Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра автоматизации обработки информации (АОИ)

**СОЗДАНИЕ КОМПЬЮТЕРНОЙ ИГРЫ В ЖАНРЕ 2D ПЛАТФОРМЕР**

Курсовая работа по дисциплине

«Компьютерная графика»

Студент гр. 426-2   
  
\_\_\_\_\_\_\_\_\_\_\_\_\_ Р.Д.Карасев

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

|  |  |  |
| --- | --- | --- |
|  | | Руководитель  канд. техн. наук, доцент каф. АОИ  \_\_\_\_\_\_\_\_\_\_\_\_ Т.О. Перемитина  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. |
|  |  | |

Томск 2018

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**Кафедра автоматизации обработки информации (АОИ)

УТВЕРЖДАЮ

Заведующий кафедрой АОИ

Д-р техн. наук, проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.П. Ехлаков

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_ г.

**ЗАДАНИЕ**на курсовую работу по дисциплине «Компьютерная графика»

студенту Карасеву Роману Дмитриевичу

группа 426-1, факультет систем управления

1. Тема проекта: Создание компьютерной игры в жанре 2D платформер
2. Срок сдачи студентом законченного проекта: 12.12.2018
3. Исходные данные к проекту:
4. Содержание пояснительной записки (перечень подлежащих разработке вопросов): анализ предметной области, средства реализации, проектирование игры, программирование и отладка, результаты работы, заключение.
5. Дата выдачи задания: 07.09.2018

Руководитель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 (подпись)

доцент каф. АОИ ТУСУР, Перемитина Т.О.

Задание принял к исполнению

(дата)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

# Оглавление

[**Оглавление 3**](#_Toc532328825)

[**Введение 4**](#_Toc532328826)

[**1 Обзор аналогов реализации и существующих игр 5**](#_Toc532328827)

[**3 Проектирование игры 13**](#_Toc532328828)

[**3.2** **Окружение** 14](#_Toc532328829)

[**3.3** **Взаимодействующие объекты** 16](#_Toc532328830)

[**4 Программирование и отладка** 18](#_Toc532328831)

[**4.1** **Реализация класса Unit** 19](#_Toc532328832)

[**4.2** **Реализация класса Character** 19](#_Toc532328833)

[**4.3** **Реализация классов Monster, ShootableMonster, MoveableMonster и Obstacle** 19](#_Toc532328834)

[**4.4** **Реализация классов Bullet, Heart, LiveBar, Buttons, CameraController и LevelEnd** 20](#_Toc532328835)

[**4.5 Тестирование и отладка** 21](#_Toc532328836)

[**5** **Результат работы** 22](#_Toc532328837)

[**6 Заключение** 23](#_Toc532328838)

[**Список использованной литературы** 24](#_Toc532328839)

[**Приложение А** 25](#_Toc532328840)

# Введение

В современном мире очень быстро и обширно развивается игровая индустрия, это связанно с сильно возросшей популярностью игр. Современные компании зарабатывают огромные суммы на продаже игр и внутриигровых предметов. Эксперты считают, что доходы с игр по итогу 2018 года составят $137,9 млрд. Это на 13,3% больше, чем индустрия заработала в прошлом году [1]. Поэтому данная отрасль разработки на данный момент является очень перспективной и популярной среди молодых разработчиков.

Несмотря на огромное разнообразие жанров в настоящее время, большую популярность среди пользователей до сих пор имеют игры одного из первых жанров – платформер. Это связанно с простотой игрового процесса, но при этом сложность такой игры может быть очень высокой, это и привлекает пользователей, незамысловатая игра, для победы в которой необходимо показать хороший уровень концентрации, реакции и смекалки. **Платформер** (англ. platformer, platform game) — жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня [2]. Поэтому темой данного проекта была выбрана разработка игры в жанре 2D (от англ. *2-dimensional*, 2-х мерный*)* платформер.

Для проекта были поставлены следующие задачи:

* анализ предметной области и проектирование будущей игры;
* выбор инструментального программного обеспечения;
* изучение выбранных средств разработки;
* разработка игры.

# Обзор аналогов реализации и существующих игр

Для выполнения задания был проведен обзор готовых решений для разработки игр.

**Unity** - это передовой игровой движок от компании Unity Technologies, позволяющий разрабатывать не только 2D, но и 3D-игры. По мнению большинства инди-разработчиков игр на данный момент это лучшая платформа разработки игр.

На официальном сайте движка имеются бесплатные обучающие материалы, проекты, онлайн-обучение и документация. У движка очень большое сообщество пользователей, поэтому вы можете черпать ответы, советы и вдохновение на официальных и неофициальных форумах.

**Game Maker** - это один из самых известных и удобных конструкторов игр. Создание игр в нем достаточно простое и не требует предварительного знакомства с каким-либо из языков программирования.

