|  |
| --- |
| Распоряжение Правительства Российской Федерации от 27 декабря 2010 г. №2387-р  Федеральное государственное бюджетное образовательное учреждение высшего образования  **«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА И ГОСУДАРСТВЕННОЙ СЛУЖБЫ ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**  КОЛЛЕДЖ МНОГОУРОВНЕВОГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ |

**К У Р С О В О Й П Р О Е К Т**

**Тема: Проектирование и разработка Компьютерной игры “Night City”**

**Специальность: 09.02.07 «Информационные системы и программирование»**

**Исполнитель: студент группы 31ИС-20 Есаян А. Э.**

**Руководитель: Гусятинер Л. Б.**

**Москва 2022**

Оглавление

[1. Аналитический раздел 4](#_Toc120070901)

[1.1Предпроектное обследование разработки игрового приложения 4](#_Toc120070902)

[1.1.2 Хоррор 5](#_Toc120070903)

[1.1.3 Шутер 5](#_Toc120070904)

[1.1.4 Battle Royale 6](#_Toc120070905)

[1.2 Выбор языка программирования 7](#_Toc120070906)

[1.2.1 C# 7](#_Toc120070907)

[1.2.2 C++ 7](#_Toc120070908)

[1.2.3 Java 8](#_Toc120070909)

[1.3 Выбор IDE 8](#_Toc120070910)

[1.3.1 Visual Studio 9](#_Toc120070911)

[1.3.2 Eclipse 9](#_Toc120070912)

[1.3.3 Visual Studio Code 9](#_Toc120070913)

[1.4 Выбор игрового движка 10](#_Toc120070914)

[1.4.1 Unity 11](#_Toc120070915)

[1.4.2 Unreal Engine 11](#_Toc120070916)

[1.4.3 Godot 11](#_Toc120070917)

[1.5 Постановка задачи 12](#_Toc120070918)

[2. Проектный раздел 12](#_Toc120070919)

[2.1 Разработка алгоритмов 12](#_Toc120070920)

[2.1.1 Передвижение 13](#_Toc120070921)

[2.1.2 Поднятие предмета 13](#_Toc120070922)

[2.1.3Прыжок 14](#_Toc120070923)

[2.2Разработка программного кода для программного продукта 14](#_Toc120070924)

[2.3 Тестирование и отладка программного продукта. 21](#_Toc120070925)

[Заключение 22](#_Toc120070926)

[Список используемых источников 23](#_Toc120070927)

**Введение**

Целью данного курсового проекта является написание игры «Night City». Эта тема является актуальной на данный момент, потому что в наше время пользователям компьютеров нравится играть в игры. Данный курсовой проект поможет сбросить стресс, расширить кругозор, логическому развитию. Так же в данной игре будет простой для понимания интерфейс.

Игровые приложения — неотъемлемая часть жизни современных подростков, в том числе и логических. Игровые приложения используются для развития и развлечения людей всех возрастов. Игровые приложения набирают все большую популярность. Киберспортивные соревнования (компьютерные игровые приложения) собирают столько же зрителей, сколько и футбольные матчи. Выставки игровых приложений проходят постоянно. Самая популярная из них — Electronic Entertainment Expo (E3). В частности, приложения для логических игр очень популярны, и есть целые компании, основной целью которых является разработка игровых приложений.

# **1. Аналитический раздел**

## **1.1 Предпроектное обследование разработки игрового приложения**

Предпроектное исследование игрового приложения — это первый этап, с которого начинается формирование игры.

Этап включает в себя:

1)Анализ основных жанров компьютерных игровых приложений, представленных на рынке.

2)Анализ различий между 2D и 3D компьютерными игровыми приложениями.

3)Выбор языка.

4)Выбор IDE

По результатам предпроектного исследования определяются и утверждаются основные параметры реализации проекта: постановка задачи, способ реализации проекта.

Выбор лидирующих жанров в игровой индустрии

Жанры в игровой индустрии — это классификация игровых приложений, разделяющая игры по их визуальной, сюжетной и механической составляющим.

