Коновалов В.В.
студент 3 курса факультета естествознания, математики и информатики
Бужинская Н.В.
доцент кафедры ИТ, к.п.н.
Нижнетагильский государственный социально-педагогический институт, (филиал) ФГБОУ ВО «Российский государственный профессионально-педагогический университет»
Россия, Нижний Тагил

ПРИМЕНЕНИЕ UNITY ДЛЯ РАЗРАБОТКИ ПРОГРАММ ДОПОЛНЕННОЙ РЕАЛЬНОСТИ

Аннотация

В данной статье повествуется о процессе разработки мобильного приложения, использующего технологии дополненной реальности с функцией распознавания меток. Когда пользователь направляет камеру устройства на метку, в его окружении появляются анимированные трехмерные модели. Эти модели обладают заранее определёнными параметрами урона и здоровья. В случае их взаимного приближения, модели активируют анимацию боевого взаимодействия, а при изменении угла взаимного расположения они ориентируются друг к другу, создавая реалистичное впечатление присутствия.

Ключевые слова: дополненная реальность, мобильное приложение, Unity, распознавание меток, анимация 3D моделей, пользовательский интерфейс.

Using Unity to develop augmented reality programs

Abstract

This abstract discusses the process of developing a mobile application that utilizes augmented reality technologies with a marker recognition feature. When the user points their device's camera at a marker, animated three-dimensional models appear in their environment. These models have predefined parameters of damage and health. In the event of their mutual approach, the models activate combat interaction animation, and when the angle of mutual positioning changes, they orient themselves towards each other, creating a realistic impression of presence.

Keywords: augmented reality, mobile application, Unity, marker recognition, 3D model animation, UI.

В современном информационном обществе дополненная реальность (AR) стала одной из наиболее актуальных и перспективных технологий, позволяющей объединить реальный и виртуальный миры для создания уникальных пользовательских впечатлений [2].

Дополненная реальность (AR) — это технология, которая объединяет реальный мир с виртуальными объектами и информацией, создавая гибридное окружение, в котором пользователь может взаимодействовать с виртуальными элементами в реальном времени [1]. В отличие от виртуальной реальности, которая полностью погружает пользователя в виртуальное окружение, AR расширяет реальный мир, добавляя к нему дополнительные визуальные и аудиоэффекты.

Для начала работы требуется сделать подготовительные действия. Сперва необходимо установить Unity [5]. Во время установки, если у вас отсутствует Visual Studio, всплывет окно об её установке — устанавливаем её тоже. Не забываем при установке выбрать опцию «Game development with Unity» или скачать пакет уже после установки.

Далее, необходимо посетить официальный сайт библиотеки и зарегистрироваться [6]. После регистрации необходимо получить базовый лицензионный ключ, а также добавить заранее подготовленные метки для распознавания.

Также для реализации приложения понадобится получить 3D модели и их анимации. Воспользуемся сайтом Mixamo, на нём можно выбирать и анимировать 3D модели, и даже добавлять свою [4]. Выбираем подходящие модели и анимируем их с помощью удобных инструментов. Скачиваем в формате «FBX For Unity (.Fbx)» (рис. 1).

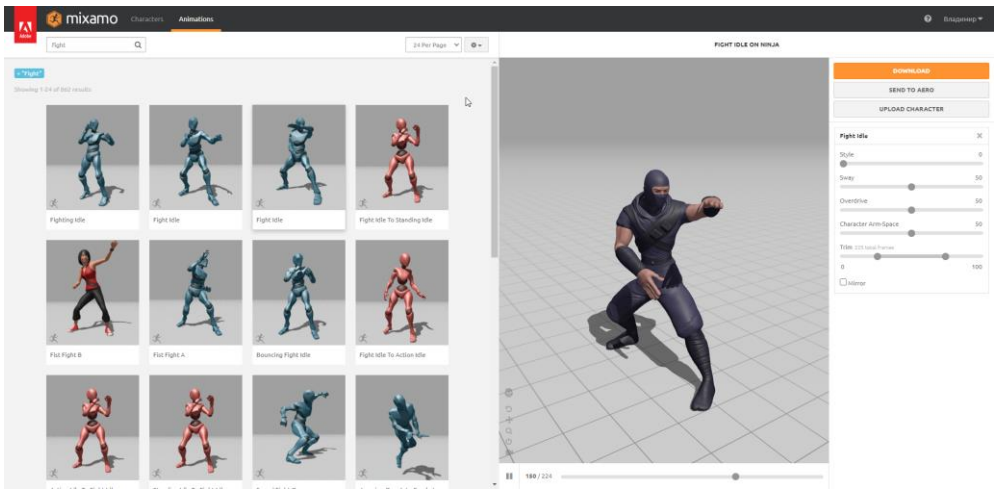

Рис. 1. Сайт Mixamo

Перейдем к непосредственной разработке приложения. Создаем новый проект, выбираем 3D приложение. Переходим в настройки сборки, меняем платформу на «Android», а также добавляем текущую сцену в сборку.