Игра в GameMaker строится как набор игровых объектов, поведение которых задаётся путём программирования реакции на события. Программировать можно, используя графическое представление программ. Это представление отличается от обычного, например, тем, что для того чтобы начать условный оператор, нужно перетащить на панель действия восьмиугольник с иконкой, обозначающей тип проверки, а затем, возможно, ввести какие-либо значения в появившуюся форму. Имеется в нём и скриптовый язык GML похожий на JavaScript.

Рассчитан в основном он на создание двухмерных (2D) игр любых жанров. Также подойдёт для создания различных презентаций и т.п. Начиная с 6-й версии появилась ограниченная возможность работать с 3D. Для разработки 3D игр в Game Maker’е необходимо использовать встроенный язык программирования Game Maker Language.

**Construct 2** – это конструктор 2D-игр, позволяющий разрабатывать игры любого жанра и направленности в формате технологии HTML5. Он достаточно прост в освоении и удобен в использовании, благодаря чему любой желающий создать свою игру сможет это сделать даже без знаний в программировании. Данный конструктор подойдёт не только для начинающих, но и для профессионалов, желающих отойти от технологии Flash к HTML5, которые хотят сделать качественные web-игры, либо прототипы. Получаемые посредством Construct 2 игры поддерживаются популярными web-браузерами и мобильными устройствами ОС типа Android или iOS [3].

Для реализации проекта была выбрана среда разработки Unity. Такой выбор был сделан исходя из того, что ранее уже были реализованы проекты с помощью Unity и уже имеется необходимый опыт работы с ним, для реализации данного проекта.

* 1. **Обзор аналогов игры**

Для анализа существующего рынка мобильных приложения и выявления особенности и недостатков приложений схожего жанра был проведен обзор существующих аналогов.

**Limbo** — мультиплатформенная компьютерная игра в жанре платформер с элементами survival horror, разработанная независимой датской студией Playdead. Технически проект является 2D сайд-скроллером, в котором используется физический движок Box2D, позволяющий управлять объектами окружающей среды и персонажем. Протагонистом является маленький мальчик, который ищет свою сестру. Студия называла геймплейный стиль проекта методом «проб и смертей» — в течение игрового процесса главный герой из-за неверных решений игрока постоянно умирает. Limbo создана в монохроматических чёрно-белых тонах, с минималистским окружением и эмбиентным звуковым сопровождением.

Игра получила множество призов на игровых фестивалях и конференциях, в том числе за лучшую визуальную стилистику, техническое исполнение, звук, а также номинировалась на премию «Игра года».



Рисунок 1.1 – Скриншот игры Limbo

**Mario** — серия видеоигр, разработанная компанией Nintendo, крупная медиафраншиза и одна из самых больших серий за историю индустрии компьютерных развлечений. Игры основной серии выполнены в жанре платформера, главным героем в них является американский водопроводчик итальянского происхождения Марио.

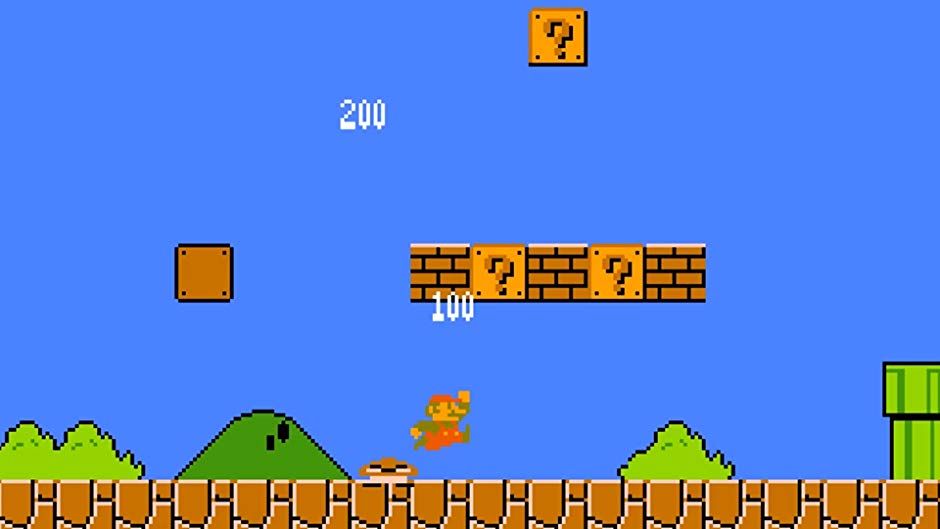


Рисунок 1.2 – Скриншот из игры Mario

**Rayman** — компьютерная игра, разработанная французской компанией Ubisoft Montpellier. Rayman является классическим 2D-платформером. Цели игры — освобождение всех электунов (electoons) и победа над фаворитами Мистера Мрака по всем шести мирам. Только при освобождении каждого электуна Рэйман сможет войти в укрытие Мистера Мрака. В противном случае игрок должен снова попытаться собрать их в уровнях и добраться до указателя с восклицательным знаком.



Рисунок 1.3 – Скриншот из игры Rayman

Rayman получил награды «Лучшая музыка в CD-ROM» и «Лучшая анимация» в Electronic Gaming Monthly’s в 1995 году Rayman также получил титул самой продаваемой PlayStation-игры всего времени в Великобритании.

