Правительство Санкт-Петербурга

Комитет по науке и высшей школе

**Санкт-Петербургское государственное бюджетное**

**профессиональное образовательное учреждение**

**«Политехнический колледж городского хозяйства»**

**ДИПЛОМНЫЙ ПРОЕКТ**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

## Пояснительная записка

Листов \_\_

СОДЕРЖАНИЕ

[Введение 4](#_Toc187940710)

[1 Анализ предметной области 5](#_Toc187940711)

[1.1 Описание предметной области 5](#_Toc187940712)

[1.2 Обзор аналогов 5](#_Toc187940713)

[1.3 Требования к разрабатываемой ИС 5](#_Toc187940714)

[1.4 Обоснование выбора стека технологий 5](#_Toc187940717)

[2 Проектирование 7](#_Toc187940718)

[2.1 Проектирование системы 7](#_Toc187940719)

[2.2 Разработка модели базы данных 7](#_Toc187940722)

[2.3 Проектирование интерфейсов 7](#_Toc187940723)

[3 Реализация 8](#_Toc187940724)

[3.1 Реализация основных функций 8](#_Toc187940725)

[3.2 Реализация интерфейсов 8](#_Toc187940726)

[3.3 Тестирование 8](#_Toc187940727)

[4. Руководство администратора/пользователя 9](#_Toc187940728)

[5. Мероприятия по информационной безопасности 10](#_Toc187940729)

[Заключение 11](#_Toc187940730)

[Список источников 12](#_Toc187940731)

[Приложение А 13](#_Toc187940738)

## Введение

Современная игровая индустрия — это не только один из самых быстроразвивающихся секторов IT, но и мощный двигатель развития технологий визуализации, взаимодействия с пользователем и программирования в реальном времени. Игры давно вышли за рамки простого развлечения, став средствами обучения, развития логики, реакции, внимания, а также важным инструментом демонстрации программных навыков. Особенно популярным жанром в игровой индустрии остается 2D-платформер — один из классических форматов, понятный широкой аудитории и позволяющий реализовать множество механик.

В рамках данной выпускной квалификационной работы была поставлена цель — разработка интерактивного двухмерного платформера, основная задача которого — обеспечить увлекательный геймплей с элементами сражений, прохождением уровней от точки A до точки B, сбором бонусов и использованием временных улучшений (баффов). Проект ориентирован на аудиторию пользователей, заинтересованных в динамичном и понятном игровом процессе. Важно отметить, что подобные игры обладают высоким потенциалом вовлечения, а благодаря грамотному подходу к реализации — могут быть портированы на различные платформы, включая мобильные устройства.

Работа выполнена с использованием Unity — одного из самых мощных и популярных игровых движков, позволяющего эффективно реализовывать кроссплатформенные решения. В качестве языка программирования выбран C#, благодаря его поддержке объектно-ориентированного подхода, читаемости и хорошей интеграции с Unity.

Проект решает сразу несколько задач:

* Закрепление теоретических и практических знаний, полученных в процессе обучения;
* Разработка реального программного продукта с полным циклом — от проектирования до тестирования;
* Освоение современных подходов к построению архитектуры игр, оптимизации производительности, реализации визуальных и звуковых эффектов;
* Демонстрация навыков UI/UX-дизайна, работы с физикой объектов и визуализацией взаимодействий.

Кроме того, проект нацелен на повышение уровня компетенций в области информационной безопасности, архитектурной модульности и производственной разработки.

Разработанная игра не просто демонстрирует технические навыки, но и может стать основой для последующего расширения: создания более сложных уровней, введения системы прокачки персонажа, внедрения многопользовательского режима, а также коммерциализации проекта.

