Федеральное государственное образовательное бюджетное учреждение
высшего образования
**«Финансовый университет при Правительстве
Российской Федерации»
(Финансовый университет)**
Колледж информатики и программирования
(наименование структурного подразделения)

Дипломный проект

Тема «Разработка компьютерной игры «Маленькие кошмары» в жанре платформер с элементами квеста и хоррора»
(наименование)

Студент Носкова Полина Александровна
(фамилия, имя, отчество полностью)

Учебная группа 4ПКС-120

Специальность 09.02.03 Программирование в компьютерных системах
(код и наименование специальности)

Руководитель дипломного проекта	_____ (подпись)	И. В. Сибирев (инициалы, фамилия)
Консультант дипломного проекта (при наличии)	_____ (подпись)	_____ (инициалы, фамилия)
Председатель предметной (цикловой) комиссии	_____ (подпись)	Т.Г. Аксёнова (инициалы, фамилия)

Москва – 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данного дипломного проекта является проектирование и разработка компьютерной игры «Маленькие кошмары» в жанре платформер с элементами квеста и хоррора.

Главной задачей проекта является создание интересной линейной компьютерной игры, включающей в себя разнообразие уровней, головоломок, препятствий и врагов с продвинутым искусственным интеллектом с различным набором поведений.

Для достижения поставленной цели необходимо решить ряд задач:

1) составить описание предметной области;
2) провести сравнительный анализ существующих программ-аналогов;
3) постановка задачи разработки программы;
4) составить характеристику инструментальных средств разработки;
5) осуществить проектирование и реализацию программы;
6) анализ требований и разработка спецификаций программы;
7) спроектировать программное обеспечение;
8) разработать программное обеспечение;
9) провести тестирование программного обеспечения;
10) создать руководство по использованию программы.

Данная тема являлась и продолжает быть актуальной, так как в настоящее время игры служат популярным развлечением для общества, предлагая игрокам различные жанры, уровни сложности, возможности совместного прохождения и многое другое. Таким образом, игры – неотъемлемая часть современной культуры, приносящие впечатления миллионам людей, а также являющиеся возможностью коммерческого дохода.

Жанр платформер сохраняет популярность по многим причинам, например, игры данного жанра не требуют особых навыков или сложных правил, потому игроки могут быстро начать играть и получать удовольствие без траты большого количества времени на обучение. Кроме того, платформеры часто предлагают разнообразные механики, потому каждая игра

является по своему уникальной. Ввиду характерной черты данного жанра, а именно наличие игровых уровней, каждый из которых становится сложнее предыдущего, в процессе прохождения происходит наращивание навыков игрока, что в последствии дает ощущение прогресса и делают игру увлекательной и разнообразной.

Объектом исследования являются компьютерные игры в жанре «платформер».

Предметом исследования являются средства разработки игры-платформера «Маленькие кошмары».

В ходе работы будут применяться следующие методы:

- анализ предметной области;
- анализ и сравнение существующих программ-аналогов;
- моделирование компьютерной игры на основе анализа требований.

Источниковой базой исследования будут литература по разработке игр на Unity, официальная документация Unity, онлайн-ресурсы, онлайн-форумы, видеоуроки по программированию, ГОСТы по оформлению документации, а также практические проекты и примеры.

Основным функциональным назначением игры на Unity является обеспечение увлекательного игрового процесса для пользователей. Главная цель разработки – предложить игрокам интересные сбалансированные по сложности уровни с уникальными и нестандартными головоломками и препятствиями.

Для разработки будут использованы язык программирования C#, среда разработки Microsoft Visual Studio, среда разработки игр Unity.

Таким образом, разработка компьютерной игры в жанре платформер представляет собой многогранный процесс, требующий не только технической грамотности и использование современных инструментов разработки компьютерных игр, но и творческого подхода, ведь, помимо программной реализации стоит важная задача придумать интересный сюжет, нестандартные головоломки и уровни с различными препятствиями.

Глава 1. Предпроектное исследование

1.1. Описание предметной области

В современном мире игры являются практически частью нашей жизни, становясь с каждым годом все более популярными и разнообразными. Актуальность игр обосновывается прежде всего тем, что для общества игра – развлечение. Игры позволяют отвлечься от забот реального мира и дать возможность погрузиться в увлекательный сюжет. Кроме того, ввиду разнообразия жанров, к примеру: платформер, приключенческие игры, аркада, шутер, симулятор, квест, RPG, стратегии и так далее, абсолютно каждый человек, вне зависимости от возраста, может найти интересную для него игру.

Создание игры «Маленькие кошмары» является актуальным и востребованным в настоящее время заданием, так как, прежде всего, представляет собой захватывающую компьютерную игру в жанре приключенческие игры. Данная игра - платформер с элементами квеста и хоррора. Подобные игры пользуются популярностью ввиду простого игрового процесса, как правило легкой и при этом увлекательной истории, атмосферы, разнообразия уровней и заданий.

Платформер – жанр компьютерных игр, в которых основу игрового процесса составляют прыжки по платформам, лазанье по лестницам, сбор предметов, необходимых для победы над врагами или завершения уровня [11].

Игра «Маленькие кошмары» рассчитана на игроков старше 16 лет ввиду нагнетающего саспенса в процессе игры.

Результатом выполнения данного дипломного проекта будет игра, представляющая собой мрачную сказку. Игроку будет необходимо помочь главному герою выбраться из западни.

1.2. Сравнительный анализ программ-аналогов

Рассмотрим и сравним существующие программные продукты: игры «Among the Sleep» и «Stray».

Игра «Among the Sleep» - приключенческая игра ужасов от первого лица, в которой игрок играет за маленькую девочку, проснувшуюся среди ночи от

зловещего шума и отправившуюся на поиски мамы. Функционал данной игры включает в себя исследования, головоломки, интерактивность, сюжет, атмосферу, уровни. Игрок может свободно перемещаться по игровому миру, находить ключевые предметы, которые в дальнейшем оказывают влияние на сюжет и помогает разгадывать головоломки. Кроме того, есть возможность взаимодействия, к примеру, с предметами, дверьми. Также игра имеет несколько уровней, отличающихся между собой местом, загадками, опасностями.

Интерфейс игры «Among the Sleep» минималистичный, направленный на погружение в атмосферу, а также на то, чтобы дать возможность почувствовать игроку ощутить себя в роли главного героя, то есть маленькой девочки. Вид в игре от первого лица, звуковое оформление игры также направлено на погружение игрока в атмосферу и в определенные моменты помогает поддерживать напряжение.

Стоимость данного продукта довольно низкая, кроме того, есть возможность сыграть в демоверсию бесплатно.

Достоинствами игры являются атмосфера, оригинальная задумка, сюжет, графика и звук, уникальность, игровой процесс.

К недостаткам игры можно отнести короткую продолжительность, так как игра может быть пройдена за пару часов, временами игра может показаться скучной и монотонной, а также ограниченной, ввиду того что главный герой – двухлетний ребенок. Технические проблемы также могут сказаться на общем впечатлении от игры.

Игра «Stray» - приключенческая инди-игра от третьего лица, в которой игрок должен помочь оторванному от семьи бродячему коту предстоит разгадать древнюю тайну, вырваться из давно заброшенного кибергорода и найти дорогу домой [12]. Функционал данной игры включает в себя исследования, головоломки, взаимодействие с окружающими объектами. В игре представлен проработанный мир с интересными деталями, в которых

игрок может путешествовать по городу, общаться с роботами и так далее. Все это позволяет игроку погрузиться в уникальный мир, узнать его историю.

Интерфейс игры интуитивно понятный, есть отличная навигация, не позволяющая игроку потеряться в процессе прохождения игры. Кроме того, освещение всегда помогает игроку. Интерфейс также направлен на полное погружение игрока в атмосферу.

Стоимость данного продукта довольно высокая, учитывая длительность прохождения и особенности функционала игры.

Достоинствами игры является уникальность истории, оригинальная концепция. Игрокам предлагают увлекательный геймплей, головоломки, исследование проработанного мира.

К основным недостаткам следует отнести, прежде всего, тот факт, что далеко не всем может понравиться идея игры от лица кота. Управление местами может быть слишком сложным или слишком простым, что в результате может заставить игрока заскучать и испортить общее впечатление от игры.

Подводя итоги, хочется сказать, что у игры «Among the Sleep» больше функционала и он гораздо увлекательнее, общая атмосфера от погружения в игру также дополняется различными моментами с элементами хоррора. Кроме того, в отличие от игры «Stray», в игре «Among the Sleep» у игрока потенциально больше возможностей, играя за маленького, но человека, в то же время в игре «Stray» коту помогает помощник-дрон. Тем не менее, в «Stray» из плюсов можно выделить, что игра от третьего лица, что для жанра платформер является несомненным преимуществом.

1.3. Постановка задачи

Для программной реализации необходимо определить жанр и сюжет разрабатываемой игры, а также входные и выходные данные.

Жанром разрабатываемой линейной игры является платформер с элементами квеста. Сюжет разворачивается в мрачном мире, где главной героине предстоит решить головоломки и пройти испытания, чтобы спастись.

Входными данными являются управление героем и взаимодействие с предметами при помощи клавиатуры и мыши. Выходными данными являются представление игрового мира на экране, анимация героя и врагов, подсказки, системы инвентаря и частиц, а также экран поражения.

Среди функциональных требований следует выделить управление персонажем, взаимодействие с окружающей средой, головоломки.

Игра «Маленькие кошмары» является однопользовательской, интерфейс интуитивно понятный и легкий для пользователя. Минимальными требованиями являются: операционная система Windows 7 (64-bit), оперативная память 4 ГБ ОЗУ.

1.4. Характеристика инструментальных средств разработки

В процессе дипломного проектирования будут использоваться различные инструментальные средства, описанные ниже.

Одной из возможностей создания игры является Unity. Данная среда разработки игр является популярным мультиплатформенным инструментом, так как позволяет создавать приложения, которые в результате смогут работать на более чем 25 различных платформах. Unity позволяет создать увлекательное, и, что самое главное, качественное игровое приложение.

C# является основным языком программирования, используемым в Unity. Он является высокоуровневым, объектно-ориентированным языком, который обеспечивает мощные возможности разработки игр.

В процессе проектирования создание необходимых диаграмм и схем будет осуществляться в plantuml.com и diagrams.net соответственно.

Для создания и оформления пояснительной записки и презентации к защите используются Microsoft Word и Microsoft PowerPoint соответственно.

Глава 2. Проектирование и реализация программы

2.1. Анализ требований и разработка спецификаций

На рисунке 1 представлена диаграмма последовательностей.

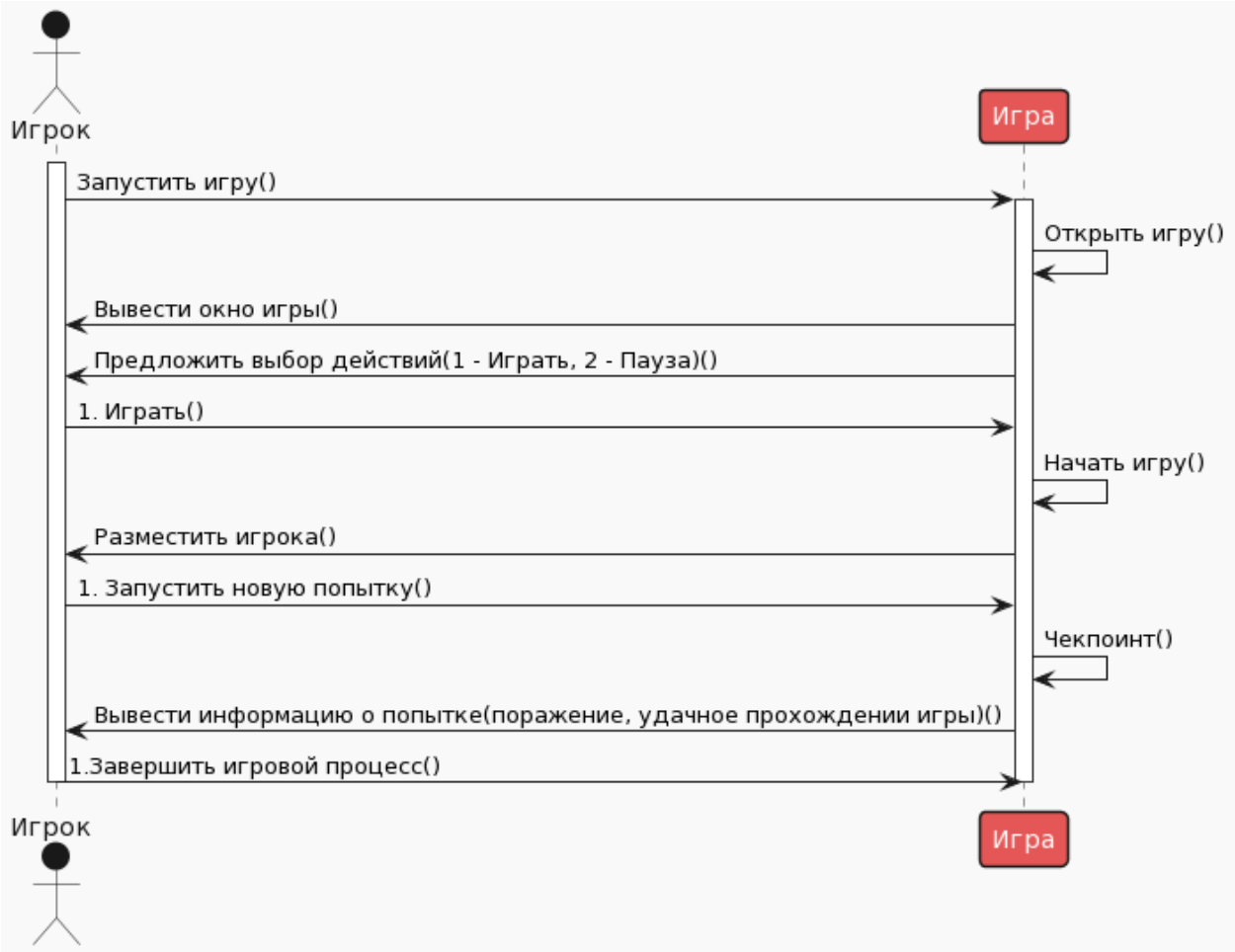

Рисунок 1. Диаграмма последовательностей

На диаграмме последовательностей показано, как игрок взаимодействует с игрой.

На рисунке 2 представлена диаграмма деятельности.