Анализируя ведущие жанры компьютерных игровых приложений, были выделены три подходящих жанра:

Хоррор;

Шутер;

Королевская битва.

## **1.1.2 Хоррор**

Чаще всего жанр представляется в 3D графике (рисунок 1), нежели в 2D и подразумевает под собой ужастик.

Рисунок 1 - Пример Симулятор выживания.

 Жанр [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), основанный на жанре [ужасов](https://ru.wikipedia.org/wiki/%D0%A3%D0%B6%D0%B0%D1%81%D1%8B_(%D0%B6%D0%B0%D0%BD%D1%80)) и обычно предназначенный для того, чтобы напугать игрока. В отличие от большинства других жанров видеоигр, которые классифицируются по их [игровому процессу](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%B9%D0%BC%D0%BF%D0%BB%D0%B5%D0%B9), игры ужасов почти всегда основаны на повествовании или визуальном представлении и используют различные типы игрового процесса.

## **1.1.3 Шутер**

Жанр игр, который может представляться в любой графике, как 2D, так и в 3D. (рисунок 2).



Рисунок 2 - Пример шутера

Представляет из себя стрельбу, зачастую приходится стрелять по игровому искусственному интеллекту, либо по неподвижным мишеням, что в нынешнее время редко встречается.

## **1.1.4 Battle Royale**

Жанр, достигший пика популярности за 3 года, который подразумевает выживание среди большого количества людей (от 64 до 100 человек) с помощью оружия, остаться единственным выжившим на большой по масштабам территории, высадившись на нее , используя для этого любые средства(рисунок3).



Рисунок . Пример Battle Royale

Жанр многогранен, так как, с ним могут сочетаться большинство остальных жанров, например, Шутер, Платформер, Строительство.

По итогам анализа представленных жанров игр было решено использовать для разработки игрового приложения жанр – Хоррор.

## **1.2 Выбор языка программирования**

Для создания более детализированного игрового приложения следует пользоваться наиболее комфортным и быстрым языком программирования.

Под подобные требования подходят сразу 3, похожие между собой, языка программирования: C#, C++, Java. Проанализировав каждый из них будет выявлен наиболее подходящий язык программирования для разработки игрового приложения.

## **1.2.1 C#**

Является языком программирования, относящийся к семье с С–подобным синтаксисом (C++, Java).

Получил многие функции от своих предшественников, что делает его современнее, а значит и более удобнее для программирования, учитывая все недостатки его старших собратьев.

## **1.2.2 C++**

В семействе является одним из самых популярных языков программирования, поскольку широко используется для разработки программного обеспечения.

C++ настолько многофункционален, что область его применения включает в себя: создание ОС, разнообразных прикладных программ, драйверов устройств и так далее.

Даже учитывая свою многофункциональность, язык программирования не позволяет достичь нужных целей в “пару шагов” и является недостаточно удобным для написания больших программ, от чего требуется проверять каждую строку, для выявления проблемы.

## **1.2.3 Java**

Служит строго типизированным языком. Позволяет выполнять Java–приложения на любом устройстве, для которого существует виртуальная машина, что делает язык программирования особенным, относительно других.

Еще одной особенностью является гибкая система безопасности, выполнение программы полностью контролируется виртуальной машиной, например, если будет попытка несанкционированного доступа к данным, то установленные пользователем разрешения в программе немедленно вызовут прерывание. Сравнение трех языков программирования представлено ниже в виде таблицы для того, чтобы выбрать один из них (Таблица 1).

Таблица 1 - Сравнение языков программирования

|  |  |  |  |
| --- | --- | --- | --- |
| **Характеристики** | **C#** | **C++** | **Java** |
| Удобство в написании | + | – | + |
| Простота в освоении | + | – | – |
| Понятность кода | + | + | + |

Проанализировав каждый, выбранный язык программирования C#, в связи с его удобством в написании кодов и современности, относительно других членов семейства.

## **1.3 Выбор IDE**

Среда разработки – это комплекс программных средств для разработки программного обеспечения.

