Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б10-мм

Обзор библиотеки Unity

Мусаев Немат Адалат оглы

Научный руководитель:
старший преподователь кафедры системного программирования, к.т.н. Ю. В. Литвинов

Санкт-Петербург, 2024г.

Оглавление

Оглавление

1. Введение

Движок – это программа для создания разнообразных игр и развлекательных приложений. Сегодня в IT можно найти немало такого контента. Каждый движок обладает своими особенностями и областями применения. Одним из наиболее популярных программ для написания игры является платформа Unity 3D.

Unity Platform Technology – это не просто движок, а полноценная среда разработки. Можно создавать с ее помощью компьютерные и консольные игры, имея минимальный набор знаний в сфере программирования. Рассматриваемое приложение после установки на компьютер предложит самые разные программные средства для разработки:

- текстовый редактор;
•отладчик;
•готовую физику и механику;
•компилятор и другие элементы.

Unity Engine делает процедуру выпуска игр простым и комфортным. Кроссплатформенность дает возможность охватить максимальное количество игровых платформ и операционных систем.

Преимущества

Unity Engine – проект для создания развлекательного контента, который имеет определенные сильные и слабые стороны. К преимуществам этой среды разработки относят:

1. Доступность. Приложение имеет несколько тарифов – каждый предлагает отдельную функциональность. Для выпуска игр можно использовать предложение Personal, которое предназначается для частных лиц и небольших компаний. При желании можно будет переключиться на более продвинутые тарифы (Plus, Pro). Для обучения допустимо использовать бесплатную версию Unity.
2. Низкий порог вхождения. Для создания игр на рассматриваемой платформе требуется минимум знаний и навыков в области программирования. Написать развлекательный контент получится даже у того, кто не умеет писать код. У Unity имеется библиотека Asset Store, в которой поддерживаются готовые шаблоны для персонажей, звуков, фонов и так далее. Эти элементы могут использоваться для первых проектов.
3. Быстрое обучение. Разобраться с Unity Technologies самостоятельно не слишком трудно. По этой платформе создано множество видеоуроков, а также документации и гайдов на русском языке. Сама среда разработки предлагает раздел Learn. В нем собраны обучающие материалы. С их помощью разработчик научится создавать проекты, размещать персонажей, формировать различные уровни сложности и собирать продукт в единое целое. Все это бесплатно.

Недостатки

Несмотря на то, что игры через Unity создаются быстро и с минимальными навыками в области разработки, недостатки у этого ПО тоже есть. К ним относят следующие моменты:

1. Низкая производительность. Чтобы получить на выходе хороший проект, необходимо учитывать тонкости разработки пользовательского интерфейса. Обычно приложения на «Юнити» менее быстрые, чем написанные с помощью других game programming technologies.
2. Оптимизация. Кроссплатформенный движок имеет меньшую производительность по сравнению с узконаправленными программами. Это сказывается на скорости работы игры, качестве графики и FPS (частоте кадров в секунду). Оптимальный кадровый диапазон – 30-60 FPS. Unity3D подойдет для создания элементарных проектов, но для игры класса AAA лучше подобрать другой инструмент разработки.
3. Отсутствие сложных шаблонов. Простой проект можно собрать на классических встроенных шаблонах. Обычно они используются для обучения. Как только игра становится более сложной, нужно тщательно продумывать архитектуру.

Несмотря на наличие весьма существенных недостатков разработки при создании игры, Unity3D Technologies пользуется спросом как у новичков, так и у более опытных программистов. Особо сложные проекты и известные компании в области games programming нередко пишут собственные движки (пример – Capcom и их RE Engine). Но Unity не исключает возможность создания качественного проекта. Эта платформа используется только при разработке игрового контента.

Вот несколько самых известных игр, написанных при помощи Unity:

- PockemonGO;
- Among Us;
- Outlast;
- Hearthstone;
- Genshin Impact.