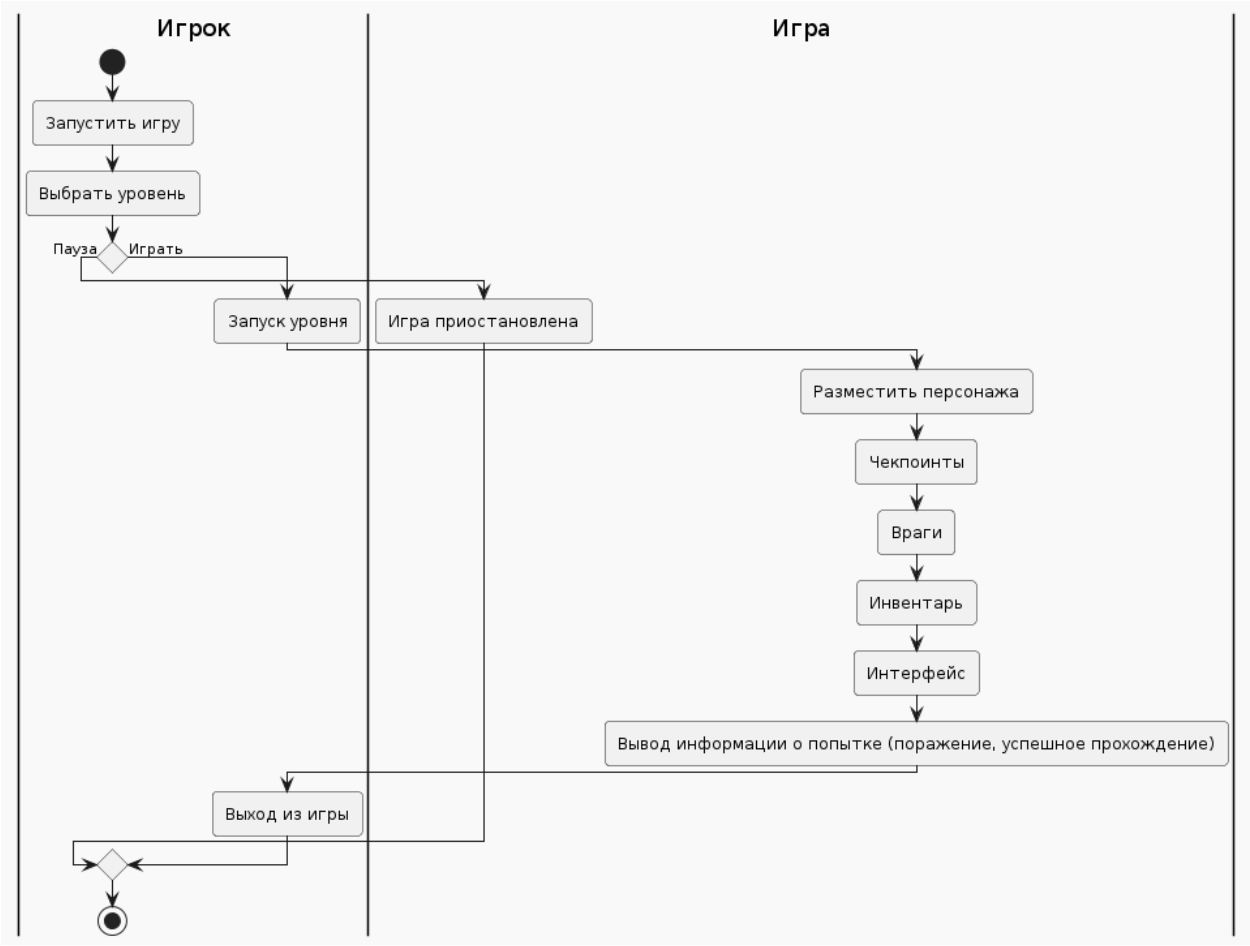

Рисунок 2. Диаграмма деятельности

На диаграмме деятельности наглядно показан рабочий процесс, логика программного продукта.

На рисунке 3 представлена диаграмма вариантов использования.

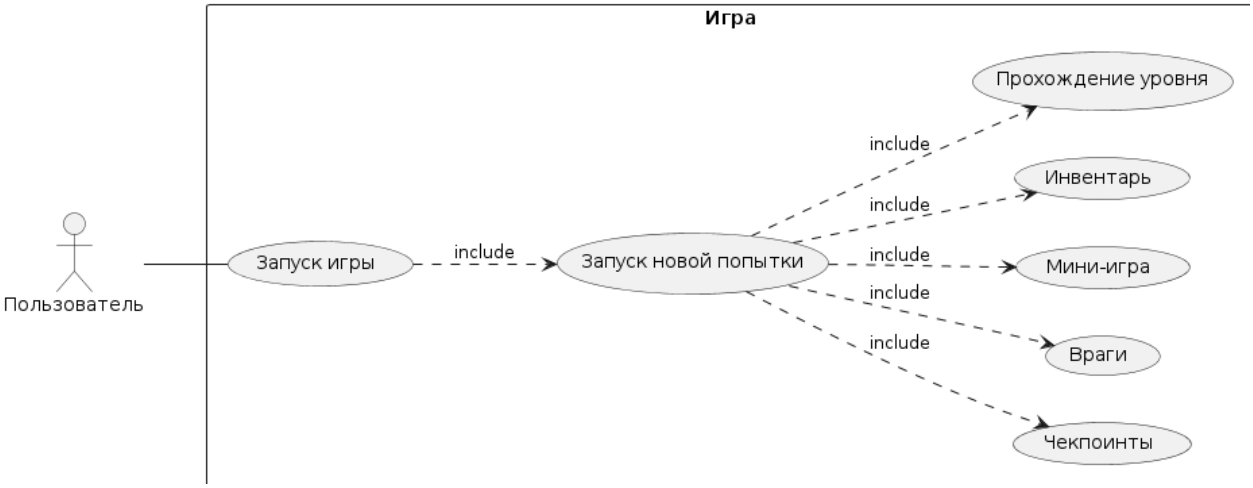

Рисунок 3. Диаграмма вариантов использования

На диаграмме вариантов использования отображены все возможные действия пользователя, а именно: запуск игры, который включает в себя

запуск новой попытки с возможность прохождения уровня, содержащего в себе головоломки ввиду жанра игры, чекпоинты, инвентарь, мини-игру, врагов.

2.2. Проектирование программного обеспечения

На рисунке 4 представлена структурная схема компьютерной игры «Маленькие кошмары».

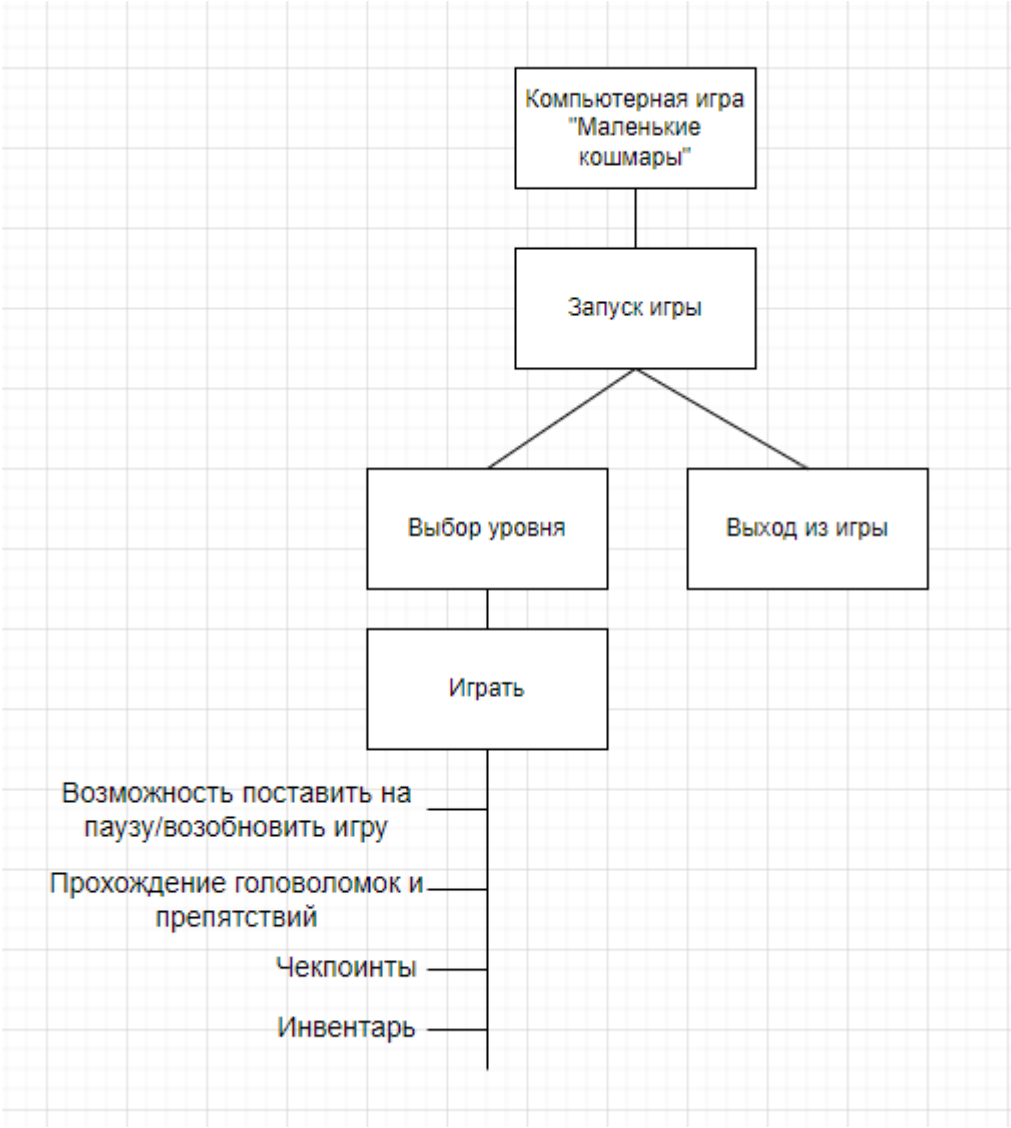

Рисунок 4. Структурная схема компьютерной игры «Маленькие кошмары»

На представленной структурной схеме отображено полное представление о проектируемой компьютерной игре.

На рисунке 5 представлена функциональная схема компьютерной игры «Маленькие кошмары».

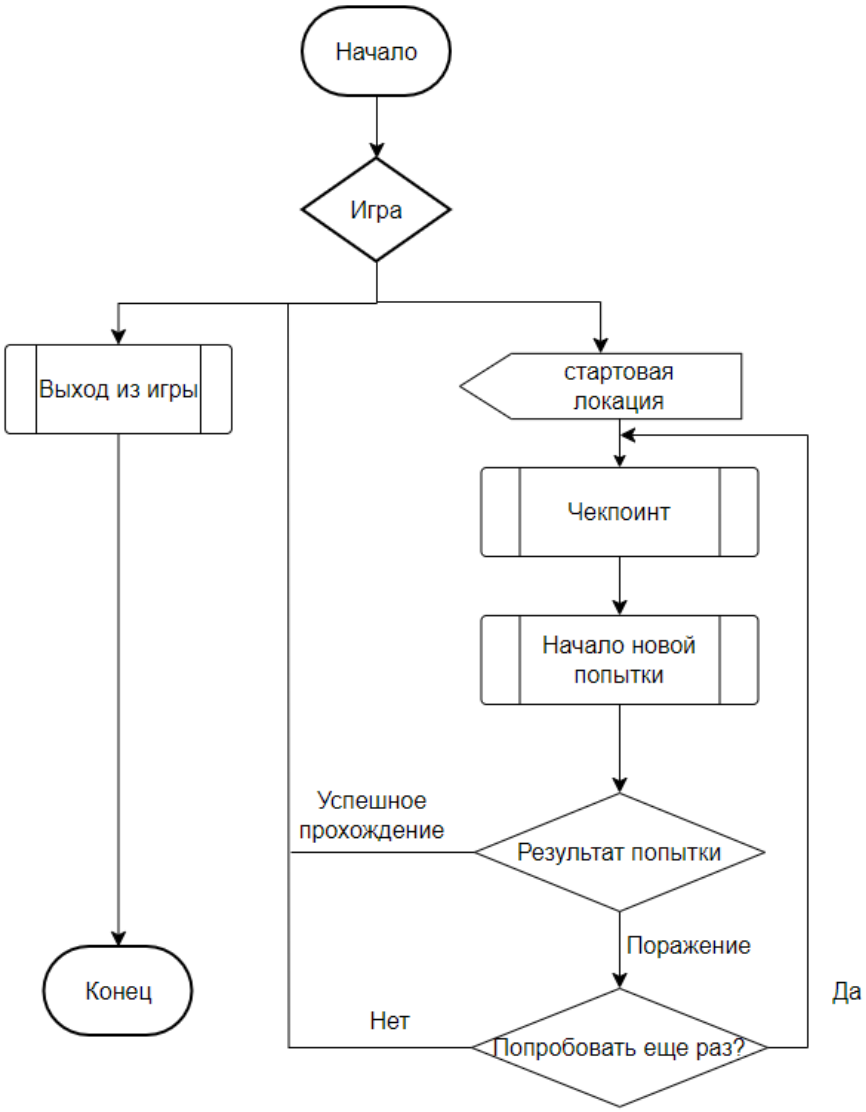

Рисунок 5. Функциональная схема компьютерной игры «Маленькие кошмары»

На представленной функциональной схеме отображено взаимодействие компонентов программного продукта с описанием информационных потоков.

2.3. Разработка программного обеспечения

Визуальная часть игры была реализована при помощи готовых, находящихся в открытом доступе материалов. Модель главного героя, а также его анимации были взяты с сайта mixamo.com. Материалы, использованные для создания сцены, в том числе предметы декора, были взяты из Unity Asset Store и Sketchfab.com.

Уровни были построены с помощью ProGrids и ProBuilder - двух мощных инструментов для Unity, предназначенных для улучшения процесса моделирования и построения.

Одним из основных функциональных требований является контроллер управления героем, в котором реализовано передвижение персонажа, включающее в себя ходьбу и бег при нажатии кнопки, одинарный прыжок, возможность красться, а также полностью настроенная анимация. Листинг программы представлен в приложении №1.

Для некоторых механик были созданы специальные проверки, например, в случае если герой крадется, он не сможет делать это быстро при помощи нажатия на клавишу «Shift» а в случае, если над его головой находится что-либо, что может помешать встать, у игрока не получится вернуть героя в исходное положение.

Кроме того, чтобы добавить анимацию, необходимо было создать Animator Controller, который представлен на рисунке 6.

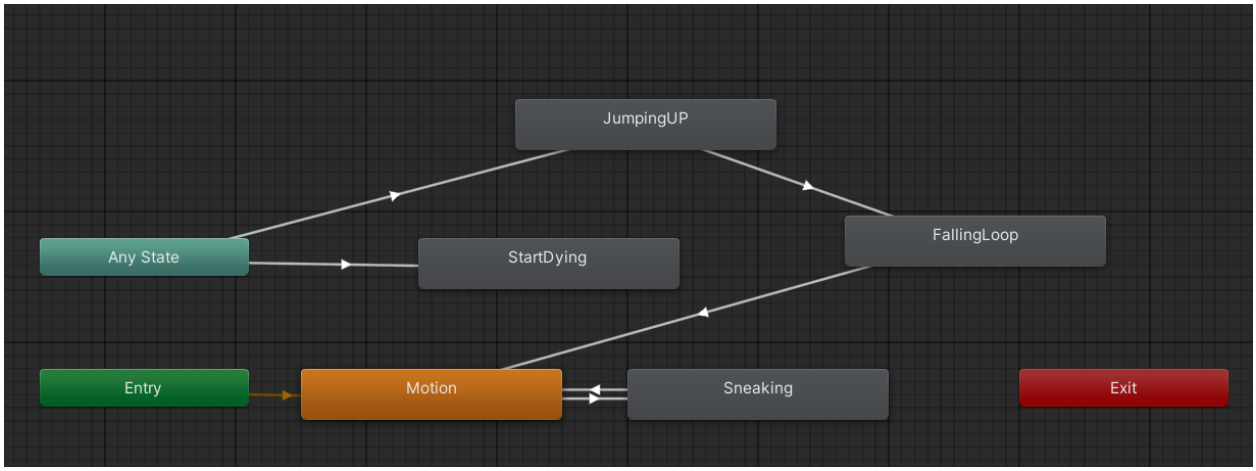

Рисунок 6. Animator Controller

В Blend Tree, который назван «Motion», настроены нейтральное положение героя, ходьба и бег. Более подробно это показано на рисунке 7.

