ЧОУ ВО «Тольяттинская академия управления»

Кафедра Прикладной информатики и высшей математики

Б1.В.18

ОЦЕНОЧНЫЕ МАТЕРИАЛЫ (СРЕДСТВА)

Учебная дисциплина	*Разработка приложений в Unity*
По направлению подготовки	*09.03.03*
	«Прикладная информатика»
Профиль (программа бакалавриата)	*«Дизайн и разработка цифровых программных продуктов»*
Форма обучения	*Очная*

Оценочные материалы (средства) дисциплины рассмотрены (актуализированы) и утверждены на заседании кафедры прикладной информатики и высшей математики

Протокол заседания № 10 от «25» мая 2026 г.

Заведующий кафедрой Стрекалова Наталья Борисовна

2025

1. ПЕРЕЧЕНЬ КОМПЕТЕНЦИЙ И ЭТАПЫ ИХ ФОРМИРОВАНИЯ В ПРОЦЕССЕ ОБУЧЕНИЯ ПО ДИСЦИПЛИНЕ

Оценочные материалы (средства) сформированы в соответствии с ФГОС ВО - бакалавриат по направлению подготовки 09.03.03 «Прикладная информатика», утвержденный приказом Министерства образования и науки Российской Федерации от 19.09.2017 № 922 (с изменениями и дополнениями от 26.11.2020, 08.02.2021). В соответствии с матрицей компетенций основной профессиональной образовательной программы «Дизайн и разработка цифровых программных продуктов» (уровень бакалавриата) в процессе обучения по дисциплине «Разработка приложений в Unity» происходит формирование закрепленных за дисциплиной компетенций обучающихся. Оценка сформированности компетенций на каждом этапе обучения происходит через оценку планируемых результатов обучения по дисциплине (знаний, умений, навыков).

Результаты освоения образовательной программы (компетенции)	Индикаторы достижения компетенций	Планируемые результаты обучения по дисциплине	Этапы формирования компетенции (семестры и темы)	Оценочное средство
ПК-3 Способен разрабатывать цифровой программный продукт	**ПК-3.1** - Осуществляет разработку программного кода и компонентов цифрового программного продукта с помощью специализированных цифровых платформ для индивидуальной и совместной разработки	**Знать:** - теоретические основы разработки 2D-приложений и 3D-приложений; виды объектов и компонентов для создания сцен; порядок создания программных скриптов, их привязки к объектам. **Уметь:** - создавать 2D-сцены и 3D-сцены; - разрабатывать скрипты для 2D-сцены и 3D-сцены; **Владеть:** - навыками кодирования на языке C#; работы с объектами, компонентами и текстурами; отладки программного кода.	**6 семестр** Тема 1. Введение в Unity, основные инструменты платформы Тема 2. Разработка 2D-приложений Тема 3. Разработка 3D-приложений Тема 4. Создание пользовательского интерфейса	- подготовка глоссария по теме 1 - практические задания по темам 2- 4 - ответ на теоретический вопрос экзамене - выполнение творческого задания на экзамене

2. ОЦЕНОЧНЫЕ МАТЕРИАЛЫ (СРЕДСТВА) ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ И ОЦЕНКИ ЗНАНИЙ, УМЕНИЙ, НАВЫКОВ, ОБЕСПЕЧИВАЮЩИХ ФОРМИРОВАНИЯ КОМПЕТЕНЦИЙ В ПРОЦЕССЕ ОБУЧЕНИЯ ПО ДИСЦИПЛИНЕ

Подготовка глоссария по теме «Введение в Unity, основные инструменты платформы»

Задание:

1. Используя учебно-методические материалы глобальной сети, разработать глоссарий ключевых понятий (не менее 15 терминов):
2. Глоссарий оформить в виде таблицы:

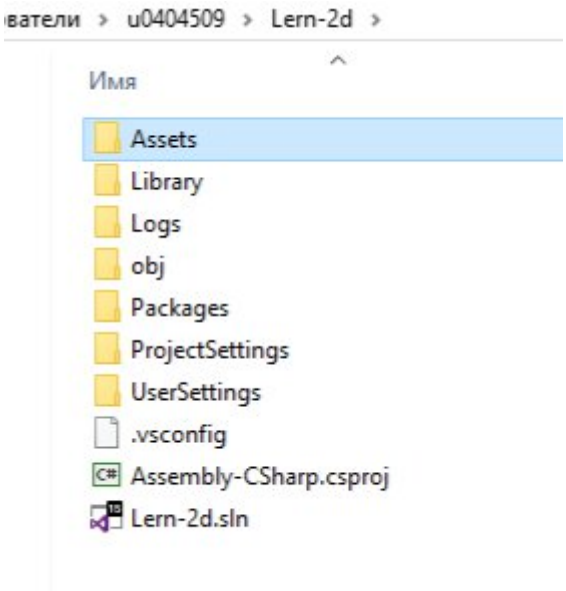
Термин / понятие	Определение	Источник

Критерии оценивания глоссария:

- оценка «отлично» выставляется студенту, если он представил не менее 15 ключевых понятий с «кликабельными» ссылками и корректным оформлением таблицы;
- оценка «хорошо» выставляется студенту, если он представил не менее от 10 до 14 ключевых понятий с «кликабельными» ссылками и корректным оформлением таблицы;
- оценка «удовлетворительно» выставляется студенту, если он представил от 10 до 14 ключевых понятий, но либо ссылками не «кликабельны», либо не корректное оформление таблицы;
- оценка «неудовлетворительно» выставляется студенту, если он не представил глоссарий или количество понятий в глоссарии менее 10.

Практическая работа по теме «Создание пользовательского интерфейса»

1. Запустить Unity3D
2. Создать новый 2D-проект
3. Ознакомиться с окнами среды Unity3D (Hierarchly, Scene, Game, Inspector, Project, Console)
4. Проверить работоспособность пустого проекта
5. Ознакомиться с перечнем файлов, создаваемых в проекте

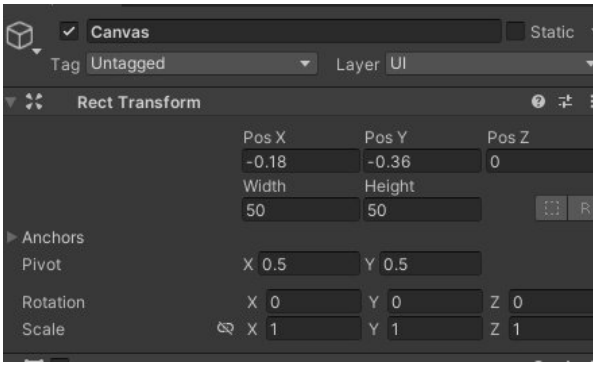


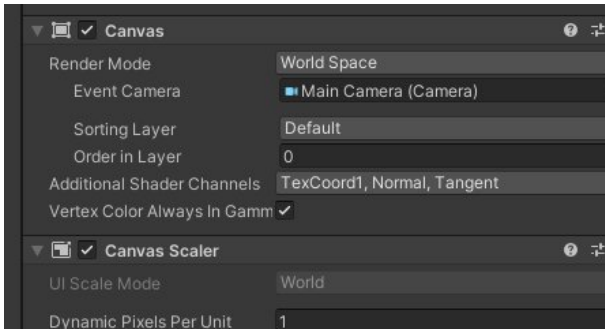
6. Используя окно Inspector, настроить камеру сцены (MainCamera): Clear Flags (Solid), Background (цвет фона), Size (расстояние до камеры лучше сразу увеличить в 2-3 раза)
7. Проверить работу сцены (приложения) в разных режимах и симуляторах.
8. Выполнить команду File-Build and Run. Убедиться в наличие exe файла (или apk – при настройке на мобильное устройство) и проверить его работу на устройстве.
9. Разработать скрипт для вывода в консоль сообщения о запуске приложения:
    a. В окне Project в папке Assets с помощью правой кнопки мыши создать пустой C#-скрипт под названием «scrConsole»
    b. Открыть двойным щелчком скрипт (в текстовом редакторе или Visual Studio, что зависит от настроек проекта)
    c. В функцию Start вписать команду Print(“Приложение начало свою работу!”);

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

public class Hello : MonoBehaviour
{


    void Start()
    {

        print("Всем привет!");
    }


    void Update()
    {
        print("Работаю :)");

