ОТЧЕТ

по практическому занятию №1

«Каркас игрового движка и базовая интеграция рендера»

Студент: Иманаков Владислав Павлович

Группа: J3300

Репозиторий проекта: https://github.com/Vladislavim/Engine_AID

2026 г.

Цель и постановка задачи

Цель работы — создать минимальный рабочий каркас игрового движка с поддержкой игрового цикла и визуализации графического вывода через адаптер рендеринга.

В рамках практического задания требовалось реализовать класс Application, систему состояний игры, расчет deltaTime, систему логирования, интерфейс RenderAdapter и отрисовку тестового примитива в графическом окне. Дополнительно требовалось реализовать обработку ввода с клавиатуры и мыши для управления объектом с учетом deltaTime.

Используемые средства

- Язык программирования: C++17.
- Система сборки: CMake.
- Графическая библиотека: raylib (через FetchContent).
- Среда разработки: Visual Studio 2022.
- Контроль версий: Git.
- Репозиторий проекта: https://github.com/Vladislavim/Engine_AID.

Описание общей архитектуры решения

Проект реализован как минимальный каркас игрового приложения с разделением ответственности между модулями.

Класс Application управляет жизненным циклом программы: выполняет инициализацию, запуск главного цикла, вычисление deltaTime, обновление и отрисовку текущего состояния, переключение состояний и завершение работы.

Для расчета deltaTime используется std::chrono::steady_clock. Значение deltaTime периодически записывается в журнал.

Система состояний построена на базовом интерфейсе GameState. В проекте реализованы состояния LoadingState, MenuState и GameplayState. Каждое состояние содержит методы входа, выхода, обновления и отрисовки, что упрощает организацию логики и дальнейшее расширение проекта.

Рендеринг вынесен в интерфейс IRenderAdapter, который отделяет игровую логику от конкретной графической библиотеки. В рамках работы реализован RaylibRenderAdapter, выполняющий создание окна, обработку ввода, начало и завершение кадра, отрисовку примитива и вывод текста. Такой подход позволяет заменить библиотеку рендеринга без изменения основной архитектуры приложения.

Для логирования реализован отдельный модуль Logger, который записывает сообщения в консоль и в файл app.log. В журнале фиксируются запуск и завершение приложения, переходы между состояниями и значения deltaTime.

Обработка управления вынесена в класс InputController. Реализовано перемещение объекта стрелками клавиатуры, масштабирование по левой кнопке мыши и вращение по правой кнопке мыши. Изменение параметров выполняется плавно с учетом текущего значения deltaTime.

Структура проекта

Основные файлы проекта:

- CMakeLists.txt — конфигурация сборки проекта;
- main.cpp — точка входа в приложение;
- Application.h, Application.cpp — каркас приложения и главный цикл;
- Logger.h, Logger.cpp — система логирования;
- GameState.h — базовый интерфейс состояния;
- States.h, States.cpp — состояния Loading, Menu, Gameplay;

- IRenderAdapter.h — интерфейс адаптера рендера;
- RaylibRenderAdapter.h, RaylibRenderAdapter.cpp — реализация адаптера на raylib;
- InputController.h, InputController.cpp — обработка ввода и управление объектом.

Реализованная функциональность (результат работы)

В результате выполнения задания разработано приложение, которое:

- Открывает графическое окно.
- Работает на основе главного игрового цикла.
- Вычисляет deltaTime и выводит его в журнал.
- Поддерживает переключение состояний Loading, Menu, Gameplay.
- Отрисовывает тестовый примитив (квадрат).
- Позволяет управлять примитивом с клавиатуры и мыши.
- Обеспечивает плавное изменение параметров объекта с учетом deltaTime.

Управление в приложении:

- 1 / 2 / 3 — переключение состояний;
- Enter — переход из Menu в Gameplay;
- Esc — возврат из Gameplay в Menu;
- стрелки — перемещение объекта;
- ЛКМ — масштабирование;
- ПКМ — вращение.