1. **Средства реализации**
   1. **Язык программирования C#**

**C#** - это язык программирования, предназначенный для разработки самых разнообразных приложений, предназначенных для выполнения в среде .NET Framework. Язык C# прост, типобезопасен и объектно-ориентирован. Благодаря множеству нововведений C# обеспечивает возможность быстрой разработки приложений, но при этом сохраняет выразительность и элегантность, присущую языкам C [4].

* 1. **Unity**

**Unity** - кроссплатформенный игровой движок для разработки двухмерных и трехмерных приложений и игр под различные платформы. **Unity**имеет очень простой **Drag and Drop** интерфейс, который разбит на несколько окон: **Hierarchy**, где находятся названия всех объектов на сцене, которые можно группировать и легко переходить по ним, **Scene**, где можно рассмотреть определенную сцену под нужным вам ракурсом, **Inspector**, который поможет с настройкой выделенного объекта, **Project**, где видны все материалы проекта, **Toolbar**(или меню с инструментами).

**Unity** поддерживает два языка: **C#** (наиболее используемый) и **Javascipt**. Unity использует компонентно-ориентированный подход. Все в игре — это обьект, куда добавлены различные компоненты. Например, если мы делаем платформер, мы добавляем GameObject, и к этому GameObject добавляем графический компонент (для отрисовки игрока) и компонент управления (чтобы можно было управлять игроком клавиатурой или мышкой). Таких различных компонентов можно добавить любое количество к любому GameObject. То есть, создание игры в Unity — это добавление GameObject-ов, и добавление им нужных компонентов [5].

**2.3 Visual Studio**

**Visual Studio** - это программная среда по разработке приложений для ОС Windows, как консольных, так и с графическим интерфейсом. К достоинствам данной среди можно отнести:

* Поддержка множества языков при разработке. Visual Studio позволяет писать код на своем языке или любых других предпочитаемых языках, используя все время один и тот же интерфейс (IDE);
* интуитивный стиль кодирования. По умолчанию Visual Studio форматирует код по мере его ввода, автоматически вставляя необходимые отступы и применяя цветовое кодирование для выделения элементов типа комментариев;
* более высокая скорость разработки. Многие из функциональных возможностей Visual Studio направлены на то, чтобы помогать разработчику делать свою работу как можно быстрее. Удобные функции, вроде функции IntelliSense (которая умеет перехватывать ошибки и предлагать правильные варианты), функции поиска и замены (которая позволяет отыскивать ключевые слова как в одном файле, так и во всем проекте) и функции автоматического добавления и удаления комментариев (которая может временно скрывать блоки кода), позволяют разработчику работать быстро и эффективно;
* возможности отладки. Предлагаемые в Visual Studio инструменты отладки являются наилучшим средством для отслеживания загадочных ошибок и диагностирования странного поведения. Разработчик может выполнять свой код по строке за раз, устанавливать интеллектуальные точки прерывания, при желании сохраняя их для использования в будущем, и в любое время просматривать текущую информацию из памяти [6].

# Проектирование игры

* 1. **Логика игры**

Игра предоставляет пользователю управление над персонажем, с помощью него пользователь взаимодействует с игровым миром. Геймплей состоит из продвижения персонажа по уровню, преодоление препятствий и уничтожении разных типов врагов.

Основной цикл игры представляет собой неограниченное время, в течение которого пользователь управляет персонажем, который длится до тех пор, пока пользователь не захочет прекратить играть, или же не пройдет уровень. Персонаж, находящийся под управлением пользователя, имеет ряд возможностей: движение в двух направлениях, прыжок и выстрел.