    }
}
```

d. Сохранить скрипт

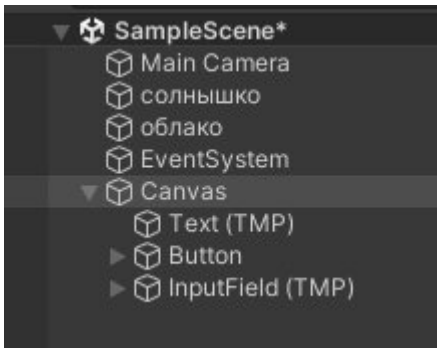
10. Добавить «Солнышко» в виде спрайта на сцену:
    a. Скачать с интернета любое двухмерное изображение солнца без фона в формате jpg.
    b. В окне Project в папке Assets подпапку Sprite и разместить изображение в ней
    c. Перетащить изображение на сцену, подкорректировать размеры при необходимости в окне Inspector. В окне Hierarchly переименовать спрайт в objSun (если изначально название не отражает суть)
11. Добавить на сцену любой 2D-объект (например, круг) под названием objCloud. Настроить внешний вид по своему усмотрению (вытянутый овал, голубой цвет и т.п.). Убедиться, что при запуске проекта объект виден в камеру. При необходимости изменить местоположение объекта.
12. Добавить скрипт scrConsole к либо объекту objCloud обычным перетаскиваем мыши (либо в окно Hierarchly, либо в окно Inspector)
13. Убедиться, что текст сообщения выводится в консольное окно. Проконтролировать время отражения сообщений.
14. Внести изменения в текст скрипта scrConsole – выводить текст сообщения (например, Приложение продолжает работать) в функции Update. Убедиться, что текст сообщения выводится в консольное окно. Проконтролировать время отражения сообщений.
15. Добавить на сцену объект Canvas (холст). Настроить холст следующим образом:
    a. Используя окно Inspector, добиться корректного размещения холста на сцене (координаты местоположения должны совпадать с центром сцены, размеры холста могут быть чуть больше сцены)
    b. Настроить холст на камеру (она одна и ее достаточно перетащить мышкой в окно Inspector для холста)

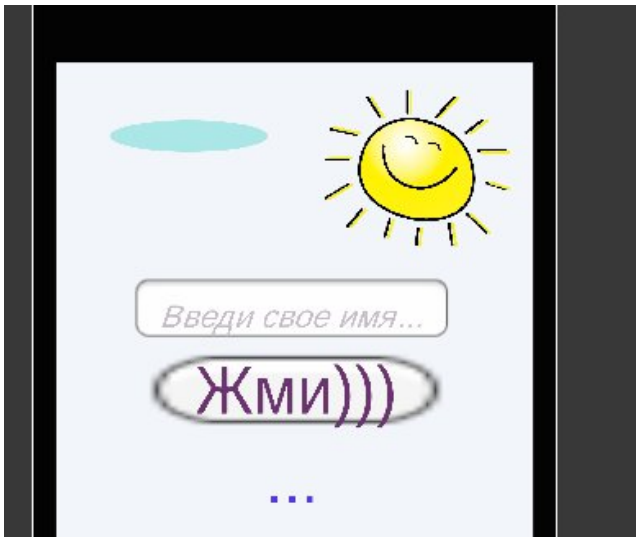




16. Добавить на холст элементы пользовательского интерфейса (UI): текстовое окно (Text) – для вывода приветствия, окно ввода (InputField) – для ввода ФИО, кнопку (Button) – для генерации приветствия. Во время установки объекта Text может понадобиться импорт данного объекта (дважды, подтверждаем). Дать всем объектам новые названия. Используя

окно Inspector, добиться корректного размещения всех компонентов на холсте, обращая особое внимание на размер текста (он, как правило, слишком большой; также помогает свойство Scale).

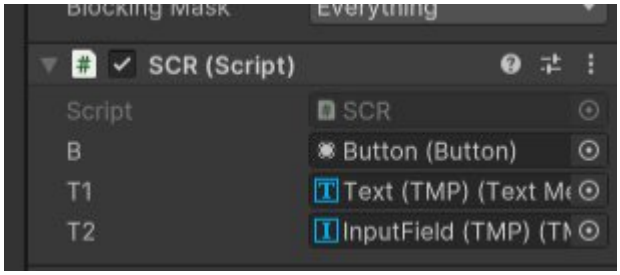




17. Создать новый скрипт для вывода приветствия при старте приложения под названием scrCanvas. Связать пустой скрипт с объектом Canvas
18. Открыть скрипт scrCanvas. Добавить библиотеки: UnityEngine.UI, TMPro. Объявить три открытые переменные для каждого объекта пользовательского интерфейса. Внимание: для более ранних версий Unity3D могут потребоваться другие типы текстовых полей (Text, InputField)

```
public Button b;
public TextMeshProUGUI t1;
public TMP_InputField t2;
```

19. Сохранить скрипт, вернуться в Unity3D, выделить в Иерархии объект Canvas и убедиться, что в окне Inspector в разделе скрипта отражаются объявленные переменные. С помощью мыши перетащить нужные объекты в пустые поля переменных.



20. Вернуться в код скрипта и в функции Start в текстовое поле внести сообщение «Привет». Сохранить скрипт, запустить приложение в Unity3D, убедиться в работоспособности скрипта
21. Отредактировать скрипт: в функции Update внести изменения в текстовое окно (можно добавлять один символ). Проверить работоспособность скрипта.
22. Добавить событие onClick для кнопки
    a. Добавить в скрипт scrCanvas новую функцию для обработки события (имя может быть любым)

```
public void Click()
{
    t1.text += ", "+t2.text+"!";
}
```

b. К объекту Button в окне Inspector в разделе OnClick добавить объект со скриптом – Canvas, его скрипт и разработанную функцию для события

23. Проверить работоспособность кнопки.
24. Отредактировать функцию обработки события – добавить проверку наличия текста в InputField. Проверить работоспособность.
25. Выполнить команду File-Build and Run. Убедиться в наличие exe файла (или apk – при настройке на мобильное устройство) и проверить его работу на устройстве.
26. Показать работу преподавателю.

Домашнее задание: построить глоссарий всех новых понятий
Внести изменения в код: добавить в скрипт scrCanvas событие EndEdit для изменения консольного вывода: «Теперь работает»+имя. По аналогии с п.22 привязать скрипт к InputField. Проверить работу скрипта.

Критерии оценивания практической работы:

- оценка «отлично» выставляется студенту, если он полностью выполнил задания, включая самостоятельную работу;
- оценка «хорошо» выставляется студенту, если он полностью выполнил задания, но не выполнил самостоятельную работу;
- оценка «удовлетворительно» выставляется студенту, если он выполнил задания частично и не выполнил самостоятельную работу;
- оценка «неудовлетворительно» выставляется студенту, если он не представил выполненную работу.

Практическая работа по теме «Разработка 2D-приложений»

Занятие 2 – Построение 2D-приложений (версия редактора 2022)

Общее задание: разработать игровое приложение, в котором Игрок, двигаясь по сцене, собирает «мандарины», при этом убирает или обходит потенциальные препятствия. Для интереса на сцену накладываем звук.
Технология выполнения:

1. **Стартовые действия**
   1.1. Запустить Unity3D
   1.2. Создать новый 2D-проект (версия редактора 2022)
   1.3. Выбираем режим игры (Симулятор или Game)
   1.4. Настраиваем камеру (MainCamera) в окне Inspector: Clear-Flags – Solid Color (заливка экрана текущим фоновым цветом); Background – белый (или любой другой по желанию), Clipping-Planes: Near-0, Far - 20
   1.5. Проверяем работоспособность проекта
2. **Работа с 2D графическими изображения – создание «Земли» и «Игрока»**
   2.1. <u>Создание «Земли»</u>
      2.1.1. Создать в окне Project в папке Assets папку Sprite
      2.1.2. Через любой поисковик нахожу и скачиваю графические изображения (спрайты): для Игрока – player sprite (массив изображений), для Земли – platform png (без фона)

2.1.3. Разместить найденные изображения в папке Sprite (достаточно перетащить мышкой)

2.1.4. Устанавливаем платформу (Землю) на сцену – перетаскиваем мышкой в окно сцены или в окно Hierarchy. Выполняем следующие настройки в окне Inspector или прямо на сцене:

2.1.5. Переименовываем спрайт в «Земля»

2.1.6. Добиваемся корректного изображения на сцене (внизу окна)

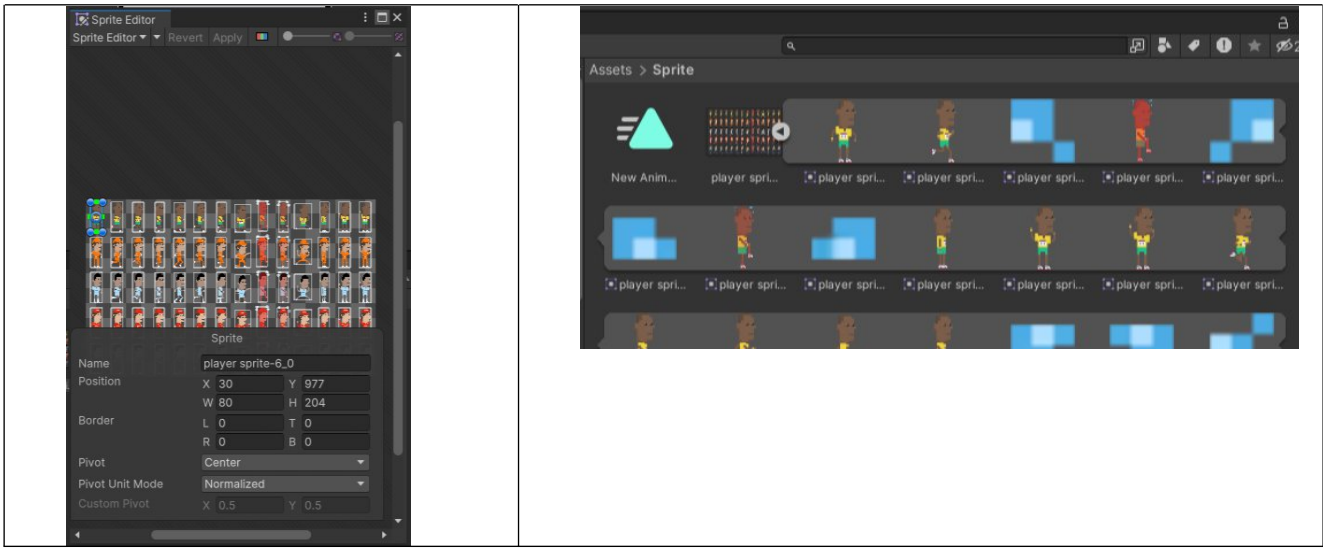
2.1.7. Настраиваем масштаб Scale (можно уменьшать/увеличивать X и У, изменять Z не имеет смысла, т.к. спрайт абсолютно плоский по данной размерности). Если Scale не помогает, то можно поработать с размером камеры – изменить Size (увеличить). Внимание: камера не может находиться в одной точке со спрайтом, т.к. ничего не будет видно.

2.1.8. Проверяем работу приложения.

2.2. Создание «Игрока»

2.2.1. Для спрайта «Игрок» (массив изображений) в окне Inspector устанавливаем Sprite Mode в значение Multiple

2.2.2. В окне Inspector нажимаем Sprite Editor (может выдаться ошибка, может помочь перезапуск проекта), в появившемся окне нажимаем кнопку со стрелочкой вниз, выбираем автоматический режим разбивки массива на отдельные изображения и нажимаем Apply (или Slice?)



2.2.3. В окне Project проверяем, что спрайт Игрока разбит на фрагменты, выделяем нужную последовательность изображений мышкой (не всегда нужны все изображения), на запрос о папке хранения изображений игрока вводим любой название (например, Player-animy)

2.2.4. В итоге Игрок должен появится на сцене и в окне Hierarchy. С помощью мыши и/или свойств в окне Inspector можно изменить размер Игрока, отзеркалить его (свойство Flip), изменить название и т.п.

2.2.5. В окне Inspector для Игрока появились компоненты: SpriteRenderer (нужен для корректного отображения изображений в виде спрайтов в 2D- или 3D-сцене) и Animator (нужен для управления анимацией двухмерных спрайтов). Убедимся, что оба компоненте содержат настройку на спрайт-игрока

2.2.6. В окне Project выделяем папку с изображениями игрока Player-animy, в окне Inspector включаем галочку LoopTime (зацикливание движения объекта)

2.2.7. Проверить непрерывность движения Игрока

2.3. Добавление физических свойств двухмерным изображениям

2.3.1. Для твердости «Земли» поверхности надо добавить компонент BoxCollider2D (создает невидимый контур с физическими свойствами тела, чаще всего используется для обработки физических столкновений объектов) – нужен, когда на «Землю» что-то «падает» - прыгает Игрок, падает камень и т.п. В окне Inspector для «Земли» добавляем компонент – Add Component, выбираем из большого списка компонентов BoxCollider2D. В компоненте BoxCollider2D нажимаем EditCollider для уточнения размеров контура с помощью мыши.

2.3.2. Для «Игрока» создаем точно такой же компонент BoxCollider2D. Внимание! Единожды настроенный компонент BoxCollider2D у одного объекта (спрайта) можно скопировать в другой объект (спрайт), если «физика поведения» идентичная.

2.3.3. Для других физических свойств Игрока (например, воздействие гравитации или столкновения) добавляем компонент RigidBody2D (отвечает за физическое поведение объекта в двухмерной среде).

2.3.4. Для воздействия силы гравитации на Игрока (притяжение к «Земле») в окне Inspector в компоненте RigidBody2D включаем галочку Simulated (нужна для обеспечения движения объекта с применением гравитационных и физических сил) и корректируем массу объекта Mass (при необходимости). Поднимаем Игрока над поверхностью, проверяем движение вниз.

2.3.5. Для подпрыгивания Игрока на «Земле» во время падения помимо компонента RigidBody2D на Игрока нужно наложить физический материал. Для этого:

2.3.6. Создаем в окне Project в папке Assets папку Materials. В этой папке правой кнопкой мыши создаем физический материал – Create-2D-PhisicalMaterial2D (позволяет настраивать трение и упругость, возникающие между двухмерными физическими объектами при их столкновении)

2.3.7. Мышью перетаскиваем на Игрока в окне Hierarchy либо на BoxCollider2D Игрока в окне Inspector

2.3.8. В окне Inspector для физического материала (выбираем в окне Project) настраиваем свойства: Friction – к-т трения (например, 0.4) и Bounciness – степень отскакивания (например, 0.5)

2.3.9. Проверяем физику движения Игрока.

3. **Добавление 3D-объекта на сцену – создание «Мандарина»** (можно добавить спрайт Мандарин и настроит его также как Игрока)

3.1. В окне Hierarchy правой кнопкой мыши добавляем трехмерную сферу: 3D Object – Sphere (у сферы уже есть встроенный коллайдер)

3.2. Все 3D-объекты на сцене черные, т.к. нет освещения. Добавляем источник света в окне Hierarchy правой кнопкой мыши: Light-DirectionLight. Желательно поставить два источника света – один направлен вверх, другой вниз (поворот направления света осуществлять мышью). Для источника, который смотрит вверх немного уменьшить интенсивность света Intensity (принимает значение от 0 до 1). Цвет источников света лучше оставить белым.

3.3. Для добавления текстуры (внешнего вида) скачаем файл текстуры формата jpg в ранее созданную папку Materials в окне Project

3.4. В окне Inspector для скачанной текстуры настроим свойство Texture Type – Default.

3.5. Перемещаем текстуру на сферу в окне Hierarchy или в окне Inspector в раздел Materials.

3.6. Для настройки физических свойств падающего мандарина надо добавить коллайдер и физику:

3.7. В окне Inspector правой кнопкой мыши удаляем Sphere Collider (иначе будет конфликтовать) и добавляем CircleCollider2D, проверяем, что свойство isTrigger

отключено (это свойство задает возможность пролетать одному объекту сквозь другой)

3.8. В окне Inspector правой кнопкой мыши добавляем компонент RigidBody2D и в нем добавляем физический материала с настройками (можно добавить ранее созданный материал для Игрока) с настройкой свойств Friction и Bounciness (если ничего не изменить, то физика Игрока и Мандарина будут идентичными).

3.9. Проверяем физику Игрока и Мандарина.

4. **Добавление движения Игроку** – игрок должен двигаться от одного края сцены до другого, потом разворачиваться и продолжать движение к другому краю сцены

4.1. Разработаем скрипт для движения Игрока: в окне Project в папке Asset создадим папку Script для аккумуляции всех скриптов приложения Create – Folder; в папке Script создадим пустой скрипт под название runIgrok (Create-C# Script); в окне Inspector появляется пустой скрипт. Перетаскиваем скрипт runIgrok на объект Игрока в окне Hierarchy или в окне Inspector.

4.2. Двойным щелчком открываем скрипт в VisualStudio, объявляем закрытые переменные для Игрока и скорости движения, получаем ссылку на Игрока (его физический компонент, отвечающий за движение), прописываем код изменения позиции игрока с заданной скоростью:

```
public class runIgrok : MonoBehaviour
{

    private Rigidbody2D _rb;
    private byte speed = 4;

    private void Awake()
    {
        _rb=GetComponent<Rigidbody2D>();
    }

    private void FixedUpdate()
    {
        _rb.position = _rb.position + Vector2.right * speed * Time.deltaTime;
    }
}
```

4.3. Сохранить код и проверить движение Игрока. Убедиться, что Игрок уходит за сцену.

4.4. Для обозначения левого и правого края сцены добавим на сцену «прозрачные стенки» в виде пустого объекта: в окне Hierarchy правой кнопкой мыши добавить два объекта Create Empty, разместить их по краям сцены и именовать WallRigth и WallLeft. Для начала нужны их координаты X, задающие края сцены.

4.5. Добавим в скрипт runIgrok код, позволяющий контролировать выход Игрока за пределы сцены:

```
public class runIgrok : MonoBehaviour
{

    private Rigidbody2D _rb;
    private byte speed = 4;
    private bool isRigth = true;


    private void Awake()
    {
        _rb=GetComponent<Rigidbody2D>();
    }

    private void FixedUpdate()
    {
        if (isRigth) _rb.position = _rb.position + Vector2.right * speed * Time.deltaTime;
        else _rb.position = _rb.position + Vector2.left * speed * Time.deltaTime;
        if (_rb.position.x>5.31) isRigth = false;
        if (_rb.position.x < -5.27) isRigth = true;
    }
}
```

4.6. Проверить движение Игрока. При необходимости можно включить свойство «прозрачности» Мандарина – isTrigger (чтобы он не останавливался на Земле, а пролетал сквозь нее) для отладки кода. Потом можно будет отключить.

4.7. Внести в код изменения для поворота Игрока при движении в противоположную сторону. Для этого добавить закрытую переменную _sr типа SpriteRenderer (массив выделенных нами ранее изображений), получить ссылку на Игрока (по аналогии с физическим компонентом) и обратиться к логическому свойству данного компонента – flipX.

4.8. Чтобы стенки работали более универсально (т.е. не зависимо от конкретного местоположения), можно добавить к ним коллайдер и прописать код для столкновения Игрока и Стены. Для этого: добавляем к объекту WallRigth компонент BoxCollider2D, редактируем его размеры с помощью EditCollider (имитируем стенку) и для отладки включаем свойство прозрачности – isTrigger, задаем объекту WallRigth тег WRigth, в скрипте runIgrok создаем обработчик события OnTriggerEnter2D:

```
private void OnTriggerEnter2D(Collider2D other)
{
    if(other.CompareTag("WRigth")) isRigth = false;

}
```

4.9. Закомментировать первый if в функции FixedUpdate и проверить движение Игрока. Добавить разворот Игрока.

4.10. Аналогично отработать с левой стеной.

4.11. Для управления Игроком мышью: один щелчок – останавливает, второй – запускает движение необходимо в скрипт runIgrok добавить реакцию на событие OnMouseDown(), в котором isClicked – это закрытая логическая переменная для отслеживания первого и второго щелчка (стартовое значение – false), CountClicked – закрытая целочисленная переменная (стартовое значение - 1), отражающая количество щелчков мыши, сделанных пользователем. Также откорректировать код функции FixedUpdate():

```
private void OnMouseDown()
{
    if (CountClicked % 2 !=0) isClicked = true; else isClicked = false;
    CountClicked++;
}
private void FixedUpdate()
{
    if (!isClicked) if (isRigth) _rb.position = _rb.position + Vector2.right * speed * Time.deltaTime;
    else _rb.position = _rb.position + Vector2.left * speed * Time.deltaTime;

}
```

4.12. Проверить работу мыши.

5. **Добавление движения Мандарину** – Мандарин должен пропадать при столкновении с Игроком и засчитываться как пойманный или при столкновении Землей, а зетам появляться сверху в новой позиции

5.1. Подсчет количества пойманных Мандаринов (в ходе столкновения с ними) осуществить по аналогии со «столкновением» со стеной: объявить открытую целочисленную переменную count (чтобы можно было контролировать в окне Inspector); объекту Мандарин присвоить тег Mandarin и в обработчик события OnTriggerEnter2D вписать проверку тега и инкремент count; (*также можно попробовать удалить объект Мандарин после того как Игрок его поймал с помощью метода Destroy(other.gameObject), но потом придется его создавать J)*.

5.2. Проверить работу счетчика (при необходимости изменить начальное положение Игрока и включить свойство прозрачности Мандарина).

5.3. Добавляем тег Player для Игрока и Finish для Земли.
5.4. Создаем скрипт scrMandarin и добавляем его к Мандарину
5.5. В скрипте создаем закрытую переменную для компонентаRigidBode2D объекта Мандарин и в методе Awake поучаем ссылку на объект.
5.6. Добавляем в скрипт событие OnTriggerEnter2D и добавляем код по перемещению Мандарина в случайные координаты верхней части сцены.