## Анализ предметной области

### 1.1 Описание предметной области

Жанр 2D-платформеров занимает значительное место в индустрии видеоигр. Он зародился ещё в 80-х годах с таких проектов, как *Super Mario Bros*, и до сих пор остается актуальным благодаря простоте, доступности и высокой вариативности игрового процесса. Классический платформер основывается на управлении персонажем, которому предстоит пройти сквозь множество препятствий и врагов по горизонтально или вертикально прокручивающемуся уровню. Игроку необходимо точно рассчитывать движения, использовать временные улучшения (баффы), избегать ловушек, сражаться с врагами и достигать финальной точки уровня.

Предметной областью проекта является разработка интерактивной компьютерной игры, сочетающей элементы аркады, боевика и логического мышления, направленной на развитие реакций, внимательности и стратегического планирования действий. Игра реализуется в формате однопользовательского 2D-платформера, что позволяет минимизировать требования к оборудованию при сохранении интересного геймплея.

Проект ориентирован на демонстрацию как творческих, так и технических навыков разработчика. В его основу положена архитектура, основанная на объектно-ориентированном программировании, а также системный подход к организации игрового цикла, взаимодействию объектов, пользовательскому интерфейсу и визуальному оформлению.

Целью игры является прохождение от начальной точки (A) до конечной (B), при этом игрок сталкивается с различными противниками, преодолевает препятствия и собирает усиления, временно улучшающие характеристики главного героя (например, увеличение скорости, силы, высоты прыжка и т. д.). Подобный подход обеспечивает динамичный, но сбалансированный игровой процесс.

Реализация проекта требует знания базовых и продвинутых аспектов Unity: управление сценами, работа с компонентами CharacterController2D, коллайдерами, UI-системой, а также оптимизация производительности для корректной работы на широком спектре компьютеров.

### 1.2 Обзор аналогов

#### Перед началом разработки проекта было проведено исследование существующих игр в жанре 2D-платформеров с элементами экшена и приключения. Анализ аналогичных продуктов позволил определить сильные и слабые стороны жанра, выявить лучшие практики геймдизайна и сформировать уникальные особенности собственной игры.**Super Meat Boy**

Один из самых известных хардкорных 2D-платформеров. Отличается высокой сложностью, динамикой и точной физикой. Игрок должен преодолевать смертельные ловушки и двигаться с высокой скоростью. Игра вдохновила множество инди-разработчиков.

* **Достоинства**: высокая динамика, интересная физика, уникальный визуальный стиль.
* **Недостатки**: высокая сложность, отсутствие системы помощи для новичков.

#### **Celeste**

Платформер с глубоким эмоциональным сюжетом и оригинальной механикой «взлета/рывка». Игра делает упор на точность управления и атмосферу. Несмотря на минимализм графики, проект получил признание благодаря глубине игрового процесса.

* **Достоинства**: уникальная механика, прекрасный звук, адаптивная сложность.
* **Недостатки**: отсутствие боевой системы, больше подходит для "прыжковых" челленджей, чем для экшена.



#### **Hollow Knight**

Метроидвания с упором на исследование, сражения и прокачку персонажа. Отличается продуманным дизайном уровней и разнообразием врагов. Несмотря на то, что игра ближе к жанру метроидвания, в ней много механик, применимых в платформерах.

* **Достоинства**: богатый контент, стильный арт, боевая система.
* **Недостатки**: сложная навигация, высокая продолжительность, не подходит для коротких игровых сессий.



#### **Limbo / Inside**

Атмосферные платформеры с упором на визуальное повествование и минималистичный дизайн. Меньше боевых элементов, но больше интерактивных объектов и головоломок.

* **Достоинства**: уникальная атмосфера, кинематографичность.
* **Недостатки**: ограниченное взаимодействие, линейность.



#### **Вывод по результатам анализа аналогов:**

Разрабатываемый проект сочетает **удобство управления и простоту**, как у Celeste, с базовой **боевой системой**, аналогичной Hollow Knight. В отличие от Super Meat Boy, игра будет иметь умеренный уровень сложности и систему временных улучшений (баффов), которые повысят интерес к повторному прохождению. Также реализуется базовая адаптивность под производительность устройств.