## **Окружение**

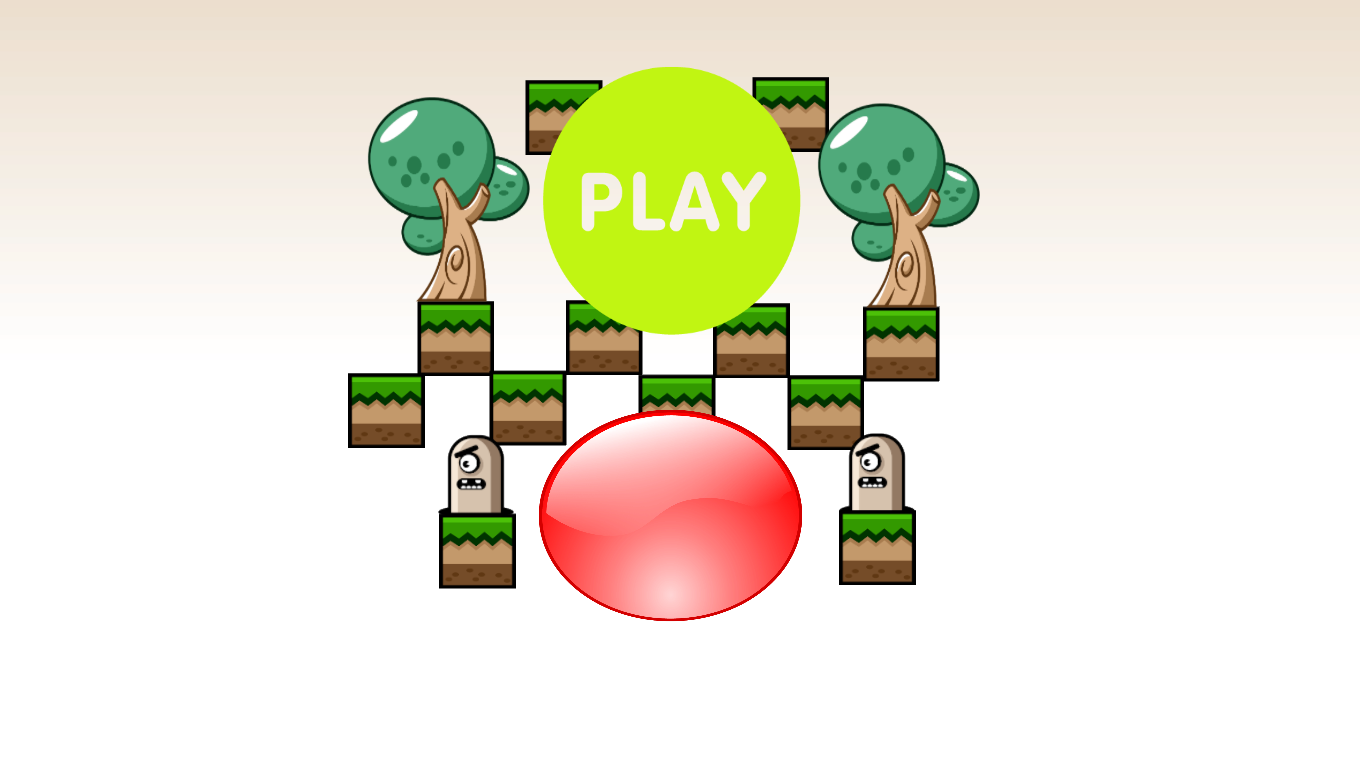
При запуске игры пользователя приветствует начальное меню, из которого он может начать игру, либо закрыть ее. 

Рисунок 3.1 – Главное меню

Когда пользователь начинает игру, запускается сцена с уровнем игры.