2. Постановка задачи

Целью работы является изучение движка Unity и его компонентов, рассмотрение работы Unity 2D/3D на основе простейшей игры, разработанной на данной платформе. Для такой цели постаавим некоторые задачи:

- Обзор на интерфейс движка
- Обзор на компоненты игровых моделей
- Обзор на сами игровые компоненты
- Создание простейшей игры на Unity

3. Обзор

3.1 Терминология Unity

Перед созданием игры важно продумать все моменты и представить общую картину, продумать опорные точки сюжета и механики. Для создания игры именно Unity также пригодится понимание некоторых базовых терминов, с которыми постоянно придется сталкиваться в процессе разработки:

•**Ассет (Asset) —** готовый компонент, который можно использовать для создания своих проектов. Это могут быть элемент интерфейса в игре, текстура, фигурка персонажа, шрифт или звук.
•**Игровой объект (GameObject) —** это любой ассет, который используется в игровой сцене. Например, изображение монетки, сам ее внешний вид — это ассет, а пять монет, которые должен подобрать персонаж в процессе прохождения уровня — это пять игровых объектов. Сам персонаж при этом тоже станет игровым объектом.
•**Компоненты (Components) —** часть игрового объекта, отвечающая за его поведение в процессе игры: перемещение или реакцию на определенные триггеры.
•**Скрипт (Script) —** код на C#, в котором прописаны конкретные условия работы компонента.

При разработке игр или приложний проще начинать с 2D-проектов, так как для этого формата создано больше готовых ассетов. Конечно, можно сразу начать делать 3D-игры, но в этом случае многие элементы и анимации придется самостоятельно создавать с нуля или выделять бюджет на то, чтобы делегировать эту часть работы другим специалистам.

3.2 Настройка интерфейса

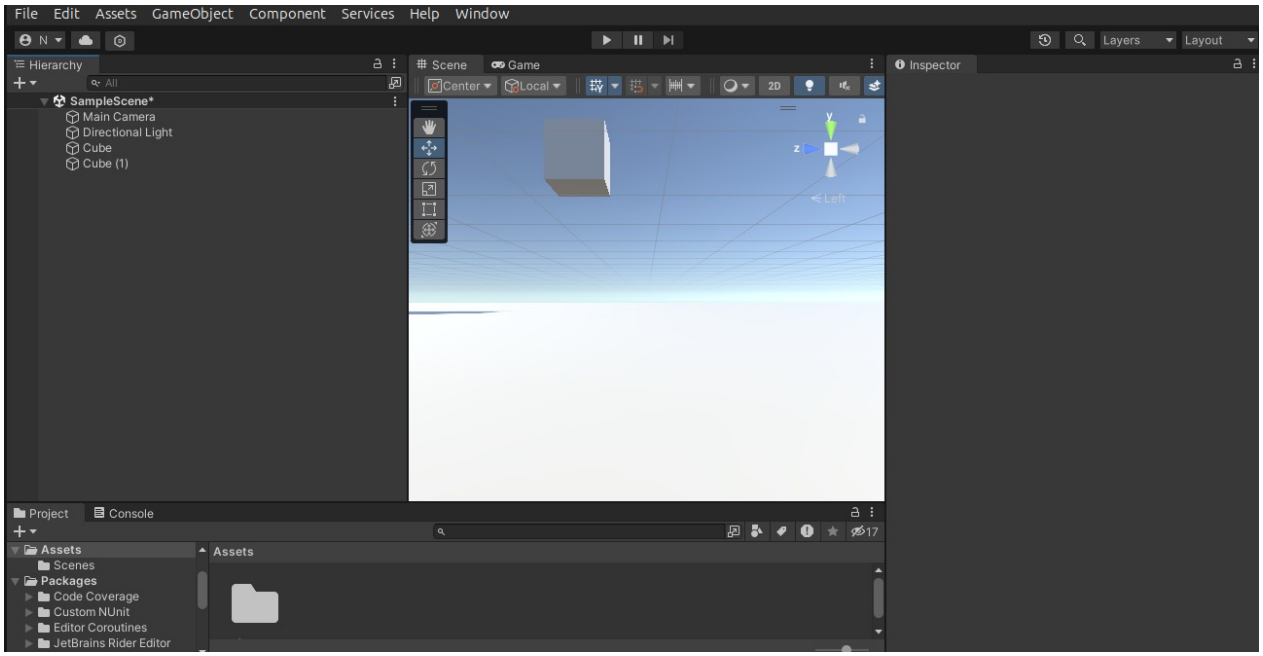
Библиотека Unity представляет собой набор различных окон с разными функциями и свойствами

