Бюджетное учреждение профессионального образования
Ханты-мансийского автономного округа – Югры
«Мегионский политехнический колледж»

Отчёт о выполнении
Практическая работа №7
По дисциплине
МДК 04.02 Обеспечение качества функционирования компьютерных систем

Проверил:
Преподаватель
Шиян М. М.
Выполнил:
Студент группы ИСП-35
Аллахяров С.С.

г. Мегион
2024г.

Практическая работа №2 Построение архитектуры программного средства.

1. Особенности реализации системы

Игровое приложение объединяет Pygame, Ursina и Tkinter для создания 3D-игры с графическим пользовательским интерфейсом. Ключевые аспекты реализации включают:

- Пользовательский интерфейс (UI): Tkinter используется для создания главного меню и внутриигрового меню для основных элементов управления игрой.
- Игровой движок: основная функциональность игры и 3D-среда созданы с помощью движка Ursina, который поддерживает расширенные функции, такие как освещение, затенение и 3D-модели.
- 3D-рендеринг и управление: Ursina 3D-рендеринг с камерой от третьего лица и персонажем-игроком, который может перемещаться и взаимодействовать с окружающей средой. К разработке Player и ThirdPersonCamera классов применяются базовые принципы объектно-ориентированного программирования.
- Обработка событий: Pygame обрабатывает жизненный цикл основного приложения и входные данные, в частности взаимодействие с меню.

2. Выбор архитектурного стиля

Учитывая структуру и требования приложения, компонентная архитектура является подходящей. Такой подход обеспечивает модульное проектирование с отдельными компонентами для пользовательского интерфейса, игрового движка и игровой логики. Компонентные архитектуры здесь выгодны, поскольку они обеспечивают четкое разделение обязанностей и гибкость при модификации или расширении отдельных компонентов без ущерба для других.

3. Визуальный дизайн архитектуры

Визуальное представление архитектуры представлено на рисунке 1.