Далее, переходим менеджер пакетов. Чтобы добавить «Vuforia» в свои ассеты, переходим на официальную страницу библиотеки на сайте Unity и нажмите кнопку «Add to My Assets» [7]. Производим импорт библиотеки в проект.

Переходим в конфигурации библиотеки, вводим ранее полученный лицензионный ключ, затем в корневую папку проекта необходимо импортировать предварительно скачанную базу данных меток.

Теперь можно приступить к работе в Unity. Теперь следует удалить камеру со сцены, так как мы будем использовать камеру дополненной реальности.

Добавляем AR камеру в проект. Для этого надо в окне, в котором перечислены объекты сцены, кликнуть правой кнопкой мыши и выбрать «Vuforia Engine», далее «AR Camera» (рис. 2).

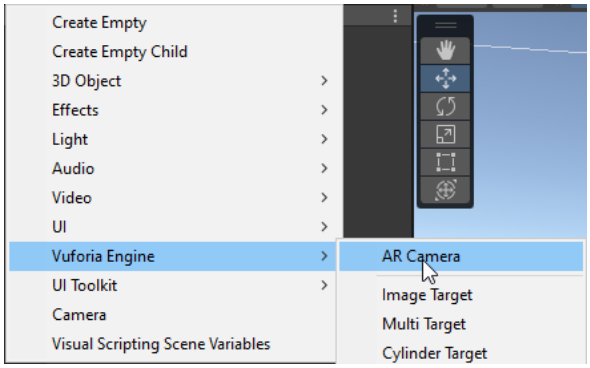

Рис. 2. AR камера

Далее, в объекте «ARCamera» создаем объект «ImageTarget», который будет использоваться в качестве метки. В свойствах меняем тип на «From Database», далее выбираем нашу базу данных и желаемое изображение (рис. 3).

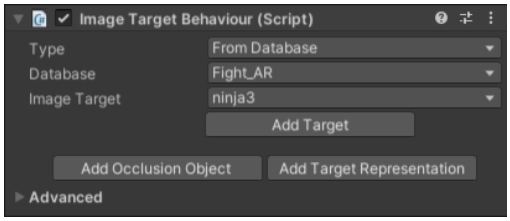

Рис. 3. Свойства изображения

Затем, импортируем модели с анимациями в удобную директорию, извлекаем текстуры для каждой анимации отдельно. И переносим модели в режиме ожидания поверх меток, прикрепляя их по иерархии к необходимой метке.

После, необходимо извлечь анимацию из каждой 3D модели (рис. 4). Для этого открываем файлы 3D моделей и копируем анимацию в ту же папку, чтобы использовать ее позже. Этот процесс повторяется для всех шести моделей.

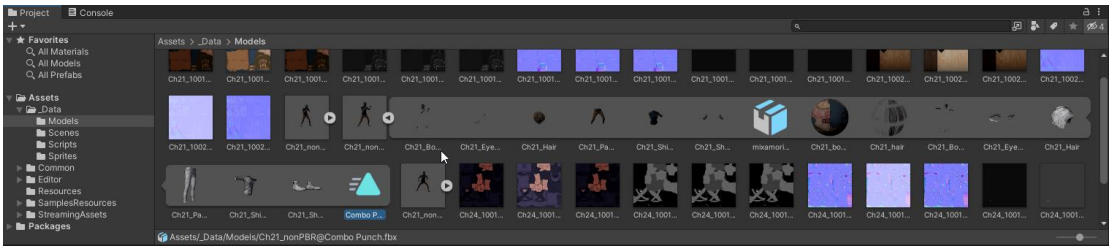

Рис. 4. Расположение анимации в модели

Следом, в папке с моделями создаем контроллер анимаций для каждого из персонажей: «Ninja» и «Woman» (рис. 5).

