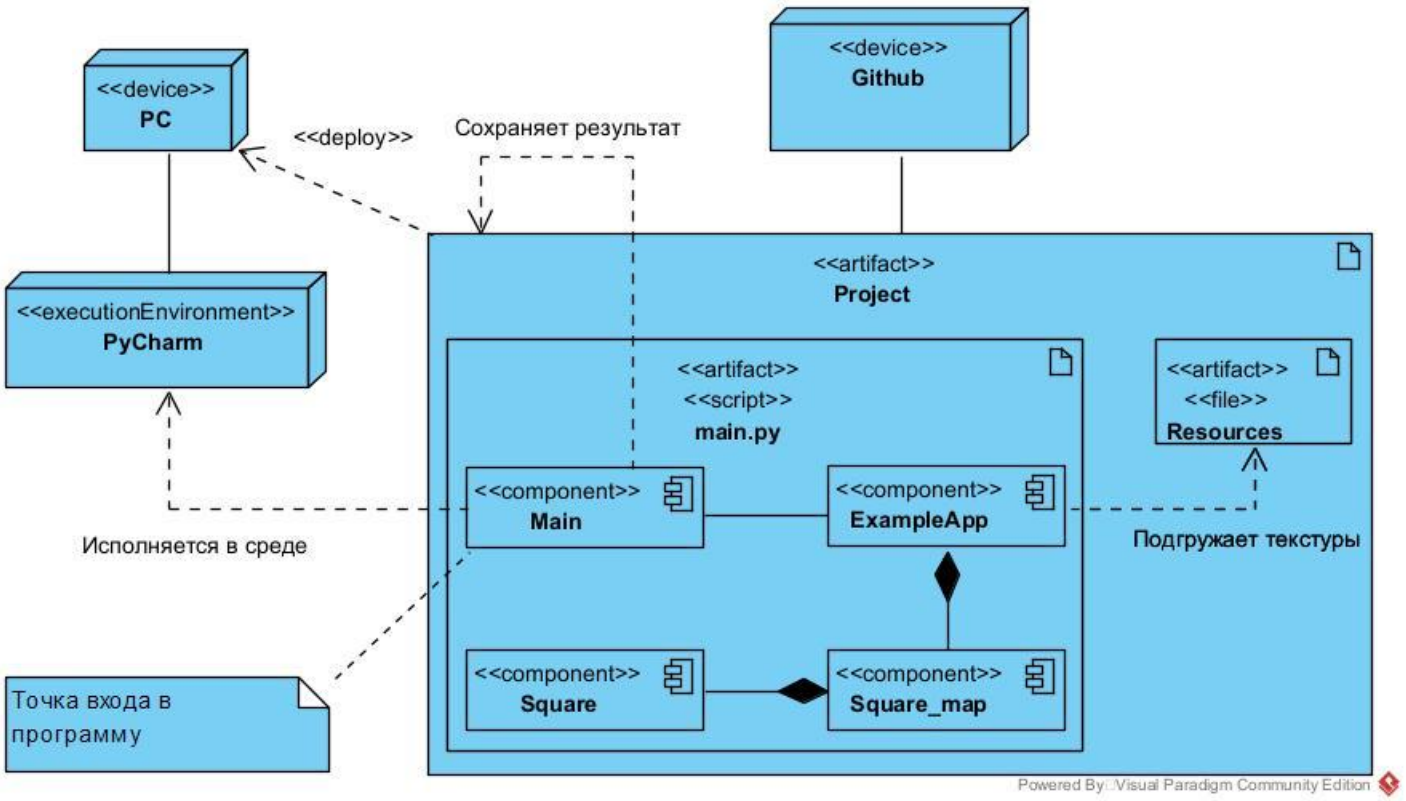

Рис. 10: Deployment диаграмма для способа «Сборка из исходного кода с использованием среды программирования»

После внесения изменений поменялся только шаг 5: в качестве параметра Script path нужно указать файл visualizator.py. Диаграмма развертывания программы после внесения изменений изображена на рисунке 11.

