МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий
кафедра Информатики и информационных технологий

направление подготовки 09.03.02 «Информационные системы и технологии»,
профиль «Технологии дополненной и виртуальной реальности в медиаиндустрии»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

БАКАЛАВРСКАЯ РАБОТА

Тема: Разработка карточной игры с применением дополненной реальности

Исполнитель Томашин Е.Д. _____
(Фамилия И.О.) (Подпись)

Руководитель Булатников Е.В., к.т.н. _____
(Фамилия И.О., степень, звание) (Подпись)

Нормоконтроль Булатников Е.В., к.т.н. _____
(Фамилия И.О., степень, звание) (Подпись)

Антиплагиат Демидова А.М., ст. преподаватель _____
(Фамилия И.О., степень, звание) (Подпись)

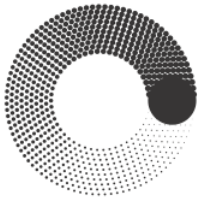
«ДОПУЩЕНО К ЗАЩИТЕ»

Зав. кафедрой ИиИТ: **Е.В. Булатников** _____
(Подпись)

Москва

2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ



Факультет информационных технологий
кафедра Информатики и информационных технологий

направление подготовки 09.03.02 «Информационные системы и технологии»,
профиль «Технологии дополненной и виртуальной реальности в медиаиндустрии»

УТВЕРЖДАЮ
Зав. кафедрой ИиИТ
Е.В. Булатников

« ____ » _____ 2022 г. _____
(Подпись)

ЗАДАНИЕ НА БАКАЛАВРСКУЮ РАБОТУ

Студент Томашин Егор Денисович **группа 181-724**
(Фамилия, Имя, Отчество)

Тема Разработка карточной игры с применением дополненной реальности

Утверждена приказом по Университету № 1657-УД от 04.05.2022 г.

1. **Срок представления работы к защите** « 08 » июня 2022 г.
2. **Исходные данные для выполнения работы:** консультации с дипломным руководителем, техническая литература, документации
3. **Содержание бакалаврской работы:** введение, аналитическая часть, практическая часть, заключение, приложения
4. **Перечень графического материала:** рисунки, блок-схемы, скриншоты
5. **Дата выдачи задания:** « 24 » декабря 2021 г.
6. **Руководитель:** _____ / Е.В. Булатников /
(Подпись) (И.О. Фамилия)
7. **Задание к исполнению принял:** _____ / Е.Д. Томашин /
(Подпись) (И.О. Фамилия)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий
кафедра Информатики и информационных технологий

направление подготовки 09.03.02 «Информационные системы и технологии»,
профиль «Технологии дополненной и виртуальной реальности в медиаиндустрии»

КАЛЕНДАРНЫЙ ПЛАН

Студент Томашин Егор Денисович **группа 181-724**
(Фамилия, Имя, Отчество)

Тема ВКР: Разработка карточной игры с применением дополненной реальности

№ п/п	Наименование этапа ВКР	Срок выполнения этапа	Примечание
1.	Составление плана ВКР	24.12.2021 – 15.01.2022	
2.	Исследование предметной области	16.01.2022 – 31.01.2022	
3.	Выполнение аналитической части	01.02.2022 – 13.02.2022	
4.	Разработка игрового проекта	14.02.2022 – 29.05.2022	
5.	Тестирование и публикация	30.05.2022 – 08.06.2022	

Руководитель: _____ / Булатников Е.В., к.т.н. /
(Подпись) (Фамилия И.О., степень, звание)

Зав. каф. ИиИТ: _____ / Е.В. Булатников /
(Подпись)

«____» _____ 2022 г.

АННОТАЦИЯ

В фантастических фильмах были различные очки, способные визуализировать виртуальный контент поверх реальности. Самым ярким примером такого эффекта являются фильмы о Железном человеке из киновселенной Марвел. То, с помощью чего это воспроизводится в наши дни, называется AR (Augmented Reality, Дополненная реальность). Данная технология стала настоящим прорывом в сфере многих индустрий и стремительно развивается.

Карточные игры популярны среди всех слоев населения на протяжении большого периода времени, так как обладают огромной вариативностью игровых процессов и исходов. С началом цифрового века всевозможные карточные игры начали переходить в формат многоплатформенных цифровых проектов.

Целью дипломной работы является разработка карточной игры с применением дополненной реальности. Эта комбинация актуальна тем, что совмещает в себе игру и технологию, которые в данный момент времени востребованы исходя из их популярности и новизны, подтверждение которых описано выше и будет аргументировано далее.

ABSTRACT

There have been various goggles in science fiction films capable of rendering virtual content on top of reality. The most striking example of this effect is the Iron Man films from the Marvel Cinematic Universe. The way that this is reproduced today is called AR (Augmented Reality). This technology has become a real breakthrough in many industries and is rapidly developing.

Card games have been popular among all segments of the population for a long period of time, as they have a huge variability of game processes and outcomes. With the beginning of the digital age, all kinds of card games began to move into the format of multi-platform digital projects.

The purpose of this thesis is to develop a card game using augmented reality. This combination is relevant because it combines the game and technology that are currently in demand based on their popularity and novelty the proof of which is presented above and will be confirmed with further facts.

РЕФЕРАТ

Цель работы: создание мобильного игрового приложения с применением технологии дополненной реальности в виде коллекционной карточной игры.

Задачи:

1. Изучение предметной области;
2. Анализ существующих аналогов приложения;
3. Выбор необходимых технологий;
4. Разработка мобильной AR-игры.

Первая глава содержит изучение предметной области, описание выбранных для реализации технологий. Вторая глава включает в себя описание этапов разработки проекта, а именно установку программного обеспечения, процесс разработки и сборки приложения. Объем пояснительной записки – 75 страниц. Количество рисунков – 40, блок-схем – 3, листингов – 10, приложений – 2.

Ключевые слова: AR, Дополненная реальность, Карты, Игра, Unity, ARCore.

ОГЛАВЛЕНИЕ

Введение

AR (Augmented Reality) переводится как «Дополненная Реальность». В фантастических фильмах были различные очки, способные поверх реальности отображать что-то информационное. Самым ярким примером являются шлем и очки Тони Старка из киновселенной Марвел. И вот уже не один год подобная сфера развивается и показанное в фильме уже становится часто используемым в мире и ускоряет многие процессы.

На сегодняшний день доступ к AR-разработка может иметь любой человек, который разбирается в IT-индустрии и способен писать код. Благодаря этому человечество может дополнить свой мир чем-то иным – тем, что нельзя либо нет возможности воплотить в реальной жизни. Любой пользователь благодаря этому может и рассмотреть автомобиль мечты «вживую», и посмотреть, как будет выглядеть его квартира после расстановки будущей мебели.

Карточные игры популярны среди всех слоев населения на протяжении большого периода времени, потому что обладают огромной вариативностью игровых процессов и исходов. В связи с этим на протяжении существования данной идеи придумывалось еще больше вариаций физических игр в карты - например, “Дурак” и “Блэкджек”.

С началом цифрового века всевозможные карточные игры начали переходить в формат многоплатформенных цифровых проектов. Например, в операционной системе стандартно была предустановлена классическая игра “Косынка”. Технологии стремительно развивались и улучшались, поэтому классика тоже не отставала - начали появляться как обычные простые мобильные игры, так и мультиплатформенные мультиплеерные игры. Самым популярным представителем такого направления стала “Hearthstone”, пользующаяся огромной популярностью среди всех.

Цифровые технологии не стоят на месте, и поэтому, учитывая вышеописанную популярность направления карточных игр, зародилась идея создания коллекционной карточной игрой с использованием дополненной реальности, которая стремительно развивается в последние годы.

Глава 1. Анализ и описание предметной области

1.1 История появления и развития карточных игр

1.1.1 Понятие и зарождение игральных карт

Игральные карты – это картонные прямоугольные листы, используемые для карточных игры, а также гаданий и фокусов. Комплект таких листов называется колодой, внутри которой каждая карта индивидуальна – одна сторона содержит в себе определенное значение. Обратная же сторона, называемая обложкой, одинакова у всех.

В настоящее время значением карты является тип (2 – 10, Валет, Дама, Король, Туз) и масть (Бубны, Черви, Трефы, Пики). Тип и масть карты указывается в верхнем левом и нижнем правом углах объекта, а вся площадь заполняется отзеркальной относительно центра фигурой (Валет, Дама, Король, Туз) или набором иконок масти (2 – 10). Пример карточной колоды изображен на рисунке 1.

Рисунок 1. Карты игральные

Происхождение игральных карты еще не выяснено. Установлено лишь, что они не были изобретены во Франции для забавы слабоумного короля Карла VI, а были известны еще раньше. Неудачной оказалась и попытка отыскать родину карт в Индии. Всего вероятнее, что К. изобретены в Китае. В словаре Чинг-цзе-Тунга, сделавшемся известным Европе в 1678 г., говорится, что К. изобретены в 1120 г. (по христианскому летосчислению), а в 1132 г. были в Китае уже в повсеместном употреблении. В Европе игральные карты появляются не ранее эпохи крестовых походов. [1]

Появление карт на Руси датируется 1600 годом — началом Смутного времени. Историческая версия связывается с ввозом карт поляками и украинскими казаками, а также немцами, «во множестве наезжающими тогда в московское государство». При царе Алексее Михайловиче карты упоминаются в Уложении 1649 года в 15 статье XXI главы: «А которые воры на Москве и в городе воруют, карты и зернью играют и проигрався воруют, ходя по улицам,

людей режут и грабят и шапки срывают... таких всяких чинов людем имая приводить в приказ... и тем вором чинить указ тот же, как писано... о татях» [272, т. 1, с. 139]. [2]

1.1.2 Понятие карточной игры

Карточная игра — игра с применением игральных карт, характеризуется случайным начальным состоянием, для определения которого используется набор (колода) карт для той или иной игры. Существует также множество наборов игральных карт, созданных под конкретные игры. Процесс определения начального состояния каждого тура (партии) игры называется раздачей карт и состоит в раскладывании определенного правилами игры количества карт по определенным местам. Пример: Игра в «Дурака» — раздать каждому из игроков (партнёров) по 6 карт «на руки» (то есть чтобы карты каждого игрока были открыты только ему), положить одну карту «в открытую» (открытую для всех) на стол, оставшиеся карты сложить стопкой «в закрытую» (закрытыми для всех). Важным принципом практически всех карточных игр является случайность порядка карт в колоде. Перед использованием той же колоды в следующей игре карты в ней перемешиваются (перетасовываются). [3] Игровой процесс изображен на рисунке 2.

Рисунок 2. Игра в карты

1.1.3 Понятие коллекционной карточной игры

Коллекционная карточная игра (ККИ, англ. Collectible Card Game, Trading Card Game) — разновидность настольных и компьютерных игр. В отличие от традиционных карточных игр, коллекционные карточные игры используют специальные карты, схожие с коллекционными карточками. В крупных коллекционных карточных играх могут существовать тысячи различных карт. Нельзя приобрести все существующие карты одновременно; вместо этого от игроков ожидается, что они будут приобретать карты небольшими наборами и составлять свои индивидуальные колоды. Собственно игра между игроками ведётся с использованием различных правил, которые могут сильно различаться для разных игр. [4]

1.1.4 Цифровизация игр в карты

С началом цифрового века всевозможные карточные игры (обычные и коллекционные) начали переходить на формат мультиплатформенных и

мультиплеерных игр и приложений. Самая большая популярность у данного жанра игр возникла с развитием операционной системы Windows 7, выпущенной 22 октября 2009 года и получившей популярность с 2011 года. В данной ОС находились изначально предустановленный набор игр, который был популярен в физическом проявлении, а теперь стал популярен еще в цифровом формате (рисунок 3).

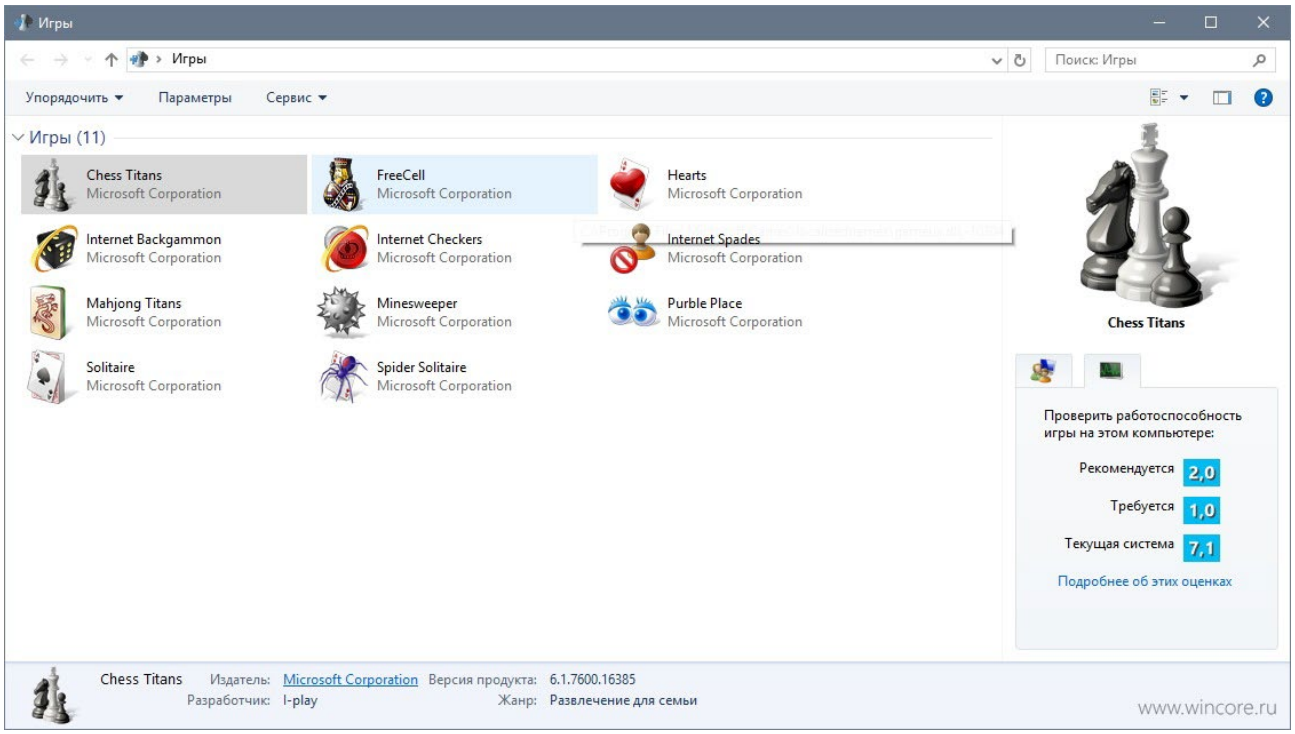


Рисунок 3. Набор классических игр Windows 7

Основными представителями карточных игр среди этого набора были пасьянсы «Косынка» и «Паук». Пасьянс – это одиночная карточная игра, в которой игроку необходимо набраться терпения и разложить карты, придерживаясь определённых правил и преследуя некоторую цель. Достижение данной цели возможно многократно благодаря случайности колоды и расклада и интеллектуальным способностям играющего.