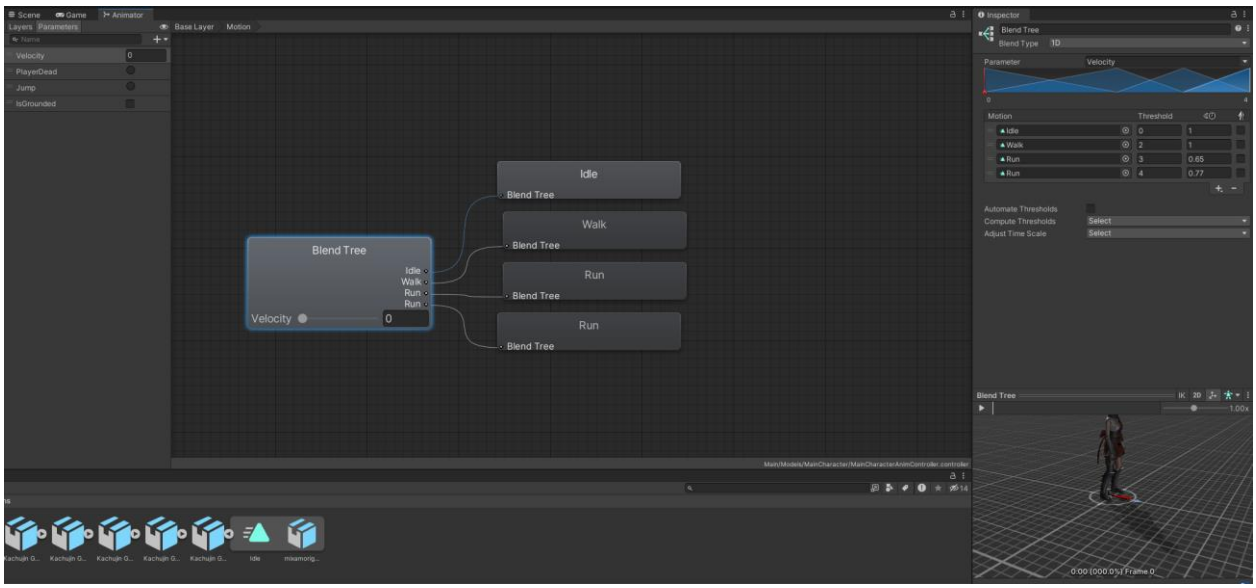

Рисунок 7. Blend Tree «Motion»

В Blend Tree, который назван «Sneaking», настроены нейтральное положение героя, когда он приседает, а также его ходьба, когда он крадется. Более подробно это показано на рисунке 8.

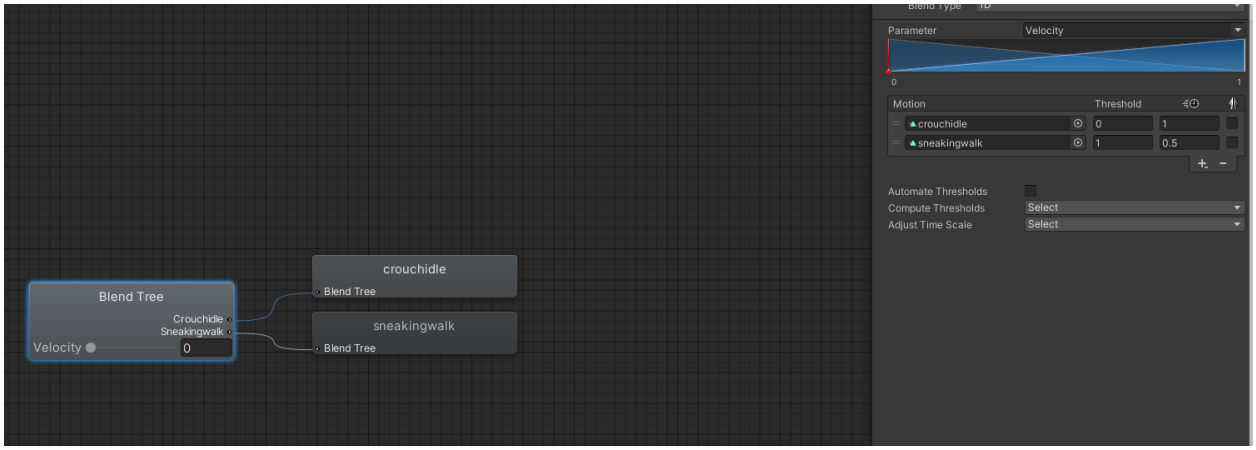

Рисунок 8. Blend Tree «Sneaking»

В приложении №2 представлен листинг программы, в котором прописано столкновение с врагом, анимация гибели, открытие экрана поражения, а также в случае падения с платформы герой не умирает, а возвращается к последнему чекпоинту. Кроме того, в данной программе реализовано взаимодействие с интерактивными предметами.

Листинг программы чекпоинта представлен ниже.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class CheckPoint : MonoBehaviour
{


    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            other.GetComponent<PlayerController>().lastCheckPoint = transform.position;
        }
    }
}
```

Игрок может поставить игру на паузу. Ниже представлен листинг программы.

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class PauseLogic : MonoBehaviour
{
    public bool isPaused = false;

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.H))
        {
            isPaused = !isPaused;
            if (isPaused)
            {
                Time.timeScale = 0;
            }
            else
            {
                Time.timeScale = 1;
            }
        }
    }
}
```

В первой головоломке игроку необходимо наступить на нажимные плиты, уворачиваясь от пилы, чтобы увидеть цифры, составляющие пароль для открытия двери, затем нужно взаимодействовать с панелью для ввода кода.

Код движения пилы представлен ниже.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SawScript : MonoBehaviour
{
    private bool flag = true;

    void Update()
    {

        if (transform.position.x <= 41.21f)
        {
            flag = false;
        }
        else if (transform.position.x >= 48.2f)
        {
            flag = true;
        }

        transform.Rotate(0, 0, 1f);
        Move();
    }

    private void Move()
    {
        if (flag)
        {
            transform.position += new Vector3(-0.005f, 0, 0) * 700 * Time.deltaTime;
        }
        else
        {
            transform.position += new Vector3(0.005f, 0, 0) * 700 * Time.deltaTime;
        }
    }
}
```

В случае столкновения с пилой, игра будет проиграна, появится экран с возможностью начать игру заново.

Листинг программы показан ниже.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneLoader : MonoBehaviour
{
    public void ReloadScene()
    {
```

```
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
}
```

Код взаимодействия игрока с нажимными плитами и отображение цифр представлен ниже.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlateWithText : MonoBehaviour
{
    public GameObject ObjectToActivate;

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            ObjectToActivate.SetActive(true);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            ObjectToActivate.SetActive(false);
        }
    }
}
```

В приложении №3 продемонстрирован листинг программы панели для ввода кода. В данной программе код-пароль задан заранее, а также есть проверка на неправильно введенный код, нельзя ввести больше четырех цифр, а также когда герой находится вблизи панели, игрок получает подсказку о том, как с ней взаимодействовать.

На рисунке 9 представлена панель для ввода кода.

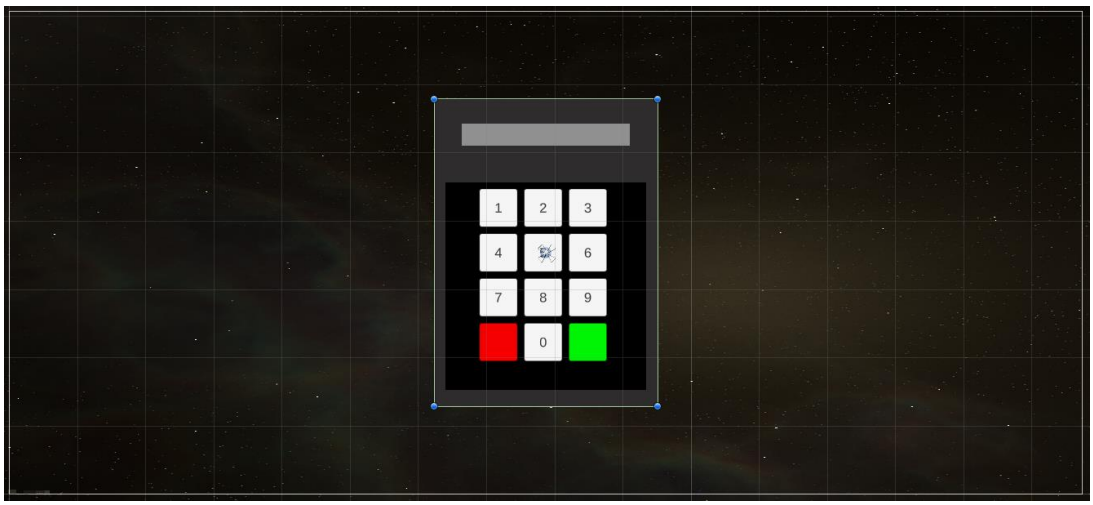

Рисунок 9. Панель для ввода кода

Листинг программы открытия двери представлен ниже.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Door : MonoBehaviour
{
    private bool isOpening = false;
    private Vector3 desiredPosition;
    private float speed = 2.5f;

    private void Start()
    {
        desiredPosition = transform.position - new Vector3(0, 4, 0);
    }

    public void OpenDoor()
    {
        isOpening = true;
    }

    private void Update()
    {
        if (!isOpening)
        {
            return;
        }

        transform.position = Vector3.Lerp(transform.position, desiredPosition, Time.deltaTime *
speed);
    }
}
```

В следующей головоломке игрок должен успеть пробежать через мост, который начинает раздвигаться, когда герой наступает на нажимную плиту. В случае, если игрок не успел, он может воспользоваться рычагом, чтобы снова соединить мост.

Ниже представлен листинг программы рычага.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Lever : MonoBehaviour
{
    public GameObject bridgeParts;
    public GameObject bridgePlate;
    public bool onStart = true;
    public bool canUse = false;
    private Vector3 startRotation = new Vector3(0, 0, 120);
    private Vector3 finalRotation = new Vector3(0, 0, 240);
    public Transform leverTransform;
    public GameObject panelHintObject;

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            canUse = true;
            panelHintObject.SetActive(true);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            canUse = false;
            panelHintObject.SetActive(false);
        }
    }

    private void Update()
    {
        if (canUse && Input.GetKeyDown(KeyCode.E))
        {
            onStart = !onStart;

            if (onStart)
            {
                leverTransform.rotation = Quaternion.Euler(startRotation);
                bridgeParts.GetComponent<Bridge>().isOpening = false;
```

```
            bridgePlate.GetComponent<BridgePlate>().canUse = true;
        }
        else
        {
        leverTransform.rotation = Quaternion.Euler(finalRotation);
        }
    }
  }
}
```

Ниже представлен листинг программы нажимной плиты, провоцирующей раздвижение мостов.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BridgePlate : MonoBehaviour
{
   public GameObject bridgeParts;
   public bool canUse = true;

   private void OnTriggerEnter(Collider other)
   {
      if (canUse && other.CompareTag("Player"))
      {
         bridgeParts.GetComponent<Bridge>().isOpening = true;
         canUse = false;
      }
   }
}
```

Ниже представлен листинг программы движения моста.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Bridge : MonoBehaviour
{
   public GameObject firstPart;
   public GameObject secondPart;
   public bool isOpening = false;

   private Vector3 finalPositionFirstPart;
   private Vector3 finalPositionSecondPart;

   private Vector3 initialPositionFirstPart;
   private Vector3 initialPositionSecondPart;
```

```
    void Start()
    {
        initialPositionFirstPart = firstPart.transform.position;
        initialPositionSecondPart = secondPart.transform.position;
        finalPositionFirstPart = firstPart.transform.position - new Vector3(5, 0, 0);
        finalPositionSecondPart = secondPart.transform.position + new Vector3(5, 0, 0);
    }

    void Update()
    {
        if (isOpening)
        {
            firstPart.transform.position = Vector3.Lerp(firstPart.transform.position,
finalPositionFirstPart, 0.5f * Time.deltaTime);
            secondPart.transform.position = Vector3.Lerp(secondPart.transform.position,
finalPositionSecondPart, 0.5f * Time.deltaTime);
        }
        else
        {
            firstPart.transform.position = Vector3.Lerp(firstPart.transform.position,
initialPositionFirstPart, 0.5f * Time.deltaTime);
            secondPart.transform.position = Vector3.Lerp(secondPart.transform.position,
initialPositionSecondPart, 0.5f * Time.deltaTime);
        }
    }
}
```

Для завершения уровня игроку необходимо подняться на лифте, движение которого реализовано в листинге программы, представленном ниже.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Elevator : MonoBehaviour
{
    public bool isGoingUp = false;
    public bool isRepeat = false;
    private Vector3 desiredPosition;
    public Vector3 startPosition;

    private void Start()
    {
        desiredPosition = transform.position + new Vector3(0, 10, 0);
        startPosition = transform.position;
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player") && !isRepeat)
        {
            isGoingUp = true;
```

```
            isRepeat = true;
        }
    }

    void Update()
    {
        if (!isGoingUp)
        {
            return;
        }
        if (Vector3.Distance(transform.position, desiredPosition) <= 0.01f)
        {
            transform.position = desiredPosition;
            isGoingUp = false;
        }

        transform.position += Vector3.up * Time.deltaTime;

    }
}
```

В случае, если игрок спрыгнул или случайно упал с лифта, его можно вернуть на место, нажав на нажимную плиту. Вблизи неё игрок так же получит подсказку, листинг программы подсказки показан ниже.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ReturnElevHint : MonoBehaviour
{
    public GameObject ElevatorHints;

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            ElevatorHints.SetActive(true);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            ElevatorHints.SetActive(false);
        }
    }
}
```

Листинг программы возвращения лифта продемонстрирован ниже.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ReturnElevator : MonoBehaviour
{
    public GameObject elevatorObjects;
    public Elevator elevator;

    private void OnTriggerEnter(Collider other)
    {
        ReturnElevatorToStart();
    }

    public void ReturnElevatorToStart()
    {
        if (elevator.isGoingUp)
        {
            elevator.isGoingUp = false;
            elevator.isRepeat = false;
        }
        else
        {
            elevator.isRepeat = false;
        }
        elevatorObjects.transform.position =
elevatorObjects.GetComponent<Elevator>().startPosition;
    }
}
```

На протяжении игры камера всегда следует за игроком. Листинг программы движения камеры продемонстрирован ниже.

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using static UnityEngine.GraphicsBuffer;

public class CameraMovement : MonoBehaviour
{
    public Transform playerTransform;
    public Vector3 offset;
    public Vector3 offsetSecondTrial;
    public bool secondTrial = false;
    public float smoothSpeed = 0.05f;

    void LateUpdate()
    {
        Vector3 desiredPosition = playerTransform.position + offset;

        if (secondTrial)
```

```
        {
            desiredPosition += offsetSecondTrial;
            transform.position = Vector3.Slerp(transform.position, desiredPosition, smoothSpeed);
        }
        else
        {
            transform.position = desiredPosition;
        }
    }
}
```

Для упрощения игрового процесса в игре присутствуют подсказки, сообщающие игроку о том, как он может взаимодействовать с предметами. Для этого был создан менеджер подсказок, являющийся синглтоном.

Синглтон это класс, являющийся глобальным для всего проекта, к которому можно обращаться и использовать в других скриптах, не давая на него ссылку. Во всём проекте он должен быть в одном экземпляре.

Ниже представлен код скрипта менеджера подсказок.