Учитывая выбор языка программирования, необходимо также выбрать среду для разработки приложения для компьютерной игры. Для таких требований на рынке огромный выбор. Перед выбором следует проанализировать каждую из сред разработки, чтобы полностью понять, на что похожи выбранные.

## **1.3.1 Visual Studio**

С Visual Studio многие начинали работать и до сих пор работают в нем. Связано это с тем, что он является самой удобной средой разработки среди остальных, поскольку данный способ создания программного обеспечения был создан Microsoft.

Официальная, бесплатная, многофункциональная, поддерживает платформы .NET, имеет широкие возможности по разработке приложений для Windows. Так же огромным преимуществом для VS является удобный и простой интерфейс.

## **1.3.2 Eclipse**

Является одной из самых популярных мультиязычных сред, которая ориентирована на разработку Java-приложений, также позволяет работать с кодами на C#.

Для Eclipse имеются множества плагинов способные помочь, благодаря тем же плагинам имеется возможность для визуальной, и не только, кастомизации. Данная среда так же является бесплатной, что позволяет пользоваться ей любому желающему, но при этом сложная для начинающих пользователей и не дает гарантий надежности, так как плагины создаются сообществом Eclipse.

## **1.3.3 Visual Studio Code**

Представляется бесплатным, легковесным, кроссплатформенным редактором кода. Для доведения его до статуса IDE пользователи используют множества плагинов, что позволяет воспользоваться программой в полной мере.

Учитывая низкий функционал и сомнительную надежность (многие надстройки не всегда выполняют даже базовые функции), их редко доводят до максимальной эффективности в сфере разработки.

Ниже представлена ​​сравнительная таблица сред разработки игровых приложений. (таблица 2).

Таблица 2 - Сравнение сред разработки

|  |  |  |  |
| --- | --- | --- | --- |
| **Характеристики** | **Visual Studio** | **Eclipse** | **Visual Studio Code** |
| Удобство пользования | + | + | – |
| Интерфейс | + | – | + |
| Быстродействие | + | – | – |
| Обучаемость | – | + | + |
| Надежность | + | + | – |

Анализируя каждый из подходящий для разработки IDE был выявлен наиболее подходящий из них – Visual Studio

## **1.4 Выбор игрового движка**

Игровой движок — это программное обеспечение, лежащее в основе компьютерной игры. Говоря простым языком, это основное ядро ​​игры, на котором построены остальные компоненты. Это влияет на все элементы игры, от физики до онлайн-аспектов. При выборе игровых движков следует учитывать огромный глобальный ассортимент, доступный каждому пользователю Интернета и не требующий финансовых затрат. Наиболее подходящими для текущих условий двигателями являются:

Unity;

Unreal Engine;

Godot.

## **1.4.1 Unity**

Значится самым популярным игровым движком на сегодняшний день. На этом движке было создано огромное количество межплатформенных игровых приложений.

Движок обладает кроссплатформенностью, является бесплатным для пользования физических лиц, выбор готовых шаблонов больше, чем у остальных рассматриваемых движков вместе взятых. К тому же обучаемость для работы на движке не вызывает трудностей – на просторе Интернета множество обучающих материалов, а также имеет большие возможности как в 2D, так и в 3D.

## **1.4.2 Unreal Engine**

Unreal Engine является движком, повлиявшим на всю игровую индустрию шутеров своей детализацией в разработке игровых приложений. Используя данный движок следует учесть, что игровое приложение будет написано на языке программирования C++, хоть этого и можно избежать благодаря особенностям данной платформы.

Движок предлагает отличную графику, хорошие инструменты для моделирования уровней, большое количество источников для обучения работы с ним.

Платформа предоставляется бесплатной для физических лиц, готовых шаблонов меньше, чем у Unity.

## **1.4.3 Godot**

Современный движок, являющийся достойным претендентом на сравнение с Unity в разработке 2D приложений, большая часть интерфейса была русифицирована, увеличивая в разы круг людей, желающих обучиться разработке игровых кроссплатформенных приложений.