Косынка (рисунок 4) – пасьянс, в котором необходимо разложить всю колоду из 52 карт в четыре стопки по мастям от туза до короля. Игра заканчивается, когда все карты разложены. Существуют различные

дополнительные модификации, добавляющие или ограничивающие различные условия и правила.

Рисунок 4. Windows 7 – Пасьянс «Косынка»

Паук (рисунок 5) – пасьянс, в котором используется 2 колоды по 52 карты. Цель игры — раскрыть все скрытые карты и, перемещая карты из одного столбца в другой, размещать карты в последовательном порядке от короля к тузу (король, дама, валет, 10, 9, 8, 7, 6, 5, 4, 3, 2, туз), используя наименьшее количество ходов. Каждая финальная последовательность должна быть одной масти.

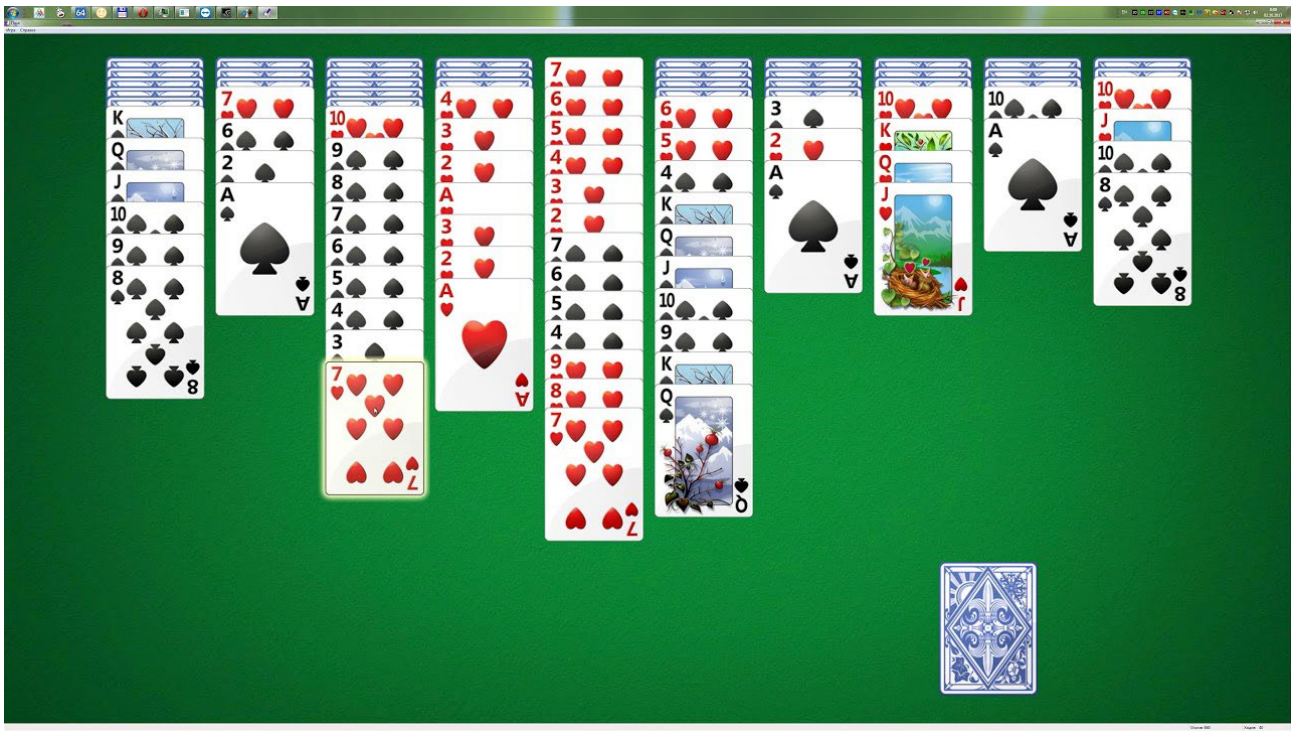


Рисунок 5. Windows 7 – Пасьянс «Паук»

С выходом новых версий операционной системы Windows популярность карточных игр в ней не угасала. Например, по состоянию на 1 сентября 2016 года число уникальных пользователей Microsoft Solitaire Collection превысило 100 (сто) миллионов человек по информации с Windows Blogs [5] (официальный блог компании Microsoft, разработчика ОС Windows). Как говорится в источнике: «В среднем каждый день в игры из пакета Microsoft Solitaire Collection играют 55 млн человек, что в сумме дает 20 млрд игроков в год или 2,29 млн каждый час и 38194 человек – каждую минуту.»

1.1.5 Карточные игры в наши дни

Воплощение карточных игр на данном этапе развития информационных технологий не остановилось, а лишь стремительно развивается с каждым днем – появляется огромное количество полноценных игровых проектов, работающих на различных устройствах (персональные компьютеры, планшеты, ноутбуки, телефоны, игровые приставки) и в различных режимах (одиночный и мультиплеерный). Самые успешные из таких проектов выходят на

киберспортивный уровень – происходит турнирное сражение игроков и полноценных команд друг против друга в реальном времени на глазах у миллионов фанатов со всей планеты. Ярким примером такого явления можно смело называть разработку игровой студии «Blizzard» под названием «Hearthstone».

Hearthstone – коллекционная карточная онлайн-игра по мотивам вселенной Warcraft, разработанная компанией Blizzard Entertainment и распространяемая по модели free-to-play. Игра была выпущена для персональных компьютеров 11 марта 2014 года. Позже она была портирована на платформы iOS и Android. Ведущими геймдизайнерами выступили Эрик Доддс и Бен Броуд. [6] Скриншот из игры приведен на рисунке 6.

Рисунок 6. Hearthstone – боевой процесс

Игровой процесс обманчиво прост и невероятно интересен. Вам необходимо собирать боевую колоду из различных типов карт со своими уникальными характеристиками и возможностями и выходить в бой против

другого реального противника. Результат зависит только от вас. С более подробными условиями игры можно ознакомиться на официальном сайте проекта [7].

1.2 История появления и развития дополненной реальности

1.2.1 Понятие дополненной реальности

AR (Augmented Reality, Дополненная реальность) – это технология, расширяющая наш физический мир, добавляя в него через экраны устройств цифровую информацию. В отличие от VR (Virtual Reality, Виртуальная реальность), система не заменяет реальный мир виртуальным, а лишь дополняет его различным виртуальным контентом (графика, видео, звуки и т.п.). Наложение компьютерного изображения поверх физической среды реального мира, позволяющее изменить его восприятие, является непосредственной «обязанностью» дополненной реальности.

Мобильные платформы являются целевыми при разработке проектов с применением AR, так как такие устройства есть практически у каждого и проще в транспортировке (рисунок 7).

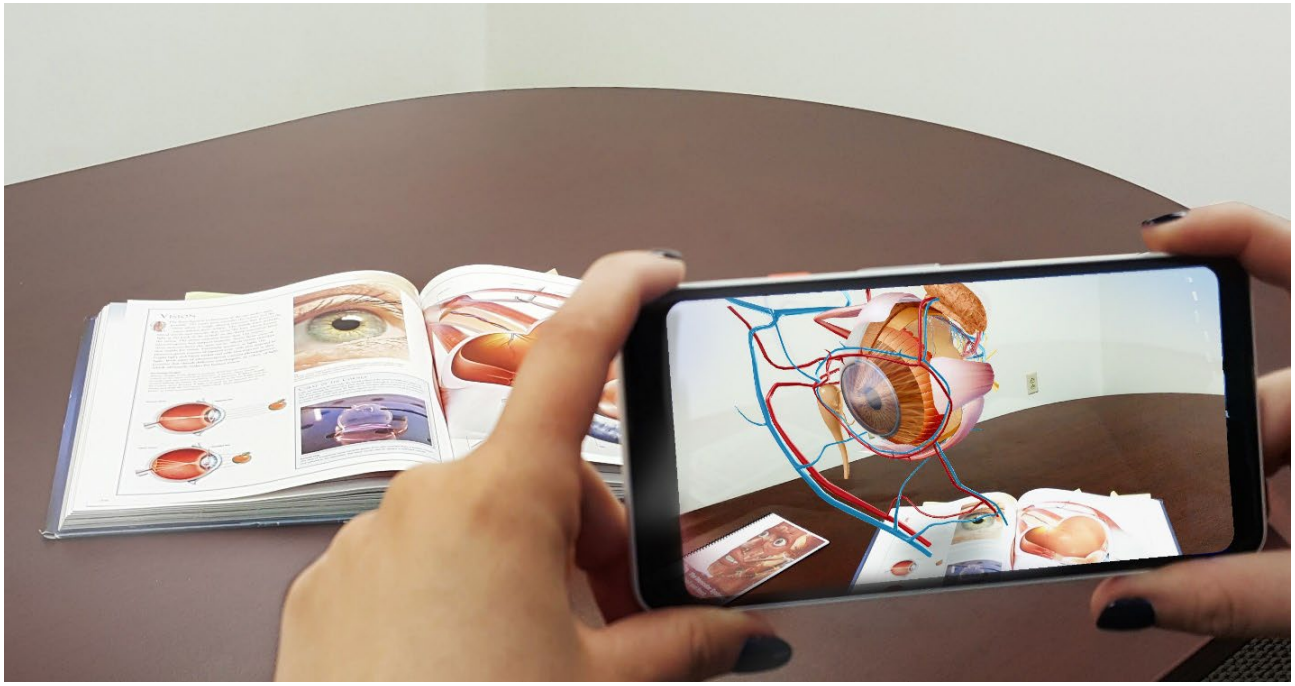


Рисунок 7. Пример работы AR

Меточная технология является основной в работе дополненной реальности, так как она основана на системе оптического трекинга аналогично стереоскопическому зрению человека. Камера устройства отслеживает специальные оптические маркеры, о которых система заранее знает, и после чего подает необходимый сигнал для проявление виртуального контента согласно сценарию приложения. Виртуальный контент способен реагировать на изменение позиции и ориентации устройства, что дает возможность видоизменяться согласно требованиям разработчика и пользователя.

AR превращает мир в то самое, что мы видим в фантастических фильмах – ему это удается, и совсем скоро то, что было в фильмах, будет у нас! Для пользователей – много новых возможностей, а для разработчиков – рост новой технической сферы, возможности трудоустройства и большие заработные платы.

1.2.2 История развития дополненной реальности

Дополненная реальность создает неполное погружение человека в виртуальный мир, тем самым отличия данную технологию от виртуальной

реальности. Соответственно, AR берет своё начало из разработок касаемо VR направления.

Считается, что создателем виртуальной реальности является кинематографист и изобретатель Мортон Хейлинг. В 1957 году миру было представлено первое устройство виртуальной реальности Sensorama (рисунок 8), запатентованное в 1962, но не получившее доверия инвесторов, из-за чего разработка была прекращена. Устройство представляло собой большой объект, содержащий в себе несколько дисплеев и звуковых устройств с посадочным местом с возможностью вибрации. Всё это позволяло пользователю частично погрузиться в виртуальный мир, проезжая по городским улицам.

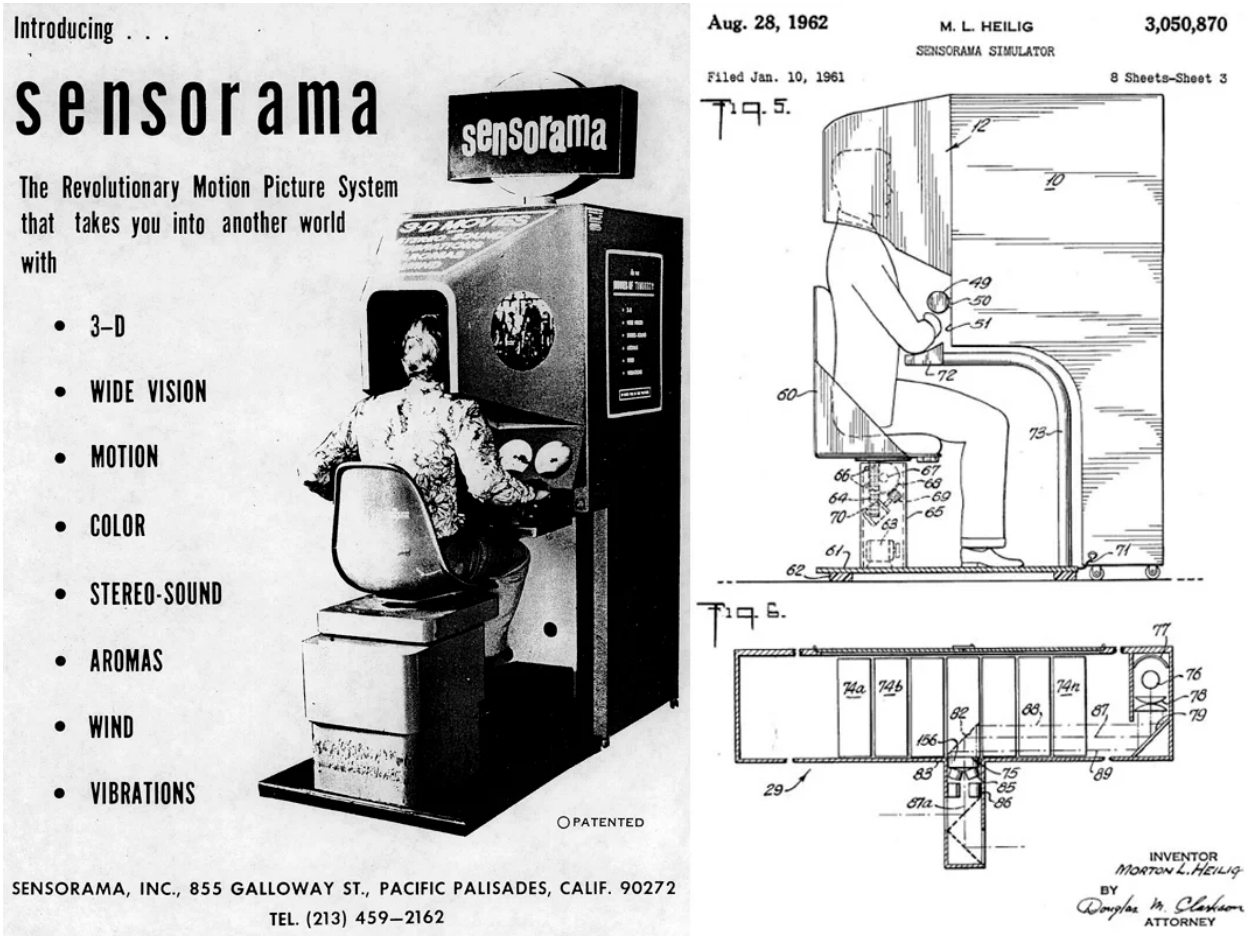


Рисунок 8. Sensorama

Первая технология дополненной реальности была разработана в 1968 году в Гарварде, когда ученый-компьютерщик Иван Сазерленд (названный «отцом

компьютерной графики») создал систему отображения дополненной реальности, устанавливаемую на голову. Устройство, состоящее из электронно-лучевых трубок и оптических элементов, проецировало на глаз пользователя сгенерированные компьютером изображения. Своё название «Дамоклов меч» механизм получил из-за очень тяжелого шлема, который крепился к потолку для меньшей нагрузки на человека. Оно продемонстрировано на рисунке 9.