```
using UnityEngine;
using TMPro;

public class HintManager : MonoBehaviour
{
    public static HintManager instance;
    public TMP_Text hintText;

    public GameObject hintObject;


    private void Awake()
    {
        instance = this;
    }

    public void ShowHint(string text)
    {
        hintText.text = text;
        hintObject.SetActive(true);
    }

    public void HideHint()
    {
        hintText.text = "";
        hintObject.SetActive(false);
    }

    public void PrintHello()
```

```
    {
        Debug.Log("Hello!");
    }
}
```

Для единого стиля подсказок также был создан макет, представленный на рисунке 10.

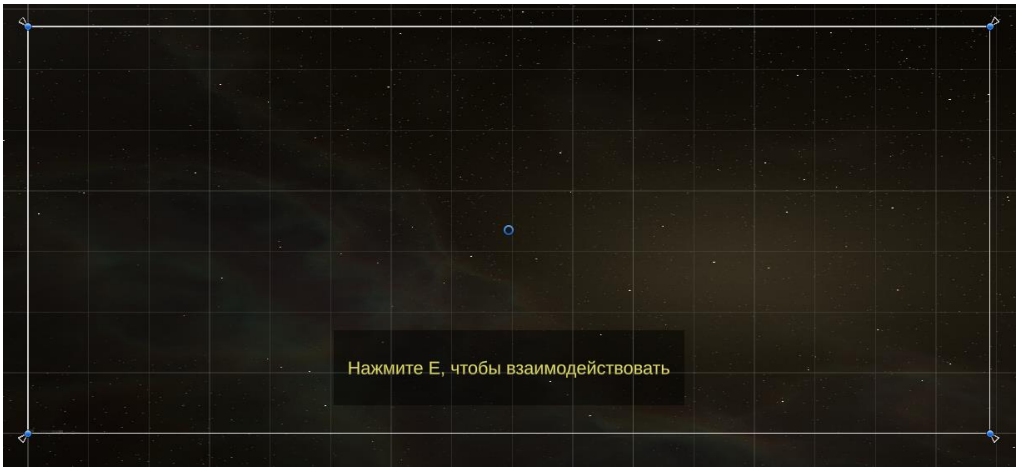

Рисунок 10. Макет подсказок игры

Также был создан интерфейс, в котором есть 3 метода, среди которых «показать подсказку», «убрать подсказку» и «взаимодействовать».

Скрипт интерфейса для подсказок представлен ниже.

```
public interface IInteractable
{
    public void ShowHint();
    public void HideHint();
    public void Interact();
}
```

После проделанной работы остается создать скрипт для объекта, с которым персонаж может взаимодействовать и где нужна подсказка, и описать методы как это необходимо для каждого объекта лично.

Пример представлен на скрипте, написанном для генератора, листинг которого представлен в приложении №6.

Таким образом, при взаимодействии с генератором, игрок увидит подсказку с текстом, соответствующую именно этому объекту, а также сможет взаимодействовать конкретным необходимым образом, отличающимся от взаимодействия с другими объектами.

Для прохождения нескольких головоломок игроку будет необходимо выключить генераторы, чтобы иметь возможность продвинуться дальше по уровню. Однако, чтобы это сделать, придется сыграть в мини-игру, внешний вид которой представлен на рисунке 11.

Рисунок 11. Мини-игра для отключения генератора

При взаимодействии с генератором у игрока на экране появляется данная игра, суть которой состоит в том, чтобы попасть по красному полю, которое каждый раз имеет разную длину в заданном диапазоне и может генерироваться в любой точке кольца. Белая стрелочка ходит по этому кольцу, и, кроме того, после начала новой попытки, она двигается с разной скоростью и также может двигаться в противоположном направлении. В случае попадания, заполняется шкала сверху, когда она будет полностью заполнена – игра завершится и генератор будет отключен, однако в случае ошибки прогресс шкалы упадет и один из трех предохранителей будет потерян. После трех ошибок игра заканчивается проигрышем, персонаж погибает.

Ниже представлен скрипт, в котором генерируются длина и местоположение красного поля, по которому игроку необходимо попасть.

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;

public class CircleGenerator : MonoBehaviour
```

```
{
    public Image filledImage;


    void Start()
    {
        Generate();
    }

    public void Generate()
    {
        GenerateShape();
        RandomRotation();
    }

    private void GenerateShape()
    {
        filledImage.fillAmount = Random.Range(0.05f, 0.1f);
    }

    private void RandomRotation()
    {
        filledImage.transform.rotation = Quaternion.Euler(0, 0, Random.Range(0, 360));
    }
}
```

В приложении №4 представлен скрипт, в котором описан алгоритм работы данной мини-игры.

Ниже представлен скрипт работы шкалы генератора.

```
using Palmmedia.ReportGenerator.Core;
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;

public class SuccessBar : MonoBehaviour
{
    public Image successBar;
    public Generator generator;
    public Generator generatorSecond;
    public PlayerController playerController;
    public AudioSource audioSource;
    public GameObject smoke;


    void Update()
    {
        if (successBar.fillAmount == 1f)
        {
```

```
            Debug.Log("Success");
            Generator.canInteract = false;
            playerController.isGameOver = false;
            audioSource.Pause();
            smoke.SetActive(false);
            successBar.fillAmount = 0f;
        }
    }

    public void Success()
    {
        successBar.fillAmount += 0.25f;
    }

    public void Fail()
    {
        successBar.fillAmount -= 0.25f;
    }
}
```

В процессе разработки был реализован инвентарь, внешний вид которого показан на рисунке 12.

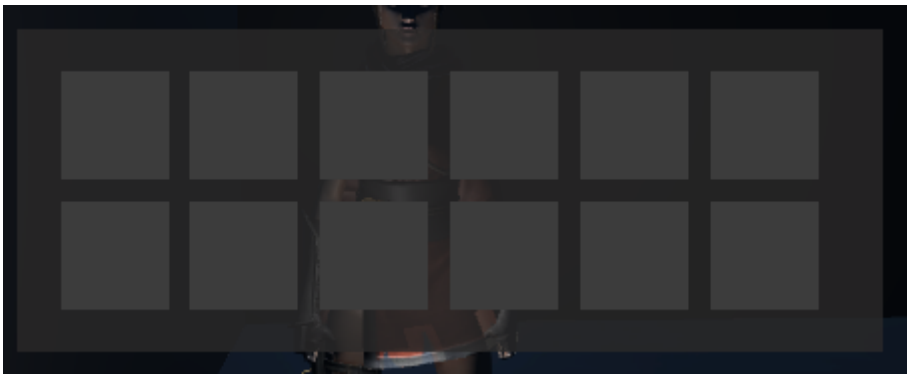

Рисунок 12. Инвентарь

Ниже представлен листинг программы, описывающей открытие инвентаря.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class InventoryUI : MonoBehaviour
{
    public GameObject inventoryRootObject;

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Tab))
        {
```

```
            inventoryRootObject.SetActive(!inventoryRootObject.activeSelf);
            //Cursor.lockState = inventoryRootObject ? CursorLockMode.None :
CursorLockMode.Locked;
            //Cursor.visible = inventoryRootObject.activeSelf;
        }
    }
}
```

Ниже представлен листинг программы, описывающий слоты в инвентаре. По умолчанию уникальный идентификатор каждого равен -1, в случае, если он пустой. После заполнения ID меняется. В случае, если игрок хочет использовать предмет, слот очищается.

```
using UnityEngine;
using UnityEngine.UI;

public class ItemSlot : MonoBehaviour
{
    public int currentID = -1;
    public Image iconImage;

    public bool isFree => currentID == -1;

    public void PutItem(InventoryItem item)
    {
        currentID = item.itemID;
        iconImage.sprite = item.icon;
    }

    private void OnEnable()
    {
        if (isFree)
        {
            iconImage.gameObject.SetActive(false);
        }
        else
        {
            iconImage.gameObject.SetActive(true);
        }
    }

    public void UseItem()
    {
        if (isFree)
        {
            return;
        }

        if (PlayerInteractionInventory.instance.currentInteractable != null)
        {
```

```
            bool result =
PlayerInteractionInventory.instance.currentInteractable.Interact(currentID);
            if (result)
            {
                ClearItem();
            }
        }
    }

    private void ClearItem()
    {
        iconImage.sprite = null;
        currentID = -1;
        iconImage.gameObject.SetActive(false);
    }
}
```

У каждого предмета, который можно подобрать и в дальнейшем использовать, есть скрипт, который показывает, что у всех них есть собственный идентификатор, а также иконка, которая в дальнейшем будет отображаться в инвентаре. В данной программе описано, что происходит с объектом после помещения его в инвентарь.

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class InventoryItem : MonoBehaviour, IPickable
{
    public int itemID;
    public Sprite icon;

    public void Pick()
    {
        if (InventorySystem.instance.PutItem(this))
        {
            Destroy(gameObject);
        }
    }
}
```

Для корректной работы представленной выше программы был создан интерфейс, представленный ниже.

```
public interface IPickable
{
    public void Pick();
}
```

Ниже представлен листинг программы, описывающей помещение предметов в инвентарь. Данная программа проверяет наличие свободных мест в инвентаре.

```
using UnityEngine;

public class InventorySystem : MonoBehaviour
{

    public static InventorySystem instance;
    public ItemSlot[] slots;


    private void Awake()
    {
        if (instance != null)
        {
            Destroy(gameObject);
            return;
        }
        instance = this;
    }

    public bool PutItem(InventoryItem itemToPut)
    {
        for (int i = 0; i < slots.Length; i++)
        {
            if (slots[i] != null && slots[i].isFree)
            {
                slots[i].PutItem(itemToPut);
                return true;
            }
        }

        return false;
    }
}
```

Для того, чтоб определенные предметы в инвентаре взаимодействовали с определенными объектами в игре, например, конкретный ключ открывал только одну дверь, был создан следующий интерфейс, листинг которого представлен ниже.

```
public interface IInteractableWithInventory
{
    void Activate();
    void Interact(int id);
}
```

Ниже представлен листинг программы, демонстрирующий работу данной системы, когда игрок оказывается вблизи объектов, с которыми можно взаимодействовать.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerInteractionInventory : MonoBehaviour
{
   public static PlayerInteractionInventory instance;
   public IInteractableWithInventory currentInteractable;
   private void Awake()
   {
      if (instance != null)
      {
         Destroy(gameObject);
         return;
      }
      instance = this;
   }
   private void OnTriggerEnter(Collider other)
   {
      IPickable pickable = other.GetComponent<IPickable>();
      if (pickable != null)
      {
         pickable.Pick();
      }

      IInteractableWithInventory interactable =
other.GetComponent<IInteractableWithInventory>();
      if (interactable != null)
      {
         currentInteractable = interactable;
      }
   }

   private void OnTriggerExit(Collider other)
   {
      IInteractableWithInventory interactable =
other.GetComponent<IInteractableWithInventory>();
      if (interactable != null)
      {
         currentInteractable = interactable;
      }
   }
}
```

На примере двери ниже продемонстрирован листинг программы, показывающий, что у объекта есть свой уникальный идентификатор, и в

случае, если его идентификатор совпадает с идентификатором предмета в инвентаре, объект активируется.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DoorKeyOne : MonoBehaviour, IInteractableWithInventory
{
    public int itemNeedID;
    public void Activate()
    {
        Debug.Log("Door is opening");
    }

    public void Interact(int id)
    {
        if (itemNeedID != id)
        {
            return;
        }

        Debug.Log("Open");
    }
}
```

Для демонстрации работы генератора были добавлены частицы, а именно дым и звуковые эффекты. На рисунке 13 продемонстрирован внешний вид работающего генератора.

Рисунок 13. Работающий генератор

После успешного прохождения мини-игры звуки и эффекты дыма отключаются, демонстрируя выключение генератора.

Кроме того, на одном из испытаний главного героя встречает враг с продвинутым искусственным интеллектом, в способности которого входит патрулирование определенных точек, преследование персонажа, его поиск. Изначально враг передвигается по заданным разработчиком точкам, причем делает он это каждый раз хаотично. В случае, если враг видит героя и находится с ним в зоне максимально допустимого расстояния, он переходит в стадию преследования, начиная бежать с увеличенной скоростью. В случае, если дистанция между героем и врагом слишком маленькая, враг наносит смертельный удар, после чего следует анимация гибели и экран поражения. Однако, если герою удается сбежать, спрятаться где-либо, то по прошествии некоторого времени, если враг так и не сможет увидеть героя, он вернется к патрулированию.

Также в процессе прохождения игрок спускается в подвал, где ему нужно найти предмет для дальнейшего прохождения. После того, как в комнату зайдет игрок, громкий звук оповестит врага о недоброжелателе. Враг придет и попытается найти героя, ориентируясь в том числе на его последнее местонахождение. В случае неудачи он уйдет.

Ниже представлен листинг программы, в которой перечислены возможные состояния врага.

```
public enum EnemyState
{
    IDLE,
    Patrul,
    Chase,
    Seek,
}
```

В приложении №5 продемонстрированы описанные состояния врага, то есть спокойное состояние, патрулирование, преследование, поиск, атака. Также описаны некоторые дополнительные методы, например, «EnemyOverview», проверяющий, находится ли игрок в зоне видимости врага.

После того, как игрок спустится, листинг программы, представленной в приложении №7, заставит врага начать искать героя. Кроме того, для лучшего обзора происходящего у игрока будет отдалена камера в момент поиска и преследования.

С помощью найденного ключа в вентиляции, игрок сможет открыть дверь на первом этаже второго уровня. Далее его встретит комната, где три двери ведут разным по сложности испытаниям. Пройдя каждое из них, герой сможет выбраться на свободу, а игрок завершить прохождение.

Первая дверь отправит игрока на самое легкое испытание, где ему необходимо будет выключить генератор и ничего не будет угрожать жизни героя. Вернувшись с испытаний, в случае их успешного завершения, обратно в комнату игрок зайти не сможет.

Второе испытание заставит игрока найти вход в вентиляцию, проползти по ней, обойти врага, найти ключ, открыть дверь с генератором и так же отключить его, как и предыдущий.

Отправившись на третье испытание, герой окажется в полностью темной комнате, где ему придется ходить по тонкой дороге, ориентируясь на свет, который будет гаснуть и появляться в зависимости от того, как далеко зашел игрок.