```
public class scrMandarin : MonoBehaviour
{
    private Rigidbody2D _rb;
    private Vector2 _newPoint;
    void Awake ()
    {
        _rb = GetComponent<Rigidbody2D>();
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player") || other.CompareTag("Finish"))
        {
            Debug.Log("Я сработал от " + other.name.ToString());
            _newPoint = new Vector2(Random.Range(-5,5),5);
            _rb.MovePosition(_newPoint);
        }
    }
```

5.7. Проверим работу сцены (*почему-то ускоряется падение Мандарин – может быть надо все-таки удалять Мандарин и создавать объект/префаб вновь?*).

6. Добавление звука на сцену (а точнее – игроку)

6.1. Скачать мелодию в формате mp3 и разместить в отдельной папке в Assets (заранее создать папку, например, Audio)
6.2. В окне Inspector добавить объекту Игрок компонент AudioSource (кнопка AddComponent). Перетащить скачанную мелодию в данный компонент (свойство AudioClip).
6.3. Выполнить настройку следующих свойств: включить галочку PlayOnAwake (для запуска звука при старте сцены), включить галочку Loop (для проигрывания звука по кругу), настроить громкость в свойстве Volume.
6.4. Проверить работу звука.

Самостоятельная работа:

1. Добавить на сцену спрайт «Камень» (скачать изображение, добавить на сцену, настроить местоположение и размеры, добавить коллайдер).
2. Написать код, с помощь которого можно убирать «камень» с дороги.
3. Написать код реакции на столкновение Игрока с камнем (остановка движения).

Критерии оценивания практической работы:

- оценка «отлично» выставляется студенту, если он полностью выполнил задания, включая самостоятельную работу;
- оценка «хорошо» выставляется студенту, если он полностью выполнил задания, но не выполнил самостоятельную работу;
- оценка «удовлетворительно» выставляется студенту, если он выполнил задания частично и не выполнил самостоятельную работу;
- оценка «неудовлетворительно» выставляется студенту, если он не представил выполненную работу.

Практическая работа по теме «Разработка 3D-приложений»

1. Создание проекта и сцены

1.1 Создаем 3D проект

1.2 Справа снизу в панели Project в папке Assets -> Scenes, изменяем название сцены на _0Scene (после выбора SampleScene, можно нажать F2)

2. Импорт ассетов

2.1 Заходим на сайт под своим аккаунтом https://assetstore.unity.com/packages/3d/characters/creatures/dragon-for-boss-monster-hp-79398 и добавляем ассеты

2.2 На верхней панели выбираем Windows -> Package Manager. В фильтрах выбираем My Assets и скачиваем появившийся ассет с драконами

2.3 После скачивания импортируем

3. Добавление дракона

3.1 На панели Project в папке Assets -> FourEvilDragonsHP -> Prefab -> DragonTerrorBringer найти зеленого дракона и создать его дубликат (нажать на него мышью -> нажать Ctrl+D)

3.2 Перетащить созданную копию Green 1 в папку Scenes

3.3 Переименовать копию в Enemy

3.4 Добавить дракона на сцену (перетащить мышью)

3.5 Во вкладке Inspector задать позицию
-Rotation 0,0,0
-Position 0,0,0
-Scale 1,1,1

3.6 В Project, в открытой папке Scenes нажать ПКМ и создать Create -> AnimatorController, задать ему имя EnemyCTRL

3.7 Из папки Assets -> FourEvilDragonsHP -> Animations -> DragonTerrorBringer найти анимацию FlyIdle и создать его копию (кликнуть по нему и нажать Ctrl+D)

3.8 Перетащить копию в папку Scenes

3.9 Открыть EnemyCTRL и перетащить туда анимацию дракона

3.10 Выбрать Enemy и во вкладке Inspector -> Animator напротив поля Controller выбрать EnemyCTRL

4. Создание драконьего яйца

4.1 На верхней панели во вкладке GameObject -> 3D Object создать Sphere, назвать ее DragonEgg и задать ей параметры
-Rotation 0, 0, 0
-Position 0, 0, 0
-Scale 1, 1.5, 1

4.2 В папке Scenes нажать ПКМ и в меню создать Material. Назвать Mat_Egg

4.3 В окне Inspector задать материалу произвольные свойства

4.4 Назначить материал объекту яйца, перетащив его мышью на DragonEgg

4.5 Выбрать DragonEgg и в окне Inspector нажать add component. В поиске выбрать Rigidbody

4.6 В окне инспектора напротив Tag в выпадающем списке нажать Add tag

4.7 В открывшемся окне нажать на «+» и задать имя тега DragonEgg, нажать Save

4.8 Вернуться на объект яйца и задать ему тег DragonEgg

4.9 Из окна Hierarchy перетащить DragonEgg на панель Project в папку Assets -> Scenes, после этого удалить его в Hierarchy

5. Создание объекта – энергетического щита

5.1 Создать 3д объект – сферу, назвать его EnergyShield

5.2 Задать ему параметры

– Position: 0, -6, 0;

– Rotation: 0, 0, 0;

– Scale: 3, 3, 3;

5.3 Создать материал Mat_Shield

5.4 В свойствах установить Rendering mode – Transparent

5.5 Перетащить материал на объект EnergyShield

5.6 Добавить на EnergyShield компонент RigidBody. Внутри компонента снять флажок Use Gravity и установить Is Kinematic

5.7 Создать префаб объекта (перетащить из Heirarchy на панель Project в папку сцены)

5.8 Сохранить сцену (ПКМ по _0Scene -> Save Scene)

6. Настройка камеры

6.1 Установить параметры камеры

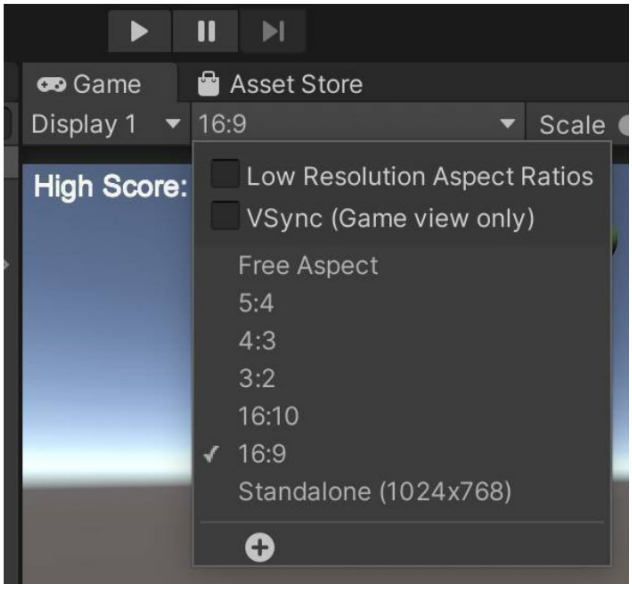
– Position: 0, 5, 10;

– Rotation: 20, – 180, 0;

– Scale: 1, 1, 0;

6.2 В инспекторе для компонента Camera установить в поле Projection – Orthographic, в поле Size значение 14

6.3 На вкладке Game сверху выбрать соотношение сторон 16:9



7. Скрипт файл EnemyDragon

7.1 на вкладке Project создать скрипт C# и назвать его EnemyDragon

7.2 Открываем скрипт и вводим следующий код:

EnemyDragon.cs

No selection

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

public class EnemyDragon : MonoBehaviour
{
    [Header("Set in Inspector")]
    public GameObject dragonEggPrefab;
    public float speed = 1f;
    public float timeBetweenEggDrops = 1f;
    public float leftRightDistance = 10f;
    public float chanceDirections = 0.1f;

    void Start()
    {
    }

    void Update()
    {
    }
}
```

В листинге мы добавили следующие строки кода:
– с помощью команды [Header("Set in Inspector")] создализаголовок в окне Inspector;
– public GameObject dragonEggPrefab создает префаб для драконьего яйца DragonEgg;
– созданная переменная speed определяет скорость движения объекта DragonEgg;
– переменная timeBetweenEggDrops определяет скорость генерации объектов DragonEgg;
– переменная leftRightDistance определяет расстояние от края экрана, на котором меняется направление движения дракона;
– chanceDirections определяет вероятность изменения направления движения.

7.3 Подключаем скрипт к Enemy, перетаскиванием его на объект Enemy. Теперь можно менять параметры дракона прямо в Inspector

7.4 Внутри Update пишем следующий код:

EnemyDragon.cs

No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class EnemyDragon : MonoBehaviour
6   {
7       [Header("Set in Inspector")]
8       public GameObject dragonEggPrefab;
9       public float speed = 1f;
10      public float timeBetweenEggDrops = 1f;
11      public float leftRightDistance = 10f;
12      public float chanceDirections = 0.1f;
13
14      void Start()
15      {
16      }
17
18      void Update()
19      {
20          Vector3 pos = transform.position; //1
21          pos.x += speed * Time.deltaTime; //2
22          transform.position = pos; //3
23      }
24  }
25
```

Каждому закомментированному номеру соответствует следующее:
– Vector3 pos – это локальная переменная для хранения текущий позиции (//1);
– x увеличивается на произведение скорости и временного интервала Time.deltaTime (//2). Это встроенная переменная, которая будет изменять свое значение при разной частоте
кадров. Если говорить простыми словами, то она отвечает за плавность игры при разном FPS;
– изменение значения pos записывается обратно в transform.position (//3).

7.5 Чтобы дракон не улетал за край экрана добавляем в Update следующий код:

EnemyDragon.cs

EnemyDragon ▸ No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class EnemyDragon : MonoBehaviour
6   {
7       [Header("Set in Inspector")]
8       public GameObject dragonEggPrefab;
9       public float speed = 1f;
10      public float timeBetweenEggDrops = 1f;
11      public float leftRightDistance = 10f;
12      public float chanceDirections = 0.1f;
13
14      void Start()
15      {
16      }
17
18      void Update()
19      {
20          Vector3 pos = transform.position;
21          pos.x += speed * Time.deltaTime;
22          transform.position = pos;
23
24          if (pos.x < -leftRightDistance) //1
25          {
26              speed = Mathf.Abs(speed);
27          }
28          else if (pos.x > leftRightDistance) //2
29          {
30              speed = -Mathf.Abs(speed);
31          }
32      }
33  }
```