Рисунок 9. Sword of Damocles

В 1974 Майрон Крюгер, компьютерный исследователь и художник, построил в Университете Коннектикута лабораторию под названием «Videoplace» (рисунок 10), полностью посвященную искусственной реальности. Основной задумкой было то, чтобы освободить потребителя от надевания чего-либо для взаимодействия с виртуальностью. Для работы устройства использовались видеокамеры и проекторы, которые записывали, анализировали и транслировали силуэт человека на экран, позволяя ощутить себя частью информационной системы.

Рисунок 10. VideoPlace

Термин дополненная реальность был введен исследователем Boeing Томом Коделлом в 1990 году. В последующие десятилетия лабораторные университеты, компании и национальные агентства продолжали развивать AR для носимых устройств и цифровых дисплеев. Эти ранние системы накладывали виртуальную информацию на физическую среду (например, накладывали на местность геолокальную информацию) и позволяли моделировать, которые использовались в авиации, военных и промышленных целях. Так в 1992 году Луи Розенбург, исследователь исследовательской лаборатории Армстронга ВВС США, создал «Виртуальные приспособления» (англ. Virtual Fixtures), одну из первых полнофункциональных систем дополненной реальности. Экзоскелет Розенберга позволял военным виртуально управлять машинами, находясь в удалённом центре управления. Он изображен на рисунке 11.

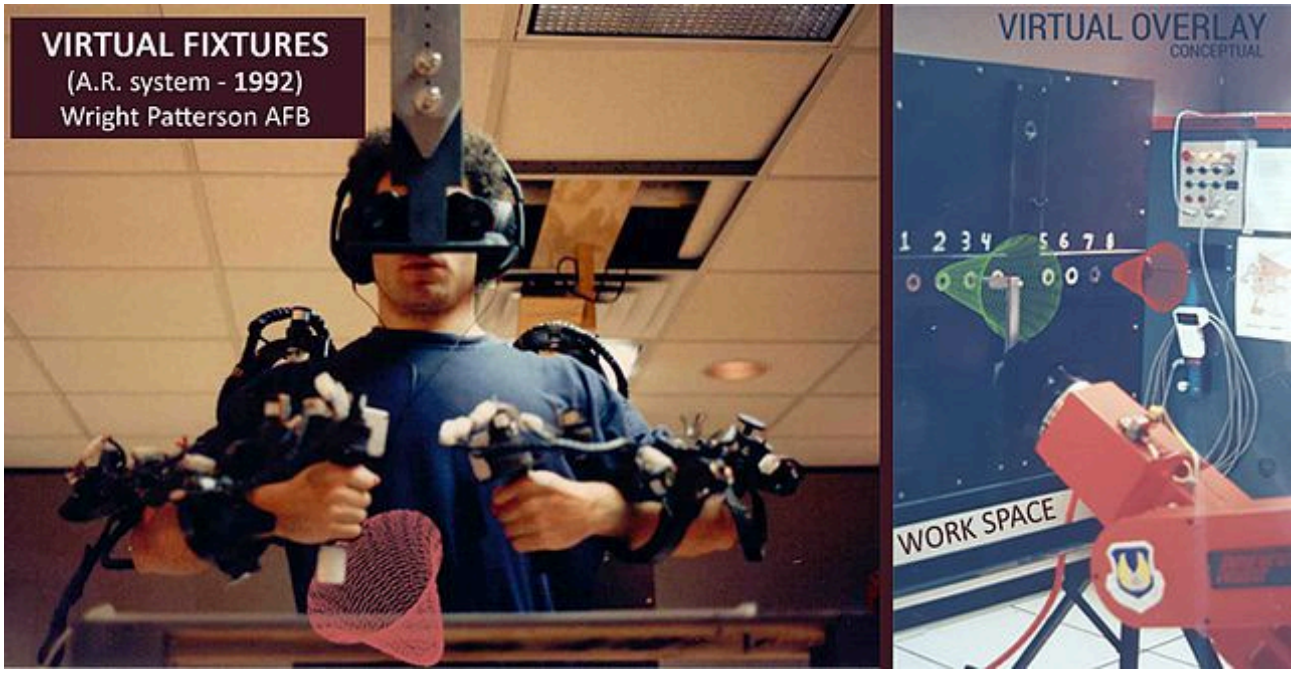


Рисунок 11. Virtual Fixtures

Первое коммерческое AR-приложение появилось в 2008 году. Он был разработан в рекламных целях немецкими агентствами в Мюнхене. Они разработали печатную журнальную рекламу модели BMW Mini (рисунок 12), которая, когда ее держали перед камерой компьютера, также появлялась на экране. Поскольку виртуальная модель была связана с маркерами на физическом объявлении, пользователь мог управлять автомобилем на экране и перемещать его, чтобы увидеть под разными углами, просто манипулируя листом бумаги. Приложение стало одной из первых маркетинговых кампаний, позволивших взаимодействовать с цифровой моделью в режиме реального времени.

Рисунок 12. Рекламная кампания BMW Mini

Моделирование цифровых продуктов, чтобы они взаимодействовали с движениями в реальном мире в режиме реального времени (обычно с помощью бумажных распечаток), было популярным подходом к AR в начале 2010-х годов, особенно для часов и ювелирных изделий. Эта технология позволяет людям виртуально «примерить» продукт. Даже часы Apple были доступны для подобной виртуальной примерки. Однако задача распечатать и вырезать специальную бумажную модель так, чтобы она помещалась на пальце или запястье, всегда была несколько громоздкой и требует от потребителя определенных усилий.

Гораздо более успешными являются те приложения, которые предлагают более удобный опыт. Виртуальная примерка продуктов с помощью мгновенного распознавания лиц стала одним из самых успешных применений дополненной реальности в коммерческом контексте до сих пор, и косметические компании лидировали в этом использовании.

Помимо примерок, большое количество исследований также показывает, что AR может быть невероятно ценным для изучения различных культурных, исторических и географических аспектов окружающей среды. Этот тип приложений обычно работает на основе того, что пользователь направляет свое мобильное устройство на объект или сайт, чтобы увидеть наложенный контент на экране.

1.2.3 Методы построения дополненной реальности

Для правильной работы технологии дополненной реальности ей необходима информация об окружающей среде. Данные могут быть получены разными способами, которые можно разделить на две категории – пространственные и оптические.

Пространственный тип ориентируется на данные, полученные с сигналов GPS, компаса, акселерометра, гироскопа, магнитометра и т.п. Обработка такой информации позволяет системе определить точное положение и ориентацию пользователя в пространстве для качественного добавления виртуального контента вокруг него.

Оптический тип подразумевает обработку изображения, поступающего с камеры устройства. Данная категория делится на несколько подкатегорий – меточная, безметочная, проекционная.

Меточная технология подразумевает использование маркеров, которые заранее определяются и имеются в реальном мире. Камера обнаруживает сходства элемента окружающей среды с меткой, определяет положение и размеры объекты, а далее система на основе данной информации размещает виртуальный контент поверх согласно сценарию программного обеспечения.

Безметочная технология анализирует окружающее пространство, пытаясь выделить плоские поверхности. В основе такого подхода лежит отслеживание

положения устройства и изменения ключевых точек полученного отображения. Достоинство этой системы кроется в том, что пользователь может размещать и сохранять цифровую информацию в любой точке обнаруженных мест.

Проекционная технология работает на основе проекции света на различные поверхности. Система анализирует изначальное и конечные изображения с камеры, чтобы просчитать совершенные потребителем изменения относительно реальности.

1.3 Описание игры, референсов и аналогов

Дипломный проект заключается в создании мобильного игрового приложения с применением технологии дополненной реальности в виде коллекционной карточной игры (ККИ). Под такой игрой подразумевается обычная карточная игра, которая портирована под мобильные устройства и имеет ряд различных карточных механик. Каждая карта имеет свои уникальные характеристики, что позволяет их улучшать – например, добавлять специальные возможности в разных ситуациях. Помимо этого, все эти карты коллекционируются и комбинируются между собой.

В наше время каждый игровой проект должен предоставлять игроку возможность соревноваться с кем-либо. По этой причине, в проекте само собой будут реализованы сражения в различных форматах и режимах, в которые игрок будет выходить в бой путем комбинирования карт из коллекции в колоду. Основными режимами являются PvP и PvE. PvP (игрок против игрока) – битва двух реальных игроков в реальном времени посредством мультиплеера. PvE (игрок против окружения) – одиночное сражение против ИИ (искусственный интеллект), либо же сражение против компьютера/бота. Основной формат битвы будет в 3D, так как 2D (интерфейсное) уже слишком часто используется и не дает игроку погрузиться полностью в процесс. Как раз-таки для полноценного погружения планируется использование технологии дополненной реальности, с

помощью которой будет возможность расположить виртуальный игральный стол в реальном пространстве, а не смотреть на него в виртуальной «бездне».

Само собой, при разработке игры её создатели должны осознавать то, насколько их проект будет ликвидным (популярным и прибыльным). Учитывая, что всевозможный функционал уже реализован в различных проекта, то есть необходимость ссылаться на эти проекты для черпания хороших идей, которые не нарушают авторские права. Помимо этого, желательно сравнение своего проекта с уже существующими аналогами, если таковые имеются.

В качестве референсов разрабатываемой игры взяты 4 проекта. Основным референсным проектом является «Hearthstone» [6, 7], который упоминался ранее в разделе об истории карточных игр. Основными особенностями данного проекта являются уникальные карты и их персонажи, возможность комбинирования нескольких карт и сражения в реальном времени против таких же игроков. «Clash Royale» [14] хвалится своими анимированными персонажами, которые «слезают» с выбранных карт и сражаются с многочисленными противниками. «Pokémon Go» [15] в свое время «взорвал» весь игровой мобильный сегмент крутой реализацией технологии дополненной реальности, с помощью которой игрокам было необходимо искать и ловить виртуальных персонажей в реальном мире. «Argy Bargy: Craft» [16] - проект, разработку которого я ввел на протяжении полугода в одной из студий. Именно после этого опыта у меня и зародилась идея разработки данного проекта – по сути замена обычных UI баталий на 3D с AR.

В ходе поиска уже существующих аналогичных проектов мне удалось найти несколько. «Drazkerz-Confrontation» [17], разработанная французской компанией, происходит перед компьютером с веб-камерой, которая считывает изображение на карточках, выкладываемых на стол, а затем на экране воспроизводятся анимированные действия, соответствующие игровой ситуации. Минус данного проекта заключается в отсутствии мобильности и необходимости

наличия физических карточек. Три других аналога, названных на портале «Голографика» [18], имеют схожие проблемы. В разрабатываемом же проекте планируется устранение этой проблеме, таким образом, что она будет максимально доступна большому количеству людей.

1.4 Анализ и выбор используемых технологий

1.4.1 Игровая среда

В наши дни разработка игровых проектов возможна на большом количестве движков. Крупные компании производят своё ПО на средах собственной разработки, но это делают лишь те, кому позволяют это финансовые средства. В основном используются уже созданные общедоступные игровые движки для разработки необходимого ПО. Основными лидерами такого направления являются Unity и Unreal Engine (рисунок 13).

Рисунок 13. Сравнение Unity и Unreal Engine

Unity – межплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах,

включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие. Выпуск Unity состоялся в 2005 году и с того времени идёт постоянное развитие.

Это универсальный движок, на котором разрабатывают игры для миллионов пользователей по всему миру: инди и проекты AAA-студий, платформеры и стратегии, симуляторы выживания и RPG, одиночные и мультиплеерные игры. На движке Unity работают известные проекты: Genshin Impact, Rust, Escape from Tarkov, Life is Strange: Before the Storm, Call of Duty: Mobile.

Игровой движок Unity принят многими пользователями, поскольку он дает пользователям возможность создавать игры в 2D и 3D, а также в виртуальной реальности. Помимо игровой индустрии, он также используется в других отраслях, таких как кино, автомобилестроение, архитектура, машиностроение и строительство.

Особенности: настройка объектов из нескольких независимых компонентов, встроенные необходимые инструменты для разработки под разные жанры, огромный магазин ассетов (готовые наборы) от других пользователей, портирование под разные платформы по клику. Интерфейс среды разработки игры продемонстрирован в виде скриншота на рисунке 14.

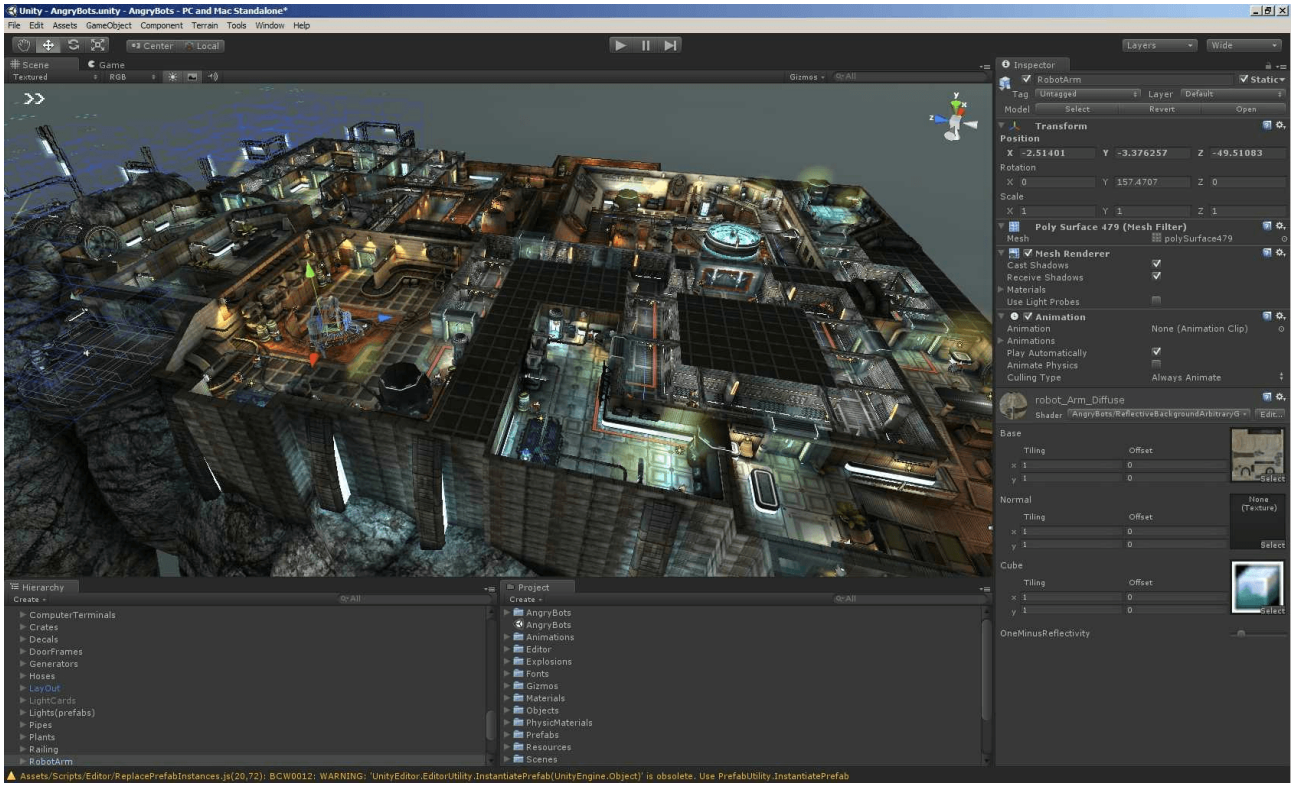


Рисунок 14. Интерфейс Unity

Unreal Engine — игровой движок, разрабатываемый и поддерживаемый компанией Epic Games. Первой игрой на этом движке был шутер от первого лица Unreal, выпущенный в 1998 году. Хотя движок первоначально был предназначен для разработки шутеров от первого лица, его последующие версии успешно применялись в играх самых различных жанров, в том числе стелс-играх, файтингах и массовых многопользовательских ролевых онлайн-играх. В прошлом движок распространялся на условиях оплаты ежемесячной подписки; с 2015 года Unreal Engine бесплатен, но разработчики использующих его приложений обязаны перечислять 5 % роялти от общемирового дохода с некоторыми условиями.