Является бесплатным, поддерживает несколько языков программирования, таких как: GDScript, C++, а C# имеет неполную поддержку. Ниже приведена таблица для сравнения игровых движков (таблица 3).

Таблица 3 – Сравнение игровых движков

|  |  |  |  |
| --- | --- | --- | --- |
| **Характеристики** | **Unity** | **Unreal Engine** | **Godot** |
| Понятный интерфейс | + | - | + |
| Удобство пользования | + | + | – |
| Наличие ассетов | + | + | – |
| Кроссплатформенность | + | + | – |
| Русификация | – | – | + |

По результатам анализа был определен игровой движок Unity, подходящий для создания 3D-приложения. Ведь в нем огромное количество доступных ресурсов и удобный и интуитивно понятный интерфейс, что является огромным преимуществом перед конкурентами.

## **1.5 Постановка задачи**

Целью создания проекта является написание игры и приобретение навыков 3д моделирования. Также система переключения между камерами и правильное обращение с объектами в иерархии приложения.

# **2. Проектный раздел**

## 2.1 Разработка алгоритмов

При разработке игрового приложения требуется определить ряд функций, которые оно должно выполнять на протяжении активности. В игровом приложении “Night City” используются 3 функции для достижения рабочего состояния программы:

* передвижение;
* прыжок;
* смена камер;
* триггеры игровой сцены;

**2.1.1 Передвижение**

Для грамотного передвижения персонажа по внутриигровому миру, для назначения скорости перемещения и определения точки соприкосновения с платформой требуется применить физику игрового движка.

Используя в Unity среду разработки C# единицей измерения является – f (например, 1f). За начальное значение в переменных берется 0 для грамотной корректировки функций (рисунок 4)

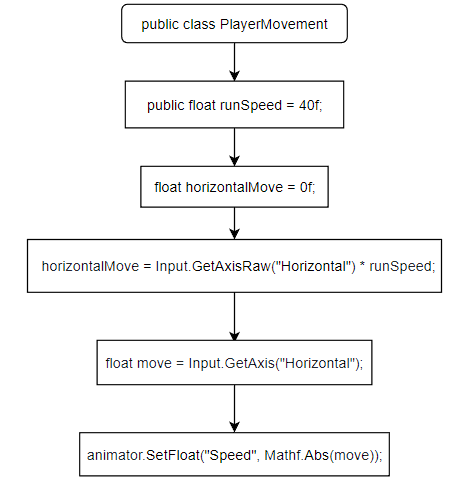
.**

Рисунок 4 – Блок-схема алгоритмов

## **2.1.2 Скримеры**

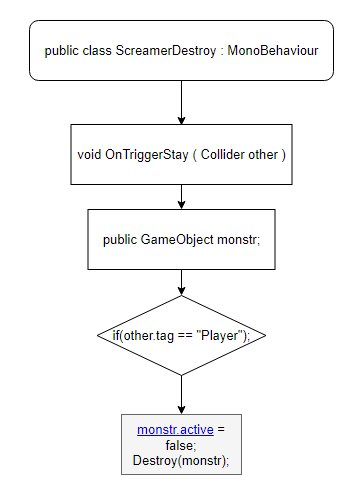
Так как жанр игрового приложения хоррор – в игре должны присутствовать скримеры, нужно будет сделать триггер в радиусе которого он будет срабатывать. (рисунок 5).

Рисунок 5 – Блок-схема алгоритма луча.

## 

## **2.1.3 Прыжок**

Так же для грамотного передвижения в игровом мире потребуется настроить прыжок персонажа для преодоления препятствий (Рисунок 7)

Функция прыжка осуществляется нажатием на кнопку “Space” на клавиатуре.