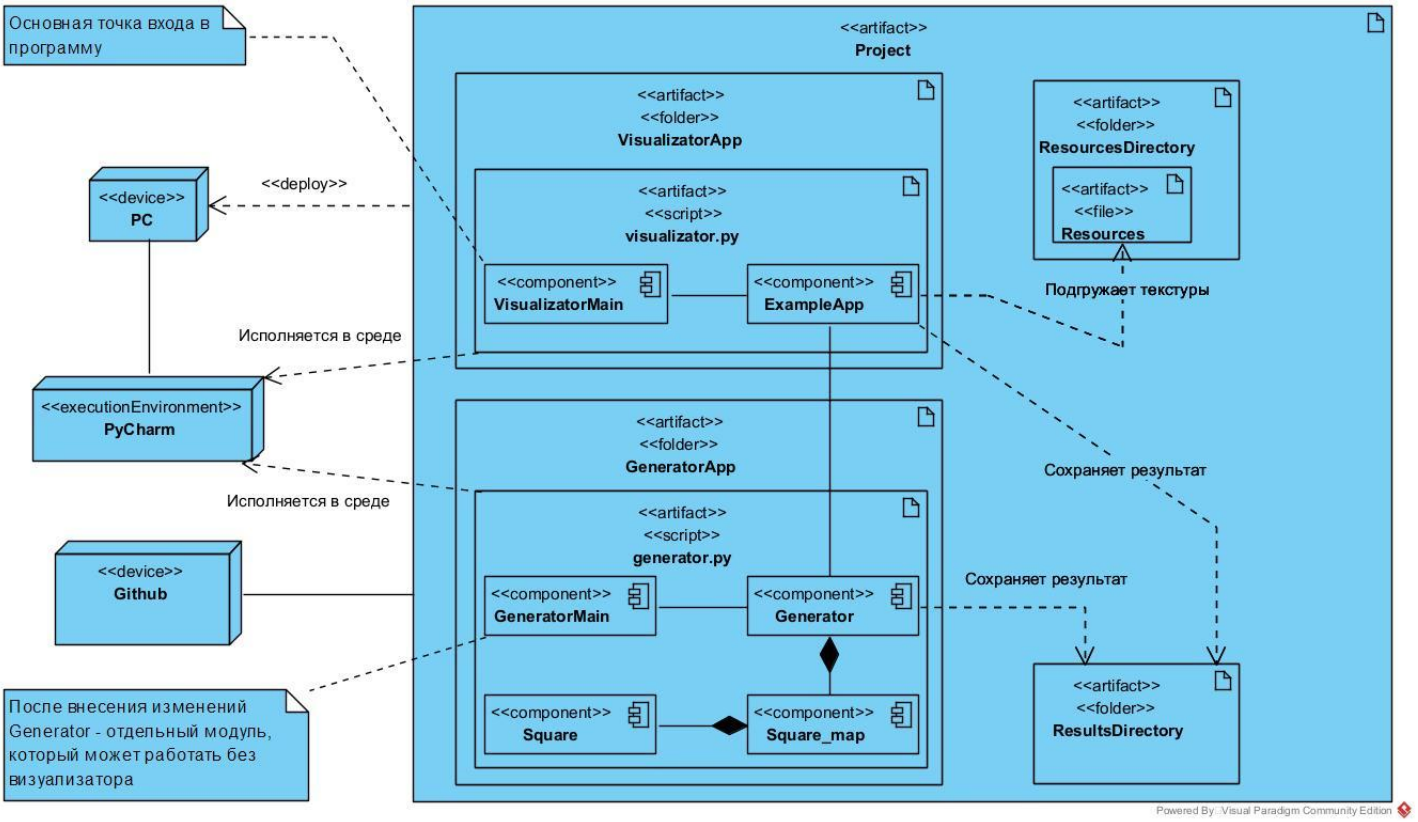

Рис. 11: Deployment диаграмма для способа «Сборка из исходного кода с использованием среды программирования»

- Сборка из исходного кода с помощью командной строки.

Для запуска программы необходимо установить язык python.

До внесения изменения программу таким способом можно было запустить с помощью следующих последовательных шагов:

1. Скачать исходный код с Github’a с помощью кнопки «<> Code»;
2. Разархивировать проект;
3. Открыть командную строку;
4. Ввести команду python “путь к файлу main.py”.

После внесения изменений на этапе П3 точка входа в приложение переместилась, поэтому теперь команда из шага 4 выглядит так: python3 visualizator.py.

При выходе обновлений пользователю нужно будет заново проделать всю процедуру развертывания.

4.6. Представление архитектуры данных

У приложения нет потребности в базе данных, была использована локальная файловая система для работы с файлами, вся работа велась в корневой директории.