Ниже представлен листинг программы, в которой описана работа света в третьей комнате.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LightLevelThree : MonoBehaviour
{
    public GameObject lightNext;
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            if (lightNext == null)
            {
                return;
            }
```

```
            lightNext.SetActive(true);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            gameObject.SetActive(false);
        }
    }
}
```

Собрав все необходимые предметы, игрок отправиться к двери, за которой будет находиться последний генератор. Чтобы ее открыть, ему необходимо вставить найденные ранее предметы.

2.4. Отладка и тестирование программы

Отладка и тестирование программы являются неотъемлемыми стадиями создания игрового продукта, так как, прежде всего, гарантируют его функциональность.

Прежде всего, был проверен контроллер управления, так как он является одним из ключевых функциональных требований игры «Маленькие кошмары». Главный герой должен ходить, бегать, прыгать, сидеть, а также должна отсутствовать возможность совершить двойной прыжок, встать, если над головой препятствие, а также бежать, когда герой крадется. Проверка прыжка представлена на рисунке 14.

Рисунок 14. Проверка на отсутствие двойного прыжка

Проверка возможности красться с отстуствием возможности встать, если над головой есть препятствие, представлена на рисунке 15.

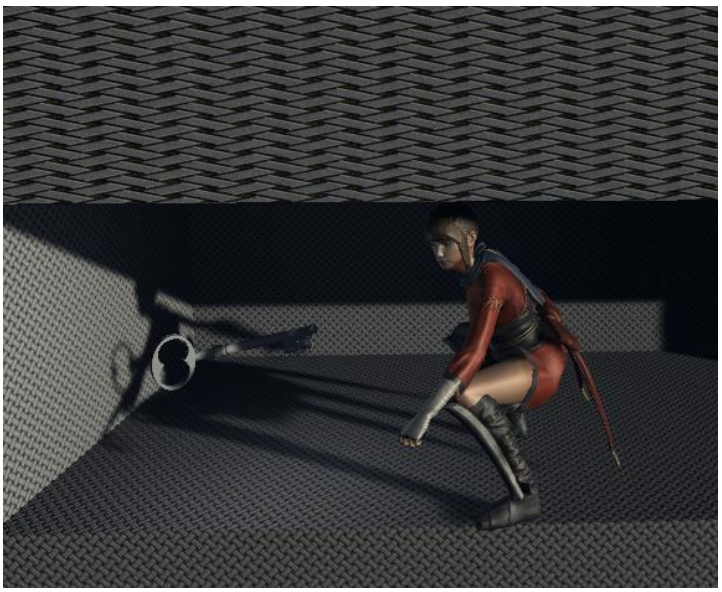

Рисунок 15. Проверка на отсутствие возможности встать, если есть препятствие

В комнате с головоломкой герой должен наступить на нажимную плиту, после чего на табличке отображается цифра. Пример игрового процесса и проверка работы нажимной плиты представлены на рисунке 16.

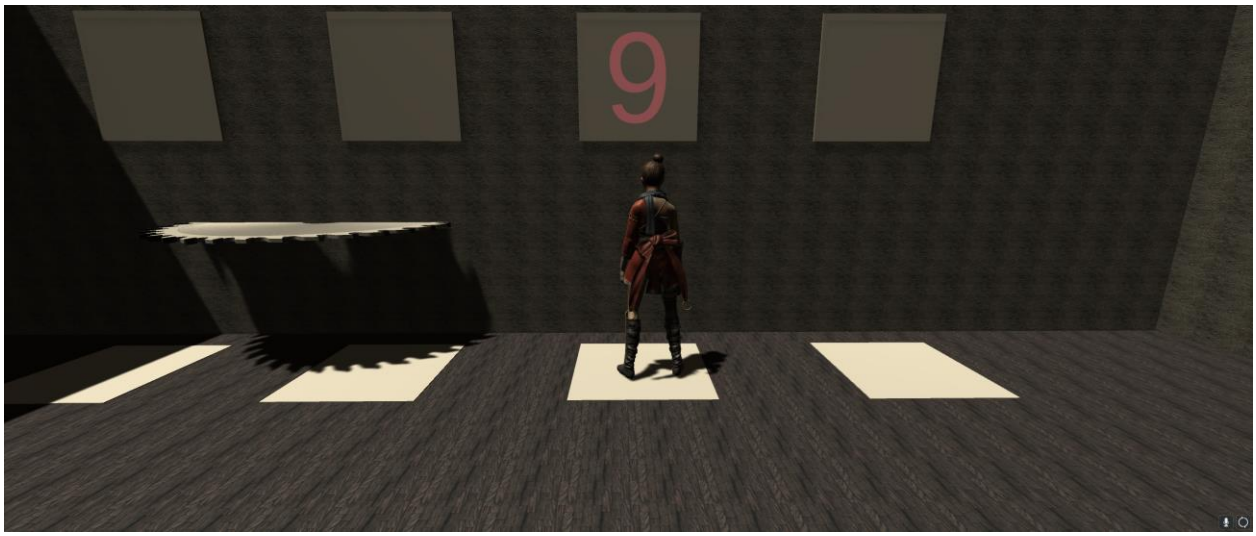

Рисунок 16. Проверка работы нажимной плиты и таблички

На рисунке 17 показан экран поражения с возможностью начать игру с самого начала после гибели от столкновения с пилой или от врага.

Рисунок 17. Экран поражения

На рисунке 18 для решения головоломки продемонстрирована возможность взаимодействия игрока с панелью для ввода кода.

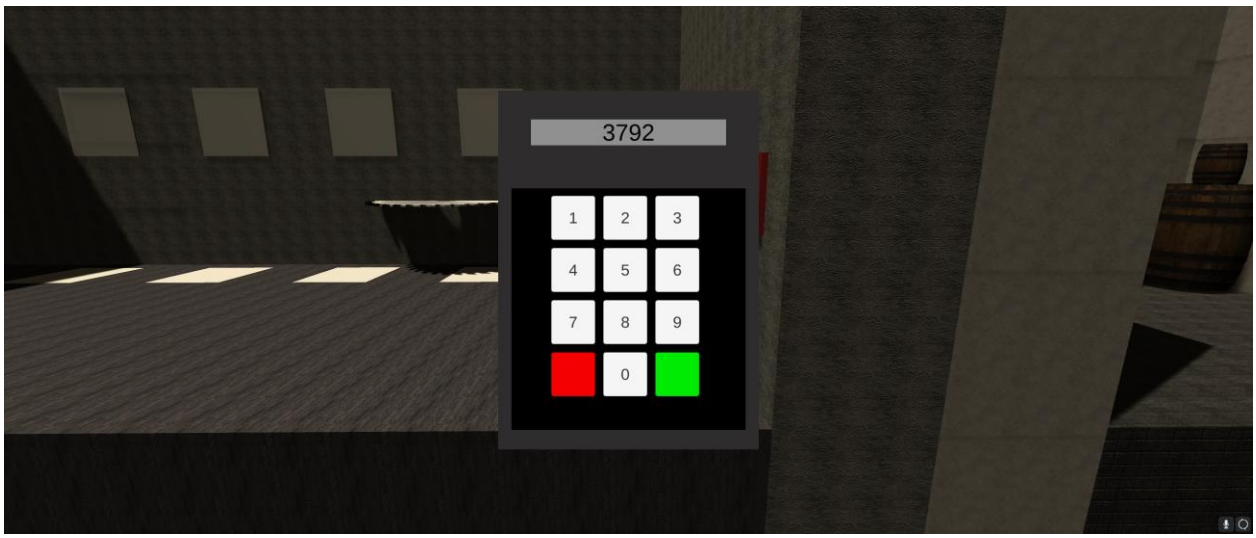

Рисунок 18. При нажатии на кнопку «E» отображается панель

На рисунках 19 и 20 проверена работоспособность рычага, соединяющего мост, а также нажимной плиты, раздвигающей мост.

Рисунок 19. Возможно взаимодействовать с рычагом

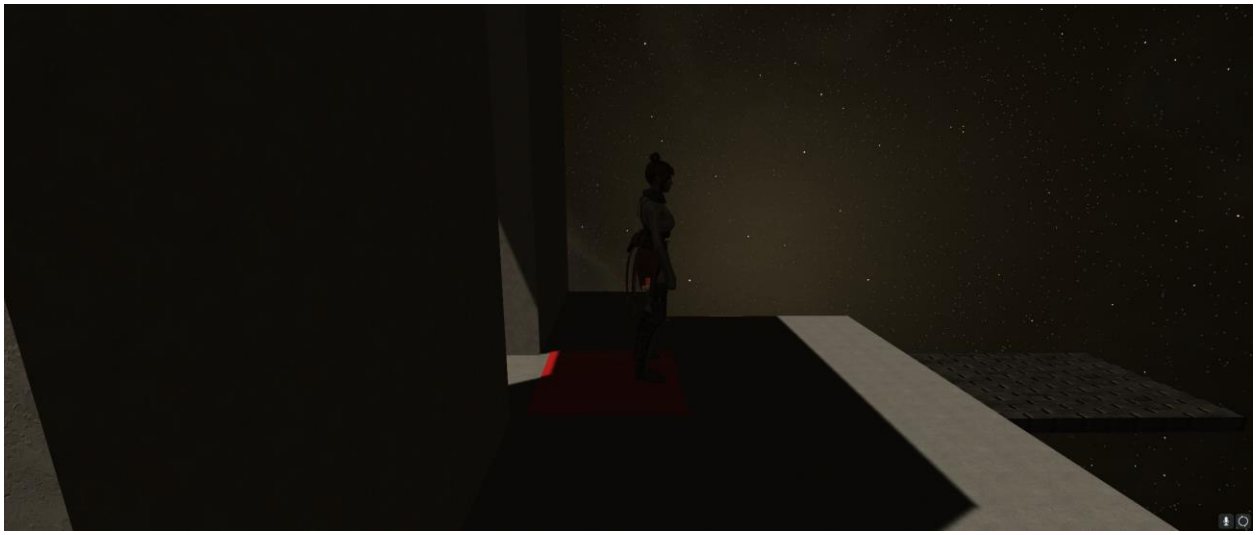

Рисунок 20. После нажатия на плиту мост раздвинулся

В конце необходимо убедиться, что лифт поднимается, а в случае, если игрок спрыгнет или упадёт, есть возможность вернуть лифт при помощи нажимной плиты.

На рисунке 21 продемонстрирована работа лифта.

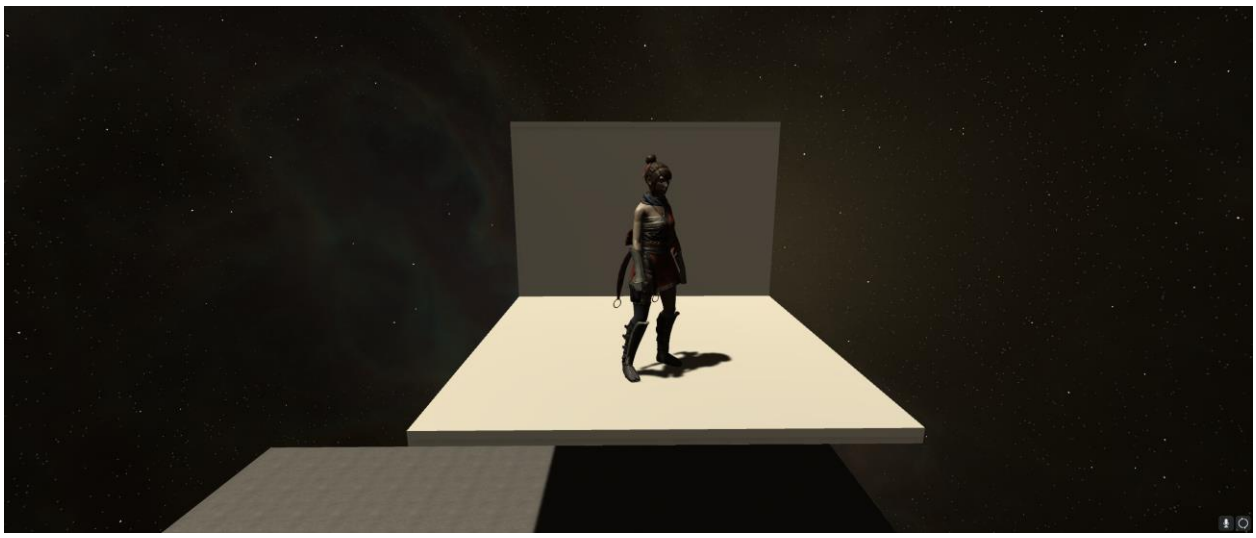

Рисунок 21. Лифт поднимается

На рисунке 22 продемонстрирована работа нажимной плиты для возвращения лифта.

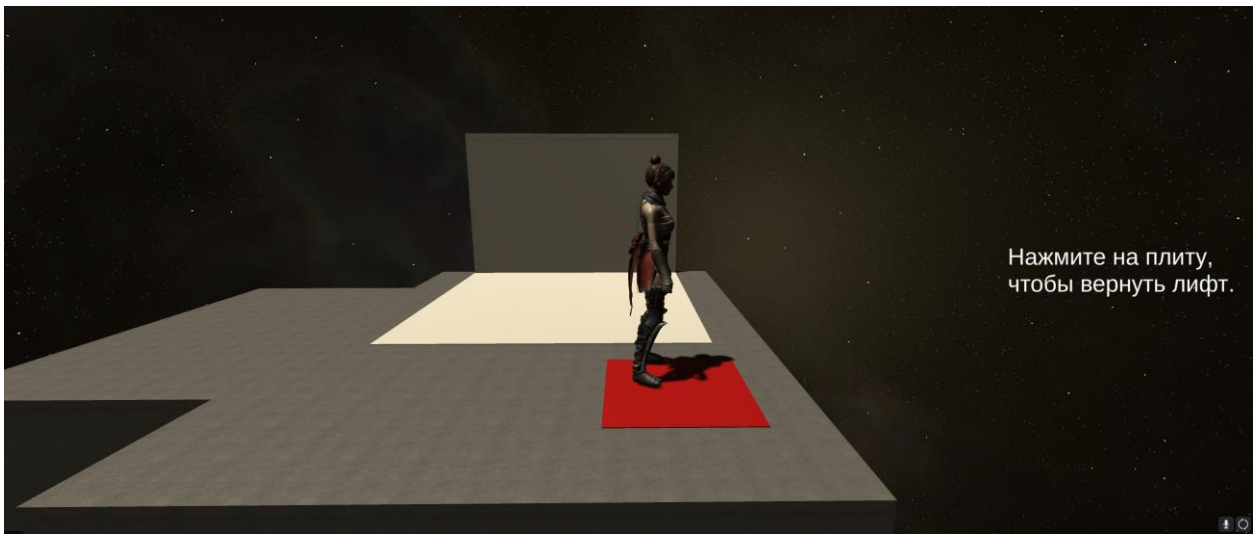

Рисунок 22. При нажатии на плиту лифт вернулся назад

Воспользовавшись подсказкой и перейдя на второй уровень, игрок окажется на другой карте. Забравшись в дом, его встретит дверь с подсказкой, которую он сможет открыть. Пример представлен на рисунке 23.

Рисунок 23. Дверь с подсказкой

Подойдя к двери, игрок сможет спуститься вниз, увидев перед этим экран загрузки, представленный на рисунке 24.

Рисунок 24. Экран загрузки

В вентиляции герой находит ключ, необходимый для открытия двери на верхнем этаже. Ключ представлен на рисунке 25.

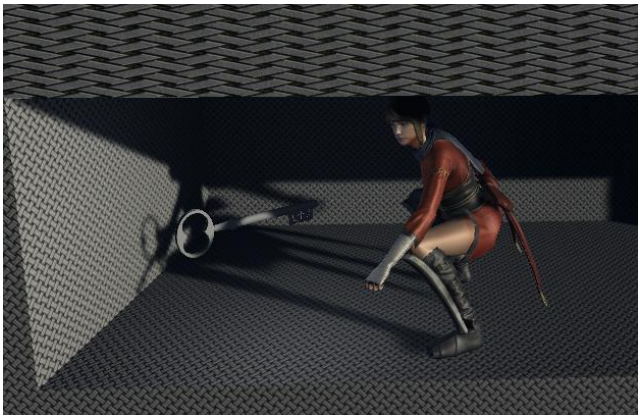

Рисунок 25. Найденный в вентиляции ключ

Чтобы подобрать ключ, герою необходимо столкнуться с ним.

После громкого сигнала враг начал нас искать. Чтобы спастись, у игрока есть возможность спрятаться в шкаф, возле которого он увидит подсказку о взаимодействии. Враг в подвале представлен на рисунке 26.

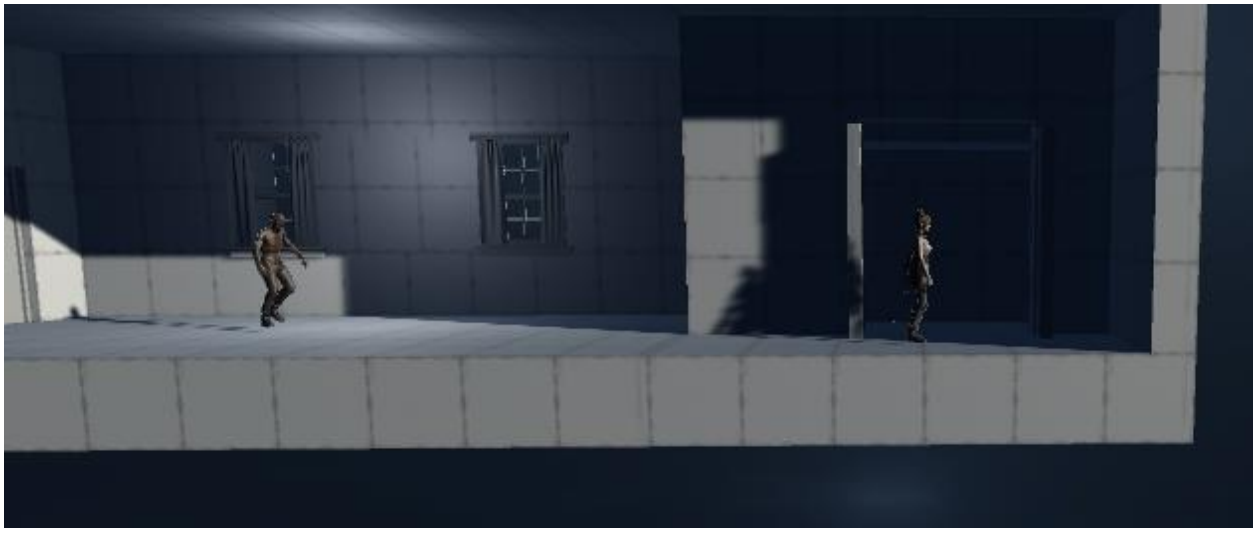

Рисунок 26. Враг в подвале

Возле закрытой двери появляется подсказка, что открыть ее можно только с помощью ключа. Открыв инвентарь и нажав на ключ, в случае, если он подходит, дверь откроется. В противном случае, дверь останется закрытой. Закрытая дверь представлена на рисунке 27.

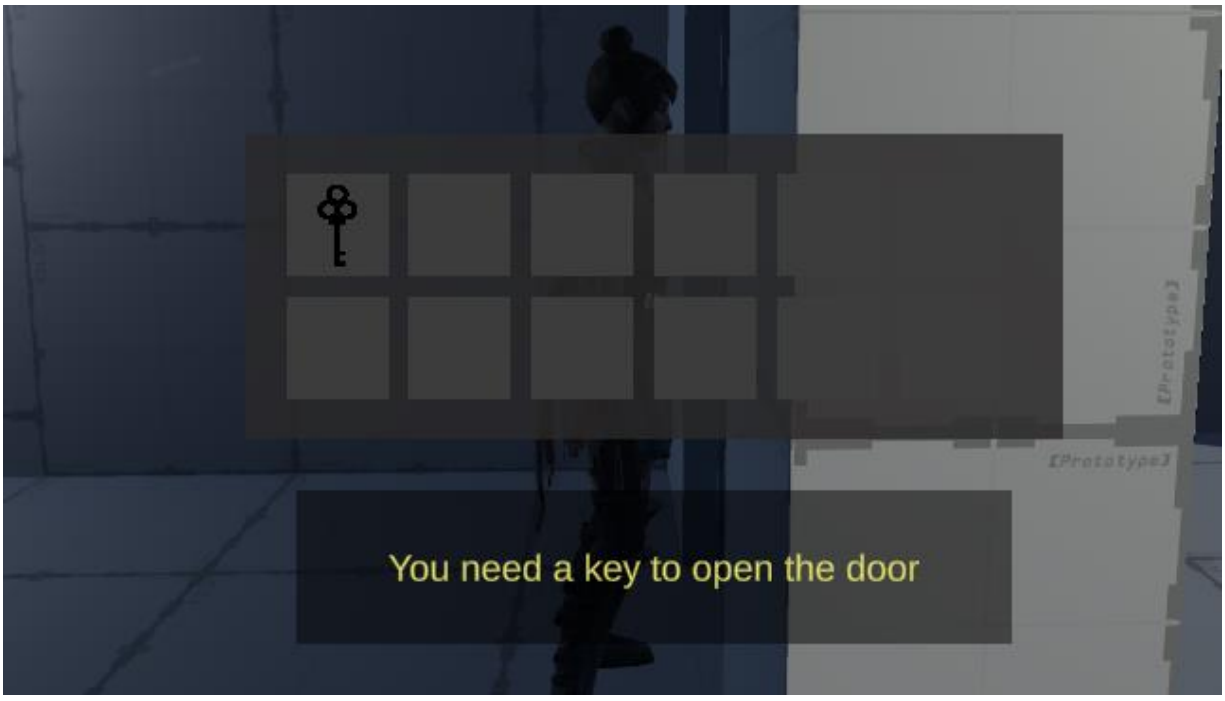

Рисунок 27. Закрытая дверь, которой необходим ключ

После того, как мы нажали на ключ, дверь открылась. Пример работы представлен на рисунке 28.

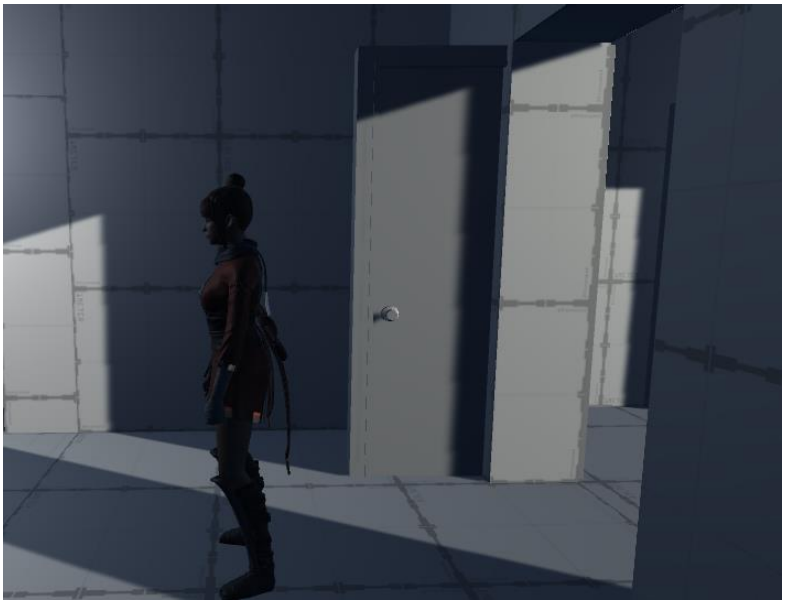

Рисунок 28. Открытая ключом дверь

В открывшейся комнате есть три двери с подсказками о том, как попасть на испытание. Пример представлен на рисунке 29.

Рисунок 29. Дверь на испытание

После активации генератора начинается игра, успешно ее завершив невозможно больше взаимодействовать с генератором, но можно вернуться в предыдущую комнату, воспользовавшись дверью. Пример взаимодействия с генератором представлен на рисунке 30.

Рисунок 30. Взаимодействие с генератором

Вернувшись и подойдя к той же двери, отсутствует возможность попасть в комнату первого испытания, а также меняется подсказка, сообщающая, что испытание уже пройдено. Пример представлен на рисунке 31.

Рисунок 31. Дверь после пройденного испытания

На втором уровне необходимо обойти врага, подобрать ключ и открыть дверь с генератором. После отключения нужно вернуться обратно.

Настигнув героя, враг убьет его одни ударом. Пример представлен на рисунке 32.

Рисунок 32. Гибель героя от врага

На третьем уровне герою предстоит передвигаться, ориентируясь на свет. Пример третьего уровня представлен на рисунке 33.

Рисунок 33. Третий уровень

Подойдя к двери, необходимо разместить найденные кубики в соответствии с цветом на стене. В случае, если предмет выбран правильно,

сигнал на стене окрасится в зеленый. Когда обе картины буду зеленые, дверь можно будет открыть. Пример представлен на рисунках 34 и 35.

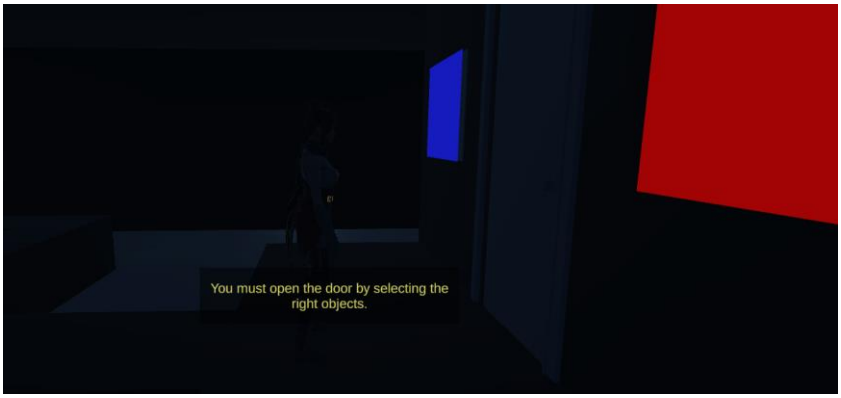

Рисунок 34. Закрытая дверь и задание, чтобы её открыть

Рисунок 35. Успешно выполненное задание

Решив все головоломки и пройдя все препятствия, герой сможет завершить прохождение игры, подойдя к последней двери.

2.5. Руководство по использованию программы

Руководство по использованию программы представлено отдельно для системного программиста, программиста и оператора.

Руководство системного программиста содержит следующие пункты:

- общие сведения о программе;
- структура программы;
- настройка программы;
- проверка программы;

- дополнительные возможности;

- сообщения системному программисту.

Общие сведения о программе. «Маленькие кошмары» представляет собой игру в жанре платформер. Функции программы включают в себя возможность управления главным героем, наличие чекпоинтов, препятствий, головоломок и врагов.

Технические и программные средства:

- операционная система Windows 7 (64-bit) и выше;

- оперативная память 4 ГБ ОЗУ.

Настройка программы. Для установки игры необходимо воспользоваться инсталлятором. На первом этапе необходимо выбрать язык установки, затем прочитать лицензионное соглашение, принять условия и нажать «Далее». После установки появится возможность запустить игру.

Проверка программы. Проведите тестирование работоспособности панели для ввода кода, а также наличие проверок введенных данных. Для проверки попытайтесь ввести код с длиной больше или меньше четырех символов, введите неверную комбинацию кода, попробуйте выйти, нажав на красную кнопку, введите верную комбинацию кода.

В ходе проверки панели для ввода кода невозможно будет ввести значение длиной более 4 символов, а при вводе значения меньшей длиной код будет считаться неверным. В случае, если код неверный, поле ввода кода обнуляется. Нажатие на красную кнопку закроет панель ввода кода, а в случае ввода верной комбинации и нажатии на зеленую кнопку откроется дверь.

Сообщения системному программисту. При запуске инсталлятора необходимо принять Лицензионное Соглашение, выбрать папку установки, выбрать, создавать ли ярлык на рабочем столе.

Руководство программиста содержит следующие пункты:

- назначение и условия применения программы;

- характеристики программы;

- обращение к программе;

- входные и выходные данные;

- сообщения.

Назначение и условия применения программы. «Маленькие кошмары» представляет собой игру в жанре платформер. Функции программы включают в себя возможность управления главным героем, наличие чекпоинтов, препятствий, головоломок и врагов.

Технические и программные средства:

- операционная система Windows 7 (64-bit) и выше;

- оперативная память 4 ГБ ОЗУ.

Характеристики программы. Игра представляет собой платформер с элементами квеста и хоррора. Игрок управляет героем, который пытается выбраться из мира, полного опасностей и загадок. Геймплей основан на преодолении препятствий, решении головоломок и избегании врагов. Музыкальное сопровождение «Маленькие кошмары» создает атмосферу страха и напряжения.

Обращение к программе. Программа вызывается при помощи открытия исполняемого файла. Для прохождения игры пользователю необходима клавиатура и компьютерная мышь.

Входные и выходные данные. Входными данными являются управление героем и взаимодействие с предметами при помощи клавиатуры и мыши. Выходными данными являются представление игрового мира на экране, анимация героя, враги, подсказки, экран поражения и загрузки.

Руководство оператора содержит следующие пункты:

- назначение программы;

- условия выполнения программы;

- выполнение программы;

- сообщения оператору.

Назначение программы. «Маленькие кошмарыге» представляет собой увлекательную игру в жанре платформер, предназначенную для пользователей старше 16 лет. Управляя главным героем, игрок обладает

возможностью прохождения препятствий и решения головоломок, параллельно исследуя загадочный и пугающий мир.

Условия выполнения программы.

Системные требования:

- операционная система Windows 7 (64-bit) и выше;

- оперативная память 4 ГБ ОЗУ.

Выполнение программы. Для загрузки игры необходимо скачать и запустить инсталлятор, при помощи которого возможно установить игру на компьютер. После того, как игра установилась на компьютер, необходимо запустить файл программы. После запуска игроку будет представлен игровой экран. Управление главным героем осуществляется с помощью клавиш WASD. Взаимодействовать с окружающими предметами игрок может при помощи клавиатуры или компьютерной мыши. Реакция программы происходит на запуск новой попытки игры и взаимодействие игрока с предметами. В случае поражения игрок увидит экран с возможностью начать игру заново. После успешного прохождения уровней игрок сможет закрыть программу.

Сообщения оператору. В игре выводятся сообщения о поражении, прохождении уровня и подсказки.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта была разработана игра в жанре платформер «Маленькие кошмары» с использованием кроссплатформенной среды разработки Unity.

Для достижения поставленной цели необходимо было решить ряд задач.

Описание предметной области позволило изучить востребованность компьютерных игр в современном мире, жанры и их популярность, а также актуальность темы данного дипломного проекта.