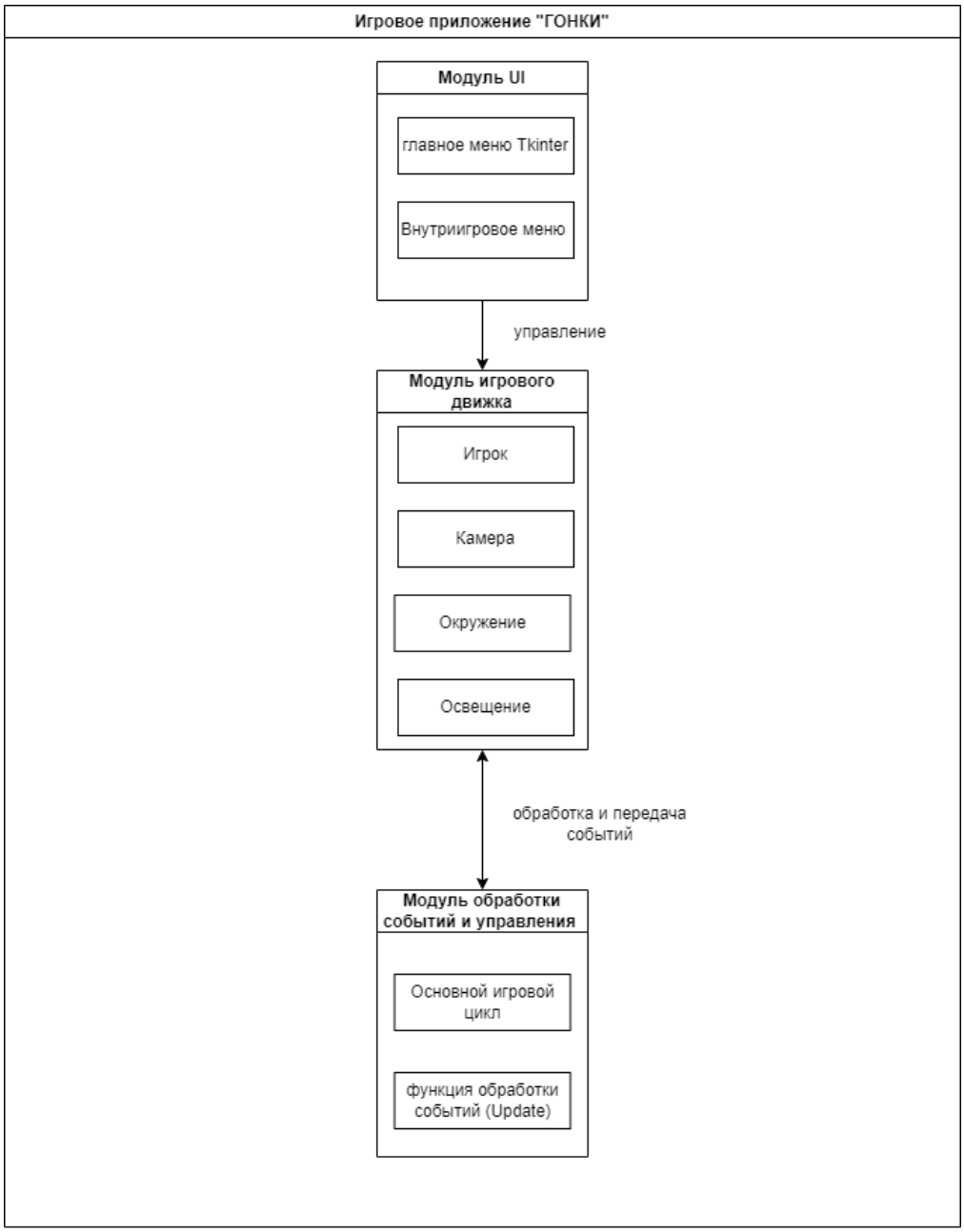


Рисунок 1 – Архитектура приложения

Предлагаемая архитектура состоит из следующих модулей:

1. модуль пользовательского интерфейса (Tkinter /Pygame);
2. модуль игрового движка (Ursina);
3. модуль обработки событий и управления.

4. Текстовое описание архитектуры

Архитектура состоит из трех основных уровней:

- уровень пользовательского интерфейса: этот уровень напрямую взаимодействует с игроком, предоставляя возможность запустить игру, получить доступ к настройкам или выйти из игры. Tkinter управляет компонентами пользовательского интерфейса, предоставляя пользователю меню. Обработка событий в Pygame позволяет уровню пользовательского интерфейса реагировать на действия пользователя, например на нажатие кнопок;
- уровень игрового движка: Ursina обрабатывает 3D-окружение, управляя игровыми объектами и взаимодействием с игроком. Классы Player и ThirdPersonCamera отвечают за перемещение игрока и позиционирование камеры, обеспечивая игровой процесс от третьего лица. Окружение заполнено такими объектами, как деревья, земля и освещение, что обеспечивает динамичный и захватывающий опыт;
- уровень обработки событий и управления: Pygame координирует жизненный цикл игры и реагирует на действия игрока. Он переключает игровые состояния (например, между игрой и паузой), управляет обновлением объектов в игровом движке и синхронизирует пользовательский интерфейс с состоянием игры.

## Контрольные вопросы

1. Что подразумевается под созданием архитектуры приложения?
Создание архитектуры приложения подразумевает процесс проектирования структуры системы, который включает в себя определение компонентов, их взаимодействия, выбор технологий и методов реализации. Архитектура задает основу для разработки, обеспечивает соответствие требованиям и позволяет интегрировать различные модули.

2. Каково основное назначение архитектуры приложения?
Основное назначение архитектуры приложения заключается в обеспечении стабильной и масштабируемой основы для разработки и внедрения программного обеспечения. Она служит каркасом, который определяет, как различные части системы будут взаимодействовать друг с другом, а также обеспечивает соответствие бизнес-требованиям.

3. Перечислите основные принципы разработки архитектуры программного обеспечения.

- Разделение ответственности — каждый компонент должен отвечать за свою часть функциональности.
- Инкапсуляция — скрытие внутренних деталей реализации от других компонентов.
- Модульность — система должна состоять из независимых и взаимозаменяемых модулей.
- Слабая связность — минимизация зависимости между компонентами системы.
- Переиспользуемость — возможность использования компонентов в различных системах.

4. Перечислите основные типовые архитектурные стили.

- Монолитная архитектура
- Клиент-серверная архитектура
- Микросервисная архитектура

- Событийно-ориентированная архитектура
- RESTful архитектура
- Слоевая архитектура
- Архитектура на основе функций

5. Перечислите основные архетипы приложений.

- Веб-приложения
- Мобильные приложения
- Настольные приложения
- Системы управления контентом (CMS)
- Игровые приложения
- Облачные приложения

6. Перечислите основные элементы, которые можно использовать для построения архитектуры программного обеспечения.

- Компоненты (модули, сервисы)
- Интерфейсы (API, контракты)
- Базы данных (системы хранения данных)
- Протоколы (для коммуникации между компонентами)
- Слои (презентация, бизнес-логика, доступ к данным)
- Инфраструктура (серверы, сети, облачные платформы)

7. Что такое системы контроля версий (СКВ) и для решения каких задач они предназначаются?