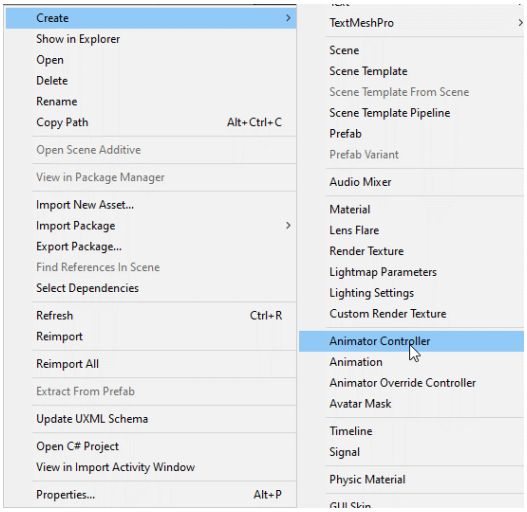

Рис. 5. Создание контроллера анимаций

К расположенной над меткой 3D модели добавляем новый компонент – «Animator». Переносим в поле «Controller» контроллер анимаций, проделываем так для двух моделей. На всех анимациях, кроме анимаций поражения, необходимо поставить флажок о зацикленности анимации – «Loop Time».

Далее необходимо открыть контроллер анимаций и создать схему следующего вида (рис. 6). При запуске приложения анимация ожидания (IDLE) будет автоматически активна, в то время как остальные анимации будут ожидать своего вызова до момента их активации.

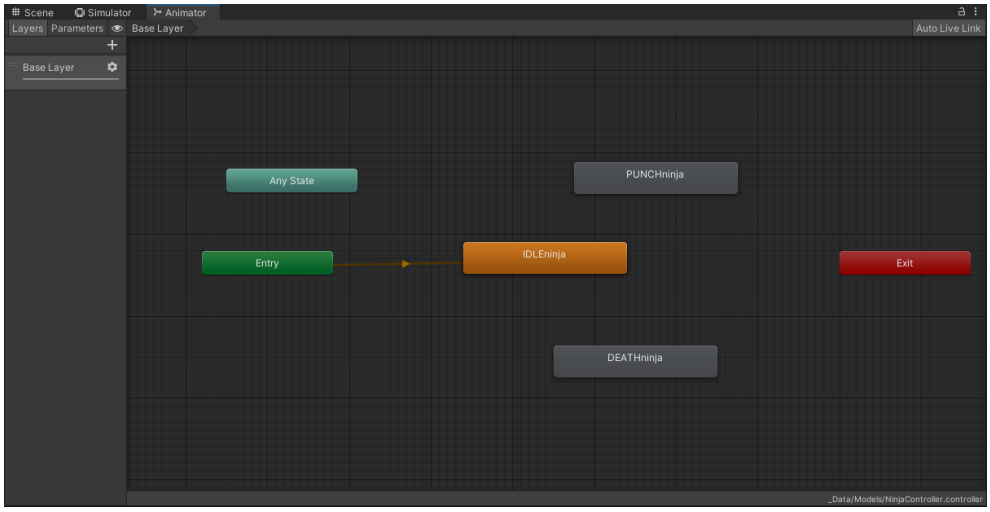

Рис. 6. Контроллер анимаций

Теперь требуется добавить куб в качестве триггера для каждой из меток. Куб будет использоваться в качестве события, срабатывающего триггер. Прикрепляем его к метке и назовем «Trigger». Далее требуется открыть свойства триггера и снять флажок с компонента «Mesh Renderer», чтобы сделать куб

прозрачным. Добавляем новый компонент к триггеру — «Rigidbody». Снимаем флажок с настройки «Use Gravity» и устанавливаем все флажки в «Freeze Position» и «Freeze Rotation», чтобы ограничить перемещение триггера. Создаем копию триггера и прикрепляем его к каждой из меток.

Рассмотрим создание пользовательского интерфейса здоровья в отдельной публикации.

Перейдем к созданию главного скрипта с названием «Fight», открываем его. Этот скрипт будет основным и влиять на функциональность проекта, за здоровье и урон персонажей (листинг 3).