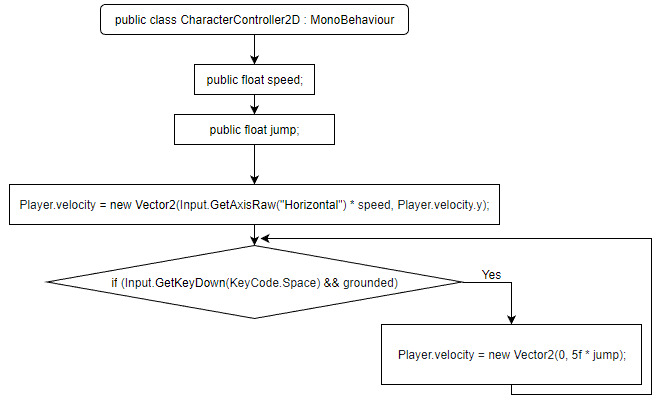


Рисунок 6 – Блок-схема алгоритмов прыжка

## **2.2 Разработка программного кода для программного продукта**

Зачастую пользователи не знают о том, что находится в корне программы, какая использовалась среда разработки, как был написан код.

В данном разделе будет рассматриваться код и его написание. Для грамотного создания персонажа требуется два кода, для контроля им и для обозначения полного здоровья, получения им урона. Контроль персонажем (передвижение и прыжок) осуществляется при помощи данного кода:

using System;  
using UnityEngine;  
using UnityStandardAssets.CrossPlatformInput;  
using UnityStandardAssets.Utility;  
using Random = UnityEngine.Random;  
  
[#pragma](https://vk.com/im?sel=357113956&st=%23pragma) warning disable 618, 649  
namespace UnityStandardAssets.Characters.FirstPerson  
{  
[RequireComponent(typeof (CharacterController))]  
[RequireComponent(typeof (AudioSource))]  
public class FirstPersonController : MonoBehaviour  
{  
[SerializeField] private bool m\_IsWalking;  
[SerializeField] private float m\_WalkSpeed;  
[SerializeField] private float m\_RunSpeed;  
[SerializeField] [Range(0f, 1f)] private float m\_RunstepLenghten;  
[SerializeField] private float m\_JumpSpeed;  
[SerializeField] private float m\_StickToGroundForce;  
[SerializeField] private float m\_GravityMultiplier;  
[SerializeField] private MouseLook m\_MouseLook;  
[SerializeField] private bool m\_UseFovKick;  
[SerializeField] private FOVKick m\_FovKick = new FOVKick();  
[SerializeField] private bool m\_UseHeadBob;  
[SerializeField] private CurveControlledBob m\_HeadBob = new CurveControlledBob();  
[SerializeField] private LerpControlledBob m\_JumpBob = new LerpControlledBob();  
[SerializeField] private float m\_StepInterval;  
[SerializeField] private AudioClip[] m\_FootstepSounds; // an array of footstep sounds that will be randomly selected from.  
[SerializeField] private AudioClip m\_JumpSound; // the sound played when character leaves the ground.  
[SerializeField] private AudioClip m\_LandSound; // the sound played when character touches back on ground.  
  
private Camera m\_Camera;  
private bool m\_Jump;  
private float m\_YRotation;  
private Vector2 m\_Input;  
private Vector3 m\_MoveDir = [Vector3.zero](https://vk.com/away.php?to=http%3A%2F%2FVector3.zero&cc_key=);  
private CharacterController m\_CharacterController;  
private CollisionFlags m\_CollisionFlags;  
private bool m\_PreviouslyGrounded;  
private Vector3 m\_OriginalCameraPosition;  
private float m\_StepCycle;  
private float m\_NextStep;  
private bool m\_Jumping;  
private AudioSource m\_AudioSource;  
  
// Use this for initialization  
private void Start()  
{  
m\_CharacterController = GetComponent<CharacterController>();  
m\_Camera = Camera.main;  
m\_OriginalCameraPosition = m\_Camera.transform.localPosition;  
m\_FovKick.Setup(m\_Camera);  
m\_HeadBob.Setup(m\_Camera, m\_StepInterval);  
m\_StepCycle = 0f;  
m\_NextStep = m\_StepCycle/2f;  
m\_Jumping = false;  
m\_AudioSource = GetComponent<AudioSource>();  
m\_MouseLook.Init(transform , m\_Camera.transform);  
}  
  