Системы контроля версий (СКВ) — это инструменты, позволяющие отслеживать изменения в коде и управлять версиями программного обеспечения. Они предназначены для:

- Сохранения истории изменений.
- Совместной работы над проектами.
- Возврата к предыдущим версиям.
- Разрешения конфликтов при параллельной работе.

8. Объясните следующие понятия СКВ и их отношения: хранилище, commit, история, рабочая копия.

• Хранилище — это репозиторий, где хранятся все версии проекта.

• Commit — это действие сохранения изменений в хранилище с описанием сделанных изменений.

• История — это последовательность всех коммитов, показывающая, как изменялся проект со временем.

• Рабочая копия — это локальная версия кода, с которой разработчик работает, вносит изменения и создает коммиты.

9. Что представляют собой и чем отличаются централизованные и децентрализованные СКВ? Приведите примеры СКВ каждого вида.

• Централизованные СКВ — имеют одно общее хранилище, к которому подключаются все разработчики. Примеры: Subversion (SVN), CVS.

• Децентрализованные СКВ — у каждого разработчика есть полная копия репозитория, и они могут работать независимо. Примеры: Git, Mercurial.

10. Опишите действия с СКВ при единоличной работе с хранилищем. При единоличной работе с СКВ разработчик:

• Создает рабочую копию проекта.

• Вносит изменения в код.

• Создает коммиты для сохранения изменений.

• Обновляет локальное хранилище при необходимости, синхронизируя с удаленным хранилищем.

11. Опишите порядок работы с общим хранилищем в централизованной СКВ.

В централизованной СКВ порядок работы следующий:

• Разработчик получает последнюю версию кода из общего хранилища.

• Вносит изменения в рабочую копию.

• Выполняет команду commit, чтобы отправить изменения в общее хранилище.

• Другие разработчики могут обновить свои копии для получения последних изменений.

12. Что такое и зачем может быть нужна разность (diff)? Разность (diff) — это инструмент, который показывает изменения между двумя

версиями файлов или между состоянием одного файла в различных коммитах. Он нужен для:

- Анализа изменений.
- Определения, что конкретно было изменено.
- Поддержки разработки и код-ревью.

13. Что такое и зачем может быть нужно слияние (merge)?
Слияние (merge) — это процесс объединения изменений из одной ветки в другую. Он нужен для:

- Интеграции изменений, выполненных разными разработчиками.
- Объединения новых функций и исправлений в основную ветку проекта.

14. Что такое конфликты (conflict) и каков процесс их разрешения (resolve)?
Конфликты возникают, когда два разработчика вносят изменения в одну и ту же часть кода, и СКВ не может автоматически объединить эти изменения. Процесс разрешения конфликта включает:

- Определение конфликтующих изменений.
- Ручное редактирование кода для выбора, какие изменения сохранить.
- Завершение слияния и создание нового коммита с разрешенными изменениями.

15. Поясните процесс синхронизации с общим хранилищем («обновления») в децентрализованной СКВ.
В децентрализованной СКВ процесс синхронизации включает:

- Получение последних изменений из общего хранилища с помощью команды pull или fetch.
- Обновление локальной копии репозитория с учетом последних коммитов других разработчиков.
- Разрешение возможных конфликтов, если они возникают.

16. Что такое и зачем могут быть нужны ветви (branches)?
Ветви (branches) — это отдельные линии разработки в репозитории, которые позволяют параллельно работать над различными задачами, не мешая основной линии (обычно ветке main или master). Они нужны для:

• Разработки новых функций или исправлений без риска сломать основной код.

• Упрощения работы в команде, позволяя каждому разработчику работать над своей задачей.

17. Объясните смысл действия rebase в СКВ Git. Rebase — это процесс переноса изменений из одной ветки на другую, который позволяет «переписать» историю коммитов. Он нужен для:

• Упрощения истории коммитов, делая её линейной.

• Интеграции изменений из одной ветки в другую более чистым способом, чем слияние.

18. Как и зачем можно игнорировать некоторые файлы при commit? Игнорировать файлы можно с помощью файла .gitignore, в который добавляются шаблоны для файлов и каталогов, которые не должны отслеживаться в репозитории. Это нужно для:

• Исключения временных файлов, кэша или конфиденциальной информации из репозитория.

• Упрощения работы с репозиторием, чтобы не загромождать его ненужными файлами.