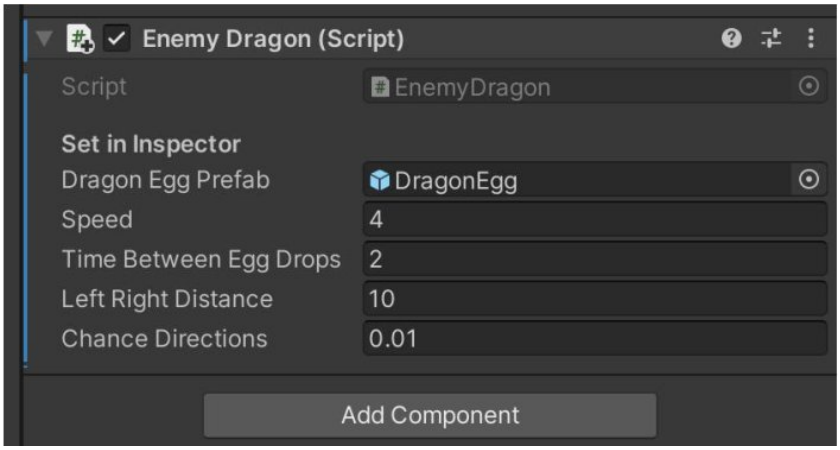
7.6 Добавляем метод для случайного изменения направления полета:

EnemyDragon.cs

EnemyDragon ▸ No selection

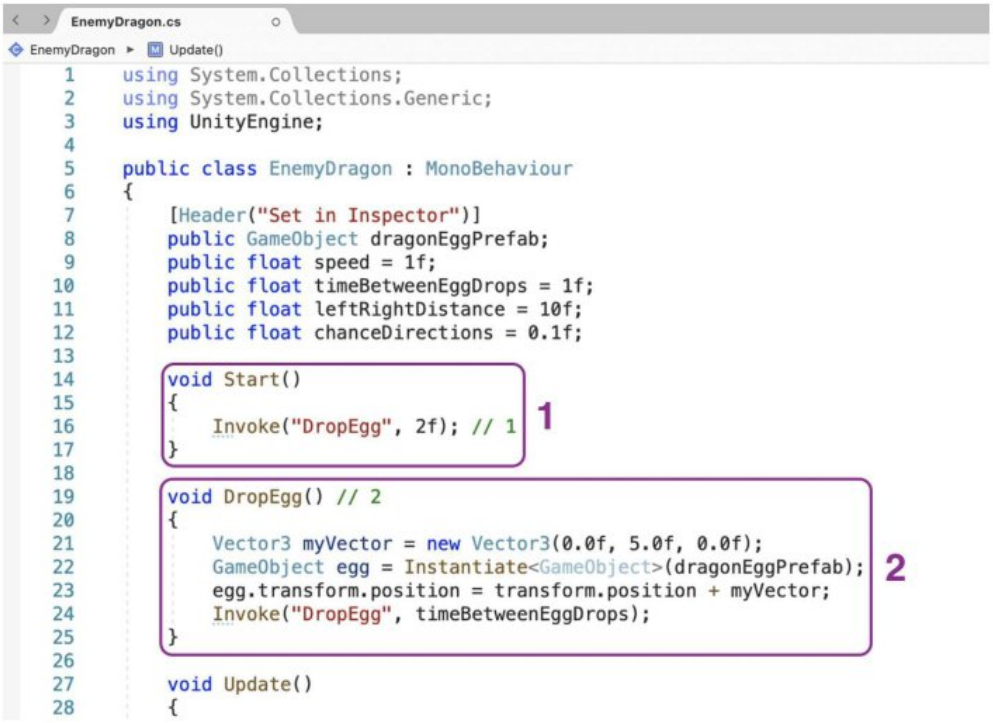
```
17
18      void Update()
19      {
20          Vector3 pos = transform.position;
21          pos.x += speed * Time.deltaTime;
22          transform.position = pos;
23
24          if (pos.x < -leftRightDistance)
25          {
26              speed = Mathf.Abs(speed);
27          }
28          else if (pos.x > leftRightDistance)
29          {
30              speed = -Mathf.Abs(speed);
31          }
32      }
33
34      private void FixedUpdate()
35      {
36          if (Random.value < chanceDirections)
37          {
38              speed *= -1;
39          }
40      }
41  }
```

7.7 Чтобы дракон менял напрвление в среднем 1 раз в секунду выствляем у него такие параметры:



7.8 Перетаскиваем DragonEgg в строку Dragon Egg Prefab для дальнейшей реализации сброса яиц.

7.9 После подключения объекта, нужно создать метод DropEgg и команду вызова этого метода в Start



– функция Invoke в методе Start() выше вызывает функцию, заданную именем через указанное число секунд. В данном случае – вызывается функция DropEgg(), с параметром 2f, т.е. с ожиданием 2 секунды перед каждым новым вызовом. Таким образом, мы можем установить наиболее подходящие значения для частоты генерации объектов. Можете самостоятельно подобрать наиболее подходящее значение.
– функция DropEgg() создает экземпляр GameObject с именем dragonEggPrefab и присваивает его переменной egg. Далее изменяется положение egg, и функция вызывает саму себя снова через интервалы времени, равные timeBetweenEggDrops.

8. Скрипт DragonEgg

8.1 Создаем файл скрипта и называем его DragonEgg

8.2 Добавляем еще один пак ассетов https://assetstore.unity.com/packages/vfx/particles/fire-explosions/fire-spell-effects-36825 не забываем импортировать в unity

8.3 Добавим плоскость Plane, которая будет играть роль поверхности земли. Также мы оформим ее в виде раскаленной лавы, падая на которую драконьи яйца будут взрываться. Чтобы добавить плоскость на сцену, в верхнем меню выберите Windows – GameObject – Plane. Переименовываем ее в Ground.

8.4 Установим размеры объекта Ground:
– Position: 0, -8, 0;
– Rotation: 0, 0, 0;
– Scale: 5, 5, 5;

8.5 В окне Inspector развернем компонент Mesh Renderer (), нажмем на “мишень” и из появившегося меню Select Material выберем материал FireParticleMeteorMaterial

8.6 Можно поменять текстуру у яйца

8.7 Напишем код в DragonEgg:

EnemyDragon.cs | DragonEgg.cs

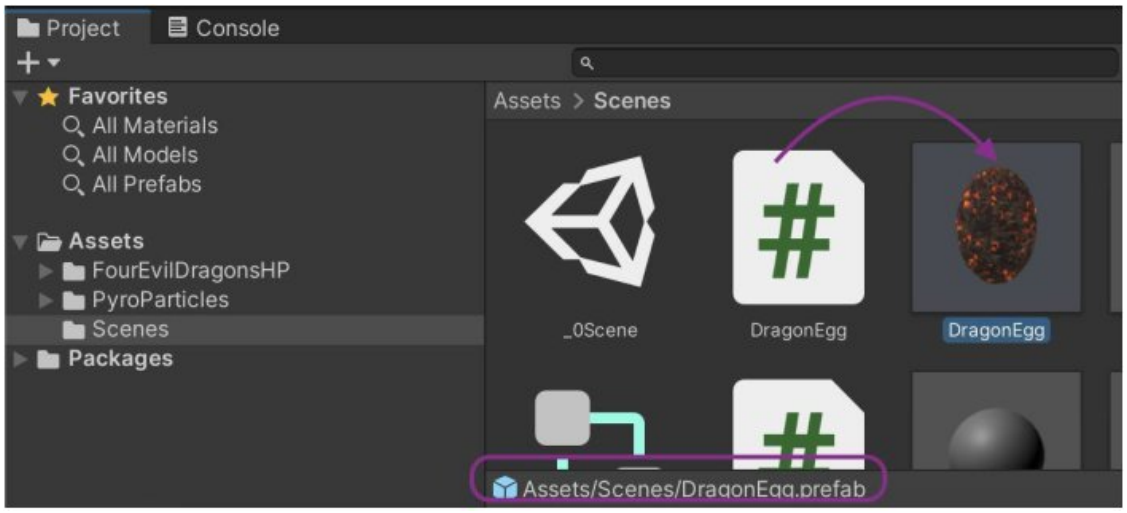
No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class DragonEgg : MonoBehaviour
6   {
7       public static float bottomY = -30f;
8
9       void Start()
10      {
11      }
12
13      private void OnTriggerEnter(Collider other)
14      {
15          ParticleSystem ps = GetComponent<ParticleSystem>();
16          var em = ps.emission;
17          em.enabled = true;
18
19          Renderer rend;
20          rend = GetComponent<Renderer>();
21          rend.enabled = false;
22      }
23
24      void Update()
25      {
26          if (transform.position.y < bottomY)
27          {
28              Destroy(this.gameObject);
29          }
30      }
31  }
```

– создается переменная bottomY, которая будет указывать, на каком расстоянии Y нужно будет удалять объекты DragonEgg.

– По названию метода OnTriggerEnter (Collision Other) можно понять, что он начинает работу, когда происходит пересечение с объектом, который работает как Trigger

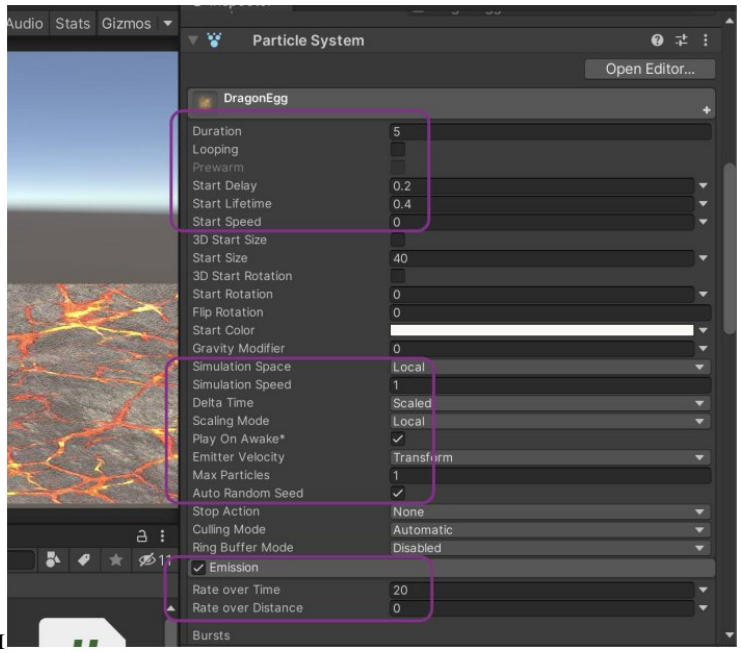
8.8 Подключим скрипт к яйцу, перетащив его



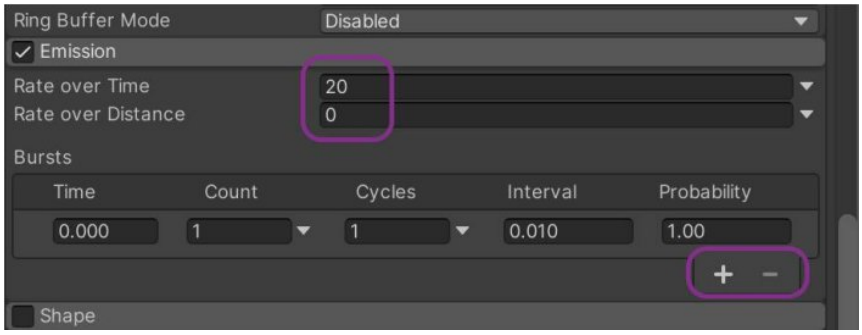
8.9 Теперь для DragonEgg добавим компонент ParticleSystem

8.10 В окне Hierarchy выберем Ground и в Inspector поставим две галочки Convex и Is Trigger

8.11 Чтобы партиклы не были стандартными зададим настройки и зададим материал

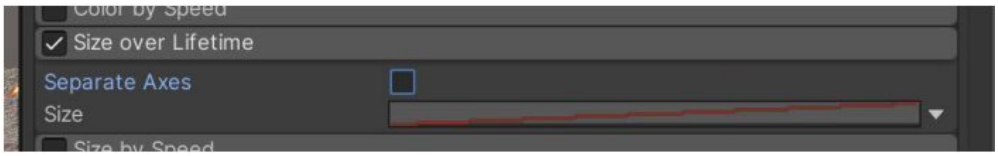


8.12 Поставьте галочку возле Emission и введите параметры, показанные на рисунке ниже (чтобы добавить новую линию с параметрами “всплесков” нажмите “+”)

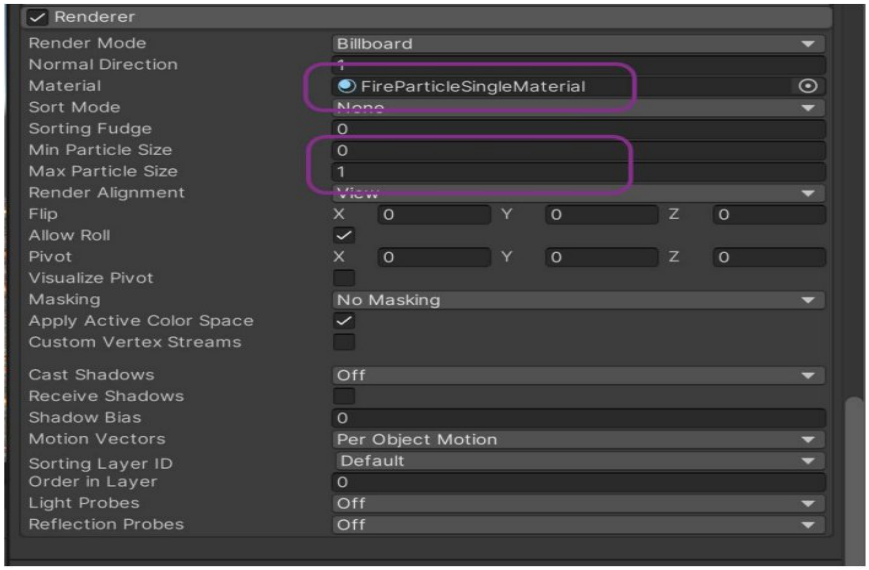


8.13 Теперь снимите галочку с Emission. Это делается для того, чтобы при запуске сцены этот параметр был выключен и объект не излучал Particles с эффектами взрыва

8.14 Поставьте галочку напротив “Size over Lifetime”, и выберите вид кривой, которая растет от минимума слева до максимума справа.



8.15 Выберем в качестве текстуры материал с названием FireParticleSingleMaterial, эта текстура также находится в скачанном ассет-паке в папке PyroParticles



9. Скрипт файл DragonPicker

9.1 Создадим файл скрипт DragonPicker

9.2 Подключаем его к MainCamera

9.3 Пишем в нем следующий код

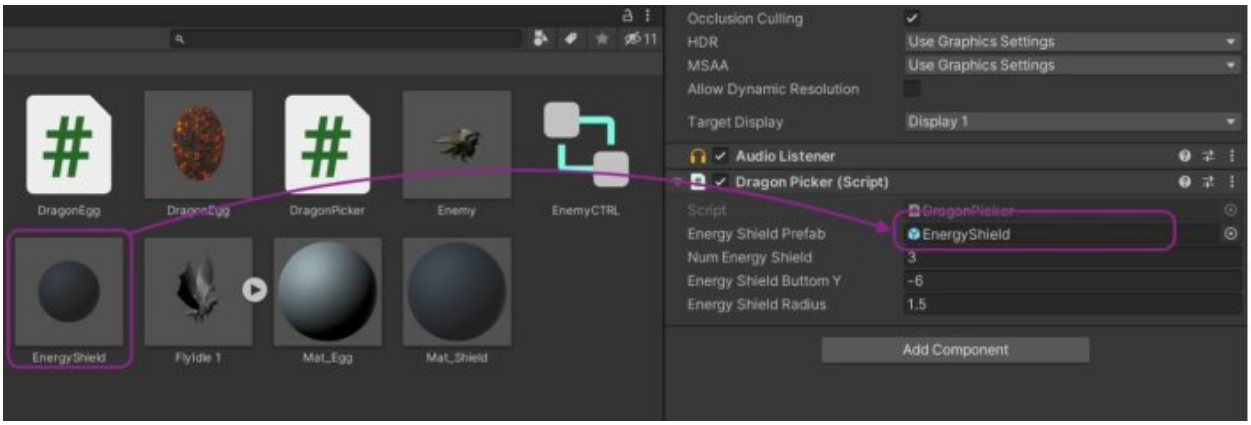
EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs

No selection

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

public class DragonPicker : MonoBehaviour
{
    public GameObject energyShieldPrefab;
    public int numEnergyShield = 3;
    public float energyShieldButtomY = -6f;
    public float energyShieldRadius = 1.5f;

    void Start()
    {
        for (int i = 1; i <= numEnergyShield; i++)
        {
            GameObject tBasketGo = Instantiate<GameObject>(energyShieldPrefab);
            tBasketGo.transform.position = new Vector3(0, energyShieldButtomY, 0);
            tBasketGo.transform.localScale = new Vector3(1 * i, 1 * i, 1 * i);
        }
    }

    void Update()
    {
    }
}
```

Код создает 3 экземпляра EnergyShield

9.4 Затем нужно подключить EnergyShield к сценарию



10. Скрипт файл EnergyShield

10.1 Создаем скрипт файл EnergyShield и прикрепляем его к шаблону Energy Shield Prefab

10.2 Чтобы щит перемещался за курсором мыши пишем следующий код

EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs | EnergyShield.cs

No selection

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

public class EnergyShield : MonoBehaviour
{

    void Update()
    {
        Vector3 mousePos2D = Input.mousePosition;
        mousePos2D.z = -Camera.main.transform.position.z;
        Vector3 mousePos3D = Camera.main.ScreenToWorldPoint(mousePos2D);
        Vector3 pos = this.transform.position;
        pos.x = mousePos3D.x;
        this.transform.position = pos;
    }
}

```

10.3 Далее реализуем ловлю яиц добавлением метода OnCollisionEnter

EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs | EnergyShield.cs

No selection

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

public class EnergyShield : MonoBehaviour
{
    void Update()
    {
        Vector3 mousePos2D = Input.mousePosition;
        mousePos2D.z = -Camera.main.transform.position.z;
        Vector3 mousePos3D = Camera.main.ScreenToWorldPoint(mousePos2D);
        Vector3 pos = this.transform.position;
        pos.x = mousePos3D.x;
        this.transform.position = pos;
    }

    private void OnCollisionEnter(Collision coll)
    {
        GameObject Collided = coll.gameObject;
        if (Collided.tag == "DragonEgg")
        {
            Destroy(Collided);
        }
    }
}
```

10.4 Сохраняем сцену и тестируем результат

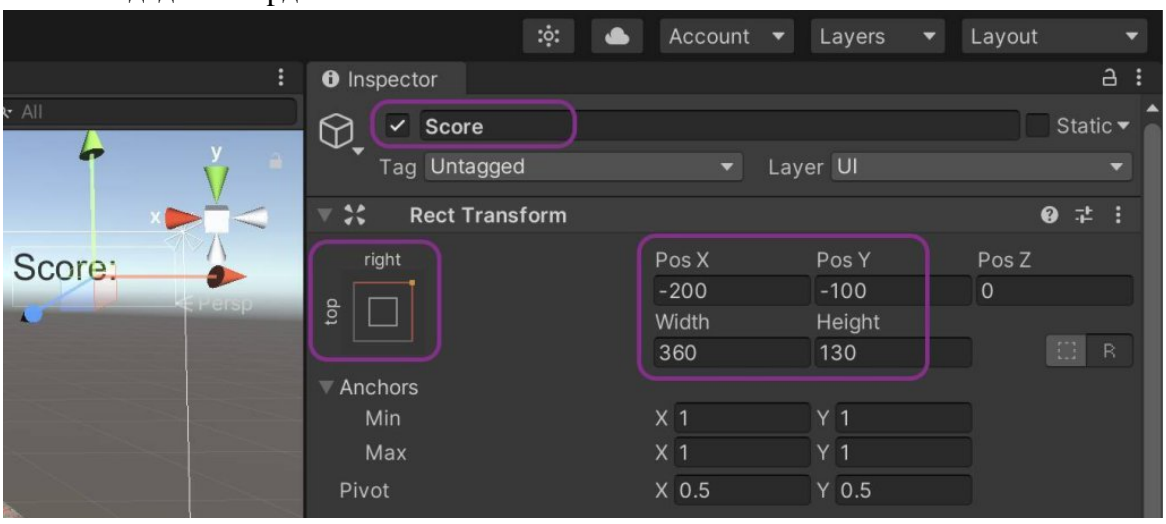
11. Создание счетчика очков

11.1 Дважды кликнем по _0Scene в панели Project, и в меню Unity выберем пункт GameObject – UI – Legacy – Text