  
// Update is called once per frame  
private void Update()  
{  
RotateView();  
// the jump state needs to read here to make sure it is not missed  
if (!m\_Jump)  
{  
m\_Jump = CrossPlatformInputManager.GetButtonDown("Jump");  
}  
  
if (!m\_PreviouslyGrounded && m\_CharacterController.isGrounded)  
{  
StartCoroutine(m\_JumpBob.DoBobCycle());  
PlayLandingSound();  
m\_MoveDir.y = 0f;  
m\_Jumping = false;  
}  
if (!m\_CharacterController.isGrounded && !m\_Jumping && m\_PreviouslyGrounded)  
{  
m\_MoveDir.y = 0f;  
}  
  
m\_PreviouslyGrounded = m\_CharacterController.isGrounded;  
}  
  
  
private void PlayLandingSound()  
{  
m\_AudioSource.clip = m\_LandSound;  
[m\_AudioSource.Play](https://vk.com/away.php?to=http%3A%2F%2Fm_AudioSource.Play&cc_key=)();  
m\_NextStep = m\_StepCycle + .5f;  
}  
  
  
private void FixedUpdate()  
{  
float speed;  
GetInput(out speed);  
// always move along the camera forward as it is the direction that it being aimed at  
Vector3 desiredMove = transform.forward\*m\_Input.y + transform.right\*m\_Input.x;  
  
// get a normal for the surface that is being touched to move along it  
RaycastHit hitInfo;  
Physics.SphereCast(transform.position, m\_CharacterController.radius, Vector3.down, out hitInfo,  
m\_CharacterController.height/2f, Physics.AllLayers, QueryTriggerInteraction.Ignore);  
desiredMove = Vector3.ProjectOnPlane(desiredMove, hitInfo.normal).normalized;  
  
m\_MoveDir.x = desiredMove.x\*speed;  
m\_MoveDir.z = desiredMove.z\*speed;  
  
  
if (m\_CharacterController.isGrounded)  
{  
m\_MoveDir.y = -m\_StickToGroundForce;  
  
if (m\_Jump)  
{  
m\_MoveDir.y = m\_JumpSpeed;  
PlayJumpSound();  
m\_Jump = false;  
m\_Jumping = true;  
}  
}  
else  
{  
m\_MoveDir +=

Physics.gravity\*m\_GravityMultiplier\*Time.fixedDeltaTime;  
}  
m\_CollisionFlags = m\_CharacterController.Move(m\_MoveDir\*Time.fixedDeltaTime);  
  
ProgressStepCycle(speed);  
UpdateCameraPosition(speed);  
  
m\_MouseLook.UpdateCursorLock();  
}  
  
  
private void PlayJumpSound()  
{  
m\_AudioSource.clip = m\_JumpSound;  
[m\_AudioSource.Play](https://vk.com/away.php?to=http%3A%2F%2Fm_AudioSource.Play&cc_key=)();  
}  
  
  
private void ProgressStepCycle(float speed)  
{  
if (m\_CharacterController.velocity.sqrMagnitude > 0 && (m\_Input.x != 0 || m\_Input.y != 0))  
{  
m\_StepCycle += (m\_CharacterController.velocity.magnitude + (speed\*(m\_IsWalking ? 1f : m\_RunstepLenghten)))\*  
Time.fixedDeltaTime;  
}  
  
if (!(m\_StepCycle > m\_NextStep))  
{  
return;  
}  
  
m\_NextStep = m\_StepCycle + m\_StepInterval;  
  