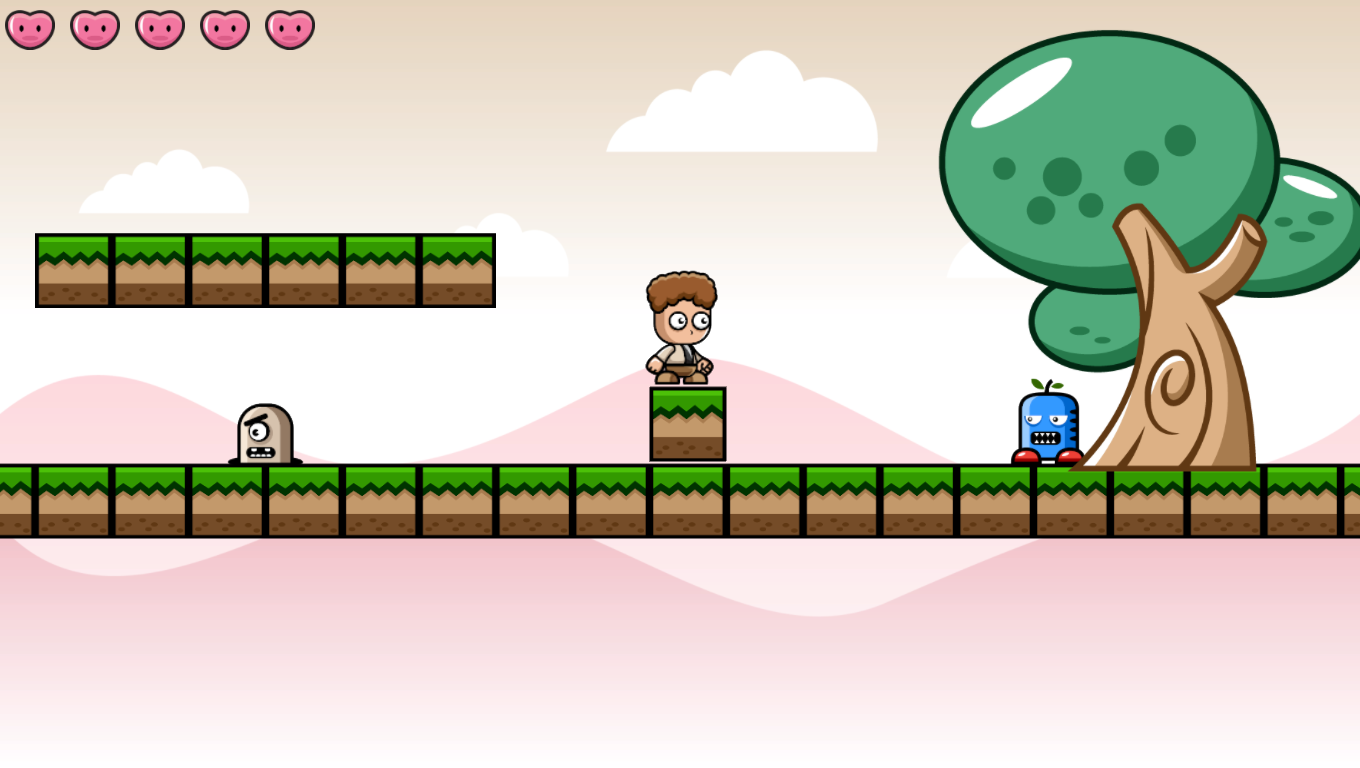


Рисунок 3.2 – Сцена уровня игры

Если в течение игры закончатся жизни, будет запущена сцена «GameOver». Существует возможность начать игру заново, либо закрыть ее.



Рисунок 3.3 – Сцена «GameOver»

## **Взаимодействующие объекты**

Пользователь управляет персонажем с помощью клавиатуры и компьютерной мыши. При нажатий клавиш «A» и «D» персонаж начинает движение влево и вправо, соответственно, при нажатии клавиши «Space» персонаж совершит прыжок, а с помощью левой кнопки мыши производится выстрел.

Изначально персонаж имеет 5 жизней, которые отнимаются при взаимодействии с врагами и препятствиями. Жизни можно восстановить, если найти и подобрать во время игры сердечко.

В игре имеется 3 типа врагов:

1. не двигающийся обычный;

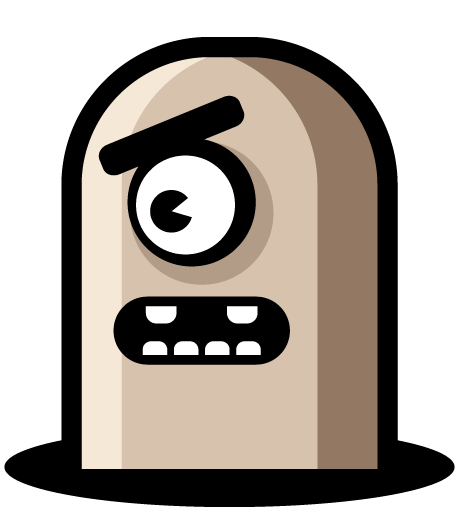


Рисунок 3.4 – Враг 1-го типа

1. не двигающийся с дальней атакой;

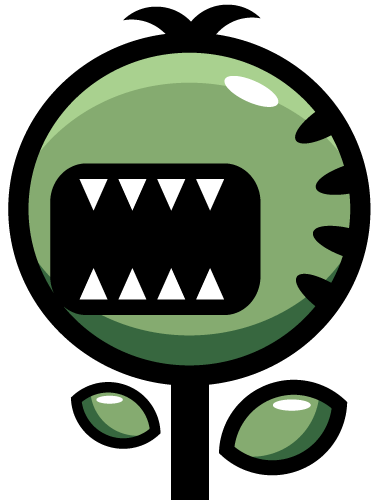


Рисунок 3.5 – Враг 2-го типа

1. двигающийся.

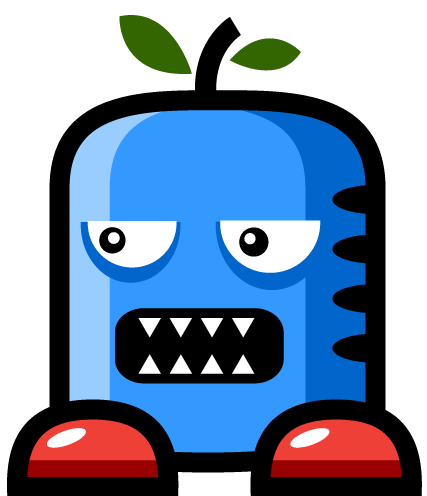


Рисунок 3.6 – Враг 3-го типа

Для уничтожения врагов 1-го типа достаточно произвести выстрел. Враги 2-го и 3-го типов не реагируют на выстрелы, для того, чтобы их уничтожить, необходимо прыгнуть на них сверху.

Также в игре есть препятствия, при соприкосновении с которыми, будет наноситься урон персонажу.

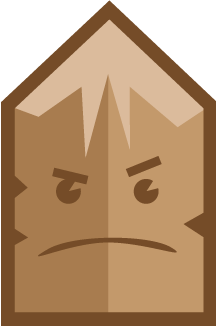


Рисунок 3.7 – Препятствие

При дальнейшей доработке планируется создание новых врагов и препятствий.

# Программирование и отладка

Объекты представляют собой унаследованные классы, базовым для которых является специальный класс из библиотеки, использующийся для видимых игровых объектов.