Особенности: визуальное программирование, ориентированность на ААА-проекты (высококачественные), много встроенных улучшений для разработчиков, проработка анимаций и эффектов. Интерфейс среды разработки игры продемонстрирован в виде скриншота на рисунке 15.

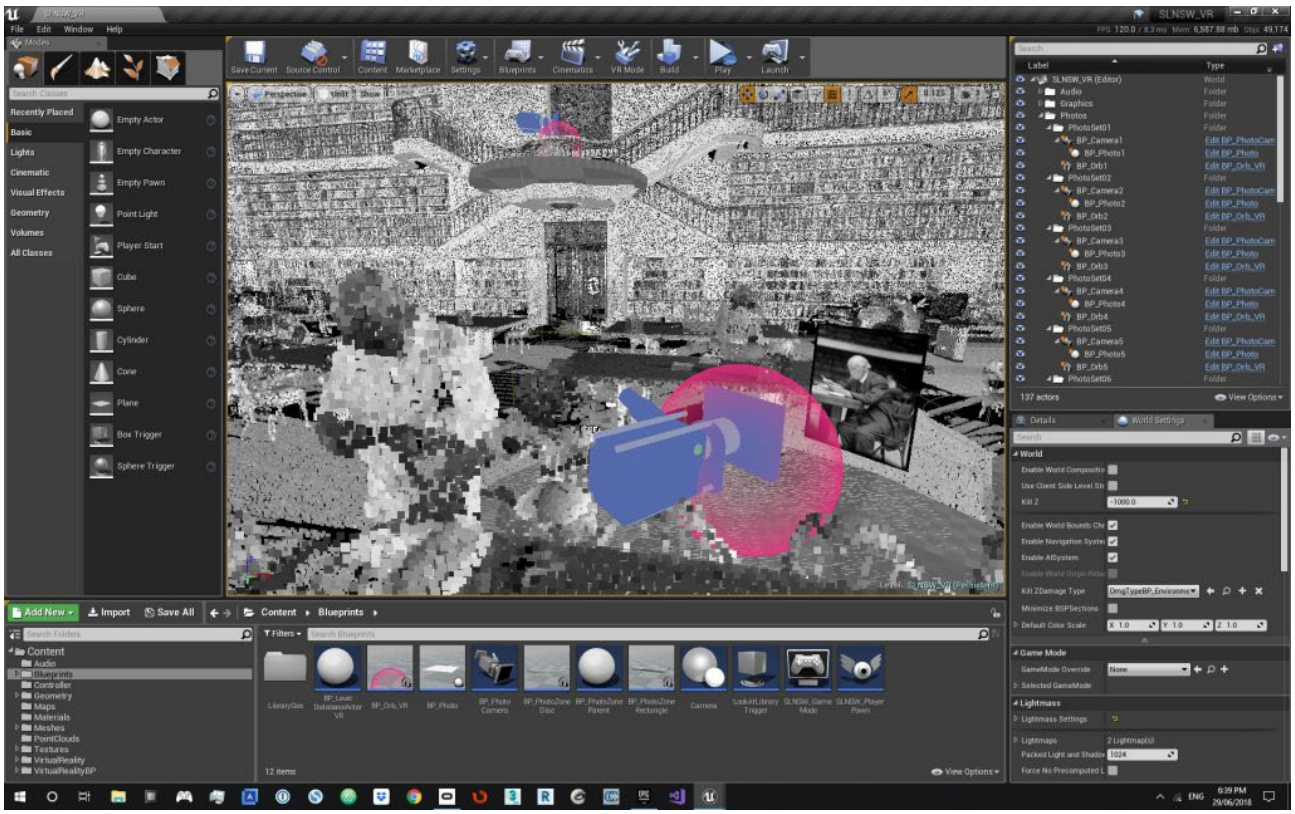


Рисунок 15. Интерфейс Unreal Engine

Оба игровых движка имеют много общего, так как должны как можно сильнее привлечь к себе потенциального разработчика. Но Unity привлекает к себе большинство инди-разработчиков в связи с тем, что используемый C# считается более подходящим для разработки игр, чем C++, поэтому Unity работает быстрее. Помимо этого, данная среда более оптимизирована под мобильную разработку, а соответственно под работу с дополненной реальностью, что есть важный фактор для работы над выбранным проектом.

1.4.2 Дополненная реальность

В настоящее время для создания приложения с дополненной реальностью вовсе необязательно писать код программы с нуля – можно воспользоваться специальными платформами SDK (от англ. Software Development Kit – набор средств разработки), которые облегчат этот процесс.

На сегодняшний день самыми распространенными инструментами работы с дополненной реальностью являются Vuforia и ARFoundation (ARCore + ARKit), если рассматривать касаемо разработки на Unity. Логотипы данных технологий на рисунке 16.

Рисунок 16. ARKit / ARCore / Vuforia

Vuforia — одна из старейших AR-компаний на рынке. После приобретения в 2015 году компанией PTC Inc. Vuforia расширила свою линейку инструментов, ориентированных на дополненную реальность. Эти инструменты теперь включают такие продукты, как Vuforia Engine и Vuforia Studio, которые используются при разработке приложений дополненной реальности.

Использование данной платформы дает возможность располагать виртуальные объекты, которыми могут быть 3D-модели, видео, изображения, в реальной сцене, отображающейся на мобильных устройствах. Причем виртуальный объект располагается так, чтобы быть визуально представленным как часть реального мира. Этот эффект достигается за счёт позиционирования объекта в определенном положении относительно точки зрения пользователя.

Недавно Vuforia Engine представила достижения в области дополненной реальности без маркеров. Это означает, что виртуальные объекты, размещенные в физической среде, более стабильны по сравнению с предыдущими версиями Vuforia Engine (рис. 17).

Рисунок 17. Vuforia

AR Foundation — это пакет, который позволяет создавать кроссплатформенные приложения дополненной реальности в Unity. Этот инструмент включает в себя пакеты ARKit и ARCore XR, что означает возможность разработать свое приложение AR в Unity3D, а затем портировать его под Android или iOS. Сравнение использования этих технологий отображено ниже на рисунке 18.

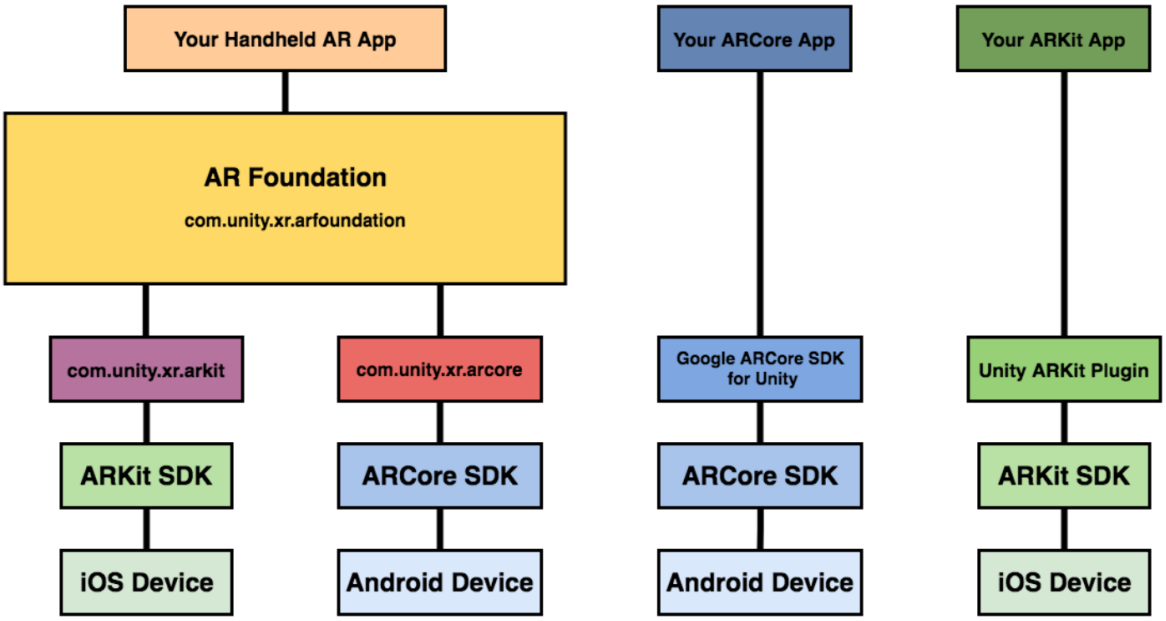


Рисунок 18. ARFoundation

Появились данные комплекты средств совсем недавно и друг за другом. В 2018 году прошли большое обновление, и теперь не нужно самому работать с метками/маркерами, привязкой к определенным предметам или изображениям. Этому всему разработчики научили свои технологии – теперь они способны на анализ и распознавание окружающей среды. Это огромные плюсы, но есть и небольшой минус – SDK по понятным техническим причинам не могут работать со слабыми старыми устройствами, что, в принципе, не является проблемой в наши дни, так как любое бюджетное устройство уже поддерживает данные технологии.

ARCore — это инструмент для разработки программного обеспечения, произведенный Google, позволяющий создавать игры и приложения дополненной реальности. Этот комплект средств AR-разработки способен на многое: отслеживать движение, в том числе различать целенаправленное перемещение устройства и случайную тряску; подразделять мир на условные сущности; понимать окружающий мир: находить объекты, определять их

размеры и местоположение всех типовых поверхностей (вертикальные, горизонтальные и угловые); оценивать освещённость: это позволяет смартфону определить текущие условия освещения окружающей среды, находить источники света и различать тени; определять источники звука, распознавать голоса, лица, жесты и так далее (рисунок 19).

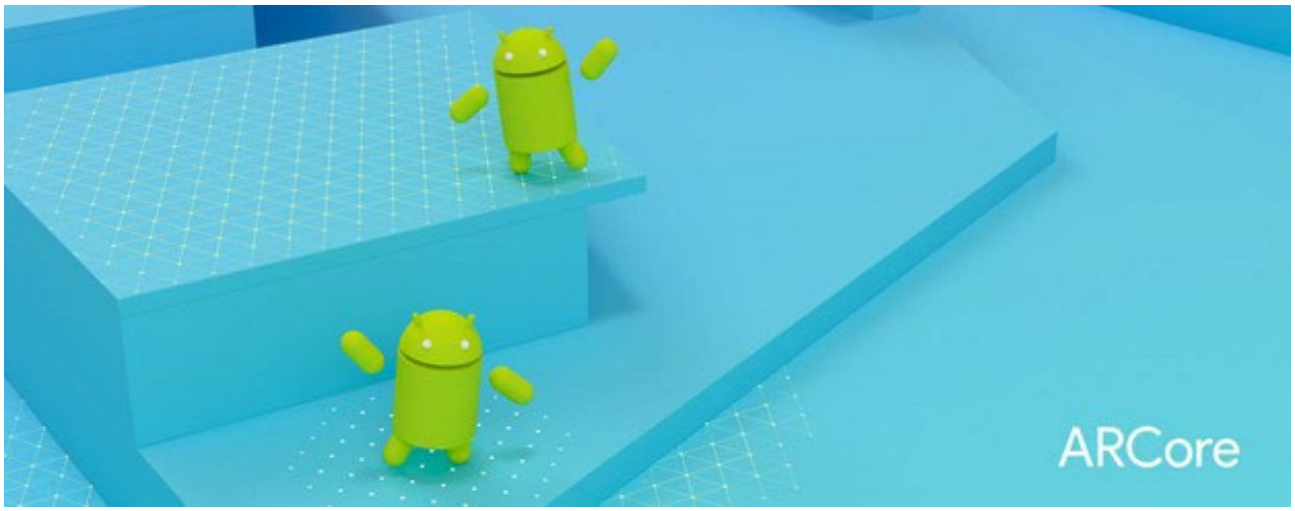


Рисунок 19. ARCore

Дела у разработки от Apple, ARKit, обстоят тоже неплохо. На WWDC 2017 (Worldwide Developers Conference - Всемирная конференция для разработчиков на платформах Apple) яблочная компания анонсировала ARKit (рисунок 20) — SDK для работы с дополненной реальностью для пользователей свой операционной системы IOS. Благодаря ему войти в эту технологию стало значительно легче и пользователям становится доступно большое количество игр и приложений с использованием AR. То, что разработчики Apple смогли развернуть на обычном столе с помощью данной технологии, поразило всех неравнодушных. Это был не просто пробник, а уже отлично работающая технология, над которой славно трудятся по сей день. С помощью большого количество демо мы можете сами в этом убедиться. К слову, для использования данной функции требуется техника Apple с процессором A9 и выше.