Листинг 3

Объявление переменных и ссылок на объекты

```
public GameObject Ninja;
public GameObject Woman;
public int maxNinjaHealth = 100;
public int currentNinjaHealth;
public int maxWomanHealth = 100;
public int currentWomanHealth;
public int ninjaDamageMin = 15;
public int ninjaDamageMax = 22;
public int womanDamageMin = 12;
public int womanDamageMax = 18;
public HealthBar ninjaHealthBar;
public HealthBar womanHealthBar;
```

В методе Start() устанавливаются начальные значения здоровья для ниндзи и женщины, а также устанавливаются максимальные значения для шкалы здоровья SetMaxHealth (листинг 4).

Листинг 4

Установка начального значения здоровья

```
void Start(){
    currentNinjaHealth = maxNinjaHealth;
    ninjaHealthBar.SetMaxHealth(maxNinjaHealth);
    currentWomanHealth = maxWomanHealth;
    womanHealthBar.SetMaxHealth(maxWomanHealth);
```

```
}
```

Напишем следующий метод для смены анимации при столкновении коллизий (листинг 5).

Листинг 5

Обработка столкновений

```
private void OnCollisionEnter(Collision collision){
    if (!isNinjaDead && !isWomanDead){
        float distance =
Vector3.Distance(Ninja.transform.position,
Woman.transform.position);
        if (distance < 200f){
            Vector3 lookDirection = Woman.transform.position -
Ninja.transform.position;
            lookDirection.y = 0f;
            Ninja.transform.rotation =
Quaternion.LookRotation(lookDirection);
            Woman.transform.rotation =
Quaternion.LookRotation(-lookDirection);
            Ninja.GetComponent<Animator>().Play("PUNCHninja");
            Woman.GetComponent<Animator>().Play("PUNCHwoman");
            isFighting = true;
        }
    }
}
```

Напишем второй метод для смены анимации при прекращении столкновения коллизий (листинг 6).

Листинг 6

Обработка прекращения столкновений

```
private void OnCollisionExit(Collision collision){
    if (!isNinjaDead && !isWomanDead){
        Ninja.GetComponent<Animator>().Play("IDLEninja");
        Woman.GetComponent<Animator>().Play("IDLEwoman");
        isFighting = false;
    }
```

```
}
```

В методе Update() проверяется состояние отслеживает состояние драки, урон и здоровье персонажей, а также запускает соответствующие анимации в зависимости от событий столкновения и состояния здоровья (листинг 7).

Листинг 7

Обновление состояния драки

```
    private void Update(){
      if (isFighting && currentNinjaHealth > 0 &&
currentWomanHealth > 0){
        timer += Time.deltaTime;
        if (timer >= damageInterval){
          int ninjaDamage = Random.Range(ninjaDamageMin,
ninjaDamageMax + 1);
          int womanDamage = Random.Range(womanDamageMin,
womanDamageMax + 1);
          currentNinjaHealth -= womanDamage;
          currentWomanHealth -= ninjaDamage;
          ninjaHealthBar.SetHealth(currentNinjaHealth);
          womanHealthBar.SetHealth(currentWomanHealth);
          if (currentNinjaHealth <= 0){
            Ninja.GetComponent<Animator>().Play("DEATHninja");
            Woman.GetComponent<Animator>().Play("IDLEwoman");
            isFighting = false;
            isNinjaDead = true;
          }
          if (currentWomanHealth <= 0){
            Ninja.GetComponent<Animator>().Play("IDLEninja");
            Woman.GetComponent<Animator>().Play("DEATHwoman");
            isFighting = false;
            isWomanDead = true;
          }
          timer = 0f;
        }
      }
    }
```

Перейдем к созданию пользовательского интерфейса в Unity, а именно полосы здоровья в дополненной реальность, для этого необходимо создать холст «Canvas», а в нем разместим изображение «Image» (рис. 7).

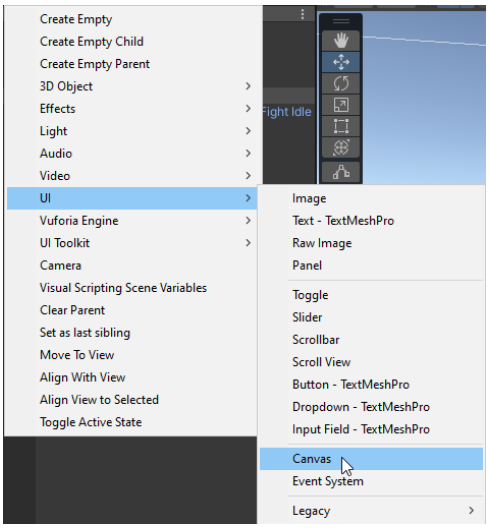

Рис. 7. Создаем холст

Изменим наложение холста на мировое пространство, чтобы можно было редактировать его размер. Для этого надо изменить свойство «Render Mode» на «World Space» (рис. 8).