Реализация игры представляет из себя 12 взаимодействующих классов – Unit (методы получения урона и уничтожения объектов), Character (действия персонажа), Monster, ShootableMonster и MoveableMonster (действия монстров и их взаимодействие с персонажем), Obstacle (взаимодействие персонажа с препятствиями), Bullet (снаряды), Heart и LiveBar (классы для жизней персонажа), а также Buttons, CameraController (действия кнопок и камеры) и LevelEnd (завершение уровня).

Алгоритмы движений и выстрела были взяты из официального источника по обучению созданий игр на Unity [7].

Ниже представлен код алгоритмов передвижения, прыжка и выстрела.

Передвижение:

private void Run()

{

Vector3 direction = transform.right \* Input.GetAxis("Horizontal");

transform.position = Vector3.MoveTowards(transform.position, transform.position + direction, speed \* Time.deltaTime);

sprite.flipX = direction.x < 0.0F;

if (isGrounded) State = CharState.Run;

}

Прыжок:

private void Jump()

{

rigidbody.AddForce(transform.up \* jumpForce, ForceMode2D.Impulse);

}

Выстрел:

private void Shoot()

{

Vector3 position = transform.position; position.y += 0.8F;

Bullet newBullet = Instantiate(bullet, position, bullet.transform.rotation) as Bullet;

newBullet.Parent = gameObject;

newBullet.Direction = newBullet.transform.right \* (sprite.flipX ? -1.0F : 1.0F);

}

Полный листинг скриптов предоставлен в приложении А.

## **Реализация класса Unit**

Класс Unit используется для двух методов - RecivedDamage() – получение урона и Die() – уничтожение игровых объектов.

## **Реализация класса Character**

Класс Character наследуется от класса Unit для получения методов RecivedDamage() и Die(), также в нем находятся методы: Run() – движение персонажа при нажатии клавиш, Jump() – прыжок персонажа, Shoot() – выстрел, CheckGround() – проверяет находится ли персонаж в воздухе и переходы между анимациями движений. Также хранит в себе значения скорости персонажа и его начальное количество жизней.

## **Реализация классов Monster, ShootableMonster, MoveableMonster и Obstacle**

Данные классы хранят в себе методы для движения врагов, их действий и взаимодействия с персонажем. Класса Monster хранит метод OnTriggerEnter2D(Collider2D collider), который проверяет, какой объект касается врага, снаряд или персонаж, и определяет кто должен получить урон.

В классе ShootableMonster находятся два метода. Метод Shoot(), который дает возможность монстру стрелять, и метод OnTriggerEnter2D(Collider2D collider), который уничтожает врага, если персонаж прыгнет на него. Класс MoveableMonster также содержит два метода. Первый OnTriggerEnter2D(Collider2D collider), такой же как и у класса ShootableMonster, а второй Move(), дает возможность монстру двигаться в двух направлениях. У класса Obstacle всего один метод OnTriggerEnter2D(Collider2D collider), который вызывает метод RecivedDamage() у персонажа.

## **Реализация классов Bullet, Heart, LiveBar, Buttons, CameraController и LevelEnd**

Класс Bullet хранит в себе скорость снаряда, его время существования и метод OnTriggerEnter2D(Collider2D collider), который вызывает метод RecivedDamage() у некоторых объектов, в которые попадает снаряд.

Класса Heart один метод, который восстанавливает персонажу одну жизнь при подборе сердечка.

Класс LiveBar отображает и изменяет, при получении урона и подборе сердечка, количество жизней персонажа.

В классе buttons описаны переходы между сценами, при нажатии кнопок Play, Exit и Restart.

Класс CameraController хранит в себе метод для движения камеры вслед за персонажем.

Класс LevelEnd хранит метод OnTriggerEnter2D(Collider2D collider), который запускает сцену LevelEnd.

## **4.5 Тестирование и отладка**

Тестирование проводилось контрольной группой в размере 5 человек. В ходе тестирования проблемы не были выявлены. Следовательно, продукт был одобрен контрольной группой и готов к презентации.

# Результат работы

Результатом работы стала полноценная игра в жанре 2D Платформер, которая на данный момент имеет один уровень.

# Заключение

Создание игр является хорошим опытом для начинающего разработчика. Это позволяет ему наглядно увидеть взаимодействие объектов, усовершенствовать профессиональные навыки, более углубленно освоить языки программирования. Во время работы также существует возможность закрепить знания математики и физики, ведь они постоянно используются в создании игр. После завершения работы разработчик получает не только новый опыт и знания, но и продукт, который может принести не только прибыль, но и удовольствие.

В ходе курсовой работы были получены навыки по разработке компьютерных игр. Была проанализирована предметная область, что поспособствовало выработке навыков научно-исследовательской работы, были изучены некоторые особенности проектирования игр и улучшены навыки в обращении с языком C#. Закреплены навыки самостоятельной работы и оформления пояснительной записки. Также разработанная игра станет хорошим проектом в портфолио.