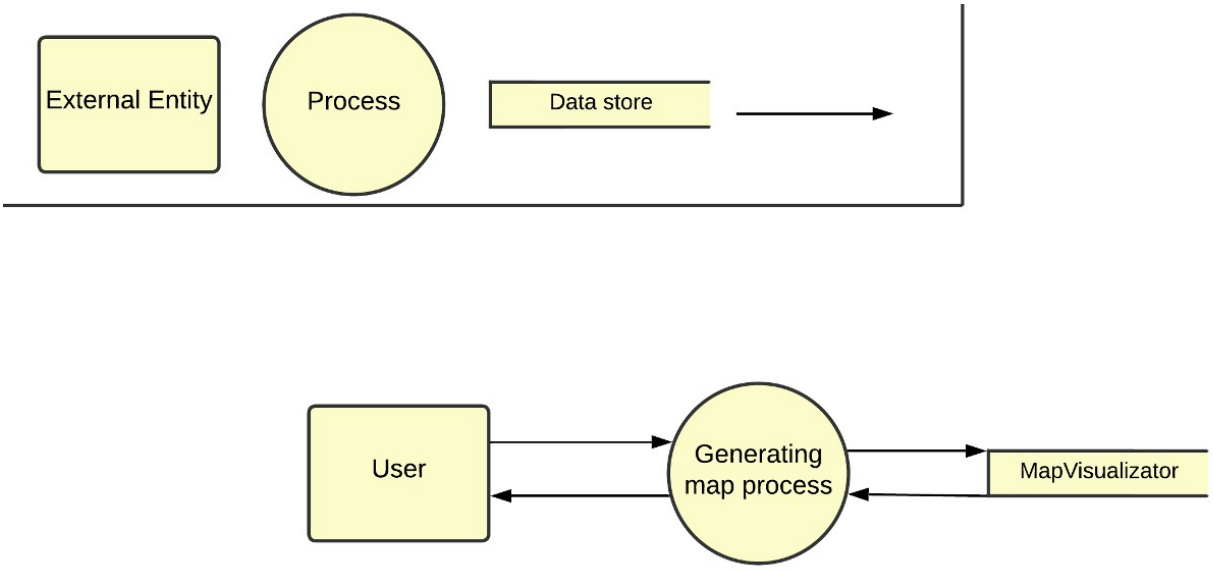

Рис. 13.1: Верхнеуровневая data flow диаграмма для исходного состояния системы

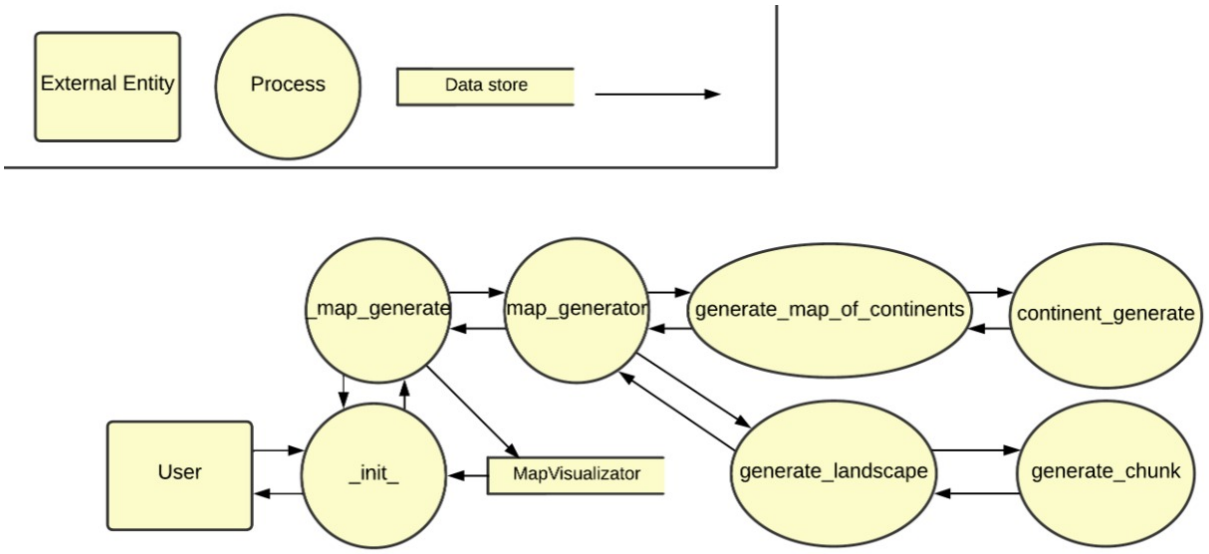

Рис. 13.2: Data flow диаграмма для исходного состояния системы

После внесения изменений корневая директория была декомпозирована на две папки: ResourcesDirectory и ResultsDirectory, описанные в разделе 3.