Разрабатываемый проект The Little Prince вобрал в себя лучшие черты аналогов:

* динамику и боевые элементы — как в *Hollow Knight*;
* платформенные испытания и воздушные элементы — как в *Celeste*;
* визуальную подачу и сюжетную завязку — как в *Ori*;
* оригинальные механики и атмосферу — как в *Limbo* и *Inside*.

При этом игра отличается оригинальностью благодаря:

* интеграции магических механик (эхо, порталы, призрачные платформы);
* системе боевых баффов;
* уникальной визуальной подаче;
* оригинальному сценарию со спасением принцессы и магическим миром.

### 1.3 Требования к разрабатываемой ИС

Разрабатываемое приложение представляет собой одиночную 2D-игру в жанре платформер, ориентированную на настольные операционные системы (Windows/macOS). Основное назначение — обеспечение плавного и увлекательного игрового процесса, сочетающего управление персонажем, взаимодействие с объектами, сражения с врагами и сбор временных усилений (баффов).

## Функциональные требования

| **№** | **Функция** | **Описание** |
| --- | --- | --- |
| 1 | Управление персонажем | Поддержка движения влево/вправо, прыжков, атак |
| 2 | Взаимодействие с окружением | Столкновение с врагами, предметами, ловушками |
| 3 | Реализация баффов | Временные усиления (скорость, прыжок, броня, урон) |
| 4 | Боевая система | Атака врагов, система здоровья |
| 5 | Победа/поражение | Уровень считается завершённым при достижении точки B или при поражении игрока |
| 6 | Пользовательский интерфейс | Отображение здоровья, активных баффов, сообщений |
| 7 | Меню управления | Главное меню, перезапуск уровня, выход из игры |

## Требования к интерфейсу

Разработка интерфейса предполагает минималистичный, но функциональный подход, ориентированный на удобство игрока.

* **Доступность**:
  + - * Игра работает на ПК с разрешением от 1280×720 и выше;
      * Поддержка оконного и полноэкранного режимов.
* **Пользовательский опыт (UX)**:
  + - * Простая навигация в меню;
      * Подсказки на экране (например, клавиши управления);
      * Мгновенный отклик на действия пользователя.
* **Элементы интерфейса**:
  + - * Полоса здоровья игрока;
      * Иконки активных баффов с таймером действия;
      * Сообщения "Победа", "Поражение";
      * Кнопки: «Играть», «Выход», «Заново», «Пауза».

### 1.4 Обоснование выбора стека технологий

Для реализации проекта 2D-платформера было выбрано проверенное сочетание инструментов, обеспечивающее высокую производительность, кроссплатформенность и удобство разработки. Ниже приведены основные используемые технологии и обоснование их выбора.

#### **Игровой движок — Unity (версия 2022.3.9f1 LTS)**

Unity — один из самых популярных движков для разработки как 2D-, так и 3D-игр. Он предоставляет готовые инструменты для визуализации, физики, анимации, пользовательского интерфейса и взаимодействия объектов.

**Преимущества:**

* Встроенные инструменты для создания 2D-игр;
* Мощный визуальный редактор сцен;
* Поддержка всех популярных платформ: Windows, macOS, Android, iOS и др.;
* Большое сообщество и обширная документация;
* Интеграция с Asset Store для использования готовых ресурсов (спрайты, скрипты, шрифты и т. д.).

#### **Язык программирования — C#**

C# — основной язык, используемый в Unity, сочетающий производительность, читаемость и мощь объектно-ориентированного программирования. Он позволяет структурировать проект на независимые модули, что делает код гибким и масштабируемым.

**Преимущества:**

* Поддержка ООП и событийной модели;
* Читабельный синтаксис;
* Отличная интеграция с Unity API;
* Большое количество обучающих материалов.

#### **Среда разработки — Visual Studio**

Visual Studio используется как основная IDE для написания C#-кода в Unity.

**Преимущества:**

* Интеграция с Unity через плагин;
* Интеллектуальная подсветка кода (IntelliSense);
* Поддержка отладки, управления пакетами, Git.