В стандартном интерфейсе проекта шесть элементов рабочей области:

1.**Верхняя панель инструментов —** в ней находятся стандартные вкладки File, Edit, Help, как во многих других интерфейсах, а также вкладки Assets, GameObject, Components и Window.
2.**Scene —** окно сцены, в котором выстраивается игровое пространство (элементы игрового мира, текстуры, фигурки персонажей и прочее).
3.**Games —** это окно игры, в котором можно посмотреть глазами пользователя, как будут двигаться элементы и работать игровые механики.

4.**Hierarchy —** окно иерархии, в нем перечислен список всех элементов (GameObject), которые помещены в окно Scene.
5.**Project —** это система папок, в которых хранятся ассеты по категориям (текстуры, шрифты, звуки и т.д.).
6.**Inspector —** окно для изменения элементов игры, их размера, цвета, положения в пространстве и других характеристик.

7.**Toolbar** Ближе к левому краю размещаются кнопки Pan, Move, Rotate, Scale, а в центре — Play, Pause, Advance Frame. Play почти мгновенно запускает игру без выполнения отдельных сборок. Pause приостанавливает игру, а Advance Frame обеспечивает покадровое выполнение, предоставляя вам очень жесткий отладочный контроль



3.3 Структура Проекта

Проекты содержат папки Assets, Library, ProjectSettings и Temp, но в интерфейсе появляется только папка Assets.

Папка Assets содержит все ваши ресурсы: художественную графику, код, звуки; любой отдельный файл, добавляемый в проект, попадает именно в эту папку. Она всегда является папкой верхнего уровня в Unity Editor. Изменения следует вносить только через интерфейс Unity, — никогда не пытайтесь делать это через файловую систему.

Папка Library — это локальный кеш для импортированных ресурсов; она содержит все метаданные для ресурсов. В папке ProjectSettings хранятся настройки, заданные вами в Edit | Project Settings. Папка Temp используется для временных файлов, создаваемых Mono и Unity в процессе сборки проекта.

3.4 Inspectores (компоненты игровых объектов)

Вы добавляете функциональность к объектам GameObject добавлением компонентов (объектов Component). Все, что вы добавляете, — это Component, и все они показываются в окне Inspector. Существуют компоненты MeshRender и SpriteRender, компоненты для звука и функционала камеры, компоненты, относящиеся к физике (коллайдеры [colliders] и твердые тела [rigidbodies]), системы частиц, системы поиска пути, сторонние пользовательские компоненты и др. Чтобы назначить код какому-то объекту вы используете скриптовый Component. Компоненты — это как раз то, что оживляет ваши