Диаграмма потоков данных изображена на рисунке 14.

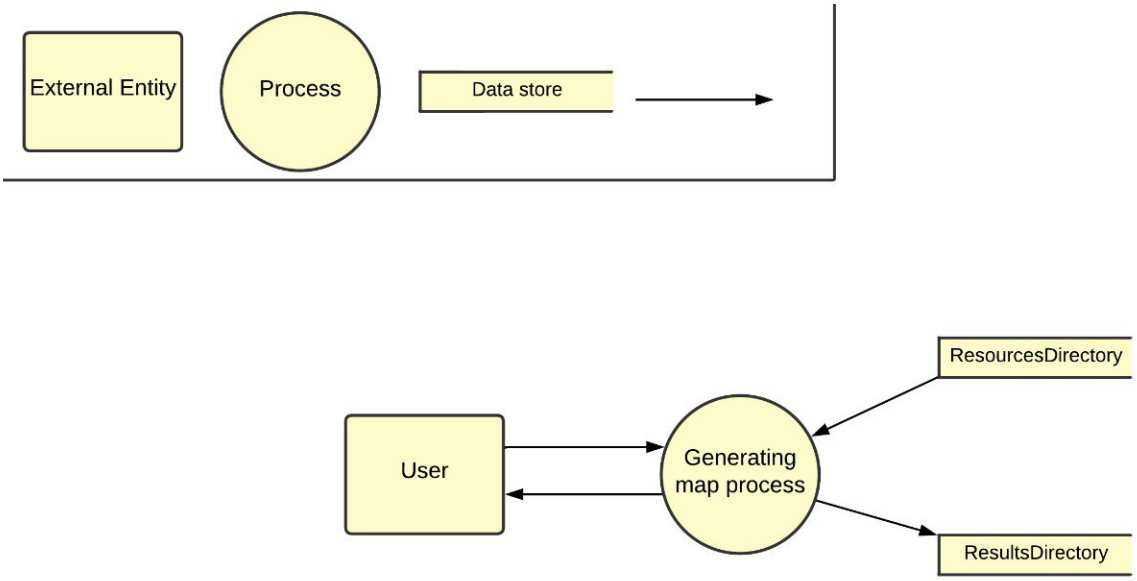

Рис. 14.1: Верхнеуровневая data flow диаграмма для системы после внесения изменений