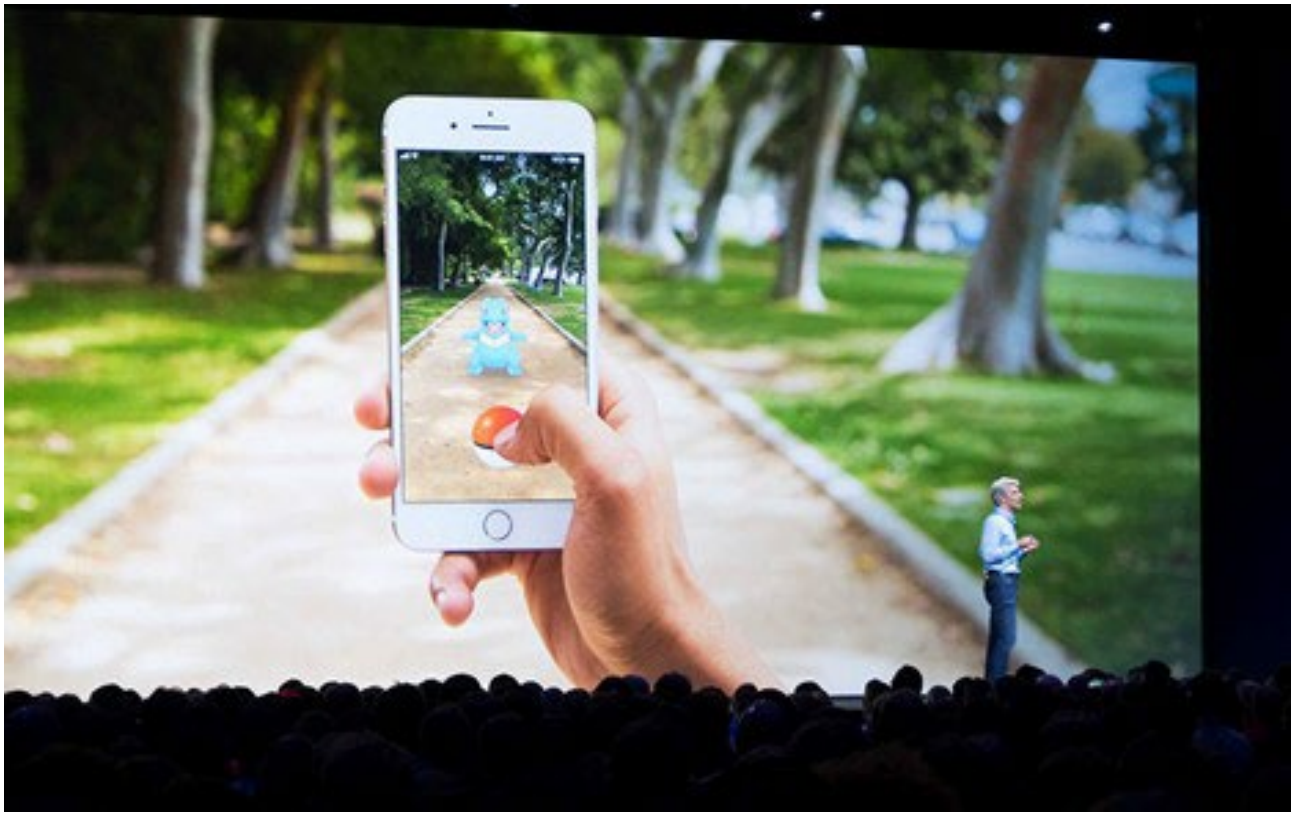


Рисунок 20. ARKit

Учитывая описанные возможности обоих фреймворков для работы с дополненной реальностью, выбор однозначный – ARFoundation. Данный выбор очевиден исходя из истории технологий… Vuforia изначально является меточной технологией, и только недавно начала делать упор на безметочность и фиксированность виртуальных объектов. Техногиганты Apple и Google разрабатывали свои технологии, соревнуясь между собой, что позволило ускорить темпы роста доступного функционала, который объединяется с помощью ARFoundation.

1.4.3 База данных

В разрабатываемом проекте большое внимание уделяется игровым коллекционным картам, которые необходимо где-то хранить. Необходимо хранение как профилей игроков, так и их статистики (очки, валюта и т.д.) и расширяемые базы карт (колода и коллекция). Функционал написания базы

данных через, например, phpMyAdmin и дальнейшее взаимодействие не клиенториентированы, а сложно соединяется с игровыми движками, так как всё это по большей части не про игры. Но компания Microsoft не стоит на месте и представила Azure PlayFab – то, что решает данные проблемы.

PlayFab – это полноценная серверная платформа для игр в реальном мире с управляемыми игровыми сервисами и аналитикой. Данный игровой облачный сервис был приобретён компанией Microsoft в 2018 году для развития облачной платформы Microsoft Azure. По данным Microsoft, сервисом воспользовались 700 млн пользователей, на платформе воспроизводятся более 1200 игр от таких компаний, как Disney, Rovio и Atari.

Сервисные услуги Azure PlayFab уменьшают барьеры запуска проектов для игровых разработчиков путем предложения больших экономически выгодных и эффективных решений для разработки, которые масштабируют их игры и помогают привлекать, удерживать и монетизировать игроков. PlayFab позволяет разработчикам использовать интеллектуальное облако для создания игр и управления ими, анализа игровых данных и улучшения общего игрового опыта (рисунок 21).

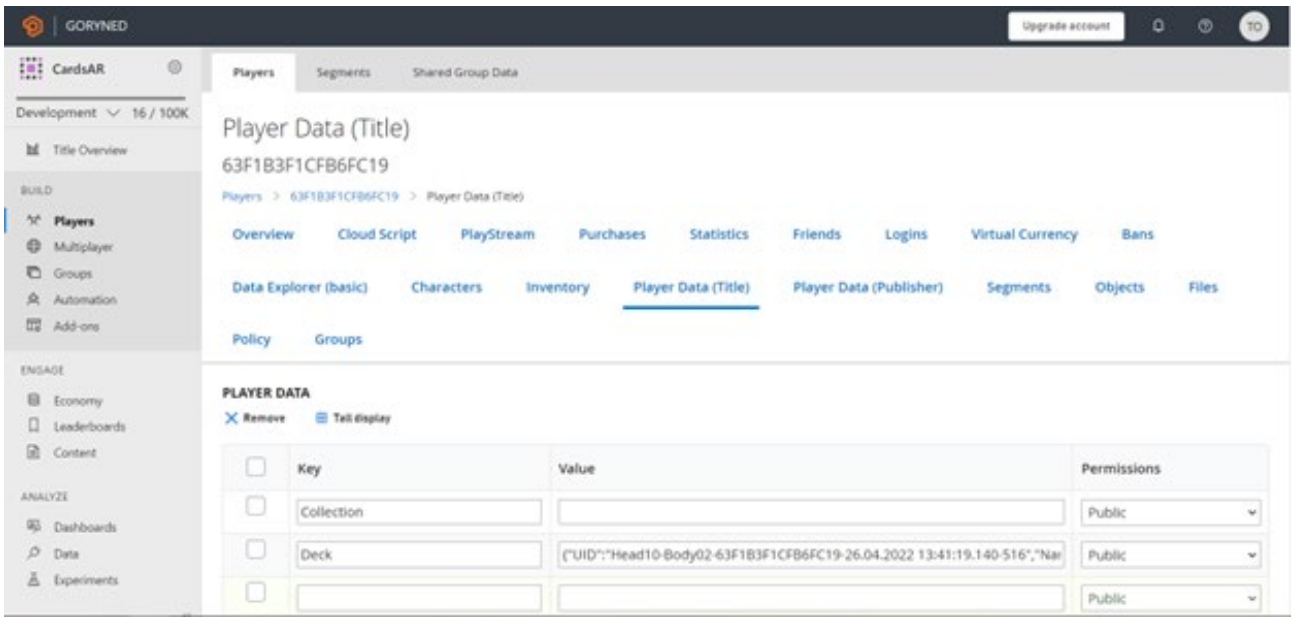


Рисунок 21. PlayFab – Player Data (Title)

Данная платформа отлично подходит для создания и хранения необходимых наборов данных по каждому пользователю, с возможностью взаимодействия с ними во всех направлениях. Для игроков есть возможность регистрации/авторизации как с помощью стандартных логина-пароля, так и через всевозможные социальные сети и не только. Помимо этого, PlayFab очень хорошо и эффективно работает с Unity, что очень важный фактор при разработке проекта.

1.4.4 Мультиплеер

Для разнообразия карточных баталий в проекте необходима реализация мультиплеера. По опыту работы над прошлыми проектами я решил безоговорочно использовать для этой цели Photon Unity Networking (PUN). Это реализация простого использования мультиплеера в Unity с производительностью и надежностью Photon Realtime. Схема его работы изображена на рисунке 22.

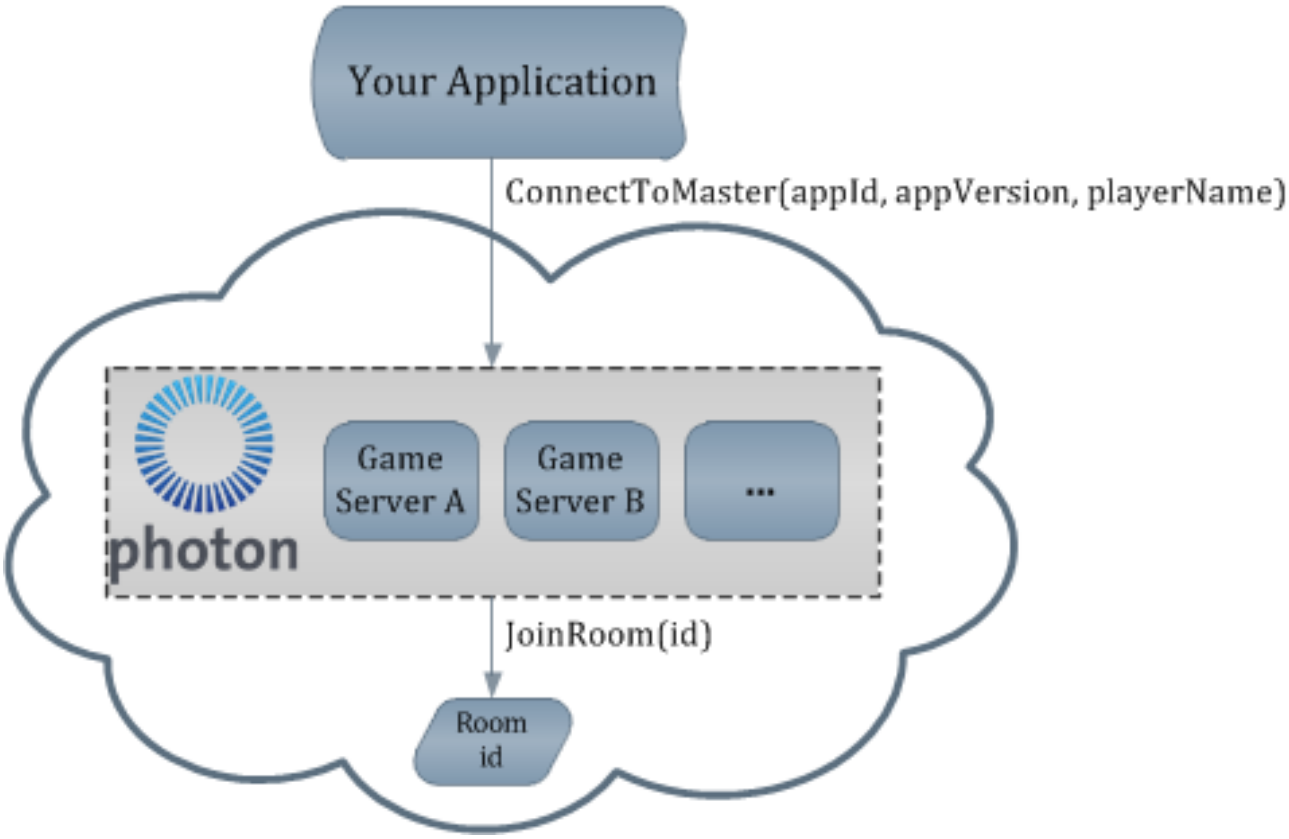


Рисунок 22. Photon Pun – схема работы

Особенности PUN, по информации с официального сайта [13]:

- Тесная интеграция с Unity;
- Хостинг находится в глобальном дистрибьютере Photon Cloud, гарантирующий низкую задержку и кратчайшее время приема-передачи данных для пользователей;
- Прочная основа для многопользовательских игр Unity любого типа – разработчик концентрируется на проекте, а с серверной частью платформа разберется сама;
- Является стандартным кроссплатформенным многопользовательским сервисом и №1 в мире для игр Unity;
- Игры, созданные с помощью PUN, плавно и автоматически масштабируются в Photon Cloud: от нескольких до десятков тысяч одновременных пользователей;

- Подбор соперников рандомно либо настраиваемым поиском через создание комнат с необходимыми параметрами;
- Большинство функций, которые необходимы для хорошего старта проекта, бесплатны, но расширение лимитов доступно по адекватным ценам.

1.5 Выводы

В этой главе было проделано следующее:

- Изучена история появления и развития карт и карточных игр с их дальнейшим развитием в цифровом формате. Выяснено почему они такое продолжительно время сохраняются свою популярность.
- Разобрана тема технологий дополненной реальностей с осознанием того, что за этим направлением большое будущее в связи с её применимостью практически во всех сферах жизнедеятельности человека.
- Проанализированы и выбраны технологии для разработки приложения: игровой движок Unity, язык программирования C#, ARFoundation (ARCore и ARKit) для дополненной реальности, PlayFab для хранения данных и Photon Pun для реализации мультиплеера. Основная целевая платформа игры – ОС Android в связи с простотой цифровой дистрибьюции.

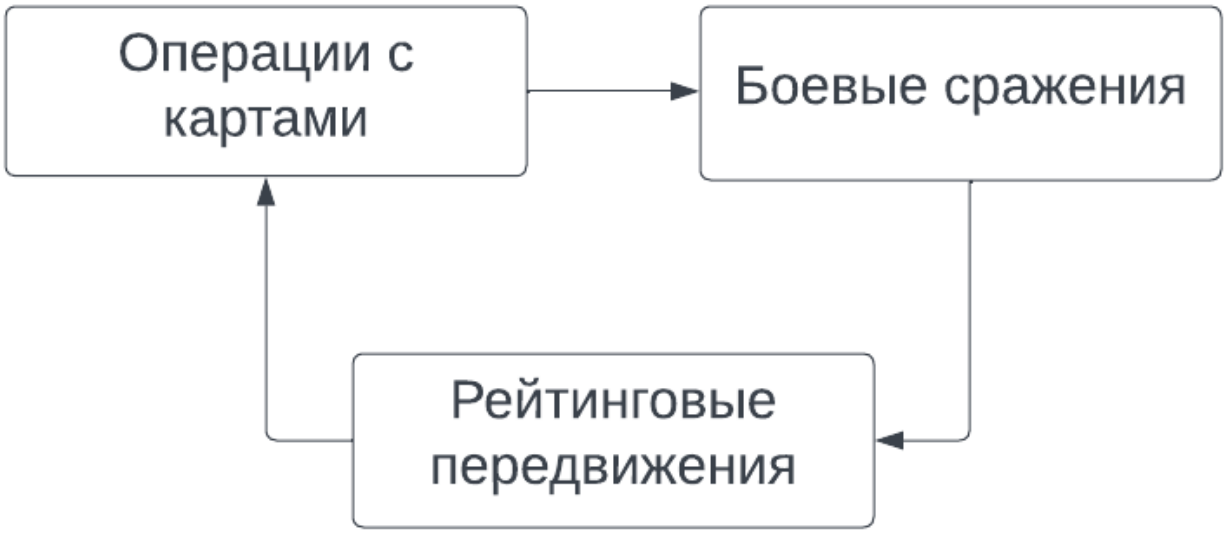
Глава 2. Проектно-конструкторская часть

2.1 Базовый геймплей

Дипломный проект заключается в создании мобильного игрового приложения с применением технологии дополненной реальности в виде коллекционной карточной игры. На текущий момент времени для снижения временных затрат на разработку игры будет урезанный функционал, так как есть понимание ненужности его с осознанием затрачиваемых времени и сил по опыту проекта-родителя, о котором говорилось ранее.