PlayFootStepAudio();  
}  
  
  
private void PlayFootStepAudio()  
{  
if (!m\_CharacterController.isGrounded)  
{  
return;  
}  
// pick & play a random footstep sound from the array,  
// excluding sound at index 0  
int n = Random.Range(1, m\_FootstepSounds.Length);  
m\_AudioSource.clip = m\_FootstepSounds[n];  
m\_AudioSource.PlayOneShot(m\_AudioSource.clip);  
// move picked sound to index 0 so it's not picked next time  
m\_FootstepSounds[n] = m\_FootstepSounds[0];  
m\_FootstepSounds[0] = m\_AudioSource.clip;  
}  
  
  
private void UpdateCameraPosition(float speed)  
{  
Vector3 newCameraPosition;  
if (!m\_UseHeadBob)  
{  
return;  
}  
if (m\_CharacterController.velocity.magnitude > 0 && m\_CharacterController.isGrounded)  
{  
m\_Camera.transform.localPosition =  
m\_HeadBob.DoHeadBob(m\_CharacterController.velocity.magnitude +  
(speed\*(m\_IsWalking ? 1f : m\_RunstepLenghten)));  
newCameraPosition = m\_Camera.transform.localPosition;  
newCameraPosition.y = m\_Camera.transform.localPosition.y - m\_JumpBob.Offset();  
}  
else  
{  
newCameraPosition = m\_Camera.transform.localPosition;  
newCameraPosition.y = m\_OriginalCameraPosition.y - m\_JumpBob.Offset();  
}  
m\_Camera.transform.localPosition = newCameraPosition;  
}  
  
  
private void GetInput(out float speed)  
{  
// Read input  
float horizontal = CrossPlatformInputManager.GetAxis("Horizontal");  
float vertical = CrossPlatformInputManager.GetAxis("Vertical");  
  
bool waswalking = m\_IsWalking;  
  
[#if](https://vk.com/im?sel=357113956&st=%23if) !MOBILE\_INPUT  
// On standalone builds, walk/run speed is modified by a key press.  
// keep track of whether or not the character is walking or running  
m\_IsWalking = !Input.GetKey(KeyCode.LeftShift);  
[#endif](https://vk.com/im?sel=357113956&st=%23endif)  
// set the desired speed to be walking or running  
speed = m\_IsWalking ? m\_WalkSpeed : m\_RunSpeed;  
m\_Input = new Vector2(horizontal, vertical);  
  
// normalize input if it exceeds 1 in combined length:  
if (m\_Input.sqrMagnitude > 1)  
{  
m\_Input.Normalize();  
}  
  
// handle speed change to give an fov kick  
// only if the player is going to a run, is running and the fovkick is to be used  
if (m\_IsWalking != waswalking && m\_UseFovKick && m\_CharacterController.velocity.sqrMagnitude > 0)  
{  
StopAllCoroutines();  
StartCoroutine(!m\_IsWalking ? m\_FovKick.FOVKickUp() : m\_FovKick.FOVKickDown());  
}  
}  
  
  
private void RotateView()  
{  
m\_MouseLook.LookRotation (transform, m\_Camera.transform);  
}  
  
  
private void OnControllerColliderHit(ControllerColliderHit hit)  
{  
Rigidbody body = hit.collider.attachedRigidbody;  
//dont move the rigidbody if the character is on top of it  
if (m\_CollisionFlags == CollisionFlags.Below)  
{  
return;  
}  
  
if (body == null || body.isKinematic)  
{  
return;  
}  
body.AddForceAtPosition(m\_CharacterController.velocity\*0.1f, hit.point, ForceMode.Impulse);  
}  
}  
}

**Для активации скримера:**

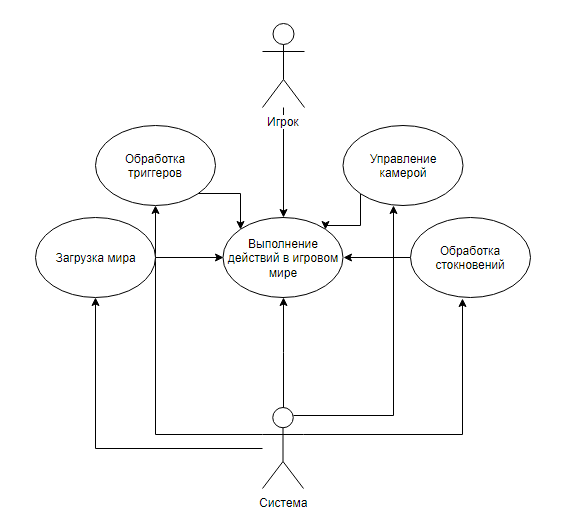
using UnityEngine;  
using System.Collections;  
  
public class ScreamerActive : MonoBehaviour {  
public GameObject monstr;  
  
  
void OnTriggerStay ( Collider other ){  
if(other.tag == "Player");  
[monstr.active](https://vk.com/away.php?to=http%3A%2F%2Fmonstr.active&cc_key=) = true;  
  