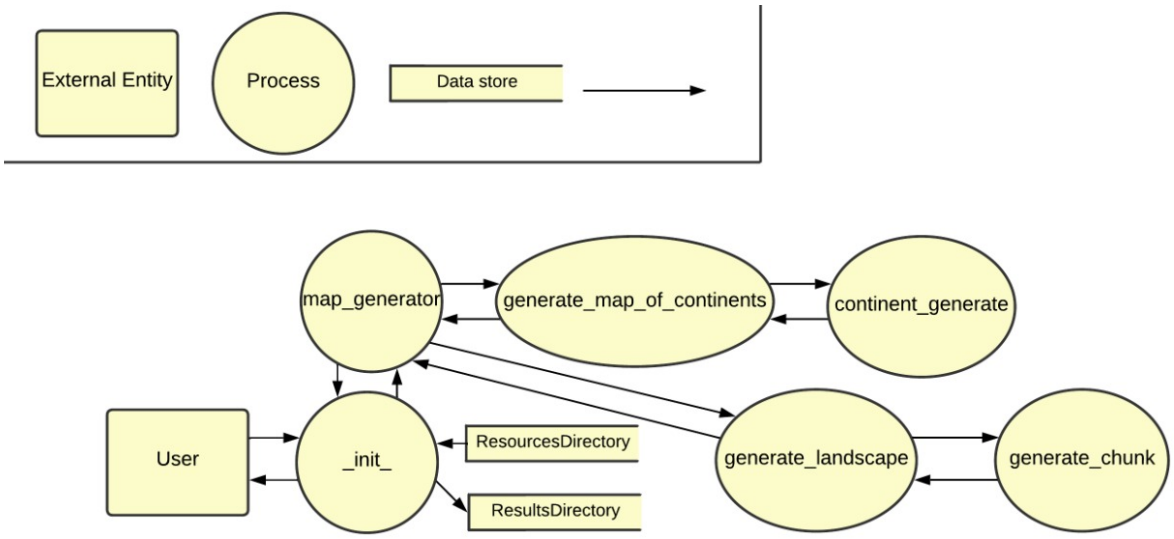

Рис. 14.2: Data flow диаграмма для системы после внесения изменений

4.7. Представление архитектуры безопасности

Программное обеспечение не требует, не запрашивает, не хранит личных данных пользователей, не производит никаких сетевых действий, в связи с чем сохранность обрабатываемых данных зависит исключительно от способа их хранения пользователем.

4.8. Представление реализации и разработки

MapVisualizator написан на языке Python. Работа велась в IDE PyCharm. Код хранится в репозитории на GitHub [1].

При разработке были использованы следующие фреймворки:

- Библиотека Tkinter [2] – используется для создания графических интерфейсов. Он предоставляет набор виджетов, таких как кнопки и поля ввода. Tkinter также обеспечивает возможности работы с графикой, включая отображение изображений (вывод текстур на экран).
- Модули Image и ImageTk из библиотеки Pillow [3] – используются для работы с изображениями в Python. Image предоставляет удобный способ чтения и записи изображений в различных форматах, а также основные функции обработки изображений, такие как изменение размера и обрезка. ImageTk, с другой стороны, используется для отображения изображений на экране.

На данный момент, история изменений такова:

1) 06.03.2023, 22:15:34 – Первичное копирование директории с локального носителя.
2) 06.03.2023, 23:24:25 – Копирование соответствующей версии Python в архив с проектом.
3) 23.04.2023 11:37 – Декомпозиция базы данных
4) 29.04.2023 12:18 – Оптимизация исходного кода программы (избавление от хардкода и устаревших технологий)
5) 29.04.2023 17:54 – Разбиение исходного монолита на два модуля (генератор и визуализатор).

Контроль версий осуществлен при помощи инструментов git.

4.9. Представление производительности

Проект не является объемным, однако операции генерации и визуализации выполняются не быстро при больших размерах карты (больше 400x400). Кроме того, на данный момент, при отрисовке участка карты, обрабатывается всё изображение, что также сказывается на производительности как генератора, так и визуализатора.

При визуализации карты размерами 20x20 время ожидания результата составляет 0.77 секунд.

При визуализации карты размерами 50x50 время ожидания результата составляет 5.43 секунд.

При визуализации карты размерами 100x100 время ожидания результата составляет 22.8 секунд.

При визуализации карты размерами 400x400 время ожидания результата составляет 719.43 секунд.

Для сохранения карты размерами 100x100 необходимо 1.6 мб памяти.

Результаты изменения производительности в результате внесения изменений в проект представлены в разделе 6.3.

4.10. Атрибуты качества системы

Ключевыми атрибутами качества системы являются:

- Производительность: Визуализатор карты должен быть быстрым (время ожидания результата меньше 5 минут для карты размером до 400x400), чтобы игрокам было комфортно перемещаться по карте и выполнять действия в рамках игрового процесса. Для перемещения от одной стороны карты размерами до 100x100 до другой необходимо не более 7 секунд.
- Реалистичность: Визуализатор карты должен создавать картину мира, которая выглядит красиво, а также соответствует ожиданиям игроков. Средняя оценка реалистичности и красоты карт игроками в анонимном независимом опросе должна составлять не менее 7 из 10.
- Адаптивность: Визуализатор должен настраиваться под разные разрешения экранов устройств пользователей. При разработке должны использоваться методы автоматического получения размеров экрана.
- Надежность: Программа должна работать без сбоев минимум в 8 запусках из 10 с генерацией карт размерами до 400x400.
- Разрешение: Визуализатор должен поддерживать высокое разрешение (до 1920x1080) для отображения детализированных карт.
- Кроссплатформенность: Визуализатор карт должен работать на операционных системах, описанных в разделе 4.4 Физическое представление архитектуры.