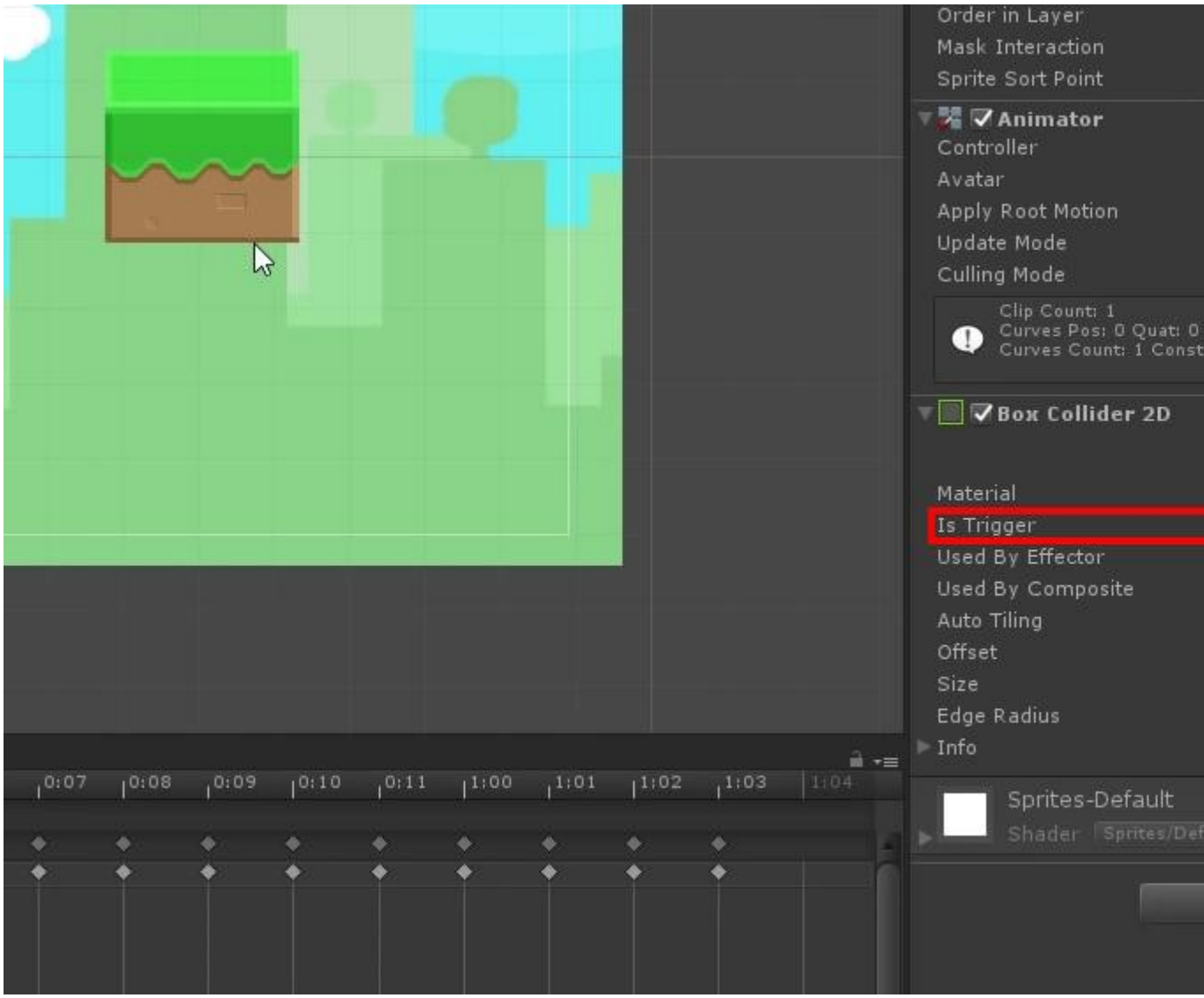
1. **Box colliders.** Это нестандартная компонента игровых объектов, но многие объекты обладают таким свойством для создания реалистичной игры. Это компонент, который присваивается объекту в пространстве игры, задает форму и делает его твердым, недоступным для прохождения сквозь него. Например, если мы разместим монетку в 2D-пространстве и захотим сделать так, чтобы она упала на платформу, то без использования компонента Collider ничего не получится — монетка пролетит сквозь платформу.

Поэтому обоим объектам необходимо присвоить компонент Box Collider 2D — это тонкая зеленая линия, которая обводит элементы по контуру, и за счет этой рамки они становятся твердыми, то есть один не может пройти сквозь другой.

Так объекты обязательно соприкоснутся и монета встанет на платформу.

Триггер (Trigger) — это пространство на карте, при попадании объекта в которое происходит действие; он тоже обводит объект или область в пространстве по краям. По сути, это тот же коллайдер, только триггер позволяет объектам проходить внутрь этой области. Представьте, что на ту же самую платформу вместе с коллайдером наброшен триггер, и при попадании персонажа внутрь триггерной области активируется телепорт — персонажа перебрасывает в другую точку карты.

Чтобы создать триггер, нужно накинуть тот же самый компонент коллайдера, но поставить галочку Is Trigger.



(https://blog.skillfactory.ru/kak-sozdat-igru-na-unity/)

Триггеры распознают три варианта взаимодействия области на карте и объекта:

•OnTriggerEnter — объект зашел в зону;
•OnTriggerStay — объект находится в зоне;
•OnTriggerExit — объект покинул зону.