Соответственно, игра содержит в себе только основные необходимые механики: примитивная работа с картами (создание и удаление рандомных (случайных) карт в коллекции, их перемещение между ней и колодой) и реализованные функционалы мультиплеера и дополненной реальности. Отсутствует, но будет сделано в случае успеха, следующее: крафт, комбинирование и улучшение карт, силовые и финансовые характеристики игрока (валюта и рейтинг).

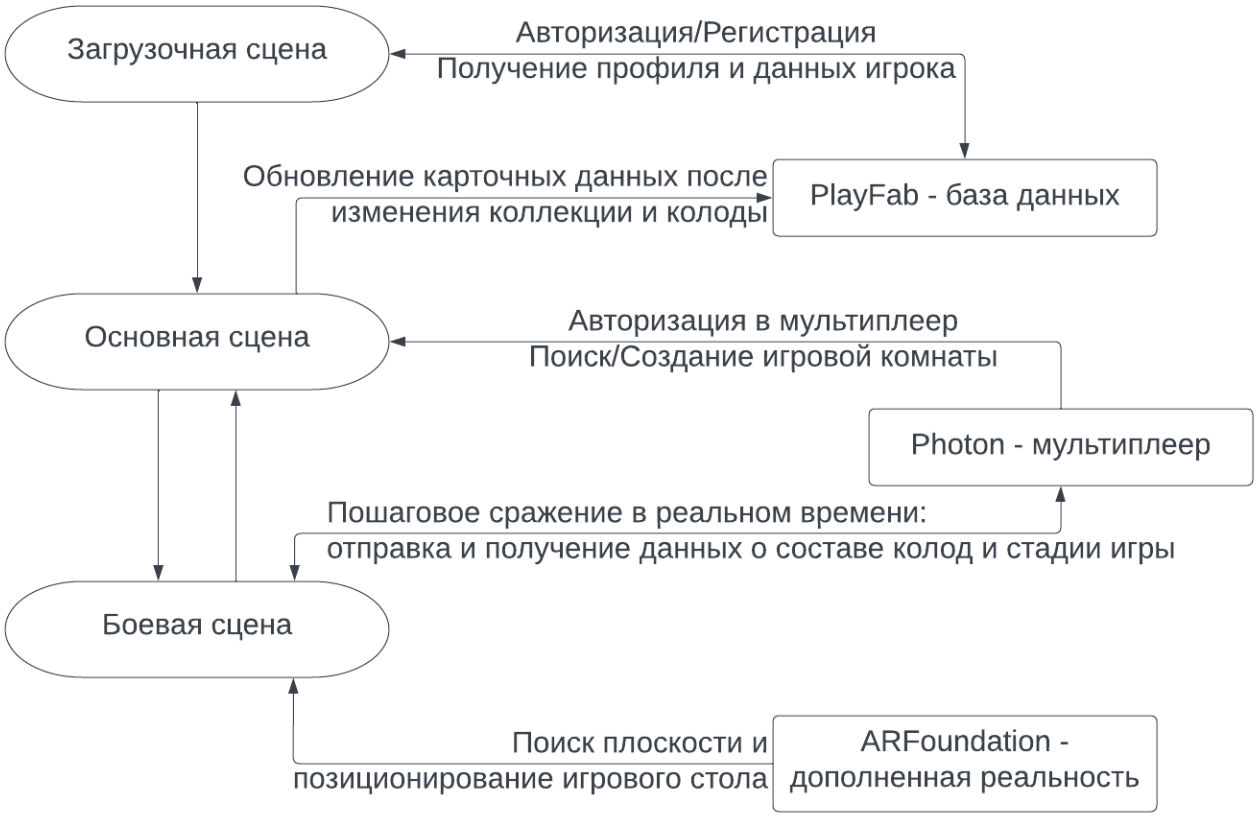
Собственно, игровой цикл проекта (блок-схема 1) заключается во взаимодействии с картами, сражении с игроками и повышение в рейтинговой таблице для получения различного рода наград. Это стандартный и вечно работающий цикл для всех игр - игроки хотят зарабатывать ресурсы, улучшать свои владения и соревноваться с другими пользователями. Но рейтинговая составляющая будет заблокирована по описанным выше причинам.



Блок-схема 1. Игровой цикл

2.2 Структура приложения

Для реализации описанного функционала требуется внедрение нескольких технологий. Основное – система хранения и управления данными игрока и игроков в целом. Далее – реализация мультиплеерной составляющей. Последнее – модуль дополненной реальности. Ниже в блок-схеме 2 представлена схема этих модулей и их взаимодействия с проектом.



Блок-схема 2. Схема взаимодействия модулей

При запуске открывается загрузочная сцена, в которой происходит авторизация/регистрация игрока в системе PlayFab, получение всех данных игрока (статистика и наборы карт) оттуда, предзагрузка следующей основной сцены.

Основная сцена, пока что, состоит из одного экрана со списками карт в коллекции и колоде. Карты из коллекции можно по удержанию перемещать в колоду, тем самым меняя выбранную и конечную карту местами. При этой операции идет запрос в PlayFab на обновление списков для корректной дальнейшей подзагрузки карточных наборов. Помимо этого, тут же располагается кнопка старта битвы, по нажатию на которую происходит авторизация в мультиплеерная среду и поиск/создание подходящей комнаты на 2 игрока. После успешного обнаружения соперника либо замены его на бота происходит синхронная для игроков загрузка боевой сцена.

Боевая сцена состоит так же из одного экрана, по краям которого располагаются боевые колоды. На каждый раунд дается определённое количество времени, отсчет которого начинается при готовности обоих игроков. Выбранная карта визуализируется в трехмерном пространстве на игровой стол и ожидает аналогичного появления другой от соперника. Всё боевое сражение происходит в анимированным форматом – присутствует анимация кулачного боя, выпрыгивающих из карт персонажей, с последующим изменением счетчика количества здоровья, учитывая параметры атаки. Переключение на формат дополненной реальности возможно в любой момент времени нахождения на боевой сцене – необходимо нажать соответствующую кнопку для переключения режима и ожидать обнаружения подходящей поверхности, на которую виртуальный игровой стол сразу переместится в зависимости от направления камеры мобильного устройства. Фиксация объекта происходит по нажатию иной кнопки в случае её активности (присутствует обнаруженная плоскость).

2.3 Установка и настройка необходимого ПО

Основным инструментом для создания проекта является игровая среда разработки Unity. Установить необходимую версию программы возможно через Unity Hub - автономное приложение, упрощающее поиск, загрузку и управление проектами и установками Unity. Данное программное обеспечение доступно для скачивания на официальном сайте [19]. После успешной установки необходимо запустить приложение, авторизоваться в аккаунт Unity, перейти во вкладку «Installs» (Установки) и нажать кнопку «Install Editor». Разработка проекта производилась в версии 2021.3.2f1 (LTS) с Android Build Support, в набор которого входит всё необходимое для успешной сборки проекта под мобильные устройства на базе операционной системы Android (рисунок 23).

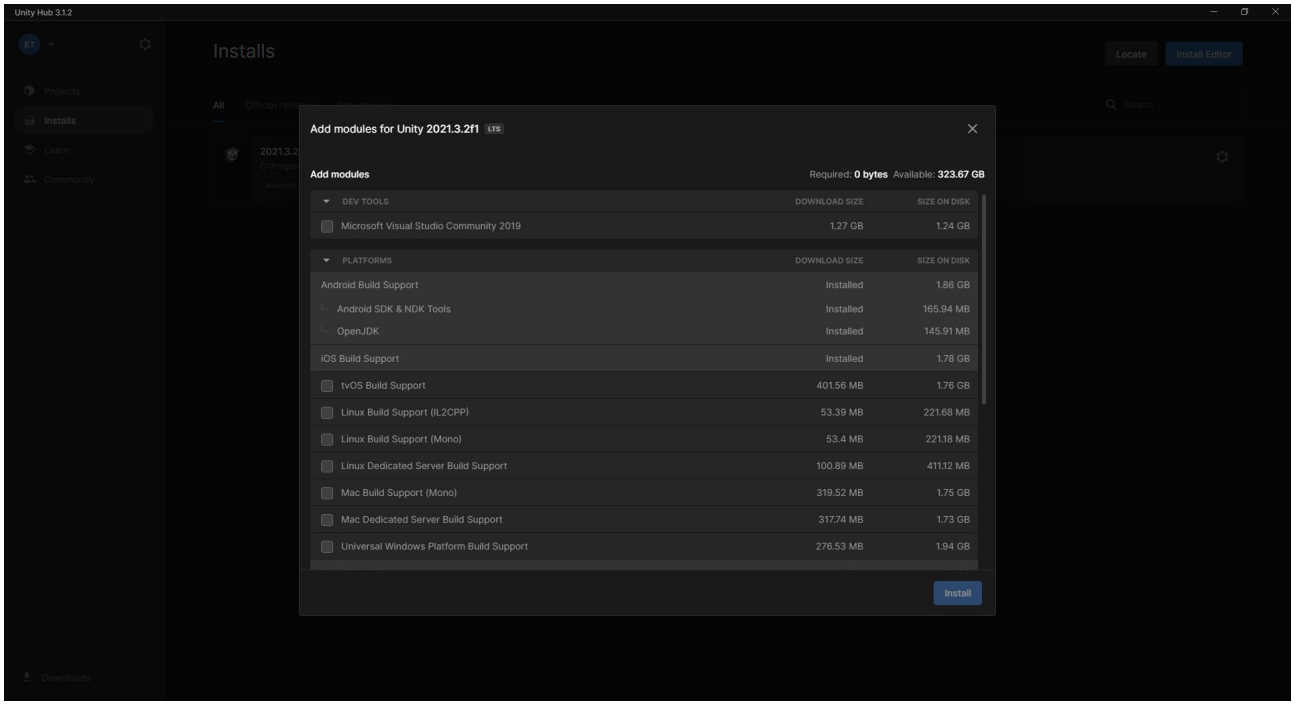


Рисунок 23. Установка Unity

После непосредственной установки необходимой версии среды разработки игр требуется создание проекта: Projects – New project – All templates – 2D. Таким образом будет произведен выбор шаблона двухмерного проекта со основными необходимыми пакетами для работы с пользовательским интерфейсом. Далее необходимо написать название проекта и выбрать его расположение (рис. 24).

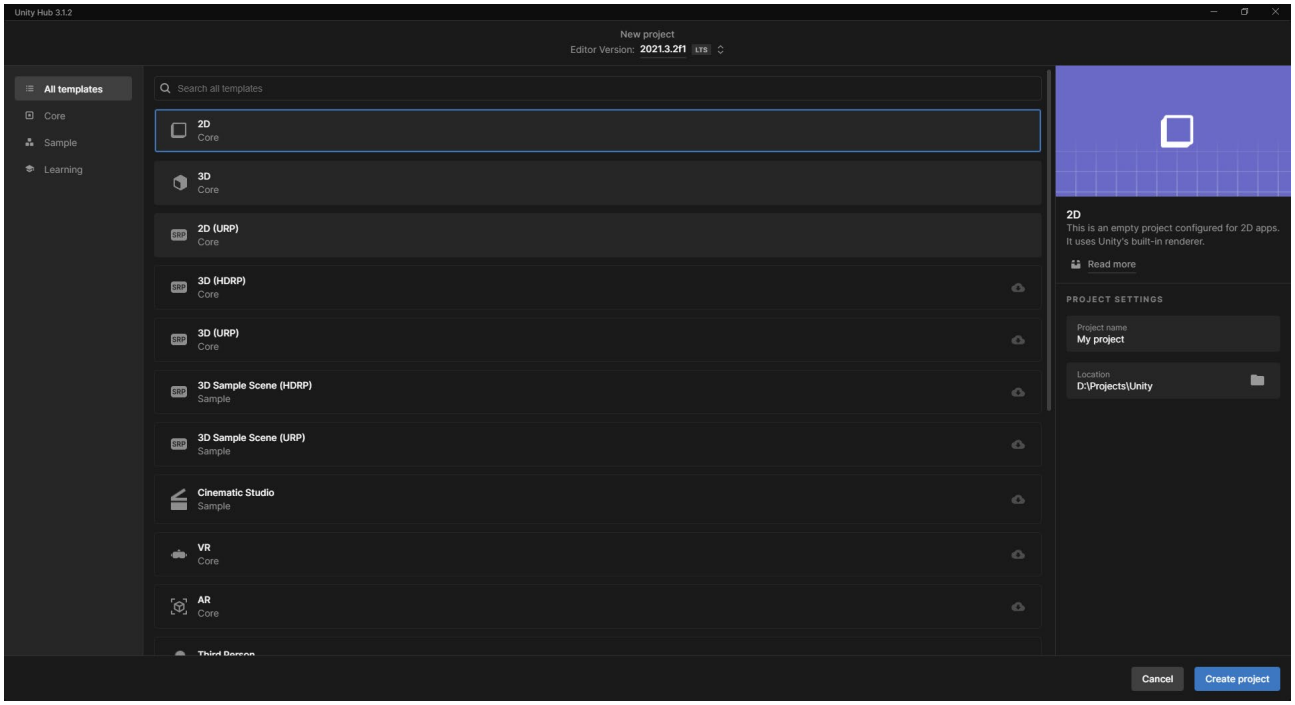


Рисунок 24. Создание проекта

При открытии созданного проекта будет видно главное окно редактора, состоящее из нескольких блоков, содержащих в себе тот или иной контент, необходимый при работе. Основные вкладки (рисунок 25):

- Scene – экран сцены, в которой производится визуальная настройка будущего проекта – расположение необходимых объектов, верстка пользовательского интерфейса;
- Game – экран конечного продукта, где можно увидеть то, как всё будет выглядеть для пользователя на устройстве;
- Project – библиотека всех элементов, находящихся в проекте – импортированные файлы, написанные скрипты, настроенные префабы и т.п.;
- Hierarchy – иерархия объектов на сцене – то, как объекты располагаются относительно друг друга;
- Inspector – детальное компонентное описание свойств выбранного объекта;

- Console – все логи проекта – ошибки, предупреждения и сообщения в ходе работы.