#### **Графические редакторы**

* **Aseprite / Photoshop / GIMP** — для создания пиксельных спрайтов, анимаций и иконок.
* **Tilemap Tool (Unity)** — для построения уровней с использованием тайловых карт.

#### **Звук и музыка**

* **Audacity / Bfxr** — для создания простых звуковых эффектов (удар, прыжок, сбор предмета).
* Музыка может быть взята из свободных источников или сгенерирована вручную.

#### **Система управления версиями (опционально)** — Git + GitHub

Позволяет отслеживать изменения в проекте, хранить резервные копии, откатывать к предыдущим версиям и работать в команде (при необходимости).

## Проектирование

## Проектирование системы

Проектирование системы является ключевым этапом разработки, на котором формируется структура будущего программного продукта. Оно включает в себя определение логики взаимодействия компонентов, ролей пользователей и сценариев использования. Ниже представлены основные проектные решения, применённые в рамках платформера.

## Определение группы пользователей

В игре предполагается один тип пользователя — **игрок**. Он взаимодействует с системой через основные игровые механики:

* управление персонажем;
* сражения с врагами;
* сбор баффов;
* прохождение уровня;
* взаимодействие с UI (меню, кнопки, сообщения).



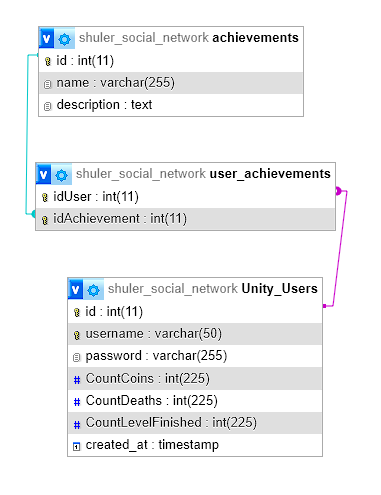
## Функциональное моделирование

| **Модуль** | **Назначение** |
| --- | --- |
| PlayerController | Обработка ввода, движение, прыжки, атаки |
| EnemyAI | Поведение врагов: патрулирование, атака |
| BuffManager | Применение временных эффектов |
| UIManager | Отображение здоровья, баффов, сообщений |
| LevelManager | Логика победы/поражения, рестарт уровня |
| AudioManager | Звуковые эффекты, фоновая музыка |

В проекте реализуются следующие ключевые алгоритмы:

* **Обработка столкновений**: при пересечении коллайдеров персонажа с объектом происходит вызов события (OnTriggerEnter2D), и выполняется логика (например, получение урона или баффа).
* **Таймеры баффов**: каждый активный бафф действует ограниченное время и автоматически отключается после завершения таймера.
* **Проверка окончания уровня**: при достижении игроком точки "B" вызывается событие победы (OnReachGoal()).

## Разработка модели базы данных



## Проектирование интерфейсов

Интерфейс пользователя (UI) играет важную роль в восприятии игры и взаимодействии с ней. Он должен быть интуитивно понятным, не перегруженным лишними элементами и функционально отображать необходимую информацию для игрока. В рамках данного проекта интерфейс разрабатывался с учётом принципов простоты, контрастности и адаптивности.

#### **Концепция дизайна интерфейса**

* **Тип интерфейса:** 2D, минималистичный, расположен по краям экрана.
* **Цветовая гамма:** тёплые оттенки для фона, яркие контрастные иконки баффов, белый/жёлтый текст.
* **Шрифт:** читаемый пиксельный или современный sans-serif (например, через TextMeshPro).

