Практическая работа №2

Построение архитектуры программного средства

Цели:

1) Изучение базовых принципов и шаблонов построения архитектуры приложений, выбор стратегии и шаблона проектирования, которые помогут при проектировании слоев, компонентов и сервисов решения, а также визуальное проектирование архитектуры приложения с использованием Microsoft Visio.

Особенности реализации системы:

Игровое приложение использует Pygame для создания игры с графическим пользовательским интерфейсом. Ключевые аспекты реализации включают обработку событий: Pygame обрабатывает жизненный цикл основного приложения и входные данные, в частности взаимодействие с меню.

Текстовый архитектурный план игры:

1. Инициализация игры

Модуль: pygame

Функциональность: инициализирует Pygame, настраивает главное окно игры, загружает музыку и устанавливает игровые константы (размер экрана, размер ячейки и т.д.).

2. Константы и переменные

Размеры экрана: WIDTH, HEIGHT

Размер ячейки: CELL_SIZE

Настройки игры:

- FPS: количество кадров в секунду;
- TIME_LIMIT: ограничение времени для игры;
- COIN_COUNT: количество монет, которые нужно собрать.

Определения цветов: определены с помощью RGB-кортежей.

3. Обработка изображений

Функция: load_and_scale_image(filename, size)

Назначение: загружает и изменяет размер изображений для игровых элементов (монет, игрока).

Изображения активов:

- coin_images: список изображений монет;
- player_image: изображение игрока;
- cover_image: фоновое изображение для стартового экрана.

4. Игровые экраны

- draw_start_screen(screen) отображает стартовый экран с названием игры и инструкциями.
- draw_instructions(screen) отображает инструкции для игрока.

5. Игровые объекты

Класс: Player

Атрибуты:

- x, y: позиция на сетке;
- coins_collected: количество собранных монет;
- image: изображение персонажа.

Методы: move(dx, dy, maze): перемещает игрока, если следующая ячейка не является стеной.

6. Генерация лабиринта

- generate_maze(width, height) генерирует лабиринт с использованием алгоритма поиска в глубину.
- generate_coins(maze, exit_pos, coin_images) случайным образом размещает монеты в лабиринте, гарантируя, что они не перекрываются со стенами или выходом.

7. Отрисовка игры

- draw_game_elements(screen, maze, player, coins, exit_pos) отрисовывает лабиринт, монеты, игрока и выход.
- добавление музыкального сопровождения;

- draw_hud(screen, elapsed_time, level, coins_collected) отображает индикатор состояния (HUD), показывающий оставшееся время, текущий уровень и количество собранных монет.
- show_end_screen(screen, levels_completed, coins_collected) отображает конечный экран, когда время истекло.

8. Основной игровой цикл

Функция: main()

Рабочий процесс:

- Инициализация экрана: настройка дисплея, заголовка и часов.
- Отображение стартового экрана: вызов draw_start_screen().
- Страница с инструкциями: вызов draw_instructions().
- Цикл уровней: для каждого уровня:
  - Генерация лабиринта и монет.
  - Запуск таймера и обработка перемещения игрока.
  - Обновление HUD и отрисовка игровых элементов.
  - Проверка условий победы (достижение выхода).

9. Обработка событий

- Обрабатывает события нажатий клавиш для перемещения игрока, начала игры, паузы и выхода.
- Обнаруживает щелчки мыши для кнопки продолжения на экране инструкций.

Инициализация: настройка pygame и загрузка ресурсов.

Пользовательский интерфейс: представление стартового и инструкционного экранов игроку.

Игровой цикл:

- генерация лабиринта и монет;
- обработка ввода игрока и перемещения;
- отрисовка состояния игры (лабиринт, игрок, монеты);
- проверка условий окончания игры (время вышло);

- конец игры: показ результатов и завершение.

Архитектура игры, сделанная через draw.io, представлена на рисунке 1.

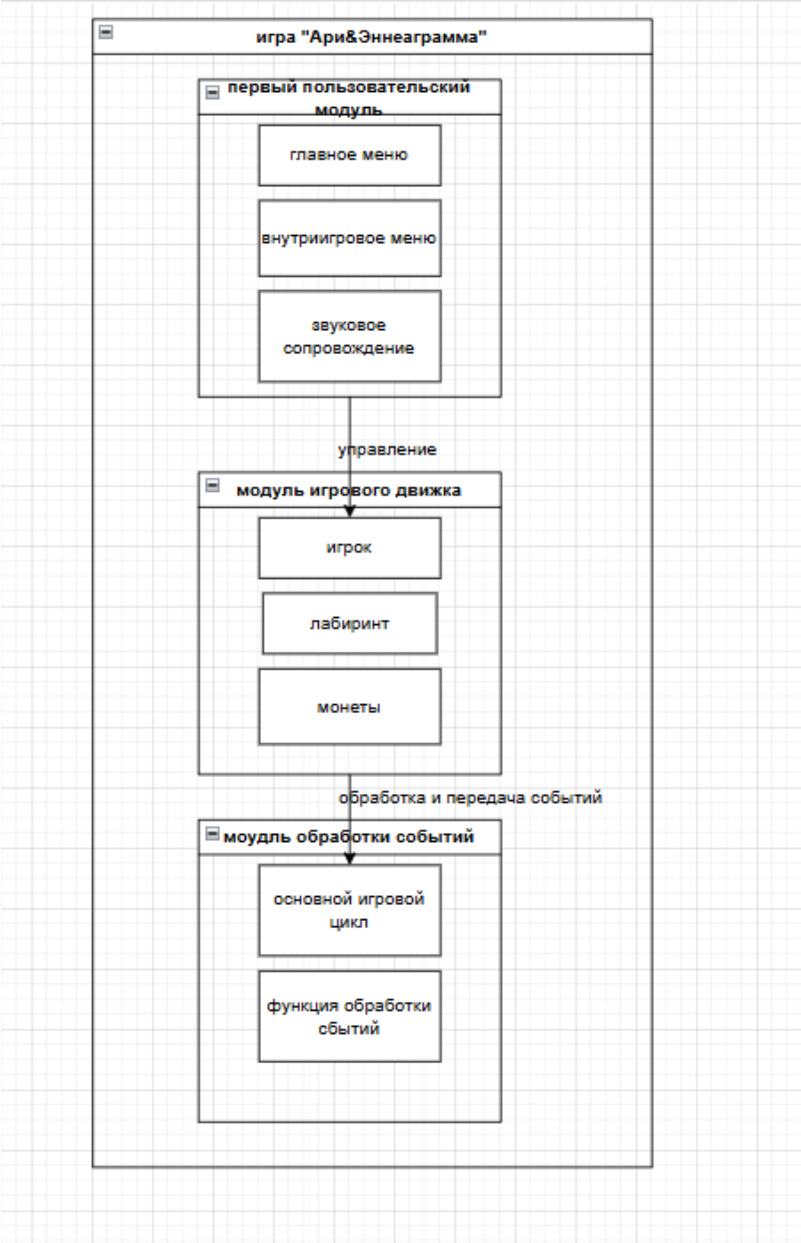

Рисунок 1 - Архитектура

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.