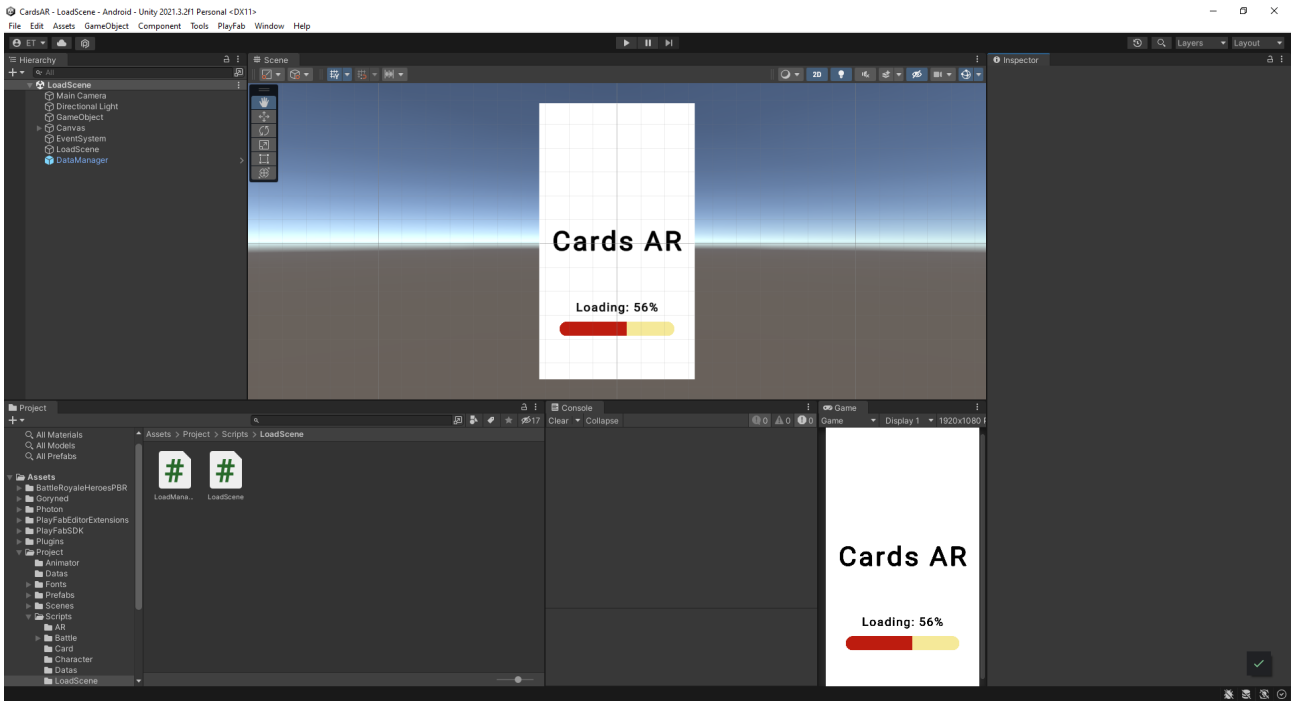


Рисунок 25. Окно Unity

Для реализации всех описанных раннее технологий необходимо их импортировать в проект. Сделать это можно через перетаскивание .unitypackage в окно Project либо же прямое скачивание посредством Package Manager (рисунок 26), расположенного в верхней панели во вкладке Window. Для комфортной дальнейшей разработки необходимо скачать с официальных сайтов (либо Unity Asset Store), установить и настроить, согласно внутренним документациям, следующие плагины: ARFoundation (работа с AR), PlayFab (работа с базами данных) и Photon PUN (работа с мультиплеером).

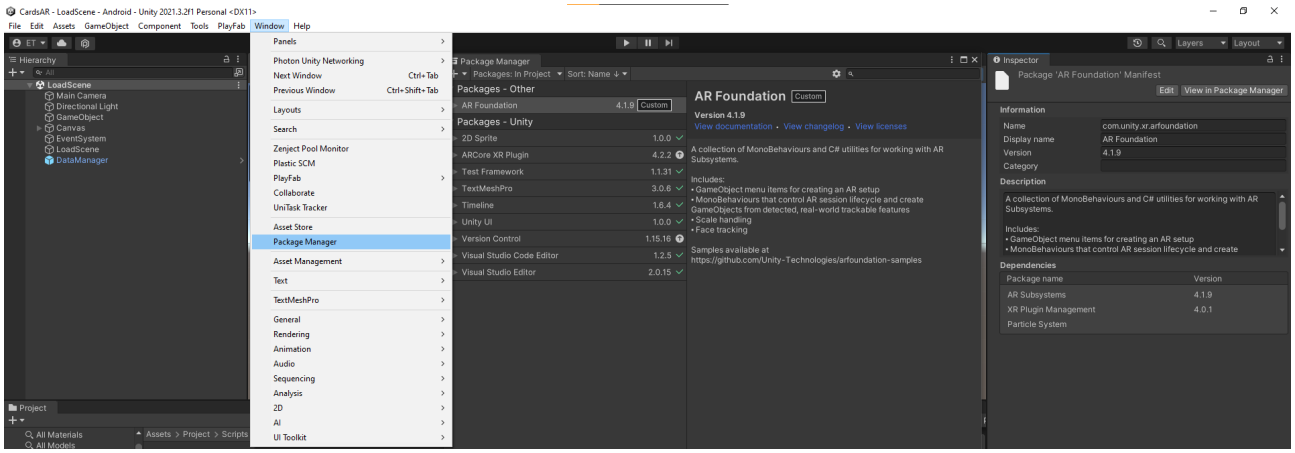


Рисунок 26. Package Manager

2.4 Верстка и программирование интерфейса

Любое программное обеспечение не может существовать без пользовательского интерфейса, через который пользователь взаимодействует с программой. Популярным средством для предварительной верстки UI является сервис Figma (рисунок 27), в котором профессиональные дизайнеры подготавливают визуализацию для упрощения разработчикам работы с его применением. Так как дизайнер я не профессиональный, я лишь примерно сверстал расположение интерфейса, чтобы проще было в Unity перенести этот первичный визуальный вид. В дальнейшем, само собой, интерфейс будет меняться в зависимости от хода разработки, во время которой появляются новые идеи реализации какого-либо функционала.

Рисунок 27. Верстка UI в Figma

Верстка в Unity происходит посредством шаблонных объектов Unity UI. Основным объектов интерфейса является Canvas (Рисунок 28), внутри которого находятся все дочерние элементы – панели, тексты, кнопки и т.д. Для того, чтобы интерфейс не «убегал» на разных разрешениях и соотношениях экранов используются различные методы в компоненте Canvas Scaler (привязка расширения по высоте/ширине изначального разрешения верстки) и грамотные привязки элементов в зависимости от иерархичности каждого.

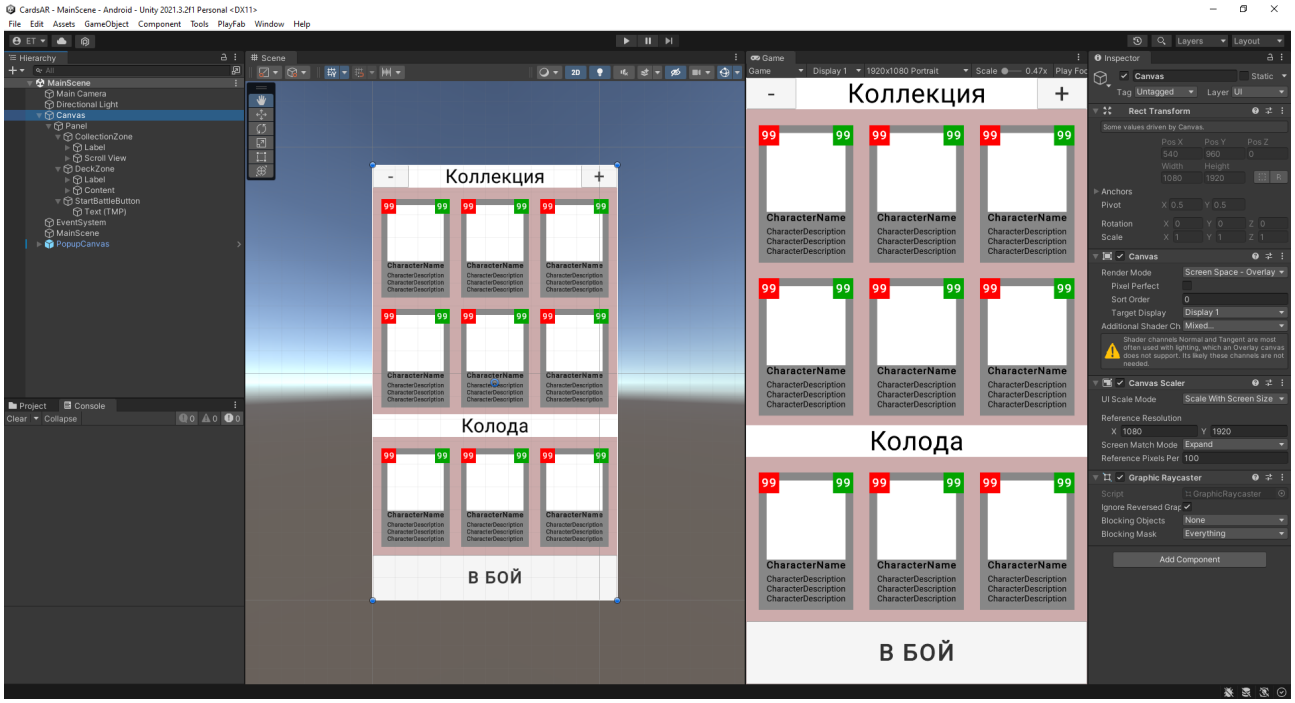


Рисунок 28. Верстка UI в Unity

Итоговый пользовательский интерфейс был сверстан в примитивном виде касаемо графических материалов, так как не имеется навыков в сфере рисования и графического дизайна для создания чего-то масштабного. Но расположение элементов является полной копией того, как это было реализовано в проекте-родителе, о котором говорилось в предыдущей главе («Argy Bargy: Craft»). Был совершен данный шаг в связи с тем, что данное расположение элементов было хорошо воспринято на выставке DevGAMM Moscow 2021, на которой я представлял тот проект. Цветовая палитра же подобрана таким образом, чтобы яркими цветами выделять важные элементы, а остальное оставлять на обычном уровне. Окончательный пользовательский интерфейс предоставлен на рисунке 29.

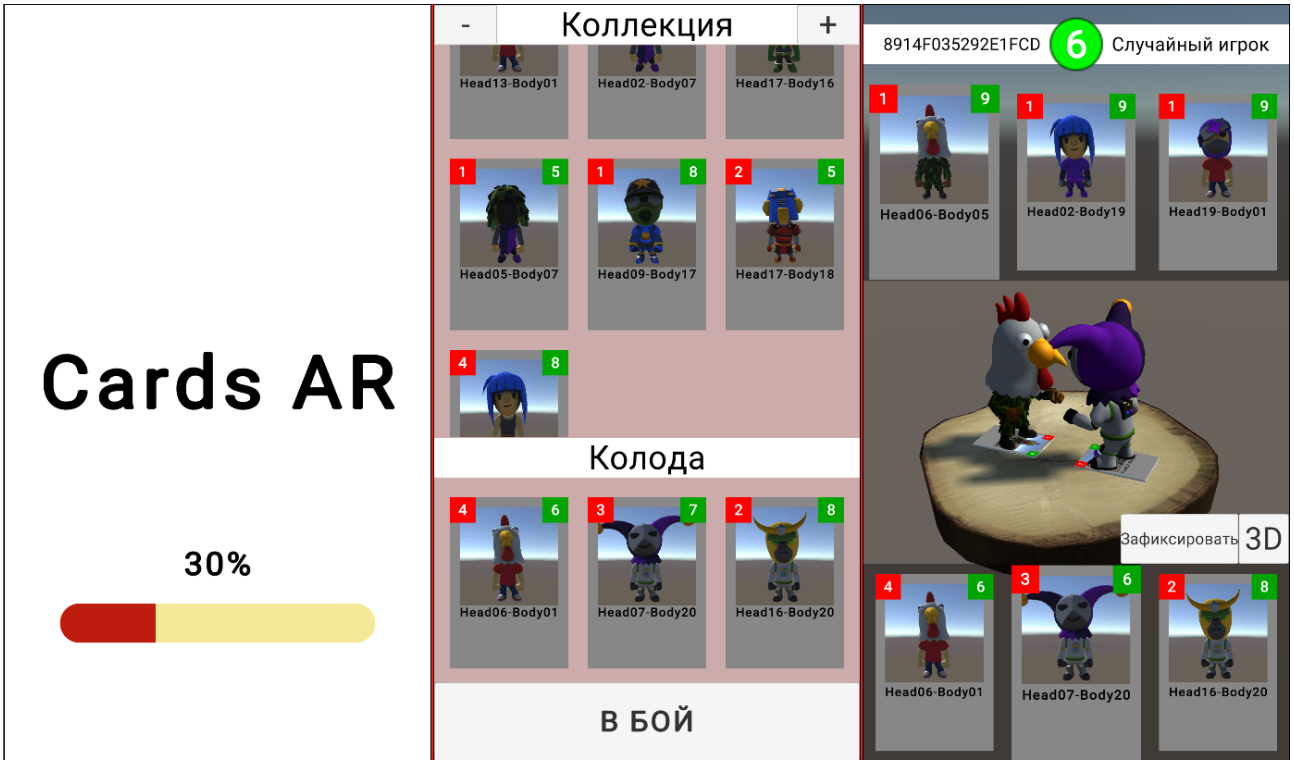


Рисунок 29. Окончательный UI проекта

2.5 Ход разработки проекта по сценам

Разрабатываемый проект строится на базе 3 сцен, где на каждую сцену приходится по одному основному канвасу (экрану), но могут быть еще и дополнительные элементы (например, всплывающие окна).

2.5.1 Загрузочная сцена

Загрузочная сцена является первой, на которую попадает пользователь при старте приложения. Соответственно, называется она «LoadScene». На ней происходят все операции касаемо авторизации/регистрации пользователя и получение/обновление его данных, а также загрузка следующей основной сцены.

Представленный ниже метод (листинг 1) запускается при инициализации объекта (старт загрузочной сцены) и продолжается до выполнения всех операций, так как является асинхронным. Тут происходит старт предзагрузки следующей сцены и обращение в следующий метод по работе с PlayFab. Когда

все запрошенные операции переходят в статус выполненных, то разрешается открытие следующей сцены. Список операций пополняется во время старта новой и переводится в совершенный статус после окончания – таким образом реализуется ожидание всех запущенных действий.

Листинг 1. Метод загрузки сцены и данных

```
public async void StartLoading()
    {
        float loadTime = 5f;
        Goryned.Load.LoadManager.Reset();
        Goryned.Scene.SceneManager.LoadScene("MainScene", loadTime, false);
        Login();

        await UniTask.Delay(TimeSpan.FromSeconds(loadTime));
        await UniTask.WaitUntil(() => Goryned.Load.LoadManager.IsAllComplete());
        Goryned.Scene.SceneManager.AllowSceneActivation();
    }
```

Вышеупомянутый метод по работе с PlayFab приводится далее (листинг 2). Он отвечает за авторизацию/регистрацию пользователя, а также получение и синхронизацию всех данных игрока – профиль, статистика и наборы карт. Аутентификация производится посредством уникального идентификационного номера устройства, который является некими логином и паролем к профилю пользователя. После того, как удалось получить уникальный идентификатор пользователя PlayFab, то происходят три запроса – получение профиля (например, игровое имя), статистики (валюта, рейтинг) и карты (коллекция и колода). При старте и окончании этих и других операций, как говорилось выше, происходит создание и закрытие статусов, по которым отслеживается сигнал для старта открытия следующей сцены.