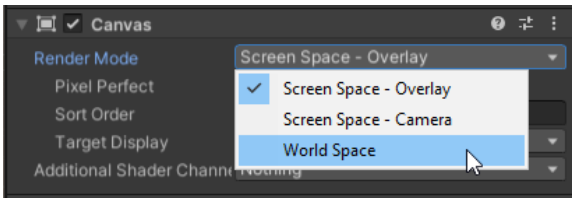

Рис. 8. Свойства холста

Размещаем холст над головой персонажа, для удобного отображения состояния здоровья персонажа (рис. 9).

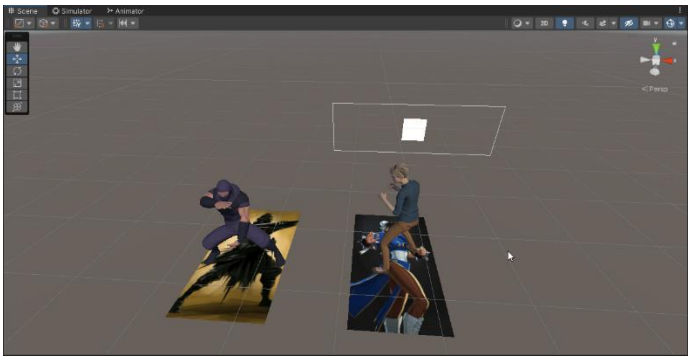

Рис. 9. Пример размещение холста

Размеры изображения внутри холста устанавливаются в свойствах изображения. Теперь заготовленные спрайты для полосы состояния здоровья

импортируются, и рамки для полосы здоровья переносятся в поле «Source image» (рис. 10). Затем изображение переименовывается в «Border».

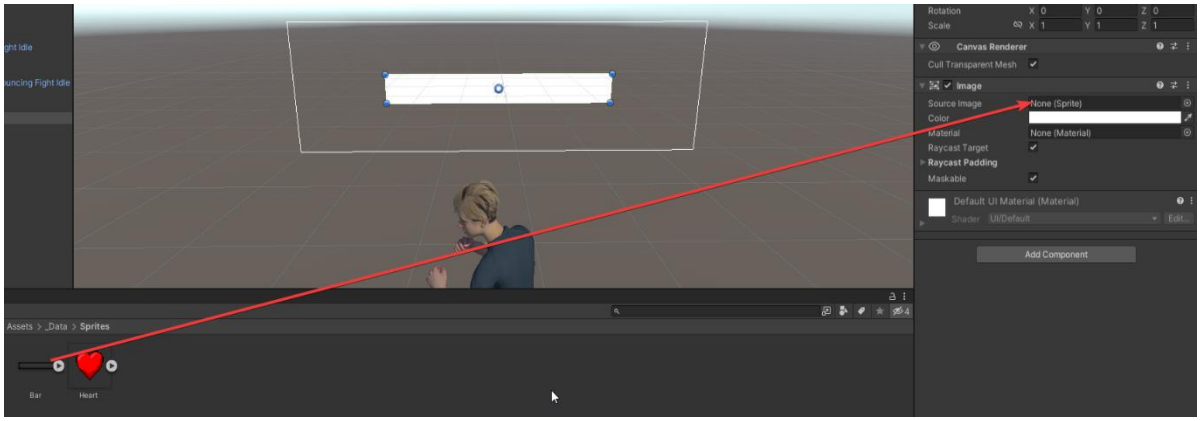

Рис. 10. Импорт спрайта как исходное изображения

После этого необходимо создать пустой объект и назовем его «Health Bar». Растянем пустой объект на рамки, чтобы границы были идентичного размера (рис. 11).

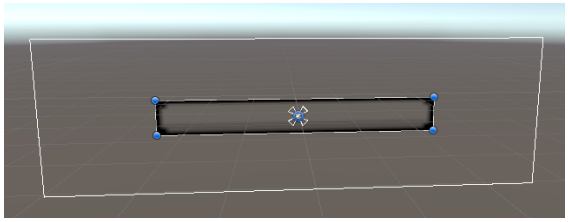

Рис. 11. Пустой объект растянут до размера рамки

Далее надо прикрепить «Border» к «Health Bar». Затем внутри «Health Bar» создается изображение, которое будет использоваться в качестве заливки полосы здоровья (рис. 12).