2. **Transport**. Свойство существует по умолчани, этот компонент задаётся для любых игровы объектов, поскольку Unity должен знать позиции объектов в сценах, где они расположены. Свойство Transform определяет позицию, поворот и масштаб любого GameObject. Unity использует левостороннюю систему координат, в которой координаты на экране компьютера рассматриваются как X (по горизонтали), Y (по вертикали) и Z (глубина, т. е. входящие или исходящие из экрана).

В разработке игр весьма распространено использование векторов. Transform.Position и Transform.Scale являются объектами Vector3. Vector3 — это трехмерный вектор; иначе говоря, в нем не более трех точек: только X, Y и Z. Манипулируя этими тремя простыми значениями, вы можете задавать местонахождение объекта и даже перемещать его в направлении вектора.

Свойство Transform определяет позицию, поворот и масштаб любого GameObject.

3. **Rigitbody.** Так же, как и Коллайдер является физическим компонентом (определяет физику объектов игр), указывает на наличие гравитации и следовательно скорости свободного падения тел.

3.5 Gameobject

Практически все в вашей сцене является GameObject. Например, System.Object в .NET Framework. От него наследуют почти все типы. Та же концепция относится и к GameObject. Это базовый класс для всех объектов в сцене Unity. Все объекты, показанные на рис. 5 (и многие другие), наследуют от GameObject.

4. Реализация

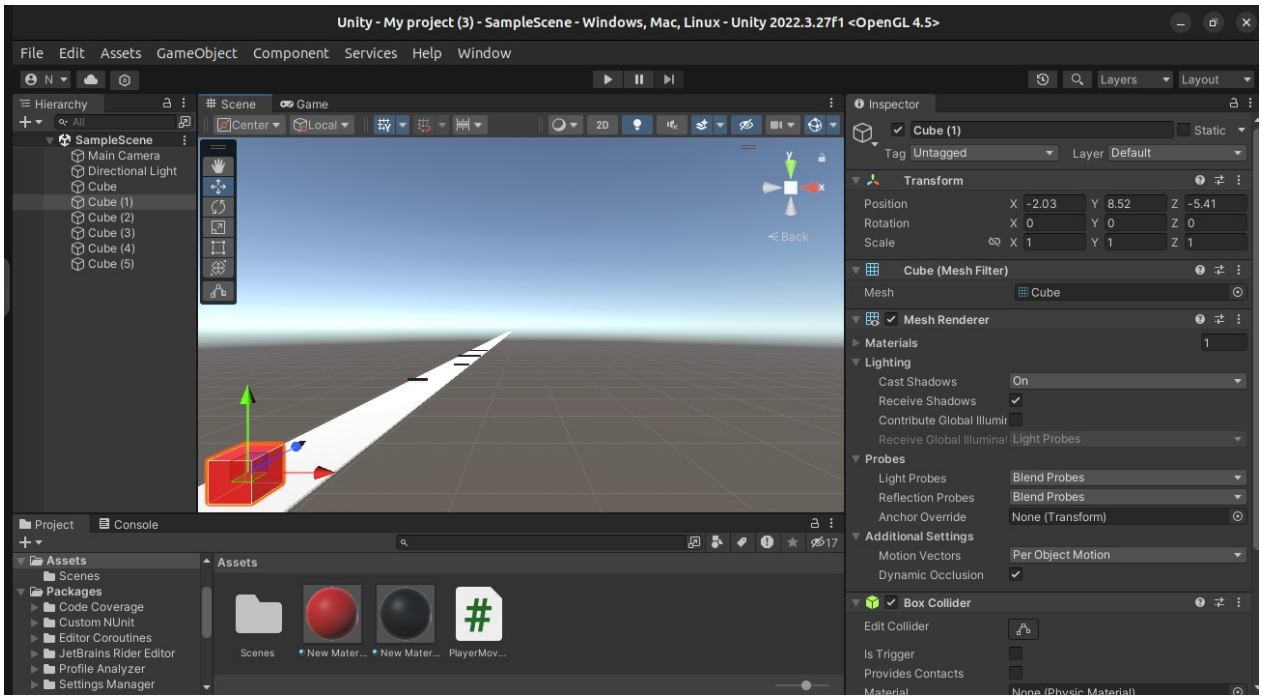
Класс нового созданного объекта наследует от базового класса MonoBehavior, что позволяет просто назначить этот класс объекту GameObject. В таком базовом классе богатая функциональность, и, как правило, вы чаще всего будете использовать несколько методов и свойств. Основными методами являются те, которые Unity будет вызывать, если они присутствуют в вашем классе. Несмотря на множество методов вы, как правило, используете лишь некоторые из них. Ниже перечислены наиболее распространенные методы, подлежащие реализации в ваших классах, производных от MonoBehavior.