При проведении сравнительного анализа программ-аналогов появилась возможность подчеркнуть достоинства и недостатки уже существующих, похожих жанром и другими особенностями проектов, что в дальнейшем помогло избежать ошибок при проектировании и разработке игры, а также позволило определить программу, наиболее точно соответствующую требованиям.

Постановка задачи, анализ требований и разработка спецификаций, проектирование программного обеспечения позволили создать полноценную картину будущего проекта. В результате выполнения данных задач были точно определены жанр, сюжет игры, входные и выходные данные, функциональные требования. С помощью разработанных диаграмм и схем наглядно показан рабочий процесс, логика программного продукта, возможности пользователя при взаимодействии с игрой, отображено полное представление о проектируемой игре и взаимодействие компонентов программного продукта.

Важную роль в процессе проектирования и реализации программы сыграло тестирование, позволившее выявить и исправить ошибки, а также обеспечить качество и надежность продукта, гарантируя его функциональность.

В результате выполнения поставленных задач был проведен тщательный анализ популярных игр данного жанра, была разработана

детальная концепция игры «Маленькие кошмары», которая включает в себя уникальный игровой мир, головоломки, врагов и испытания для игрока.

В процессе разработки были выполнены поставленные функциональные требования, являющиеся характерными для данного жанра, а именно: управление главным героем, наличие чекпоинтов, враги, препятствия, а также головоломки. Для удобства пользователей были созданы подсказки, помогающие разобраться в управлении главным героем и во взаимодействии с предметами в игре.

В ходе выполнения данного дипломного проекта существенную роль сыграла среда разработки Unity, предоставив мощный инструментарий для разработки игровых уровней, игровой логики и интерфейса. Работа с Unity принесла ценный опыт в области разработки компьютерных игр.

Основной трудностью при выполнении данного дипломного проекта оказался недостаток опыта в разработке компьютерных игр и в работе со средой разработки Unity, однако изучение литературы по разработке игр на Unity, официальная документация Unity, онлайн-ресурсы, онлайн-форумы и видеоуроки по программированию, а также практические проекты и примеры помогли достичь поставленной цели.

В заключение можно отметить, что проектирование и разработка игры «Маленькие кошмары» это многогранный процесс, требующий не только технической грамотности, но и творческого подхода, ведь, кроме программной реализации было важно придумать интересный сюжет, нестандартную головоломку, создать уровень с различными препятствиями и врагами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ИНТЕРНЕТ-РЕСУРСОВ

Законодательные и нормативные акты:

1. ГОСТ Р 7.0.12-2011 Библиографическая запись. Сокращение слов и словосочетаний на русском языке. Общие требования и правила. – М.: Стандартинформ, 2012. – 61 с.
2. ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: Стандартинформ, 2010. – 92 с.
3. ГОСТ 7.32-2017 Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: Стандартинформ, 2017. – 47 с.
4. ГОСТ 7.82-2001 Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2001. – 39 с.
5. ГОСТ Р 7.0.100-2018 Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: Стандартинформ, 2018. – 122 с.
6. ГОСТ Р 7.0.5-2008 Библиографическая ссылка. Общие требования и правила составления. – М.: Стандартинформ, 2008. – 32 с.
7. Единая система программной документации. – М.: Стандартинформ, 2005. – 128 с.

Учебная и научная литература:

8. Гуриков, С. Р. Введение в программирование на языке Visual C#: учебное пособие / С.Р. Гуриков. – МОСКВА: ФОРУМ: ИНФРА-М, 2020. – 447 с. – (Высшее образование: Бакалавриат). – ISBN 978-5-00091-458-8. – Текст: электронный. – Режим доступа: https://znanium.com/catalog/product/1092167 (дата обращения: 21.09.2023).

Интернет-документы:

9. Аксенова Т.Г. Онлайн-курс по технологиям разработки программного обеспечения. – [Электронный ресурс]. – Режим доступа: https://classroom.google.com/c/NTc2MzExNTI0MTY0?cjc=na6mvlx (дата обращения: 25.10.2023).

10. Гагарина, Л. Г. Технология разработки программного обеспечения: учебное пособие / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Сидорова-Виснадул; под ред. Л.Г. Гагариной. – Москва: ФОРУМ: ИНФРА-М, 2020. – 400 с. – (Среднее профессиональное образование). – ISBN 978-5-8199-0812-9. – Текст: электронный. – Режим доступа: https://znanium.com/catalog/product/1067012 (дата обращения: 15.10.2023)

11. Жанр платформер. – [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Платформер (дата обращения: 11.09.2023).

12. Игра Stray. – [Электронный ресурс]. – Режим доступа: https://store.steampowered.com/app/1332010/Stray/ (дата обращения: 18.09.2023).

13. Интернет-сервис для построения UML-диаграмм. – [Электронный ресурс]. – Режим доступа: https://plantuml.com/ru/ (дата обращения: 20.10.2023).

14. Интернет-сервис для построения схем и диаграмм Draw.io. – [Электронный ресурс]. – Режим доступа: https://app.diagrams.net (дата обращения: 20.10.2023).

15. Материалы для создания сцены. – [Электронный ресурс]. – Режим доступа: https://assetstore.unity.com (дата обращения: 16.11.2023).

16. Модель и анимации для главного героя. – [Электронный ресурс]. – Режим доступа: https://www.mixamo.com (дата обращения: 01.11.2023).

17. Первые шаги в Unity. – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/otus/articles/576434/ (дата обращения: 09.10.2023)

18. Передвижение персонажа в Unity 2D и 3D. – [Электронный ресурс]. – Режим доступа: https://unityhub.ru/guides/peredvizhenie-personazha-v-unity-2d-i-3d_59 (дата обращения: 01.11.2023)

19. Полное руководство по языку программирования C# 6.0 и платформе .NET 4.6. – [Электронный ресурс]. – Режим доступа: http://metanit.com/sharp/tutorial/ (дата обращения: 30.10.2023).

20. Предметы для сцены. – [Электронный ресурс]. – Режим доступа: https://sketchfab.com/feed (дата обращения: 18.03.2024).

21. Руководства и электронные книги Unity. – [Электронный ресурс]. – Режим доступа: https://unity.com/ru/how-to (дата обращения: 13.05.2024).

22. Руководство по работе в среде Visual Studio. – [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/ (дата обращения: 15.05.2024).

23. Уроки по Unity 3D, анимация персонажа, скриптинг для новичков. – [Электронный ресурс]. – Режим доступа: https://www.youtube.com/watch?v=hd_rmZiAEbw (дата обращения: 20.05.2024).

ПРИЛОЖЕНИЕ

Приложение №1

Контроллер управления. Листинг программы

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    public Transform rayCastStart;
    public Rigidbody playerRb;
    public Animator animController;
    public float force = 20;
    private float runMultipl = 2.2f;
    public Vector3 direction;
    private float jumpforce = 7;
    private bool isGrounded;
    public bool cantStand = false;
    private bool isSneaking;
    private bool isWalking;
    public bool isGameOver = false;
    public Vector3 lastCheckPoint;
    public LayerMask maskToAvoid;

    public Transform colliderTransform;

    void Update()
    {
        if (isGameOver)
        {
            force = 0;
            animController.SetFloat("Velocity", 0);
            return;
        }
        isGrounded = IsGrounded();
        canStandUp();
        GetInput();
        CheckDrag();
        Jump();
        UpdateAnimator();
    }

    private void FixedUpdate()
    {
        Movement();
    }

    private void OnCollisionEnter(Collision collision)
    {
```

```
        if (collision.transform.CompareTag("Box"))
        {
            collision.transform.GetComponent<Rigidbody>().AddForce(-collision.relativeVelocity
* 3, ForceMode.Impulse);
        }
    }

    private void Movement()
    {
        if (Input.GetKey(KeyCode.LeftShift) && isGrounded && !isSneaking)
        {
            playerRb.AddForce(direction * force * runMultipl);
        }
        else if (isGrounded)
        {
            playerRb.AddForce(direction * force);
        }

        if (direction.magnitude >= 0.1f)
        {
            transform.rotation = Quaternion.Lerp(transform.rotation,
Quaternion.LookRotation(direction), 0.2f);
        }
    }

    private void GetInput()
    {
        float hor = Input.GetAxisRaw("Horizontal");
        float ver = Input.GetAxisRaw("Vertical");
        direction = new Vector3(hor, 0, ver).normalized;

        if (Input.GetKey(KeyCode.LeftControl))
        {
            //rayCastStart.localPosition = new Vector3(0, 1.1f, 0);
            colliderTransform.localScale = new Vector3(1, 0.3f, 1);
            isSneaking = true;
            isWalking = false;
        }
        else if (!cantStand)
        {
            //rayCastStart.localPosition = new Vector3(0, 1.8f, 0);
            colliderTransform.localScale = new Vector3(1, 1, 1);
            isSneaking = false;
            isWalking = true;
        }
    }

    private void Jump()
    {
        if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
        {
            playerRb.AddForce(Vector3.up * jumpforce, ForceMode.Impulse);
```

```
            animController.SetTrigger("Jump");
        }
    }

    private bool IsGrounded()
    {
        //Debug.DrawRay(rayCastStart.position, Vector3.down * 0.6f, Color.green);
        return Physics.OverlapSphere(transform.position + new Vector3(0, 0.12f, 0), 0.22f,
~maskToAvoid).Length > 1;
        //return Physics.Raycast(rayCastStart.position, Vector3.down, 0.6f);
    }

    private void OnDrawGizmos()
    {
        Gizmos.DrawSphere(transform.position + new Vector3(0, 0.12f, 0), 0.22f);
    }

    private void CheckDrag()
    {
        if (isGrounded && !isSneaking)
        {
            playerRb.drag = 10;
            runMultipl = 2.2f;
            force = 40; //50
        }
        else if (isGrounded && isSneaking)
        {
            playerRb.drag = 10;
            runMultipl = 2.2f;
            force = 20;
        }
        else
        {
            playerRb.drag = 0;
            runMultipl = 1.5f;
            force = 16;
        }
    }

    private void UpdateAnimator()
    {
        //Debug.Log("speed " + playerRb.velocity.magnitude);
        animController.SetBool("isSneaking", isSneaking);
        animController.SetBool("isWalking", isWalking);
        if (isGrounded)
        {
            animController.SetFloat("Velocity", playerRb.velocity.magnitude);
        }
        else
        {
            animController.SetFloat("Velocity", 0);
        }
```

```
        animController.SetBool("IsGrounded", isGrounded);
    }

    private void canStandUp()
    {
        RaycastHit hitInfo;
        Debug.DrawRay(rayCastStart.position, Vector3.up * 0.5f, Color.green); // рисуем луч
для визуализации с длиной 0.3
        cantStand = Physics.Raycast(rayCastStart.position, Vector3.up, out hitInfo, 0.5f); //
запускаем Raycast с длиной 0.3
        if (cantStand)
        {
            cantStand = true;
        }
        else
        {
            cantStand = false;
        }
    }
}
```