4.10.1 Объем данных и производительность системы

MapVisualizator работает с пользовательским вводом (размеры карты, количество материков). Генерирует png-файлы размерами до 2 мб для карт размерами до 100x100. Время генерации файла варьируется в зависимости от размера карты, но может достигать нескольких минут (больше 4-5 минут) при размерах карты больших 400x400.

Объем данных ограничен размером локального хранилища данных на устройстве пользователя.

В качестве метрик производительности системы используются время перемещения от одной стороны до другой, время визуализации карты, размер файла формата .png, в котором сохраняется карта.

4.10.2 Гарантии качества работы системы

Тестирование является одной из гарантий качества системы. Тесты должны покрывать все возможные сценарии работы.

Если у пользователя возникает ошибка, он может написать на github в раздел issues или на почту разработчиков.

5. Технические описания отдельных ключевых архитектурных решений

5.1. Техническое описание решения №1: Хранение данных карты

5.1.1 Проблема

Как должно быть обеспечено хранение данных карты?

5.1.2 Идея решения

Следует хранить карту высот и расположение объектов в массиве. Остальные метаданные (текстуры, png сгенерированных карт и т.д.) будут храниться в файловой системе пользователя.

5.1.3 Факторы

Быстрая отрисовка является ключевым требованием к визуализатору карт.

5.1.4 Решение

Текстуры и карты высот, к которым нужен быстрый доступ, будут храниться в массивах, остальные в файловой системе.

5.1.5 Мотивировка

Такая система хранения удобна с точки зрения быстрого доступа к значениям. Карта высот – структура с большим количеством значений (100x100 – минимальный размер карты). Ко всем значениям нужен постоянный доступ для корректного перемещения персонажей. Расположение объектов (ресурсов, бонусов, персонажей и т.д.) также будет храниться в массиве в связи с высокой динамикой изменения координат объектов.

5.1.6 Неразрешенные вопросы

Нет.

5.1.7 Альтернативы

Карту высот и расположение объектов можно было бы также хранить в базе данных, но этот вариант не подходит, поскольку в данном случае снизится скорость отрисовки.

5.2. Техническое описание решения №2: Визуализация сгенерированных карт

5.2.1 Проблема

Стоит ли вынести части кода, отвечающие за визуализацию, в отдельный модуль?

5.2.2 Идея решения

Следует вынести в отдельный модуль.

5.2.3 Факторы

Требования технического задания предусматривают возможность дальнейшего масштабирования инструмента.

5.2.4 Решение

Компоненты визуализации будут вынесены в отдельный модуль.

5.2.5 Мотивировка

Отделение модуля генерации карты от модуля визуализации имеет несколько преимуществ:

1) Расширяемость. При такой архитектуре легче добавлять новые функции.
2) Повышение качества тестируемости. Легче проводить юнит-тестирование.

5.2.6 Неразрешенные вопросы

Нет.

5.2.7 Альтернативы

Альтернативные решения не рассматривались.

6. Оценка результативности предложенных изменений

В ходе разработки предложений по доработке системы мы выявили следующие критерии результативности:

- Сравнение модульности проекта до и после переработки.
- Успешный результат тестирования программы.
- Изменение производительности и других характеристик качества.

При доработке проекта, команда выполняла следующие задачи:

Доржиев Донир Саянович (БПИ203):

- Оценка производительности и других характеристик качества системы после внесения изменений в режиме визуализации карты.
- Итоговое тестирование приложения для проверки корректности работы.

Кондаков Семен Васильевич (БПИ201):

- Разбиение main.py на два класса: Visualizator.py и Generator.py. Настройка вызовов Generator.py визуализатором, и подгрузка в Vizualizator.py сгенерированных in-time изображений.

Насыхова Анастасия Артемовна (БПИ201):

- Оптимизация программы, устранение хардкода.
- Оценка производительности и других характеристик качества системы до внесения изменений в режиме в режиме визуализации карты.

Неймышева Юлия Петровна (БПИ201):

- Создание папок ResourcesDirectory и ResultsDirectory, перемещение в первую спрайты клеток (текстур), а во вторую – сохраненные карты. Фиксация внутри исполняемых файлов изменения директорий.
- Итоговое тестирование приложения для проверки корректности работы.

Внесенные изменения фиксировались в архитектурном документе каждым членом команды в зависимости от этапа работы.