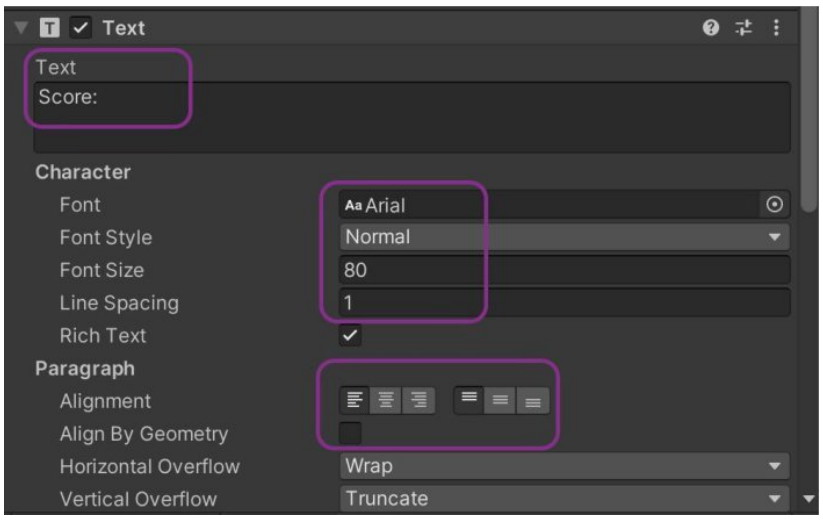
11.2 В окне Hierarchy выделим Canvas и в окне Inspector развернем компонент Canvas. Установим в поле Render Mode значение Screen Space – Camera, а в качестве Render Camera выберем Main Camera (Camera). В поле Plane Distance установим значение равное 10

11.3 Переименуем имя Text на Score

11.4 Зададим координаты очкам

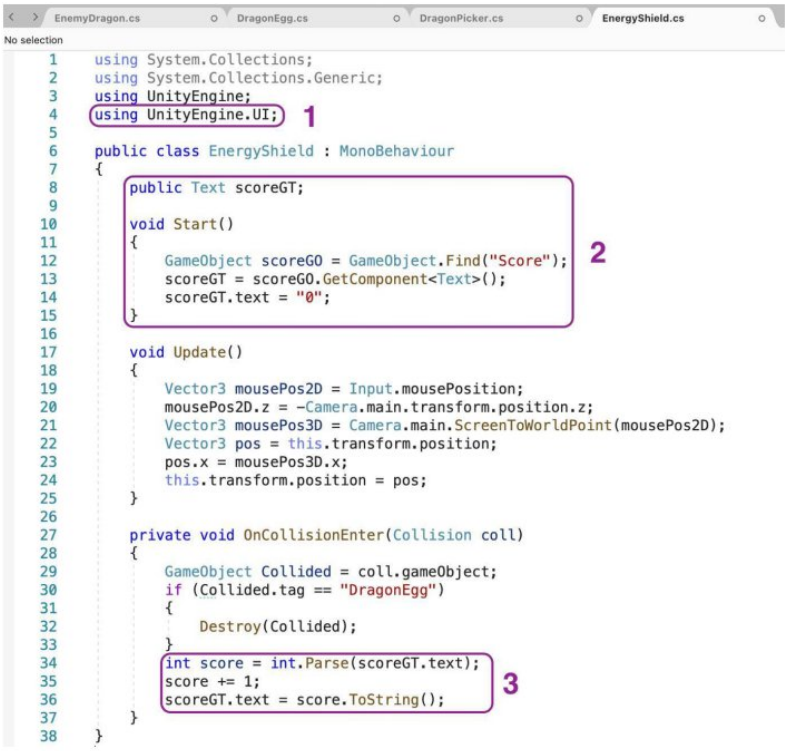


11.5 Чуть ниже зададим надпись на экране и размер шрифта



12. Добавление счетчика очков

12.1 Чтобы создать счетчик откроем скрипт EnergyShield и добавим несколько строк кода



– С помощью команды using UnityEngine.UI подключается библиотека, позволяющая использовать интерфейс пользователя .UI (User Interface), доступная внутри движка Unity.

– Далее объявляется переменная scoreGT типа public, которая будет хранить в себе счетчик пойманных объектов DragonEgg.

– GameObject scoreGO = GameObject.Find(“Score”); – находит на сцене игровой объект с именем Score, и присваивает ссылку на объект переменной scoreGO.

– scoreGT = scoreGO.GetComponent<Text>(); – находит компонент Text внутри игрового
объекта scoreGO и присваивает ссылку на него в public-поле scoreGT.

– scoreGT.text = "0"; – публичному полю scoreGT присваивается начальное значение равное нулю. Обратите внимание на то, что только благодаря строке в начале сценария using UnityEngine.UI; можно получать доступ к компоненту Text в коде на C#;

– int score = int.Parse(scoreGT.text); – получает текст, записанный в поле ScoreCounter и преобразует его в целое число типа int. Далее это число записывается в переменную с именем score;

– score += 1; – значение переменной увеличивается на 1 с каждым вызовом функции

OnCollisionEnter(Collision coll), т.е. с каждым пойманным яйцом;
– scoreGT.text = score.ToString(); – переменная score преобразуется обратно в строковую переменную и записывается в свойство text объекта scoreGT;

13. Уведомление, что яйцо не поймано

13.1 Начнем с изменений в DragonEgg

EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs | EnergyShield.cs

No selection

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

public class DragonEgg : MonoBehaviour
{
    public static float bottomY = -30f;

    void Start()
    {
    }

    private void OnTriggerEnter(Collider other)
    {
        ParticleSystem ps = GetComponent<ParticleSystem>();
        var em = ps.emission;
        em.enabled = true;

        Renderer rend;
        rend = GetComponent<Renderer>();
        rend.enabled = false;
    }

    void Update()
    {
        if (transform.position.y < bottomY)
        {
            Destroy(this.gameObject);
            DragonPicker apScript = Camera.main.GetComponent<DragonPicker>();
            apScript.DragonEggDestroyed();
        }
    }
}
```

– DragonPicker apScript = Camera.main.GetComponent< DragonPicker >(); – в локальную переменную apScript записывается ссылка на компонент DragonPicker (Script) главной камеры Main Camera.
– После этого появляется возможность напрямую обращаться к переменным и методам экземпляра класса DragonPicker.cs, подключенного к главной камере.
– apScript.DragonEggDestroyed(); – это вызов метода DragonEggDestroyed() класса DragonPicker.

13.2 Добавим метод DragonEggDestroy в DragonPicker

EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs | EnergyShield.cs

No selection

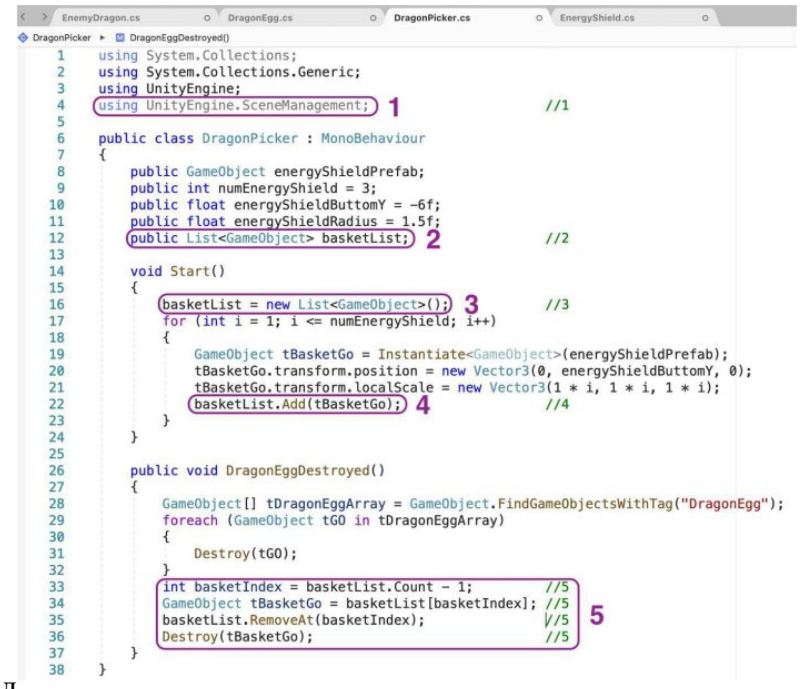
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DragonPicker : MonoBehaviour
{
    public GameObject energyShieldPrefab;
    public int numEnergyShield = 3;
    public float energyShieldButtomY = -6f;
    public float energyShieldRadius = 1.5f;

    void Start()
    {
        for (int i = 1; i <= numEnergyShield; i++)
        {
            GameObject tBasketGo = Instantiate<GameObject>(energyShieldPrefab);
            tBasketGo.transform.position = new Vector3(0, energyShieldButtomY, 0);
            tBasketGo.transform.localScale = new Vector3(1 * i, 1 * i, 1 * i);
        }
    }

    void Update()
    {
    }

    public void DragonEggDestroyed()
    {
        GameObject[] tDragonEggArray = GameObject.FindGameObjectsWithTag("DragonEgg");
        foreach (GameObject tGO in tDragonEggArray)
        {
            Destroy(tGO);
        }
    }
}
```

– метод DragonEggDestroy имеет тип public (то есть является общедоступным), чтобы его можно было вызывать из других классов, например, таких как DragonEgg.cs; – строки кода внутри созданного

метода возвращают массив всех существующих игровых объектов DragonEgg и с помощью цикла foreach выполняется обход всех найденных объектов с их последующим уничтожением.