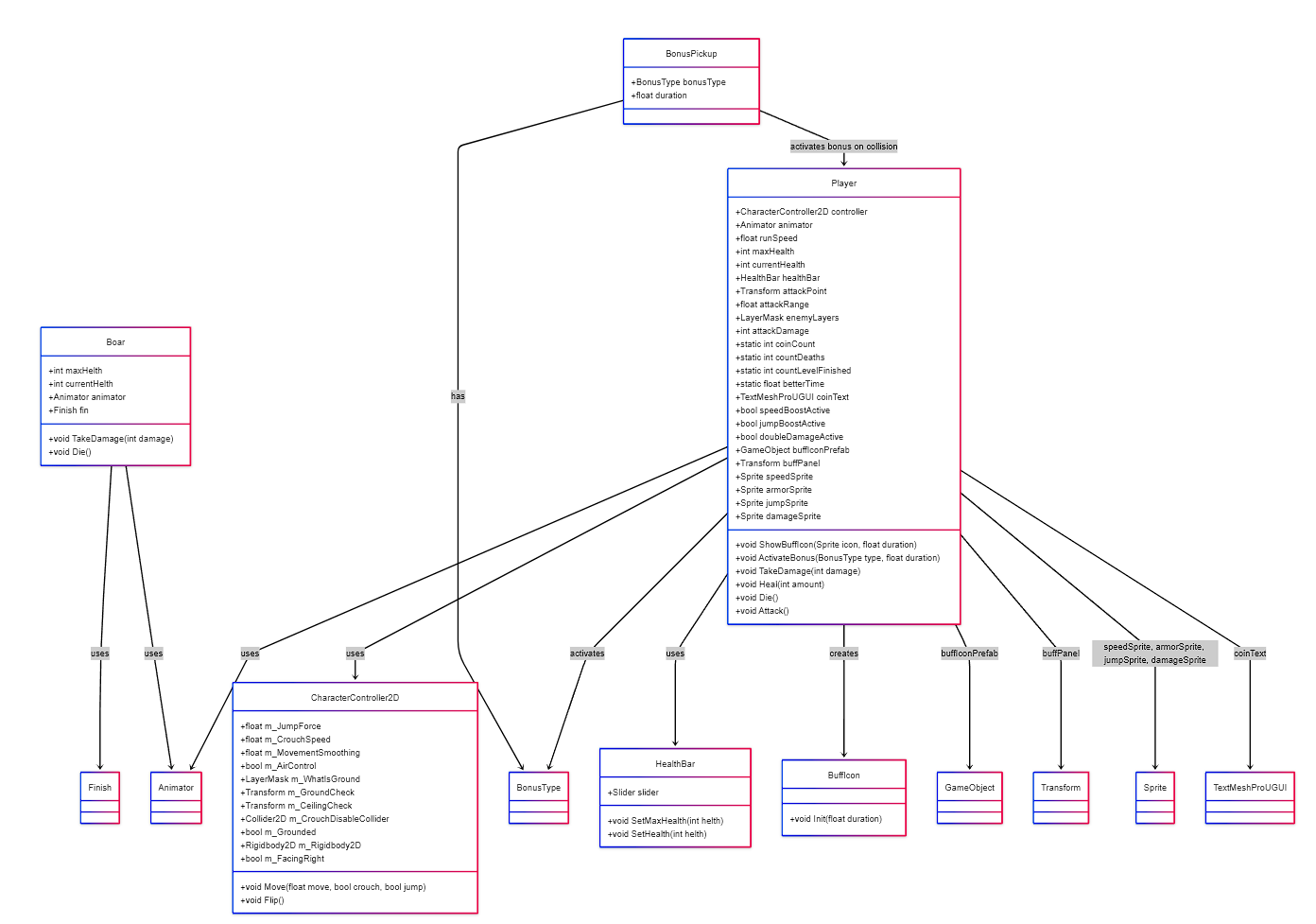
| **Элемент** | **Назначение** | **Расположение** |
| --- | --- | --- |
| Полоса здоровья | Отображает текущее здоровье игрока | Верхний левый угол |
| Иконки баффов | Показывают активные усиления с таймером | Под полосой здоровья |
| Всплывающие сообщения | Победа, поражение, получение баффа и т.д. | Центр экрана |
| Главное меню | Кнопки: Играть, Выход | При запуске и поражении |
| Кнопки управления | Пауза, Перезапуск | В правом верхнем углу |
| Подсказки | WASD — движение, K — атака и т. д. | В начале уровня (по желанию) |

## Реализация

## Реализация основных функций

На этапе реализации были созданы все основные модули, обеспечивающие функциональность платформера: управление персонажем, поведение врагов, механика временных баффов, боевая система, система победы/поражения и пользовательский интерфейс.