Приложение №2

Взаимодействие с окружающей средой. Листинг программы

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class PlayerInteraction : MonoBehaviour
{
    public Animator anim;
    public GameObject gameOverObjects;
    public GameObject endObjects;
    public GameObject hintObjects;
    //private bool interactWithGenerator = false;
    private PlayerController playerController;
    private bool canInteract = false;
    public IInteractable interactable;
    public bool explosion = false;
    public CameraMovement cameraMovement;
    public Scenes scenes;
    private bool toTheSecondLevel = false;

    public int idGenerator;

    private void OnCollisionEnter(Collision collision)
    {
        if (playerController)
        {
            return;
        }
        if (collision.transform.CompareTag("Enemy"))
        {
            playerController = transform.GetComponent<PlayerController>();
            playerController.isGameOver = true;
            Rigidbody playerRb = transform.GetComponent<Rigidbody>();
            playerRb.AddForce(collision.relativeVelocity.normalized * 10f, ForceMode.Impulse);
            playerRb.AddForce(Vector3.up * 4, ForceMode.Impulse);
            GameOver();
        }

    }

    private void OnTriggerEnter(Collider other)
    {
        if (playerController)
        {
            return;
        }
        if (other.CompareTag("Fall"))
        {
```

```
            transform.position = transform.GetComponent<PlayerController>().lastCheckPoint;
        }
        if (other.CompareTag("End"))
        {
            endObjects.SetActive(true);
            toTheSecondLevel = true;
        }
        if (other.CompareTag("Generator"))
        {
            idGenerator = other.GetComponent<Generator>().idGenerator;
        }
        if (other.CompareTag("CameraSecond"))
        {
            cameraMovement.secondTrial = true;
        }
        interactable = other.GetComponent<IInteractable>();
        if (interactable != null)
        {
            interactable.ShowHint();
            canInteract = true;
            //Debug.Log(interactable);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("End"))
        {
            endObjects.SetActive(false);
            toTheSecondLevel = false;
        }
        if (other.CompareTag("Generator"))
        {
            idGenerator = 0;
        }
        if (other.CompareTag("CameraSecond"))
        {
            cameraMovement.secondTrial = false;
        }
        interactable = other.GetComponent<IInteractable>();
        if (interactable != null)
        {
            interactable.HideHint();
            canInteract = false;
            interactable = null;
        }
    }

    public void GameOver()
    {
        hintObjects.SetActive(false);
        if (explosion)
```

```
        {
            Debug.Log("Explosion!");
            anim.SetTrigger("Explosion");
        }
        else
        {
            anim.SetTrigger("PlayerDead");
        }
        //anim.SetTrigger("PlayerDead");
        Invoke("ShowGameOverScreen", 4);
        //Cursor.lockState = gameOverObjects ? CursorLockMode.None :
CursorLockMode.Locked;
        //Cursor.visible = gameOverObjects.activeSelf;
    }

    private void ShowGameOverScreen()
    {
        gameOverObjects.SetActive(true);
    }

    private void Update()
    {
        //Debug.Log(idGenerator);
        Debug.Log(interactable);
        if (canInteract)
        {
            if (Input.GetKeyDown(KeyCode.E))
            {
                //Debug.Log(interactable);
                interactable.HideHint();
                interactable.Interact();
            }
        }

        if (toTheSecondLevel)
        {
            if (Input.GetKeyDown(KeyCode.P))
            {
                scenes.SecondLevel();
            }
        }
    }
}
```

Приложение №3

Панель для ввода кода. Листинг программы

```
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using UnityEngine;
using TMPro;

public class PanelLogic : MonoBehaviour
{
    public Door doorToOpen;
    public TMP_Text passwordText;
    public GameObject panelObjects;
    public GameObject panelHintObject;
    private bool canUse = false;
    private string currentPassword = "";
    private string correctPassword = "3792";

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            canUse = true;
            panelHintObject.SetActive(true);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            canUse = false;
            panelObjects.SetActive(false);
            panelHintObject.SetActive(false);
        }
    }

    private void Update()
    {
        if (canUse && Input.GetKeyDown(KeyCode.E))
        {
            panelObjects.SetActive(true);
            panelHintObject.SetActive(false);
        }
    }

    public void EnterNumber(string number)
    {
        if (currentPassword.Length >= 4)
        {
```

```
            return;
        }
        currentPassword += number;
        passwordText.text = currentPassword;
    }

    public void CheckPassword()
    {
        if (currentPassword == correctPassword)
        {
            doorToOpen.OpenDoor();
            panelObjects.SetActive(false);
        }
        else
        {
            currentPassword = "";
            passwordText.text = currentPassword;
        }
    }

    public void ExitFromPanel()
    {
        panelObjects.SetActive(false);
        currentPassword = "";
        passwordText.text = currentPassword;
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class ArrowMovement : MonoBehaviour
{
    public float rotationSpeed = 100f;
    private bool isRightDirection = true;
    public RectTransform fillCircleRect;
    public RectTransform arrowRect;
    public CircleGenerator circleGenerator;
    public PlayerController playerController; //
    public PlayerInteraction playerInteraction;
    private float rotationAmmount = 0;
    public SuccessBar successBar;
    private float fillAmount => circleGenerator.filledImage.fillAmount; //постоянно
подставляет значение справа
    private float currentRotationSpeed;
    private bool isOverlapping = false;
    public bool isGameStarted = false; //начинается и = true
    public bool isGameOver = false;


    public TMP_Text successText;
    public TMP_Text failText;
    public int ammountOfFails = 0;
    private int ammountOfSuccess = 0;

    public GameObject[] testMistakes;

    public AudioSource audioSuccess;
    public AudioSource audioFail;


    void Update()
    {
        rotationAmmount += Time.deltaTime * rotationSpeed;
        if (ammountOfFails == 3 && isGameStarted)
        {
            isGameStarted = false;
            isGameOver = true;
            playerInteraction.explosion = true;
            playerInteraction.GameOver();
        }
        if (!isGameStarted)
        {
```