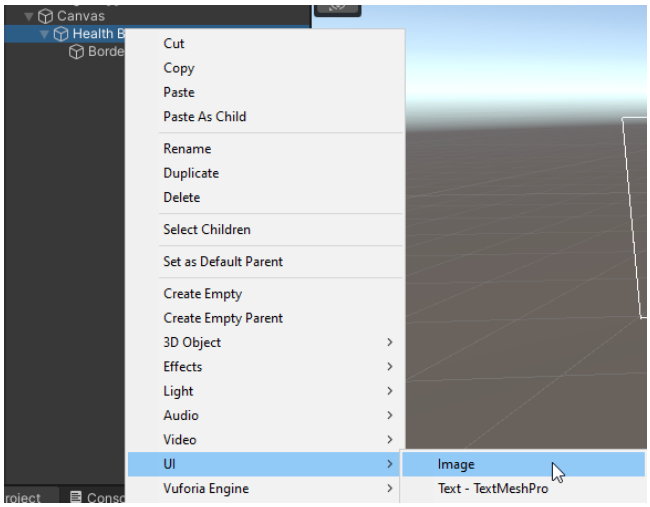

Рис. 12. Создание изображения внутри «Health Bar»

Теперь ставим рамки поверх изображения и меняем настройки привязки самого изображения. В свойствах находим необходимую опцию, зажимаем клавишу «alt» и растягиваем изображения по всей площади рамки (рис. 13). И переименуем изображение как заливка «Fill».

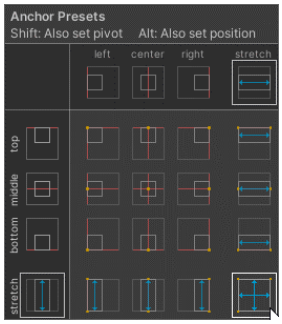

Рис. 13. Настройки привязки изображения

Добавляем объекту «Health Bar» новый компонент: «Slider», затем вносятся изменения в настройки нового компонента: отключается интерактивность, переход – отсутвует, навигация – отсутвует. Теперь нужно перетащить объект «Fill» в поле «Fill Rect», чтобы использовать его как заливку. И теперь можно изменять максимальное и минимальное значение, а также указывать стартовое значение. Изменим максимальное значение на 100. Теперь перейдем в настройки привязки «Border» и выберем заполнение.

Создадим изображение и распложим его слева от полосы здоровья, для более красивого интерфейса, а затем заменим спрайт изображение на сердце, также заменив название на «Heart». Теперь нужно установить привязку слева, чтобы при изменение маштаба, оно всегда находилось слева. Итоговый результат (рис. 14).

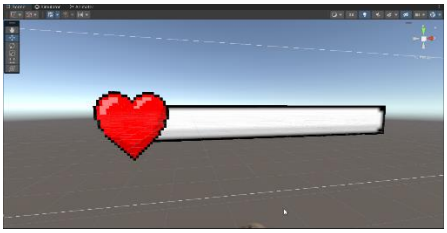

Рис. 14. Спрайт сердца

Изменим заливку «Fill» на цвет сердца при помощи пипетки (рис. 15).

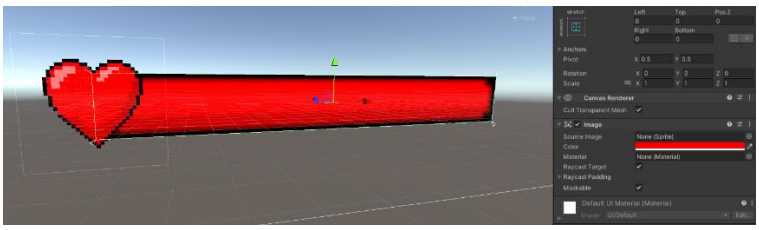

Рис. 15. Изменение цвета заливки

Продублируем «Health Bar» для второго персонажа.

Далее нужно написать скрипт для полосы здоровья. Данный код относится к компоненту «HealthBar» и отвечает за управление полоской здоровья (листиг 8).