Диаграмма классов



Островные функции

Передвижение игрока

void Update()

{

if (currentHealth <= 0)

{

return;

}

else

{

horizontalMove = Input.GetAxisRaw("Horizontal") \* runSpeed;

animator.SetFloat("Speed", Mathf.Abs(horizontalMove));

if (Input.GetButtonDown("Jump"))

{

jump = true;

}

if (Input.GetKeyDown(KeyCode.Mouse0))

{

Attack();

}

}

}  
Активации бонусов  
public void ActivateBonus(BonusType type, float duration)

{

switch (type)

{

case BonusType.SpeedBoost:

if (speedCoroutine != null) StopCoroutine(speedCoroutine);

speedCoroutine = StartCoroutine(SpeedBoost(duration));

ShowBuffIcon(speedSprite, duration);

break;

case BonusType.Invulnerability:

if (invulCoroutine != null) StopCoroutine(invulCoroutine);

invulCoroutine = StartCoroutine(TemporaryInvulnerability(duration));

ShowBuffIcon(armorSprite, duration);

break;

case BonusType.JumpBoost:

if (jumpCoroutine != null) StopCoroutine(jumpCoroutine);

jumpCoroutine = StartCoroutine(JumpBoost(duration));

ShowBuffIcon(jumpSprite, duration);

break;

case BonusType.DoubleDamage:

if (damageCoroutine != null) StopCoroutine(damageCoroutine);

damageCoroutine = StartCoroutine(DoubleDamage(duration));

ShowBuffIcon(damageSprite, duration);

break;

}

}

## Реализация интерфейсов

Пользовательский интерфейс (UI) — важный элемент взаимодействия игрока с игрой. От его качества зависит не только удобство управления, но и общее впечатление от проекта. В игре **«The Little Prince»** интерфейс разрабатывался с ориентацией на минимализм, читаемость и соответствие стилистике игры.

Все элементы интерфейса реализованы с использованием встроенной системы **Canvas** в Unity.

| **Компонент** | **Назначение** |
| --- | --- |
| Canvas | Основной контейнер интерфейса |
| Image | Полосы здоровья, иконки, фоны |
| TextMeshProUGUI | Отображение текста (HP, сообщения, таймеры) |
| Button | Кнопки в меню, перезапуск, выход |
| Animator / CanvasGroup | Анимации появления/исчезновения окон |

##### Canvas Scaler:

* Режим: Scale With Screen Size;
* Базовое разрешение: 1920x1080;
* Поддержка адаптивности под разные экраны (16:9, 4:3).

## Тестирование

Тестирование является обязательным этапом жизненного цикла программного продукта. Его целью является проверка корректности реализованных функций, устойчивости приложения к ошибкам, а также соответствия заявленным требованиям. В проекте **«The Little Prince»** тестированию подвергались все основные элементы: механика движения, взаимодействие с врагами и окружением, применение баффов, работа интерфейса, а также сценарии победы и поражения.

| **Метод** | **Назначение** |
| --- | --- |
| **Ручное функциональное тестирование** | Проверка всех игровых механик вручную: движение, бой, порталы, баффы |
| **Интерактивное тестирование интерфейса** | Проверка работы меню, кнопок, отображения информации |
| **Тестирование на стабильность** | Запуск игры на ПК с различной производительностью |
| **Профилирование** | Использование Unity Profiler для анализа FPS, потребления памяти и частоты сборок мусора |