}  
}

**Искусственный интеллект врага:**

using UnityEngine;  
using System.Collections;  
  
public class AI\_Ray : MonoBehaviour {  
  
private Transform Player;  
private UnityEngine.AI.NavMeshAgent NMA;  
  
void Start () {  
Player = GameObject.FindGameObjectWithTag("Player").transform;  
NMA = (UnityEngine.AI.NavMeshAgent)this.GetComponent("NavMeshAgent");  
}  
  
  
void Update () {  
NMA.SetDestination(Player.position);  
  
  
  
}  
}

## **2.4 Диаграмма прецедентов**

В этом разделе представлена диаграмма прецедентов. На ней показаны все возможные функциональные и поведенческие отношения и взаимодействие игрока и системы. (Рисунок4)



## **2.4 Тестирование и отладка программного продукта.**

Готовый программный продукт проверяется вручную,

Ручная проверка игрового приложения “Night City” на работоспособность осуществляется при помощи множества использований всех функций. Для ручной проверки игрового приложения будут рассматриваться функции передвижения и проверка триггеров, поскольку они считаются основными в программе.

# 

# **Заключение**

В ходе курсового проекта на тему проектирование и разработка игры «Night City» в соответствии с техническим заданием сделано следующее:

* Проведено предпроектное обследование жанров игровых приложений, языков программирования, среды разработки и игрового движка;
* Проанализированы и выбраны подходящие для разработки игрового приложения жанры, языки программирования, среды разработки и игрового движка;
* Поставлены задачи к программному продукту в виде жанра игрового приложения и 3D–графики.
* Разработаны алгоритмы, код программного продукта и интерфейс пользователя;
* Протестированы основные функции игрового приложения.

# **Список используемых источников**

1. ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления: Стандартинформ, 2017

2. ГОСТ 7.1—2003 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления

3. ГОСТ 7.9—95 Система стандартов по информации, библиотечному и издательскому делу. Реферат и аннотация. Общие требования

4. ГОСТ 7.11—2004 (ИСО 832:1994) Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Сокращение слов и словосочетаний на иностранных европейских языках

5. ГОСТ 7.12—93 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Сокращение слов на русском языке. Общие требования и правила

6. ГОСТ 7.80—2000 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Заголовок. Общие требования и правила составления

7. ГОСТ 7.82—2001 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления

8. Гейг, М. А. Разработка игр на Unity 2018 за 24 часа / М. А. Гейг. [перевод с английского М. А. Райтмана], М.: Эксмо, 2020. – 464 с.

9. Гвоздева, В. А. Введение в специальность программиста / В. А. Гвоздева. – М.: ИД “ФОРУМ” – ИНФРА, 2007. – 207 c.

10. Godot [Электронный ресурс]. – Режим доступа: https://docs.godotengine.org/en/stable/ – Загл. с экрана.

11. Unity [Электронный ресурс]. – Режим доступа: https://unity.com/ru/learn – Загл. с экрана.

12. Unreal Engine [Электронный ресурс]. – Режим доступа: https://www.unrealengine.com/en-US/learn – Загл. с экрана.

13. Visual Studio [Электронный ресурс]. – Режим доступа: https://visualstudio.microsoft.com/ru/ – Загл. с экрана.

14. Eclipse [Электронный ресурс]. – Режим доступа: https://www.eclipse.org/ide/ свободный. – Загл. с экрана.

15. Visual Studio Code [Электронный ресурс]. – Режим доступа: https://code.visualstudio.com/learn свободный. – Загл. с экрана.