# Список использованной литературы

1. Мировой рынок игр. [Электронный ресурс]. — Режим доступа: URL https://app2top.ru/industry/mirovoj-ry-nok-igr-zarabotaet-137-9-mlrd-za-2018-god-120106.html (дата обращения: 14.11.2018).
2. Википедия: Платформер. [Электронный ресурс]. — Режим доступа: https://ru.wikipedia.org/wiki/Платформер (дата обращения: 14.11.2018)
3. Pikabu: Топ программ для создания 2D игр. [Электронный ресурс]. — Режим доступа: https://pikabu.ru/story/top\_5\_programm\_dlya\_sozdaniya\_2d\_igr\_bez\_programmirovaniya\_4248324 (дата обращения: 14.11.2018)
4. Язык программирования C#. [Электронный ресурс]. Режим доступа: https://flexberry.github.io/ru/gbt\_csharp.html (дата обращения 20.11.2018)
5. О Unity: Обзор движка. [Электронный ресурс]. — Режим доступа: https://itkeys.org/about-unity/ (дата обращения: 20.11.2018)
6. Описание среды разработки Microsoft Visual Studio. [Электронный ресурс]. — Режим доступа: https://studbooks.net/2258619/informatika/opisanie\_sredy\_razrabotki\_microsoft\_visual\_studio. (дата обращения: 20.11.2018)
7. Изучение Unity. [Электронный ресурс]. — Режим доступа: https://unity3d.com/ru/learn/tutorials/topics/2d-game-creation/player-controller-script (дата обращения: 20.11.2018)

## **Приложение А**

**Unit:**

public class Unit : MonoBehaviour

{

public virtual void ReceiveDamage()

{

Die();

}

protected virtual void Die()

{

Destroy(gameObject);

}

}

**Character:**

using UnityEngine.SceneManagement;

public class Character : Unit

{

[SerializeField]

private int lives = 5;

public int Lives

{

get { return lives; }

set

{

if (value < 5) lives = value;

livesBar.Refresh();

}

}

private LivesBar livesBar;

[SerializeField]

private float speed = 3.0F;

[SerializeField]

private float jumpForce = 15.0F;

private bool isGrounded = false;

private Bullet bullet;

private CharState State

{

get { return (CharState)animator.GetInteger("State"); }

set { animator.SetInteger("State", (int)value); }

}

new private Rigidbody2D rigidbody;

private Animator animator;

private SpriteRenderer sprite;

private void Awake()

{

livesBar = FindObjectOfType<LivesBar>();

rigidbody = GetComponent<Rigidbody2D>();

animator = GetComponent<Animator>();

sprite = GetComponentInChildren<SpriteRenderer>();

bullet = Resources.Load<Bullet>("Bullet");

}

private void FixedUpdate()

{

CheckGround();

}

private void Update()

{

if (isGrounded) State = CharState.Idle;

if (Input.GetButtonDown("Fire1")) Shoot();

if (Input.GetButton("Horizontal")) Run();

if (isGrounded && Input.GetButtonDown("Jump")) Jump();

}

private void Run()

{

Vector3 direction = transform.right \* Input.GetAxis("Horizontal");

transform.position = Vector3.MoveTowards(transform.position, transform.position + direction, speed \* Time.deltaTime);

sprite.flipX = direction.x < 0.0F;

if (isGrounded) State = CharState.Run;

}

private void Jump()

{

rigidbody.AddForce(transform.up \* jumpForce, ForceMode2D.Impulse);

}

private void Shoot()

{

Vector3 position = transform.position; position.y += 0.8F;

Bullet newBullet = Instantiate(bullet, position, bullet.transform.rotation) as Bullet;

newBullet.Parent = gameObject;

newBullet.Direction = newBullet.transform.right \* (sprite.flipX ? -1.0F : 1.0F);

}

public override void ReceiveDamage()

{

Lives--;

rigidbody.velocity = Vector3.zero;

rigidbody.AddForce(transform.up \* 8.0F, ForceMode2D.Impulse);

Debug.Log(lives);

if (lives == 0)

{

SceneManager.LoadScene("GameOver");

}

}

private void CheckGround()

{

Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position, 0.3F);

isGrounded = colliders.Length > 1;

if (!isGrounded) State = CharState.Jump;

}

private void OnTriggerEnter2D(Collider2D collider)

{

Bullet bullet = collider.gameObject.GetComponent<Bullet>();

if (bullet && bullet.Parent != gameObject)

{

ReceiveDamage();

}

}

}

public enum CharState

{

Idle,

Run,

Jump

}

**Monster:**

public class Monster : Unit

{

protected virtual void Awake() { }

protected virtual void Start() { }

protected virtual void Update() { }

protected virtual void OnTriggerEnter2D(Collider2D collider)

{

Bullet bullet = collider.GetComponent<Bullet>();

if (bullet)

{

ReceiveDamage();

}

Character character = collider.GetComponent<Character>();

if (character)

{

character.ReceiveDamage();

}

}

}

**ShootableMonster:**

public class ShootableMonster : Monster

{

[SerializeField]

private float rate = 2.0F;

