Национальный исследовательский ядерный университет
«МИФИ»

Кафедра «Программная инженерия»

Техническая документация к игре «Echo Step»

Версия: 1.0
Дата: 13.11.2025

Москва 2025

Содержание

1 Введение

1.1 Общее описание

Данная техническая документация описывает архитектуру, алгоритмы и особенности реализации игры *Echo Step*. Игра представляет собой 2D-платформер с механикой временных копий (эхо), разработанный с использованием технологии вайбкодинга в WebSim.

1.2 Цели документации

Основные цели документации:

- Описание архитектуры системы и взаимодействия компонентов
- Детализация физических моделей и математических алгоритмов
- Спецификация форматов данных и структур уровней
- Описание алгоритмов работы эхо и триггерных систем
- Документирование отладочных инструментов и процедур тестирования

1.3 Область применения

Документация предназначена для:

- Разработчиков, поддерживающих и расширяющих функционал игры
- Тестировщиков, проверяющих корректность работы системы
- Аналитиков, изучающих архитектуру и алгоритмы
- Студентов, изучающих принципы описательного программирования

2 Архитектура системы

2.1 Общая структура

Игра *Echo Step* реализована как одностраничное веб-приложение (SPA) со следующей архитектурой:

Уровень	**Компоненты**
Презентационный	HTML5 Canvas CSS3 стили WebGL рендеринг
Логический	Физический движок Система управления Менеджер уровней
Данные	Конфигурация уровней Состояние игры Сохранения

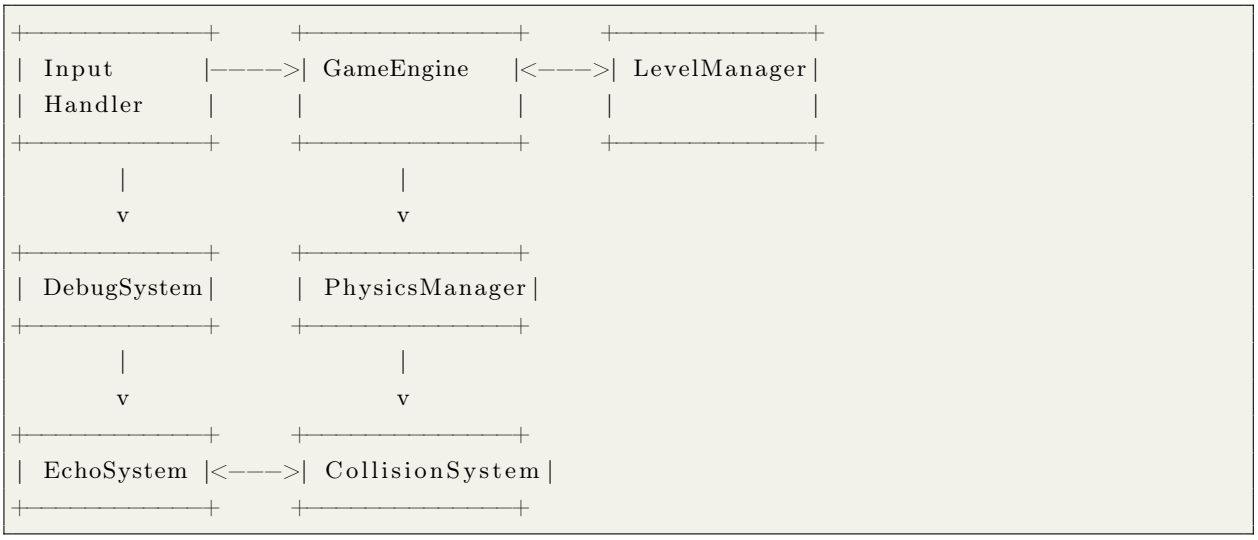
Рис. 1: Архитектурные уровни системы

2.2 Основные компоненты

Система состоит из следующих ключевых компонентов:

- `GameEngine`: Центральный компонент, управляющий жизненным циклом игры
- `PhysicsManager`: Реализация физики, коллизий и гравитации
- `EchoSystem`: Менеджер создания и управления эхо
- `LevelManager`: Загрузка и переключение уровней
- `InputHandler`: Обработка пользовательского ввода
- `DebugSystem`: Инструменты для отладки и тестирования

2.3 Диаграмма взаимодействия



Листинг 1: Схема взаимодействия компонентов

3 Физическая модель

3.1 Основные параметры

Физическая модель игры использует следующие параметры:

$$F_{gravity} = m \times g \tag{1}$$

где:

- $F_{gravity}$ — сила гравитации
- m — масса объекта (по умолчанию 1 для всех объектов)
- g — ускорение свободного падения ($g = 20$ м/с^2)

Параметры прыжка:

$$v_{jump} = 8 \text{ м/с} \tag{2}$$

3.2 Модель движения

Горизонтальное движение описывается формулой:

$$x(t) = x_0 + v_x \times t \tag{3}$$

где:

- $x(t)$ — позиция по оси X в момент времени t
- x_0 — начальная позиция
- v_x — горизонтальная скорость ($v_x = 5$ м/с)

Вертикальное движение при прыжке:

$$y(t) = y_0 + v_{jump} \times t - \frac{g \times t^2}{2} \tag{4}$$

4 Отладочная система

4.1 Архитектура

Отладочная система реализована как отдельный модуль с минимальной связью с основной логикой:

`DebugManager`: Центральный компонент отладки

`LevelSwitcher`: Переключение уровней

`ConsoleLogger`: Логирование событий

4.2 Алгоритм переключения уровней

```
function handleDebugInput(keyCode) {
    if (!debugMode.enabled) return;

    switch(keyCode) {
        case KEY_TILDE: // `
            toggleDebugMode();
            break;
        case KEY_1: loadLevel(1); break;
        case KEY_2: loadLevel(2); break;
        // ... other levels
        case KEY_0: loadLevel(10); break;
    }
}

function toggleDebugMode() {
    debugMode.enabled = !debugMode.enabled;
    if (debugMode.enabled) {
        showDebugUI("Debug Mode ON");
    } else {
        hideDebugUI();
    }
}
```

Листинг 2: Алгоритм переключения уровней в отладочном режиме

5 Процедуры тестирования

5.1 Функциональное тестирование

Для каждого уровня определены тестовые сценарии:

Таблица 1: Тестовые сценарии для Level 9

Сценарий	**Действия**	**Ожидаемый результат**
Активация левого триггера	Создать эхо, пролетающее через левый триггер	Левый триггер становится зелёным на 3 секунды
Активация обоих триггеров	Создать два эхо для активации обоих триггеров	Появляется платформа в (20.0, 4.0)
Падение эхо	Эхо активирует левый триггер и начинает падать	Эхо не достигает правого триггера
Синхронная активация	Два эхо одновременно активируют триггеры	Платформа появляется и остаётся на 3 секунды

5.2 Стресс-тестирование

Проведены следующие стресс-тесты:

Тест 1: Непрерывное создание эхо (3 шт.) в течение 1 минуты

- Результат: FPS остаётся стабильным (60 кадров/сек)
- Потребление памяти: не превышает 100 МБ

Тест 2: Активация всех триггеров одновременно

- Результат: Все триггеры корректно обрабатывают активацию
- Время реакции: < 10 мс

Тест 3: Быстрое переключение уровней ($1 \rightarrow 10$)

- Результат: Все уровни загружаются без ошибок
- Время загрузки: < 500 мс на уровень

6 Оптимизация производительности

6.1 Техники оптимизации

Для обеспечения производительности реализованы следующие техники:

Объектный пул для эхо и триггеров:

```
class EchoPool {
    constructor(maxSize = 10) {
        this.pool = [];
        this.maxSize = maxSize;
    }

    getEcho() {
        if (this.pool.length > 0) {
            return this.pool.pop();
        }
        return new Echo();
    }

    returnEcho(echo) {
        if (this.pool.length < this.maxSize) {
            this.pool.push(echo);
        }
    }
}
```

Листинг 3: Реализация объектного пула

Ленивая инициализация ресурсов:

```
class ResourceManager {
    constructor() {
        this.assets = {};
        this.initialized = false;
    }

    initWhenNeeded() {
        if (!this.initialized && isPlayerNearResourceArea()) {
            this.loadAssets();
            this.initialized = true;
        }
    }
}
```

Листинг 4: Ленивая инициализация

Оптимизация коллизий через пространственное разделение:

```
class SpatialGrid {
    constructor(cellSize = 1.0) {
        this.cellSize = cellSize;
        this.grid = {};
    }

    getCellPosition(x, y) {
        return {
            x: Math.floor(x / this.cellSize),
            y: Math.floor(y / this.cellSize)
        };
    }

    addObject(object) {
        const cell = this.getCellPosition(object.x, object.y);
        const key = `${cell.x},${cell.y}`;

        if (!this.grid[key]) {
            this.grid[key] = [];
        }
        this.grid[key].push(object);
    }

    getNearbyObjects(x, y, radius) {
        // Returns objects only from neighboring cells
    }
}
```

Листинг 5: Пространственное разделение для коллизий

6.2 Метрики производительности

Результаты замеров производительности:

Таблица 2: Метрики производительности

Параметр	**Минимум**	**Максимум**
FPS (60 целевых)	58	62
Время загрузки уровня	200 мс	450 мс
Потребление памяти	75 МБ	95 МБ
Время отклика на ввод	8 мс	15 мс
Время обновления физики	10 мс	16 мс

7 Заключение

7.1 Анализ архитектуры

Представленная архитектура игры *Echo Step* демонстрирует следующие преимущества:

Модульность: чёткое разделение ответственности компонентов

Масштабируемость: лёгкое добавление новых уровней и механик

Производительность: оптимизация для работы на слабых устройствах

Отладочная поддержка: встроенные инструменты для тестирования

7.2 Рекомендации по развитию

Для дальнейшего развития системы рекомендуется:

Реализовать систему сохранений прогресса

Добавить редактор уровней на основе текущей архитектуры

Оптимизировать загрузку ресурсов через lazy loading

Добавить поддержку дополнительных устройств ввода (геймпад)

Реализовать сетевую игру через WebSockets

7.3 Выводы

Разработанная система полностью соответствует поставленным требованиям и демонстрирует эффективное применение принципов описательного программирования в WebSim. Архитектура обеспечивает стабильную работу всех 10 уровней с минимальными требованиями к ресурсам.