Листинги наиболее значимых частей кода

В отчет рекомендуется вставить фрагменты следующих файлов:

- IRenderAdapter.h — интерфейс адаптера рендера.
- Application.cpp — главный цикл и расчет deltaTime.
- GameState.h, States.cpp — система состояний.
- InputController.cpp — обработка ввода с учетом deltaTime.

- RaylibRenderAdapter.cpp — реализация адаптера рендера.

Примечание: листинги вставляются в отчет после финальной сдачи кода (в виде фрагментов с подсветкой или моноширинным шрифтом).

Скриншоты рабочего приложения

В отчет необходимо вставить скриншоты рабочего приложения. Рекомендуемые скриншоты:

- Состояние Menu.
- Состояние Gameplay с тестовым примитивом.
- Демонстрация работы управления (перемещение/масштаб/поворот).
- Консоль с логами (deltaTime, переходы между состояниями).

Скриншоты в репозитории: menu.jpg, gameplay.jpg.

Пример подписей:

- Рисунок 1 — Состояние меню приложения.
- Рисунок 2 — Состояние игрового режима с тестовым примитивом.

Проверка работоспособности

В ходе тестирования были проверены:

- Запуск приложения и создание окна.
- Переключение между состояниями (1, 2, 3, Enter, Esc).
- Отрисовка примитива в состоянии Gameplay.
- Перемещение объекта стрелками.
- Масштабирование по ЛКМ.
- Вращение по ПКМ.
- Вывод значений deltaTime в журнал.

Результат: функциональность работает корректно, управление объектом визуально плавное, требования задания выполнены.

Выводы

В результате выполнения практического занятия реализован минимальный каркас игрового движка с базовой архитектурой и возможностью дальнейшего расширения.

Выполнены основные требования задания: класс Application; игровой цикл; расчет deltaTime; система состояний; логирование; RenderAdapter; отрисовка тестового примитива.

Дополнительно реализована обработка ввода с клавиатуры и мыши с учетом deltaTime.

Полученная архитектура позволяет в дальнейшем расширять проект (добавлять новые состояния, заменять рендерер, усложнять игровую логику).

Приложения к отчету

Репозиторий с исходным кодом: https://github.com/Vladislavim/Engine_AID

Конфигурационные файлы: CMakeLists.txt, .gitignore

Документация в репозитории: ARCHITECTURE.md, REPORT_NOTES.md, CONTROL_QUESTIONS.md

Скриншоты: menu.jpg, gameplay.jpg

Контрольные вопросы (краткие ответы)

Чем отличается фиксированный и переменный шаги игрового цикла?

Фиксированный шаг использует постоянный интервал обновления (например, 1/60 секунды), что удобно для физики и повторяемости поведения. Переменный шаг использует фактическое время между кадрами (deltaTime), что проще для базовой реализации, но требует ограничения слишком больших значений времени.

Зачем необходим RenderAdapter и можно ли обходиться без него?

RenderAdapter отделяет игровую логику от конкретной графической библиотеки. Без него можно обойтись в небольшом проекте, но код становится сильнее связан с выбранным API и хуже расширяется.

Какова роль deltaTime в управлении физическими процессами и анимациями?

deltaTime позволяет изменять параметры объектов пропорционально времени, а не числу кадров. Это делает движение и анимацию более стабильными при разном FPS.

Какие преимущества дает использование системы состояний в играх?

Система состояний разделяет поведение приложения по режимам (загрузка, меню, игра), уменьшает количество условных ветвлений и упрощает сопровождение кода.

Как технически организовать смену графического API («горячую загрузку»)?

Необходимо работать через абстракцию (например, IRenderAdapter) и отделить игровые данные от графических ресурсов. При смене API создается новый адаптер и выполняется повторная инициализация графических ресурсов, при этом логика игры продолжает работать через общий интерфейс.