Листинг 8

Скрипт HealthBar

```
public Slider slider;
public Gradient gradient;
public Image fill;
public void SetMaxHealth(int health){
    slider.maxValue = health;
    slider.value = health;
    fill.color = gradient.Evaluate(1f);
}
public void SetHealth(int health){
    slider.value = health;
    fill.color = gradient.Evaluate(slider.normalizedValue);
}
```

Теперь переходим в свойства объекта, который ранее назвали «Health Bar» появилось новое поле. Нажимаем на это поле и открывается окно. Придаем самой правой точке зеленый цвет, а левой красный. Нажимаем под полосой снизу по середине и добавляем ещё один цвет для градиента. Меняем его на желтый и получаем итоговый результат (рис. 16). Чтобы градиент начал работать, надо перенести в новое поле «Fill» заливку «Health Bar»

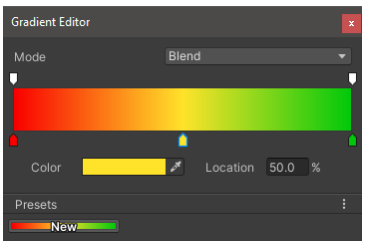

Рис. 16. Градиент полосы здоровья

Также необходимо, чтобы полоса здоровья всегда была повернута в сторону камеры. Для этого нам надо к холсту «Canvas» добавить скрипт «Billboard» (листинг 9).

Листинг 9

Скрипт Billboard

```
public Transform cam;
void LateUpdate(){
    transform.LookAt(transform.position + cam.forward);
}
```

Благодаря всем этим шагам, получилось построить мобильное приложение – карточную игру:

– обе модели находятся в режиме ожидания (рис. 17);

– при приближении к друг другу входят в боевой режим и наносят друг другу урон (рис. 18);

– при изменении положения карточек, 3D модели все равно направлены в сторону друг друга (рис. 19);

– при потери всего здоровья, один из персонажей переходит в анимацию смерти, а другой в анимацию ожидания (рис. 20).

Рис. 17. Анимация ожидания

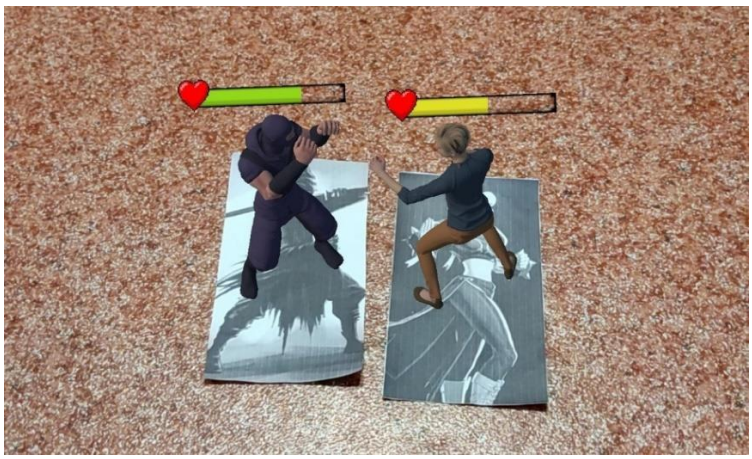

Рис. 18. Анимация удара

Рис. 19. Изменение положения карточек

Рис. 20. Итоги боя. Гибель одного из персонажей

Таким образом, получилось создать мобильное приложение дополненной реальность на основе меток с 3D моделями и их взаимодействием друг с другом, с различными сценариями развития.

Список источников информации

1. Громов, С. В. Технология дополненной реальности : методические указания / С. В. Громов. — Москва : МИСИС, 2022. — 92 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/305462 (дата обращения: 21.04.2024). — Режим доступа: для авториз. пользователей.

2. itProger — Уроки Unity C#. URL: https://itproger.com/course/unity-csharp (дата обращения: 21.04.2024).

3. Mixar — разработка дополненной реальности. URL: https://mixar.biz/blog/tekhnologiya_dopolnennoj_realnosti_ar?ysclid=lw32vnvjuk794825322 (дата обращения: 22.04.2024).

4. Mixamo — Animate 3D characters for games, film, and more. URL: https://www.mixamo.com/ (дата обращения: 22.04.2024).

5. Unity — Вперед к разработке. URL: https://unity.com/ru (дата обращения: 21.04.2024).

6. Unity — Vuforia Engine. URL: https://assetstore.unity.com/packages/templates/packs/vuforia-engine-163598 (дата обращения: 23.04.2024).

7. Vuforia — Vuforia Engine developer portal. URL: https://developer.vuforia.com/ (дата обращения: 22.04.2024).