Awake Этот метод вызывается один раз для каждого объекта при его первой инициализации. Другие компоненты могут быть еще не инициализированы, поэтому этот метод обычно используется для инициализации текущего GameObject. Для инициализации производного от MonoBehavior класса следует всегда использовать этот метод, а не его конструктор. И не пытайтесь запрашивать здесь другие объекты сцены, так как они могут быть еще не инициализированы.

Start Этот метод вызывается на первом кадре жизненного цикла объекта, но перед любыми методами Update. Он может показаться очень похожим на Awake, но в случае Start уже известно, что другие объекты инициализированы через Awake и существуют в вашей сцене, поэтому вы можете запрашивать другие объекты, **Update** Этот метод вызывается в каждом кадре. И каждую секунду частота кадров

может варьироваться. Вы можете щелкнуть кнопку Stats на вкладке Game при переключении в игровой режим, чтобы видеть текущую частоту кадров.

Напишем простейшую игру, в которой наш главны игрок в в иде куба будет преодолевать препятсвия, двигаясь по полосе.



Для начала напишем скрипт взаимодействия пользователя с игрой:

```
public float runSpeed = 500f;
public float strafeSpeed = 500f;
public float jumpForce = 15f;

protected bool strafeLeft = false;
protected bool strafeRight = false;
protected bool doJump = false;

Ссылок: 0
void Update()
{
    if(Input.GetKey("d"))
    {
        strafeLeft = true;
    } else
    {
        strafeLeft = false;
    }

    if(Input.GetKey("a"))
    {
        strafeRight = true;
    } else
    {
        strafeRight = false;
    }

    if(Input.GetKeyDown("space"))
    {
        doJump = true;
    }
}
```