14. Уничтожение EnergyShield при потере яйца

14.1 После потри яйца будем удалять один щит. Добавим в DragonPicker следующий код



– часть кода #1 подключает библиотеку SceneManagement для возможности работы со сценами.
– часть кода #2 и #3 определяет новый список типа List<GameObject>;
– часть кода #4 в конце цикла for добавляет каждый созданный EnergyShield в список basketList;
– часть кода #5 уничтожает один щит с каждым очередным срабатыванием метода DragonEggDestroyed();

14.2 Добавим условие, при котором после потери всех щитов, сцена будет перезапускаться

```
26      public void DragonEggDestroyed()
27      {
28          GameObject[] tDragonEggArray = GameObject.FindGameObjectsWithTag("DragonEgg");
29          foreach (GameObject tGO in tDragonEggArray)
30          {
31              Destroy(tGO);
32          }
33          int basketIndex = basketList.Count - 1;
34          GameObject tBasketGo = basketList[basketIndex];
35          basketList.RemoveAt(basketIndex);
36          Destroy(tBasketGo);
37          if (basketList.Count == 0)
38          {
39              SceneManager.LoadScene("_0Scene");
40          }
41      }
42  }
```

Самостоятельная работа:

1. Создание стартовую сцену с тремя кнопками: Игра, Опции, Выход
2. Разработать отл\клики на нажатие кнопок
3. Разработать переходы между стартовой и игровой сценой

Критерии оценивания практической работы:

- оценка «отлично» выставляется студенту, если он полностью выполнил задания, включая самостоятельную работу;

- оценка «хорошо» выставляется студенту, если он полностью выполнил задания, но не выполнил самостоятельную работу;
- оценка «удовлетворительно» выставляется студенту, если он выполнил задания частично и не выполнил самостоятельную работу;
- оценка «неудовлетворительно» выставляется студенту, если он не представил выполненную работу.

3. ОЦЕНОЧНЫЕ МАТЕРИАЛЫ (СРЕДСТВА) ДЛЯ ПРОВЕДЕНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ПО ДИСЦИПЛИНЕ

ОЦЕНКА ЗНАНИЙ ОБУЧАЮЩИХСЯ НА ЭКЗАМЕНЕ
ВОПРОСЫ ДЛЯ ПОДГОТОВКИ К ЭКЗАМЕНУ

ПК-3 Способен разрабатывать цифровой программный продукт

ПК-3.1 - Осуществляет разработку программного кода и компонентов цифрового программного продукта с помощью специализированных цифровых платформ для индивидуальной и совместной разработки

Обучающийся знает: теоретические основы разработки 2D-приложений и 3D-приложений; виды объектов и компонентов для создания сцен; порядок создания программных скриптов, их привязки к объектам

1. Состав платформы Unity.
2. Ассеты, объекты, компоненты, шаблоны.
3. Порядок разработки 2D-приложения. Особенности разработки 2D-приложения.
4. Порядок разработки 3D-приложения. Особенности разработки 3D-приложения.
5. Понятие сцены, создание сцены и управление ею. Используемые классы. Отличие 2D-сцен и 3D-сцен
6. Работа с освещением 2D-сцен и 3D-сцен
7. Понятие объекта, размещение объекта на сцене, настройка объекта
8. Понятие текстуры, разработка и накладывание текстур на объект. Взаимодействие с материалами.
9. Понятие и назначении шейдеров. Модификация существующих шейдеров.
10. Движение объектов по сцене, отличия 2D физики от 3D физики.
11. Создание анимации в Unity. Используемые классы.
12. Работа с анимацией в 2D, спрайты, 3D физика. Приемы и особенности работы с 2D.
13. Работа с анимацией в 3D, спрайты, 3D физика. Приемы и особенности работы с 3D.
14. Импорт и использование спрайтов.
15. Понятие скрипта, основы написания скриптов на языке C#, структура скрипта, присоединение скрипта к объекту
16. Отладка приложения. Окно Debug. Анализ ошибок в приложении.
17. Встроенный инструментарий для создания пользовательского интерфейса.
18. Канвас и три его режима, элементы UI, Layout, Event System.
19. Перенос координат из пространства Canvas (overlay) в мировое пространство, и наоборот.

ОЦЕНКА УМЕНИЙ И НАВЫКОВ ОБУЧАЮЩИХСЯ НА ЭКЗАМЕНЕ

ПК-3 Способен разрабатывать цифровой программный продукт

ПК-3.1 - Осуществляет разработку программного кода и компонентов цифрового программного продукта с помощью специализированных цифровых платформ для индивидуальной и совместной разработки

Обучающийся умеет: создавать 2D-сцены и 3D-сцены; разрабатывать скрипты для 2D-сцены и 3D-сцены**.**

Обучающийся владеет: навыками кодирования на языке C#; работы с объектами, компонентами и текстурами; отладки программного кода**.**

Оценка достижения: обучающимся запланированного результата обучения осуществляется по результатам выполнения творческого проектного задания и его защиты во время экзамена.

Творческое проектное задание

Творческое проектное задание .выполняемое на экзамене, представляет собой небольшой фрагмент игрового приолжения.

Примерный список тем для творческого проектного задания:

1. Игровое приложение «Пинпонг».
2. Игровое приложение «Прятки».
3. Игровое приложение «Лабиринт».
4. Игровое приложение «Созидание».
5. Игровое приложение «Гонки».
6. Игровое приложение «Спортивное состязание».
7. Деловая игра «Компас абитуриента».
8. Деловая игра «Тренажер знаний».
9. Деловая игра «Маршрутизатор».
10. Игровое приложение «Змейка».
11. Игровое приложение «Охота».
12. Игровое приложение «Башня».
13. Игровое приложение «Стройка».
14. Игровое приложение «Танки».
15. Игровое приложение «Комната».

Требования к творческому проектному заданию:

1. Определен игровой персонаж и его минимальный набор действий
2. Разработана сцена
3. Реализованы анимационные эффекты
4. Реализован пользовательский интерфейс

ОБРАЗЕЦ ЭКЗАМЕНАЦИОННОГО БИЛЕТА

Экзаменационный билет состоит из трех теоретических вопросов.

Тольяттинская академия управления	Дисциплина Разработка игровых приложений

Вариант 1

1. Современные инструменты для разработки игровых приложений. Жизненный цикл разработки игры.
2. Понятие персонажа и его создание. Используемые классы.
3. Демонстрация и защита творческого проектного задания.

4. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ, ОПРЕДЕЛЯЮЩИЕ ПРОЦЕДУРЫ ОЦЕНИВАНИЯ ЗНАНИЙ, УМЕНИЙ, НАВЫКОВ, ХАРАКТЕРИЗУЮЩИХ ЭТАПЫ ФОРМИРОВАНИЯ КОМПЕТЕНЦИЙ

КРИТЕРИИ ОЦЕНИВАНИЯ СФОРМИРОВАННОСТИ КОМПЕТЕНЦИЙ

Планируемые результаты обучения (показатели достижения заданного уровня освоения компетенций)	Критерии оценивания результатов обучения				
	1	2	3	4	5
ПК-3 Способен разрабатывать цифровой программный продукт ПК-3.1 - Осуществляет разработку программного кода и компонентов цифрового программного продукта с помощью специализированных цифровых платформ для индивидуальной и совместной разработки					
Знать: теоретические основы разработки 2D-приложений и 3D-приложений; виды объектов и компонентов для создания сцен; порядок создания программных скриптов, их привязки к объектам	Не знает	Фрагментарные знания функциональных возможностей платформы Unity 3D, принципов разработки игровых приложений	Общие, но не структурированные знания функциональных возможностей платформы Unity 3D, принципов разработки игровых приложений	Сформированные, но содержащие отдельные пробелы знания функциональных возможностей платформы Unity 3D, принципов разработки игровых приложений	Сформированные систематизированные прочные знания функциональных возможностей платформы Unity 3D, принципов разработки игровых приложений
Уметь: создавать 2D-сцены и 3D-сцены; разрабатывать скрипты для 2D-сцены и 3D-сцены	Не умеет	Частично освоенные умения разрабатывать прототипы 2D-приложений и 3D-приложений	В целом освоенные, но не подкрепляемые знаниями и самостоятельностью выполнения умения разрабатывать прототипы 2D- и 3D-приложения	В целом сформированные и самостоятельно выполняемые, но не всегда интеллектуально обоснованные умения разрабатывать 2D- и 3D-приложения	Успешно сформированные, самостоятельно и осознанно выполняемые умения разрабатывать 2D- и 3D-приложения
Владеть: навыками кодирования на языке C#; работы с объектами, компонентами и текстурами; отладки программного кода	Не владеет	Фрагментарно сформированные навыки кодирования на современных языках программирования	В целом сформированные, но выполняемые на элементарном уровне навыки кодирования на современных языках программирования	Сформированные, но не устойчивые в сложных ситуациях навыки кодирования на современных языках программирования	Успешно сформированные, устойчивые, технологически грамотно выполняемые навыки кодирования на современных языках программирования

ПРОЦЕДУРА ПРОВЕДЕНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ И КРИТЕРИИ ОЦЕНКИ РЕЗУЛЬТАТОВ ОБУЧЕНИЯ ПО ДИСЦИПЛИНЕ

Для контроля усвоения обучающимися данной дисциплины и достижения запланированных результатов обучения учебным планом предусмотрены экзамен. Экзамен проводится в форме устного ответа на теоретические вопросы. Результаты текущего контроля являются допуском к экзамену: обучающийся должен выполнить и подготовить к защите творческое проектное задание.

В ходе промежуточной аттестации перевод результатов работы обучающихся (результатов текущего контроля) в систему оценки знаний осуществляется с ориентацией на критерии оценивания сформированности компетенций следующим образом:

- оценка «отлично» выставляется обучающемуся, который показал сформированные систематизированные прочные знания о разработке 2D и 3D приложений в Unity; творческое задание выполнено на отметку «отлично».
- оценка «хорошо» выставляется обучающемуся, который показал сформированные, но содержащие отдельные пробелы знания о разработке 2D и 3D приложений в Unity; творческое задание выполнено на отметку «хорошо».
- оценка «удовлетворительно» выставляется обучающемуся, который показал общие, но не структурированные знания о разработке 2D и 3D приложений в Unity; творческое задание выполнено на отметку «удовлетворительно».
- оценка «неудовлетворительно» выставляется обучающемуся, который имеет фрагментарные знания о разработке 2D и 3D приложений в Unity; творческое задание не выполнено или выполнено на отметку «неудовлетворительно».

-

5. ЛИСТ СОГЛАСОВАНИЯ

Составил:

Н.Б. Стрекалова, д.п.н., доцент

(подпись)

Заведующий кафедрой

Н.Б. Стрекалова, д.п.н., доцент

(подпись)

Заведующий выпускающей кафедрой

Н.Б. Стрекалова, д.п.н., доцент

(подпись)