6.1. Сравнение модульности проекта до и после переработки

В результате внесения изменений система была разбита на 2 модуля: генератор и визуализатор. См. логическое представление в разделе 4.2.

6.2. Успешный результат тестирования программы

В результате после внесения изменений в проект программа запускается, выполняет все задачи, не выдает ошибки, корректно обрабатывает некорректные действия пользователя.

6.3. Изменение производительности и других характеристик качества

Рассмотрим влияние внесенных изменений отдельно по каждому атрибуту качества.

- Производительность.

В таблице №1 представлено изменение основных метрик производительности системы: время перемещения от одной стороны до другой, время визуализации карты, размер файла формата .png, в котором сохраняется карта.

В таблице №1 приведены усредненные показатели в результате 15 запусков, количество континентов во всех тестах = 1.

Таблица №1: Изменение основных метрик производительности

Тест	**До**	**После**	**Разница**
Перемещение от одной стороны карты размерами до 100x100 до другой	5.8 секунд	5.8 секунд	0
Визуализация карты размерами 20x20	0.77 секунд	0.72 секунд	-6.94%
Визуализация карты размерами 50x50	5.43 секунд	5.22 секунд	-4.02%

Визуализация карты размерами 100x100	22.8 секунд	22.1 секунд	-3.17%
Визуализация карты размерами 400x400	719.43 секунд	716.17 секунд	-0.46%
Сохранение карты размерами 100x100.	1.6 МБ	1.6 МБ	0

Все тесты были произведены на компьютере с характеристиками, представленными в таблице №2. Тестирование происходило с помощью библиотеки time. В ветке test на Github можно найти zip файлы: Before (проект до изменений) и After (после изменений), в которых оставлен код, с помощью которого был произведен замер времени. Замер времени производился непосредственно в файлах main.py для исходной версии и visualizator.py для новой.

Таблица №2: Характеристики компьютера для тестирования системы

Операционная система	Windows 11
Процессор	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
Оперативная память	16,0 ГБ
Тип системы	64-разрядная операционная система, процессор x64

Как можно заметить из таблицы внесенные изменения повлияли в значительной мере на время визуализации карты, другие параметры не изменились, что собственно логично, ведь основная логика перемещения по карте и ее генерации не менялась.

Можно заметить, что чем больше размер карты, тем меньше заметна разница между значениями. Это можно объяснить тем, что декомпозиция кода позволяет снизить накладные расходы, но все же основная логика приложения от этого не меняется.

- Реалистичность.

В рамках предложенных улучшений не предусматривалась работа по изменению реалистичности карты (не была произведена работа по изменению текстур карт и логики кода для генерации).

- Адаптивность

В результате внесения изменений был заменен метод получающий размеры экрана на более современный из библиотеки Tkinter. Результат этого изменения отражается на времени ожидания появления экрана.

До внесения изменений среднее время ожидания результата составляет 1.088 секунд.

После внесения изменений среднее время ожидания результата составляет 0.35 секунд.

Тестирование также производилось на компьютере с характеристиками, представленными в таблице №2.

Согласно проведенному тестированию, в результате внесения изменений в код программы, удалось значительно снизить время ожидания появления экрана (метрика улучшилась примерно в 3 раза). Избавление от хардкода и использования устаревших библиотек позволило сократить время ожидания появления экрана, что положительно отразится на пользовательском опыте в целом.

- Надежность.

  Внесенные изменения не влияют на надежность системы.
- Разрешение.

  Внесенные изменения не влияют на разрешение.
- Кроссплатформенность.

  Внесенные изменения не влияют на кроссплатформенность.

6.4. Проверка работоспособности предложенных изменений

Были написаны тесты, позволяющие оценить производительность системы. Их можно найти в ветке на репозитории в github. За счёт них получилось проверить, что система после внесённых изменений, продолжает работать корректно.

Подведение итогов:

В результате работы по улучшению проекта архитектура изменилась следующим образом:

1. main.py был разбит на два разных файла: Visualizator.py и Generator.py;

2. Корневая директория RootDirectory была декомпозирована на две папки;
3. В проекте были удалены хардкоды и использование устаревших библиотек.

Проделанная работа позволила добиться улучшения ключевой метрики производительности - времени ожидания результатов визуализации. Несмотря на то, что производительность кода при больших размерах карты (больше 100x100) улучшается незначительно, декомпозиции все же по-настоящему требовалась проекту и в будущем она принесет еще немало плодов. Так, например, будет проще масштабировать систему, отлаживать её, вводить новую функциональность и т. д.