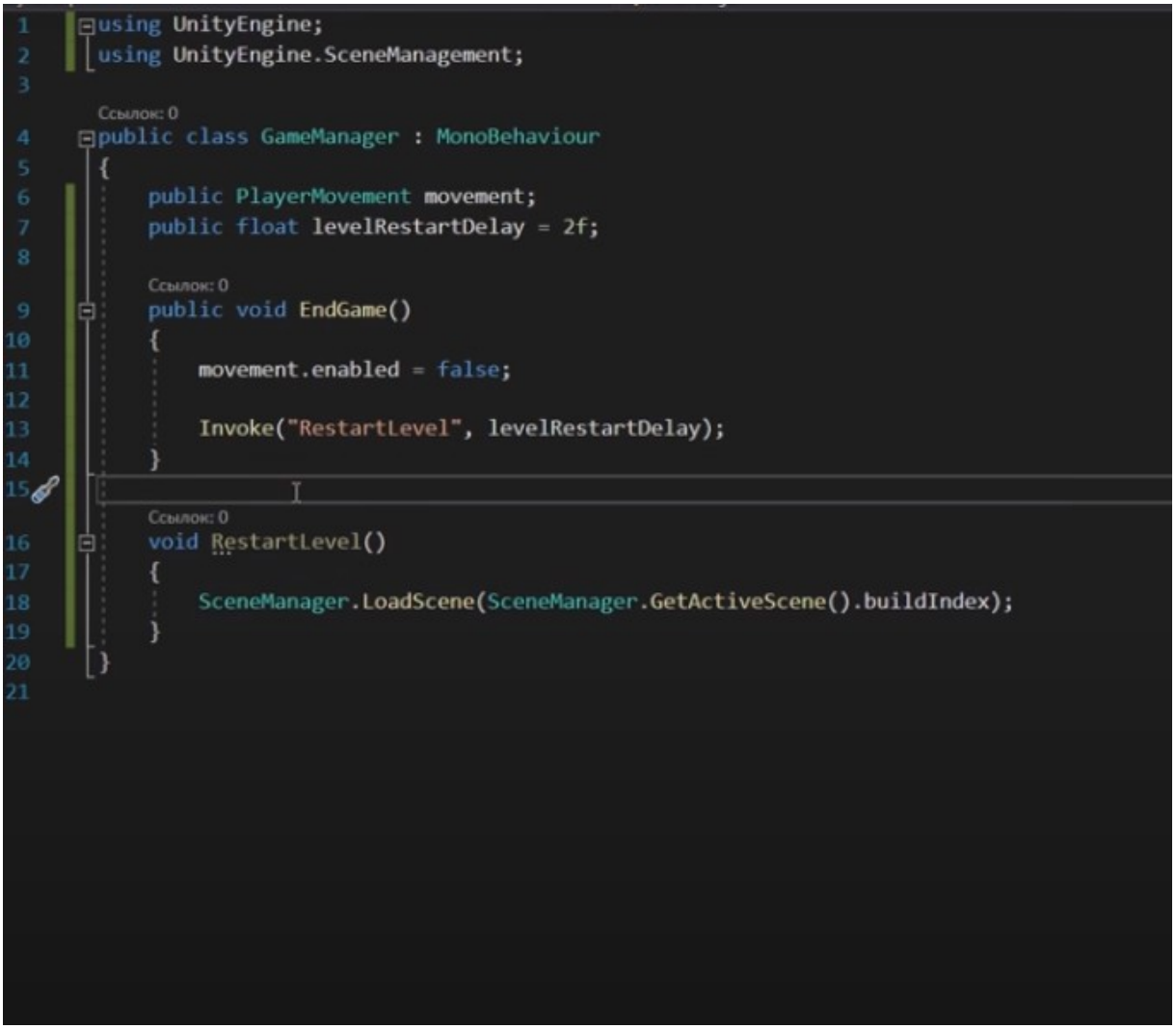
Затем в скрипте PlayerMovement используется метод AddForce компоненты Rigidbody, принимающий вектор силы и режим применения силы, в данном случае — изменение скорости. Чтобы скорость не набиралась бесконтрольно, используется метод LimitVelocity, ограничивающий ее. При беге эта максимальная скорость повышается.

```
44      void FixedUpdate()
45      {
46          rb.AddForce(0, 0, runSpeed * Time.deltaTime);
47
48          if(strafeLeft)
49          {
50              rb.AddForce(-strafeSpeed * Time.deltaTime, 0, 0, ForceMode.VelocityChange);
51          }
52
53          if (strafeRight)
54          {
55              rb.AddForce(strafeSpeed * Time.deltaTime, 0, 0, ForceMode.VelocityChange);
56          }
57
58          if(doJump)
59          {
60              rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
61
62              doJump = false;
63          }
64      }
65  }
66
```

Здесьс проверяем, если игрок падает с платформы по оси Y, то игра завершается, мы проиграли. (Дописываем в Update)

```
    if(transform.position.y < -5f)
    {
        Debug.Log("Конец игры!");
    }
}
```

Дальше создадим новый компоннет объекта игрового, который будет нам говорит на каком уровне мы игры, по-другому говоря,



```
using UnityEngine;
using UnityEngine.SceneManagement;

Ссылок: 0
public class GameManager : MonoBehaviour
{
    public PlayerMovement movement;
    public float levelRestartDelay = 2f;

    Ссылок: 0
    public void EndGame()
    {
        movement.enabled = false;

        Invoke("RestartLevel", levelRestartDelay);
    }

    Ссылок: 0
    void RestartLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
}
```

скажет, удалось ли нам пройти препятсвие или нет. Назовём его GameManager. Запишем в таком скрипте код, по которому мы будем отключать другой скрипт PlayerMovement. В другом методе текущего скрипта будем обновлять наш уровень, то есть придём к исходной точке игры

В случае поражения будем обновлять уровень через класс SceneManager и его метод загрузки сцены(LoadScene).

5. Список литературы

https://blog.skillfactory.ru/kak-sozdat-igru-na-unity/

https://habr.com/ru/articles/161463/