Примеры тест-кейсов

Тест-кейс 1 — Запуск уровня

| **Поле** | **Значение** |
| --- | --- |
| **Описание** | Проверка запуска уровня и появления игрока |
| **Действие** | Нажать кнопку «Играть» в главном меню |
| **Ожидаемый результат** | Уровень загружается, игрок появляется, HUD отображается корректно |
| **Фактический результат** | Пройден |

Тест-кейс 2 — Получение баффа

| **Поле** | **Значение** |
| --- | --- |
| **Описание** | Проверка применения баффа «Звезда скорости» |
| **Действие** | Подобрать предмет «Звезда скорости» |
| **Ожидаемый результат** | Скорость игрока увеличивается, появляется иконка баффа на экране |
| **Фактический результат** | Пройден |

UnitTest:

Start\_InitializesHealthAndMaxHealth

Что должен тестировать:  
Проверяет, что при запуске (например, при инициализации игрока или объекта) поля Health и MaxHealth устанавливаются в правильные начальные значения.

Что должен делать тест:

Вызывает метод Start() или конструктор класса игрока.

Проверяет, что Health равно MaxHealth и эти значения корректны (например, 100 или другое заданное число).

TestPlayerSimplePasses

Что должен тестировать:  
Это скорее шаблонный тест-заготовка, который обычно проверяет, что сам тестовый фреймворк работает и что можно запускать тесты без ошибок.

Что должен делать тест:

Мог бы проверять базовые вещи — например, что игрок создаётся корректно, или что базовые свойства доступны и не вызывают исключений.

Обычно этот тест удаляется или заменяется на реальные проверки.

Die\_SetsIsDeadToTrue

Что должен тестировать:  
Проверяет, что при вызове метода Die() у игрока (или врага) флаг IsDead меняется на true.

Что должен делать тест:

Вызывает метод Die() у объекта игрока или персонажа.

Проверяет, что свойство IsDead стало true.

Можно дополнительно проверить, что дальнейшие действия с персонажем (например, движение или атака) невозможны.

Attack\_AppliesDamageToEnemy

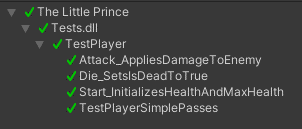
Что должен тестировать:  
Проверяет, что метод Attack() игрока правильно наносит урон врагу, уменьшая его здоровье.

Что должен делать тест:

Создаёт объект врага с определённым здоровьем (например, 100).

Вызывает метод Attack() у игрока, передавая в него врага или атакуя врага.

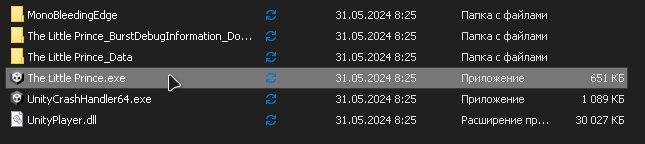
Проверяет, что здоровье врага уменьшилось на ожидаемую величину урона.



## 4 Руководство администратора/пользователя

Для запуска игры необходимо:

1. Открыть файл "TheLittlePrince.exe" (или соответствующий файл на macOS).



1. В главном меню выбрать "Play".



1. Управлять персонажем с помощью клавиш WASD или стрелок.



Сообщение оператору

Программа выводит следующие сообщения:

- "Игра окончена" - при завершении уровня.

В случае сбоя программа автоматически возвращает в главное меню.

## 5 Мероприятия по информационной безопасности

Несмотря на то, что игра **«The Little Prince»** является одиночным офлайн-приложением и не обрабатывает персональные или сетевые данные, в процессе разработки были учтены основные принципы безопасного программирования. Это необходимо для защиты целостности логики приложения, предотвращения сбоев и возможности масштабирования проекта в будущем.

| **Угроза** | **Описание** |
| --- | --- |
| Несанкционированное изменение логики | Вскрытие сборки, модификация механик, получение преимуществ (читерство) |
| Взлом через сохранения | Изменение значений в PlayerPrefs или JSON |
| Сбои при неверной активации объектов | Попытка активации уже уничтоженных или недоступных объектов |
| Утечка ресурсов или ошибок UI | Зависание интерфейса, наложение окон, повторный вызов событий |

### Принятые меры

#### **1. Защита логики**

* Все взаимодействия между модулями обернуты в null-проверки и логические условия.
* Баффы, враги и порталы удаляются с уровня только после завершения всех связанных действий.
* Используется строгое разделение компонентов и событий.