Декомпозиция корневой директории значительно повысила аккуратность кода.

Избавление от устаревших библиотек и оптимизация кода позволила еще немного сократить сопутствующие расходы по времени, создавая окно приложения быстрее. Ускорение работы приложения повысит его удобство и улучшит впечатление от использования.

Как команда разработчиков мы искренне довольны проделанной работой.

Мы молодцы!

7. Предложения по дальнейшему улучшению

В рамках дальнейших улучшений проекта мы предлагаем:

- перестроить класс ExampleApp таким образом, чтобы он следовал шаблону MVC;
- упростить развертывание;
- доработать модуль Visualizator так, чтобы он мог визуализировать уже ранее сгенерированные карты.

7.1. Предложение №1

Предлагается разделить ExampleApp на два класса: Example_View (представление карты), и на Example_App_Controller (контроллер, отвечающий за логику и работу с интерфейсом), тем самым проведя декомпозицию класса.

В данный момент в классе ExampleApp находятся такие методы, как: _draw_image_left, _draw_image_right, _draw_image_up, _draw_image_down, __init__, _map_generate, _draw_image. Планируется все эти методы вынести в Example_App_controller, оставив под Example_View только инициализацию окна, и вызов методов Controller при соответствующей интеракции с пользователем.

7.2. Предложение №2

Переработать развёртывание так, чтобы программа собиралась не из исходных файлов. Сделать это планируется, создав отдельное приложение-установщик, отвечающее за автоматическое обновление заранее скачанного приложения, а также - за удобную установку данного приложения. Выполняться она будет за счёт, предварительной автоматической установки всех библиотек, которые используются в данном приложении.

7.3. Предложение №3

Предлагается ввести режим работы “Проверка качества”, в котором будет подгружаться уже ранее сгенерированное изображение из файловой системы, и по нему с помощью такого же функционала, будет выполняться навигация.

Также можно усовершенствовать визуализатор, добавив метод “upload_map”, позволяющий выбрать произвольное изображение, и подгружать его вместо генерируемого .png файла карты.

8. Приложения

8.1. Словарь терминов

GitHub – это онлайн-платформа, которая позволяет разработчикам и командам разработчиков хранить, управлять и совместно работать над своими проектами. Она специализируется на системе контроля версий Git, что позволяет упростить процесс разработки программного обеспечения и обеспечить более эффективную командную работу.

PyCharm – это интегрированная среда разработки (IDE) для языка программирования Python. Она предназначена для упрощения процесса создания, отладки и тестирования программ на языке Python.

База данных — совокупность всех объектов БД (таблиц, процедур, триггеров и т.д.), статических данных (неизменяемых данных, хранящихся в lookup-таблицах) и пользовательских данных (которые изменяются в процессе работы с приложением).

Визуализатор карты – программа, которая выводит на экран изображение игрового мира.

Высотная карта – это тип карты, на которой показаны элементы, которые влияют на рельеф земной поверхности, такие как горы, долины, холмы и др. Эта карта используется в визуализаторе карт для создания трехмерных моделей ландшафта и других форм земной поверхности.

Генерация карты – это процесс создания игрового уровня или мира с помощью алгоритмов и случайных генераторов. Она может создавать различные типы карт, такие как поля, леса, пустыни, горы и т.д., а также управлять размещением объектов игрового мира.

Инструмент - это программа, приложение или утилита, которая используется для ускорения и упрощения разработки, тестирования, отладки или улучшения качества программного обеспечения.

Карта – игровое пространство, на котором происходят события игры.

Модуль – это самодостаточная часть программы, которая содержит определенную функциональность и может быть использована другими частями программы.

Текстура – графическое изображение, которое накладывается на поверхности мира.

Юнит-тестирование – это методология тестирования программного обеспечения, при которой отдельные единицы программного кода (юниты) тестируются на предмет соответствия спецификации. Целью юнит-тестирования является проверка корректности работы кода, установка ожидаемых результатов и обнаружение возможных ошибок на ранних этапах разработки.

8.2. Ссылки на используемые документы

[1] GitHub репозиторий - https://github.com/seemur1/MapVisualizator

[2] Tkinter GUI - https://docs.python.org/3/library/tkinter.html

[3] Pillow Library - https://python-pillow.org/