```
            return;
        }
        playerController.isGameOver = true; //
        currentRotationSpeed = isRightDirection ? rotationSpeed : -rotationSpeed;
        transform.Rotate(0, 0, Time.deltaTime * currentRotationSpeed);
        CheckOverlap(arrowRect, fillCircleRect);

        if (rotationAmmount > 360)
        {
            isGameStarted = false;
            for (int i = ammountOfFails; i < testMistakes.Length;)
            {
                testMistakes[i].SetActive(false);
                ammountOfFails += 1;
                successBar.Fail();
                break;
            }
            UpdateText();
            Invoke("ResetGame", 1);
        }
        if (Input.GetKeyDown(KeyCode.Space) && isGameStarted)
        {
            isGameStarted = false;
            currentRotationSpeed = 0;
            if (isOverlapping)
            {
                ammountOfSuccess += 1;
                successBar.Success();
                audioSuccess.Play();
            }
            else
            {
                audioFail.Play();
                for (int i = ammountOfFails; i < testMistakes.Length;)
                {
                    testMistakes[i].SetActive(false);
                    ammountOfFails += 1;
                    successBar.Fail();
                    break;
                }
            }
            UpdateText();
            Invoke("ResetGame", 1);
        }
    }

    void CheckOverlap(RectTransform rectTrans1, RectTransform rectTrans2)
    {
        float arrowAngle = rectTrans1.rotation.eulerAngles.z;
        float minArcPivot = rectTrans2.rotation.eulerAngles.z;
        float maxArcPivot = minArcPivot - fillAmount * 360.0f;
```

```
        isOverlapping = arrowAngle < minArcPivot && arrowAngle > maxArcPivot;
    }

    private void UpdateText()
    {
        successText.text = $"Успех: {ammountOfSuccess}";
        failText.text = $"Неудача: {ammountOfFails}";
    }

    private void ResetGame()
    {
        ResetArrow();
        rotationAmmount = 0;
        circleGenerator.Generate();
        isGameStarted = true;
    }
    private void ResetArrow()
    {
        rotationSpeed = Random.Range(50, 141);
        transform.rotation = Quaternion.identity;
        isRightDirection = Random.value > 0.5f;
    }
}
```

Приложение №5

Враг. Листинг программы

```
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Runtime.CompilerServices;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.AI;

public class TestAgent : MonoBehaviour
{
    public EnemyState enemyState = EnemyState.Patrul;
    public NavMeshAgent navMeshAgent;
    public Transform playerTransform;
    public Transform[] patrulPoints;
    public Transform[] pointsToHide;
    public Animator enemyAnimController;
    private Transform currentPatrulPoint;
    private Transform seekPoint;
    public Rigidbody playerRb;
    private Vector3 lastPlayerPos;
    public PlayerController playerController;

    private float idleTime = 4;
    private float patrulTime = 6;
    private float currentTimer = 0;
    private float distanceToPlayer;
    public bool canSeePlayer = true;
    private bool canAttack = true;
    public bool foundPlayer = false;
    private float timeOfLosingPlayer;

    private void Start()
    {
        GetRandomTarget();
    }
    void Update()
    {
        //Debug.Log("Player " + playerRb.velocity.magnitude);
        enemyAnimController.SetFloat("Velocity", navMeshAgent.velocity.magnitude);
        //Debug.Log(navMeshAgent.velocity.magnitude);
        switch (enemyState)
        {
            case EnemyState.IDLE:
                Idle();
                Debug.Log("IDLE");
                break;
            case EnemyState.Patrul:
                Patrul();
```

```
            Debug.Log("Patrul");
            break;
        case EnemyState.Chase:
            Chase();
            Debug.Log("Chase");
            break;
        case EnemyState.Seek:
            Seek();
            Debug.Log("Seek");
            break;
        default:
            break;
    }
    if (!canAttack)
    {
        currentTimer += Time.deltaTime;
        if (currentTimer >= patrulTime)
        {
            enemyState = EnemyState.Patrul;
        }
    }
    Debug.Log(navMeshAgent.destination);

}

private void FixedUpdate()
{
    EnemyOverview();
}
private void Seek()
{
    for (int i = 0; i < pointsToHide.Length; i++)
    {
        distanceToPlayer = Vector3.Distance(pointsToHide[i].position,
playerTransform.position);
        if (distanceToPlayer <= 1)
        {
            foundPlayer = true;
            seekPoint = pointsToHide[i];
            break;
        }
    }
    if (foundPlayer)
    {
        navMeshAgent.destination = seekPoint.position;
    }
    else
    {
        seekPoint = pointsToHide[Random.Range(0, pointsToHide.Length)];
        navMeshAgent.destination = seekPoint.position;
    }
```

```
    if (Vector3.Distance(transform.position, seekPoint.position) <= 2)
    {
        if (canSeePlayer)
        {
            enemyState = EnemyState.Chase;
        }
        else
        {
            currentTimer = 0;
            enemyState = EnemyState.IDLE;
        }
    }
}
private void EnemyOverview()
{
    DistanceToPlayer();

    RaycastHit hitInfo;
    Vector3 dirToPlayer = (playerTransform.position - GetSelfEyePosition()).normalized;
    if (Physics.Raycast(GetSelfEyePosition(), dirToPlayer, out hitInfo, 60))// true or false
    {
        canSeePlayer = hitInfo.transform.CompareTag("Player");
        Color cl = canSeePlayer ? Color.green : Color.red;
        Debug.DrawRay(GetSelfEyePosition(), dirToPlayer * 60, cl);
    }
    else
    {
        canSeePlayer = false;
    }
}

private Vector3 GetSelfEyePosition()
{
    return transform.position + Vector3.up * (navMeshAgent.height * 1f - 0.5f);
    //return transform.position;
}

private void Chase()
{
    if (!canAttack)
    {
        return;
    }
    navMeshAgent.speed = 4.5f; //3.5f
    navMeshAgent.destination = playerTransform.position;
    if (canSeePlayer)
    {
        navMeshAgent.destination = playerTransform.position;
        timeOfLosingPlayer = 0;
    }
    else
    {
```

```
            if (timeOfLosingPlayer <= 0.01f)
            {
                lastPlayerPos = playerTransform.position;
            }
            timeOfLosingPlayer += Time.deltaTime;
            navMeshAgent.destination = lastPlayerPos;
        }

        if (distanceToPlayer <= 2 && canSeePlayer)
        {
            Attack();
        }
        else if (timeOfLosingPlayer >= 2)
        {
            timeOfLosingPlayer = 0;
            enemyState = EnemyState.IDLE;
        }
    }
    private void Idle()
    {
        currentTimer += Time.deltaTime;
        if (currentTimer >= idleTime)
        {
            GetRandomTarget();
            currentTimer = 0;
            enemyState = EnemyState.Patrul;
        }
    }
    private void GetRandomTarget()
    {
        Transform temp;
        do
        {
            temp = patrulPoints[Random.Range(0, patrulPoints.Length)];
        } while (currentPatrulPoint == temp);
        currentPatrulPoint = temp;
    }
    private void Patrul()
    {
        navMeshAgent.speed = 1f;
        navMeshAgent.destination = currentPatrulPoint.position;
        if (Vector3.Distance(transform.position,currentPatrulPoint.position) <= 2)
        {
            enemyState = EnemyState.IDLE;
        }
    }
    private void DistanceToPlayer()
    {
        if (!canAttack)
        {
            return;
        }
```

```
        distanceToPlayer = Vector3.Distance(transform.position, playerTransform.position);
        if (distanceToPlayer <= 20 && canSeePlayer)
        {
            enemyState = EnemyState.Chase;
        }
    }
    private void Attack()
    {
        if (canAttack)
        {
            enemyAnimController.SetTrigger("Attack");
            canAttack = false;
            playerController.GetComponent<PlayerController>().isGameOver = true;
            playerTransform.GetComponent<PlayerInteraction>().GameOver();
            enemyAnimController.SetTrigger("Biting");
        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Generator : MonoBehaviour, IInteractable
{
    public string hint = "Press E to deactivate generator";
    public GameObject GeneratorGame;
    public ArrowMovement arrowMovement;
    public DoorToLevelOne doorToLevelOne;
    public DoorToSecondLevel doorToSecondLevel;
    public DoorToThirdLevel doorToThirdLevel;
    public static bool canInteract = true;
    public int idGenerator;
    public PlayerInteraction playerInteraction;
    public GameObject[] testMistakes;
    public void HideHint()
    {
        HintManager.instance.HideHint();
    }

    public void Interact()
    {
        if (!canInteract)
        {
            return;
        }
        GeneratorGame.SetActive(true);
        for (int i = 0; i < testMistakes.Length; i++)
        {

            testMistakes[i].SetActive(true);
        }
        arrowMovement.isGameStarted = true;
    }

    public void ShowHint()
    {
        if (!canInteract)
        {
            return;
        }
        HintManager.instance.ShowHint(hint);
    }

    void Update()
    {
        if (arrowMovement.isGameOver || !canInteract)
```

```
        {
            GeneratorGame.SetActive(false);
            if (playerInteraction.idGenerator == 1)
            {
                Debug.Log("1");
                doorToLevelOne.hint = "You have already passed this test!";
                doorToLevelOne.canInteract = false;
            }
            else if (playerInteraction.idGenerator == 2)
            {
                Debug.Log("2");
                doorToSecondLevel.hint = "You have already passed this test!";
                doorToSecondLevel.canInteract = false;
            }
            else if (playerInteraction.idGenerator == 3)
            {
                Debug.Log("3");
                doorToThirdLevel.hint = "You have already passed this test!";
                doorToThirdLevel.canInteract = false;
            }

        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using System.Net;
using UnityEngine;
using UnityEngine.AI;

public class BasementAgent : MonoBehaviour
{
    public EnemyState enemyState = EnemyState.IDLE;
    public NavMeshAgent navMeshAgent;
    public Transform playerTransform;
    public Transform[] pointsToHide;
    private float distanceToPlayer;
    public bool canSeePlayer = false;
    public bool canAttack = true;
    public bool foundPlayer = false;
    private float timeOfLosingPlayer;
    private Transform seekPoint;
    private float currentTimer = 0;
    public Animator enemyAnimController;
    private float idleTime = 4;
    private Vector3 lastPlayerPos;
    public PlayerController playerController;
    public CameraMovement cameraMovement;
    public bool heroInBasement = false;

    void Update()
    {
        enemyAnimController.SetFloat("Velocity", navMeshAgent.velocity.magnitude);
        //Debug.Log(navMeshAgent.velocity.magnitude);
        switch (enemyState)
        {
            case EnemyState.IDLE:
                Idle();
                Debug.Log("IDLE");
                break;
            case EnemyState.Chase:
                Chase();
                Debug.Log("Chase");
                break;
            case EnemyState.Seek:
                Seek();
                Debug.Log("Seek");
                break;
            default:
                break;
        }
        if (enemyState == EnemyState.Seek || enemyState == EnemyState.Chase)
```

```
        {
            cameraMovement.secondTrial = true;
        }
    }

    private void FixedUpdate()
    {
        EnemyOverview();
    }
    private void Seek()
    {
        currentTimer += Time.deltaTime;
        if (currentTimer >= idleTime)
        {
            currentTimer = 0;
        }
        for (int i = 0; i < pointsToHide.Length; i++)
        {
            distanceToPlayer = Vector3.Distance(pointsToHide[i].position,
playerTransform.position);
            if (distanceToPlayer <= 2)
            {
                Debug.Log("Rabotaet!");
                foundPlayer = true;
                seekPoint = pointsToHide[i];
                break;
            }
        }
        if (foundPlayer)
        {
            navMeshAgent.destination = seekPoint.position;
        }
        else
        {
            Debug.Log("Random!!!");
            seekPoint = pointsToHide[Random.Range(0, pointsToHide.Length)];
            navMeshAgent.destination = seekPoint.position;
        }

        if (Vector3.Distance(transform.position, seekPoint.position) <= 2)
        {
            if (canSeePlayer)
            {
                enemyState = EnemyState.Chase;
            }
            else
            {
                currentTimer = 0;
                enemyState = EnemyState.IDLE;
            }
        }
    }
```

```
private void Idle()
{
    currentTimer += Time.deltaTime;
    if (currentTimer >= idleTime)
    {
        currentTimer = 0;
        navMeshAgent.destination = new Vector3(-19f, -6f, -16f);
    }
}

private void Chase()
{
    if (!canAttack)
    {
        return;
    }
    navMeshAgent.speed = 2.5f; //3.5f
    navMeshAgent.destination = playerTransform.position;
    if (canSeePlayer)
    {
        navMeshAgent.destination = playerTransform.position;
        timeOfLosingPlayer = 0;
    }
    else
    {
        if (timeOfLosingPlayer <= 0.01f)
        {
            lastPlayerPos = playerTransform.position;
        }
        timeOfLosingPlayer += Time.deltaTime;
        navMeshAgent.destination = lastPlayerPos;
    }

    if (distanceToPlayer <= 1 && canSeePlayer)
    {
        Attack();
    }
    else if (timeOfLosingPlayer >= 2)
    {
        timeOfLosingPlayer = 0;
        enemyState = EnemyState.IDLE;
    }
}

private void Attack()
{
    if (canAttack)
    {
        enemyAnimController.SetTrigger("Attack");
        canAttack = false;
        playerController.GetComponent<PlayerController>().isGameOver = true;
```

```
            playerController.GetComponent<PlayerController>().force = 0;
            playerTransform.GetComponent<PlayerInteraction>().GameOver();
            enemyAnimController.SetTrigger("Biting");
        }
    }

    private void EnemyOverview()
    {
        DistanceToPlayer();

        RaycastHit hitInfo;
        Vector3 dirToPlayer = (playerTransform.position - GetSelfEyePosition()).normalized;
        if (Physics.Raycast(GetSelfEyePosition(), dirToPlayer, out hitInfo, 60))// true or false
        {
            canSeePlayer = hitInfo.transform.CompareTag("Player");
            Color cl = canSeePlayer ? Color.green : Color.red;
            Debug.DrawRay(GetSelfEyePosition(), dirToPlayer * 60, cl);
        }
        else
        {
            canSeePlayer = false;
        }
    }

    private Vector3 GetSelfEyePosition()
    {
        return transform.position + Vector3.up * (navMeshAgent.height * 1f - 0.5f);
        //return transform.position;
    }

    private void DistanceToPlayer()
    {
        if (!canAttack)
        {
            return;
        }
        distanceToPlayer = Vector3.Distance(transform.position, playerTransform.position);
        if (distanceToPlayer <= 20 && canSeePlayer)
        {
            enemyState = EnemyState.Chase;
        }
    }

}
```