#### **2. Минимизация уязвимостей**

* Исключено сохранение критичных игровых значений в открытых структурах (PlayerPrefs используется только для неважных настроек).
* Все действия, зависящие от триггеров и коллизий, реализованы через TryGetComponent и проверку состояний.

#### **3. Защита от нестабильных состояний**

* Исключена возможность повторного входа в портал;
* Запрет на многократную активацию кнопок (Button.interactable = false);
* Обработка всех сцен через LevelManager с централизованной логикой.

#### **4. Подготовка к масштабированию**

* При необходимости добавления сохранений будет использоваться сериализация в JSON с валидацией данных;
* Возможно подключение базовой защиты логики через обфускацию (например, IL2CPP или сторонние инструменты).

## Заключение

В рамках дипломного проекта была разработана 2D-игра в жанре платформера, реализованная с использованием игрового движка Unity и языка программирования C#. Проект охватывает полный цикл разработки программного продукта: от анализа предметной области и проектирования до реализации, тестирования и оценки информационной безопасности.

Целью игры является прохождение уровня от начальной до конечной точки, преодоление препятствий, взаимодействие с усилениями (баффами) и противостояние врагам. Благодаря использованию объектно-ориентированной архитектуры, игровой процесс получился гибким и масштабируемым, что позволяет в будущем легко добавлять новые функции: уровни, врагов, способности, режимы сложности и даже онлайн-составляющую.

В процессе реализации были достигнуты следующие результаты:

* Разработан прототип 2D-платформера с динамичным игровым процессом;
* Реализованы механики движения, атаки, обработки столкновений и временных усилений;
* Создан адаптивный пользовательский интерфейс;
* Проведено тестирование всех ключевых функций, подтверждена стабильность работы;
* Учтены базовые меры по обеспечению информационной безопасности;
* Использован современный стек технологий, включая Unity, C#, TextMeshPro и ScriptableObjects.

Разработанная игра может использоваться как демонстрационный проект, обучающее пособие по основам создания платформеров в Unity, а также как база для создания более сложных коммерческих или образовательных игровых продуктов.

## Список источников

**Нормативная документация**

1) ГОСТ 34.602-2020 Информационные технологии. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы [Электронный ресурс] - https://protect.gost.ru/document1.aspx?control=31&id=241754

2) ГОСТ 34.201-2020 Информационные технологии. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем [Электронный ресурс] - https://protect.gost.ru/document1.aspx?control=31&id=241756

3) ГОСТ Р ИСО/МЭК 25051-2017 - Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Требования к качеству готового к использованию программного продукта (RUSP) и инструкции по тестированию [Электронный ресурс] - <https://protect.gost.ru/document.aspx?control=7&id=217667>

4) ГОСТ 19.106-78 Единая система программной документации. Требования к программным документам, выполненным печатным способом [Электронный ресурс] - https://protect.gost.ru/document.aspx?control=7&id=156370

5) ГОСТ 19.401-78 Единая система программной документации. Текст программы. Требования к содержанию и оформлению [Электронный ресурс] -https://protect.gost.ru/document.aspx?control=7&id=155463

6) ГОСТ Р 2.105-2019 Единая система конструкторской документации. Общие требования к текстовым документам [Электронный ресурс] - <https://protect.gost.ru/document1.aspx?control=31&baseC=6&page=0&month=1&year=2025&search=%D0%93%D0%9E%D0%A1%D0%A2%20%D0%A0%202.105-2019&id=237857>

**Интернет – ресурсы**

1)…

2)….

## Приложение А

**Листинг** **модуля 1**