[SerializeField]

private Color bulletColor = Color.white;

private Bullet bullet;

protected override void Awake()

{

bullet = Resources.Load<Bullet>("Bullet");

}

protected override void Start()

{

InvokeRepeating("Shoot", rate, rate);

}

private void Shoot()

{

Vector3 position = transform.position; position.y += 0.5F;

Bullet newBullet = Instantiate(bullet, position, bullet.transform.rotation) as Bullet;

newBullet.Parent = gameObject;

newBullet.Direction = -newBullet.transform.right;

newBullet.Color = bulletColor;

}

protected override void OnTriggerEnter2D(Collider2D collider)

{

Unit unit = collider.GetComponent<Unit>();

if (unit && unit is Character)

{

if (Mathf.Abs(unit.transform.position.x - transform.position.x) < 0.3F) ReceiveDamage();

else unit.ReceiveDamage();

}

}

}

**MoveableMonster:**

using System.Linq;

public class MoveableMonster : Monster

{

[SerializeField]

private float speed = 2.0F;

private Vector3 direction;

private SpriteRenderer sprite;

protected override void Awake()

{

sprite = GetComponentInChildren<SpriteRenderer>();

}

protected override void Start()

{

direction = transform.right;

}

protected override void Update()

{

Move();

}

protected override void OnTriggerEnter2D(Collider2D collider)

{

Unit unit = collider.GetComponent<Unit>();

if (unit && unit is Character)

{

if (Mathf.Abs(unit.transform.position.x - transform.position.x) < 0.3F) ReceiveDamage();

else unit.ReceiveDamage();

}

}

private void Move()

{

Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position + transform.up \* 0.5F + transform.right \* direction.x \* 0.5F, 0.1F);

if (colliders.Length > 0 && colliders.All(x => !x.GetComponent<Character>())) direction \*= -1.0F;

transform.position = Vector3.MoveTowards(transform.position, transform.position + direction, speed \* Time.deltaTime);

}

}

**Obstacle:**

public class Obstacle : MonoBehaviour

{

private void OnTriggerEnter2D(Collider2D collider)

{

Unit unit = collider.GetComponent<Unit>();

if (unit && unit is Character)

{

unit.ReceiveDamage();

}

}

}

**Heart:**

using UnityEngine;

using System.Collections;

public class Heart : MonoBehaviour

{

private void OnTriggerEnter2D(Collider2D collider)

{

Character character = collider.GetComponent<Character>();

if (character)

{

character.Lives++;

Destroy(gameObject);

}

}

}

**LiveBar:**

public class LivesBar : MonoBehaviour

{

private Transform[] hearts = new Transform[5];

private Character character;

private void Awake()

{

character = FindObjectOfType<Character>();

for (int i = 0; i < hearts.Length; i++)

{

hearts[i] = transform.GetChild(i);

Debug.Log(hearts[i]);

}

}

public void Refresh()

{

for (int i = 0; i < hearts.Length; i++)

{

if (i < character.Lives) hearts[i].gameObject.SetActive(true);

else hearts[i].gameObject.SetActive(false);

}

}

}

**CameraController:**

public class CameraController : MonoBehaviour

{

[SerializeField]

private float speed = 2.0F;

[SerializeField]

private Transform target;

private void Awake()

{

if (!target) target = FindObjectOfType<Character>().transform;

}

private void Update()

{

Vector3 position = target.position; position.z = -10.0F;

transform.position = Vector3.Lerp(transform.position, position, speed \* Time.deltaTime);

}

}

**Bullet:**

public class Bullet : MonoBehaviour

{

private GameObject parent;

public GameObject Parent { set { parent = value; } get { return parent; } }

private float speed = 10.0F;

private Vector3 direction;

public Vector3 Direction { set { direction = value; } }

public Color Color

{

set { sprite.color = value; }

}

private SpriteRenderer sprite;

private void Awake()

{

sprite = GetComponentInChildren<SpriteRenderer>();

}

private void Start()

{

Destroy(gameObject, 1.4F);

}

private void Update()

{

transform.position = Vector3.MoveTowards(transform.position, transform.position + direction, speed \* Time.deltaTime);

}

private void OnTriggerEnter2D(Collider2D collider)

{

Unit unit = collider.GetComponent<Unit>();

if (unit && unit.gameObject != parent)

{

Destroy(gameObject);

}

}

}

**Buttons:**

using UnityEngine.SceneManagement;

public class Buttons : MonoBehaviour {

private void OnMouseDown()

{

if (gameObject.name == "Exit")

{

Application.Quit();

}

if (gameObject.name == "Restart")

{

SceneManager.LoadScene("Level");

}

if (gameObject.name == "Play")

{

SceneManager.LoadScene("Level");

}

}

}

**LevelEnd:**

using UnityEngine.SceneManagement;

public class LevelEnd : MonoBehaviour {

private void OnTriggerEnter2D(Collider2D collider)

{

Character character = collider.GetComponent<Character>();

if (character)

{

SceneManager.LoadScene("LevelEnd");

}

}

}