Листинг 2. Авторизация и загрузка данных в PlayFab

```
public async void Login()
    {
        Goryned.Load.LoadManager.AddStep("LoginPlayFab");
        PlayFabManager.LoginPlayFab();
        await UniTask.WaitUntil(() => PlayFabTempData.loginResult != null);

        LoginResult loginResult = PlayFabTempData.loginResult;
        PlayFabTempData.loginResult = null;

        PlayerDataManager.GetPlayFabProfile(loginResult.PlayFabId);
        PlayerDataManager.GetPlayerStatistics(loginResult.PlayFabId);
        PlayerDataManager.GetUserCards(loginResult.PlayFabId);

        Goryned.Load.LoadManager.CompleteStep("LoginPlayFab");
    }
```

Внутри каждого из трех запросов к PlayerDataManager.cs происходят запросы в PlayFab API на получение определенного вида данных. Данные приходят в сыром формате в виде словарей данных (строка-число либо строка-строка в зависимости от типа), которые далее форматируются в комфортные для дальнейшей работы форматы. Например, значения словаря карточных данных состоят из перечисления структур CardData в виде строки, которые необходимо потом парсить в этот тип.

В случае отсутствия необходимых данных на сервере (например, обнуление аккаунта или создание нового) происходит отправка туда и

подстановка на месте заранее настроенных дефолтных (стандартных) наборов данных.

2.5.2 Основная сцена

После того, как все операции с получение данным произведены, открывается данная сцена, называя MainScene (основная/главная сцена), так как она является той, на которую пользователь попадает больше всего за игровую сессию. На ней располагается скрипт MainScene.cs, отвечающий за взаимодействие пользователя с контентом сцены.

В текущей версии приложения, описанной ранее, данная сцена содержит в себе только отображение наборов карт (коллекция и колода) и стартовую кнопку для битвы, поэтому функционал скрипта заключается в работе по выводу и перемещению карт из коллекции в колоду. Данный функционал реализуется с помощью использования на сцене двух объектов с скриптом-компонентом CardsZoneFiller.cs, благодаря которому можно перезаполнять и дополнять/удалять элементы типа Card.cs (компонент карты, работающий со структурой CardData). Ниже приведен метод по полному перезаполненнию объектов.

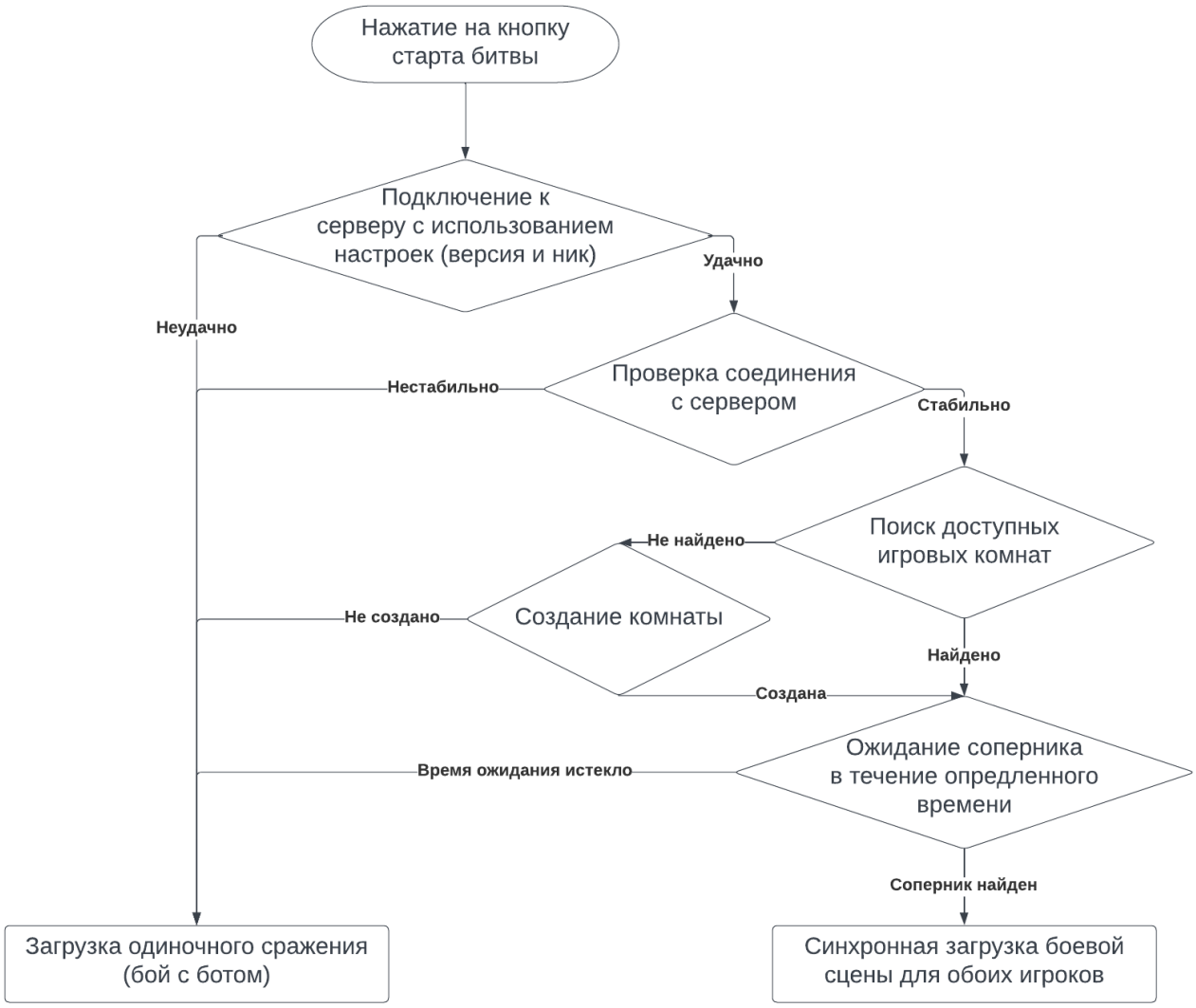
Данный компонент поддерживает принятие параметра, отвечающего за тип создаваемой карты – обычная (без каких-либо дополнительных компонентов), перемещаемая (с возможностью тащить объект при удержании его в зажатом состоянии) и боевая (для функционала боевой сцены).

Листинг 3. Заполнение набора карт данными

```
public void Fill(List<CardData> cardDatas, CardsZoneFillerType type = CardsZoneFillerType. Simple)
   {
      GameObject cardObject = DataManager.Instance.CommonDatas.Card2D;
      Goryned.Object.SpawnManager.ClearChildren(cardsHolder);
      foreach (var cardData in cardDatas)
      {
         Card card = Goryned.Object.SpawnManager.Spawn(cardObject, cardsHolder).GetComponent<Card>();
         card.transform.localScale = Vector3.one;
         card.SetCardData(cardData);

         if (type == CardsZoneFillerType.Moveable) card.gameObject.AddComponent<MoveableCard>();
         if (type == CardsZoneFillerType.Battleable) card.gameObject.AddComponent<BattleableCard>();
      }
   }
```

Вторым важным функционалом сцены является работа с запуском мультиплеера – это происходит по нажатию соответствующей кнопки запуска битвы. Здесь происходит запрос к API методам Photon Network, которые помогают подключиться/отключиться от сервера, искать/создавать комнаты и ожидать соперника. Схема работы на блок-схеме 3.



Блок-схема 3. Подключение к мультиплееру

2.5.3 Боевая сцена

После завершения всех операций, связанных с подключением к серверу, вне зависимости от результата (подключился ли к серверу, нашлась/создалась ли комната, нашелся ли соперник) происходит запуск боевой сцены (BattleScene), где как раз-таки идет проверка наличия соперника (либо реальный игрок, либо бот).

Исходя из этой булевой переменной система делает решение касаемо отправки данных – в данный момент это реализовано так, что при отсутствии реального соперника вызываются сразу коллбеки, минуя вызов через

специализированные запросы PunRPC. RPC (Remote Procedure Calls) – удаленные вызовы процедур, одна из особенностей, которая отличает PUN от других пакетов Photon.

Об использовании данной системы речь пойдет позднее, так как для начала нужно описать то, что происходит при запуске сцены. Основным скриптом данной сцены является BattleManager.cs, который занимается управлением всего, что происходит.

Первым этапом битвы является её инициализация (листинг 4). Тут происходит заполнение контейнеров данных, содержащих набор действующих карт. Для пользователя дублируются его колодные карты в боевую колоду, а для противника, временно и на случай отсутствия реального соперника создается случайный набор карт в боевую колоду, аналогично с их созданием при добавлении в коллекцию пользователя на главной сцене.

Листинг 4. Инициализация битвы

```
public void InitializeBattle()
   {
      if (DataManager.Instance.PlayerDatas.UserPlayerData.Cards == null || !DataManager.Instance.PlayerDatas.UserPlayerData.Cards.ContainsKey(CardListType.Deck))
      {
         DataManager.Instance.PlayerDatas.UserPlayerData.Cards = new Dictionary<CardListType, List<CardData>>();

DataManager.Instance.PlayerDatas.UserPlayerData.Cards.Add(CardListType.Deck, CardDataManager.GetRandomCardDatas(3, string.Empty));
      }
```

```
        userBattleData.CardDatas = DataManager.Instance.PlayerDatas.UserPlayerData.Cards[CardListType.Deck];

        enemyBattleData.CardDatas = CardDataManager.GetRandomCardDatas(3, string.Empty);
        DataManager.Instance.PlayerDatas.EnemyPlayerData.Cards = new Dictionary<CardListType, List<CardData>>();

DataManager.Instance.PlayerDatas.EnemyPlayerData.Cards.Add(CardListType.Deck, enemyBattleData.CardDatas);

        NextRound();
    }
```

После завершения этих быстрых процессов стартует первый раунд. В нем происходит сброса счетчика готовых игроков, необходимого для определения того, когда и кто выбрал карту для старта сражения непосредственно. Следующий метод отправляет циклические запросы реальному противнику, в случае его наличия, на предмет его готовности к сражению, так как бывают небольшие задержки в интернет-соединении. Так же это необходимо, чтобы удостовериться, что следующий запрос касаемо отправки боевых данных (набор действующих боевых карт) был успешно получен противником = избежание рассинхронизации. Следующая проверка идет касаемо того, что в процессе всех операций не отключился ли противник. Эта проверка так же заполняет информацию вверху экрана (никнеймы пользователя и противника). Далее идут 3 шага касаемо визуальной информации – респавн (перезаполнение) контейнеров боевых колод, состоящих в 2D визуализации карт в интерфейсе; очистка трехмерного игрового стола, включающая сброс анимаций и скрытие 3D

представления карт; запуск таймера на выбор боевой карты, если по истечению отсчета карта была не выбрана, то выбирается случайная автоматически.

Листинг 5. Метод следующего раунда

```
public async void NextRound()
    {
        BattleHelper.ReadyCount = 0;
        await BattleHelper.SendReady();
        await BattleHelper.SendBattleData();
        CheckUsers();

        RespawnBattleDecks();
        cardTable.ResetCards();
        StartCoroutine(TimerCounter());
    }
```

На данном моменте рассказа хода битвы можно сделать небольшой перерыв, чтобы рассказать о вышеупомянутых Photon Pun RPC методах. Данные методы необходимы для организации отправки необходимой информации между игроками, находящимися внутри одной комнаты. Отправка реализуется через компонент PhotonView и его метод RPC (рисунок 30), принимающий в себя имя метода обратной связи, тип получателей и массив отправляемых данных, которые принимает коллбек.

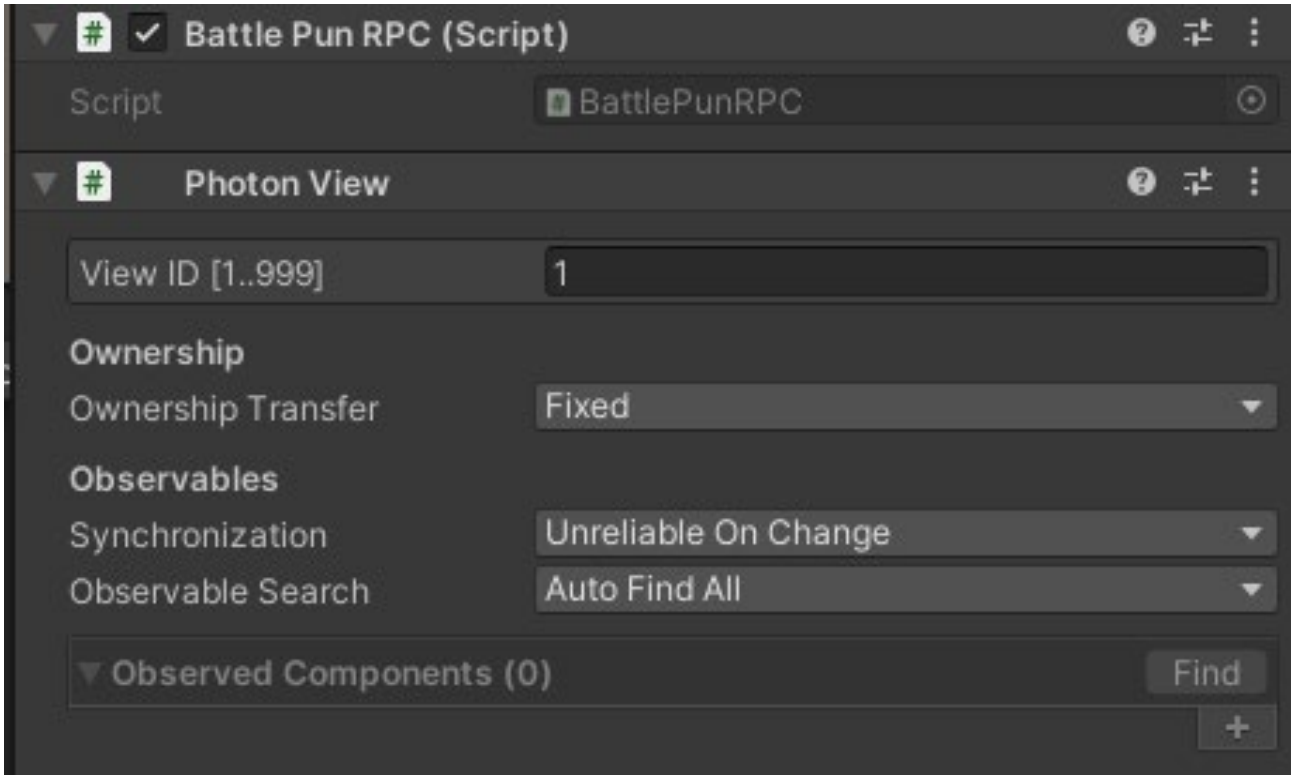


Рисунок 30. Компоненты BattlePunRPC и PhotonView

Для простоты работы с отправками всей информации мною были написаны несколько скриптов-посредников, которые помогают упростить эти вызовы. Изначально, запрос идет в BattleHelper.cs, который содержит в себе различные проверки (наличие противника и его готовность) и базовые методы отправки данных, обращающиеся в специализированный класс, который размещен на сцене в виде компонента – BattlePunRPC.cs. Ниже приведен пример отправки боевых данных игрока.

Листинг 6. Отправка боевых данных сопернику

```
public async static UniTask SendBattleData(){
        OpponentReady = false;
        BattleData battleData = BattleManager.Instance.userBattleData;
        BattlePunRPC.instance.Call(BattlePunRPC.instance.SendBattleData,
RpcTarget.Others, JsonUtility.ToJson(battleData));
        await UniTask.WaitUntil(() => OpponentReady);    }
```

По методу видно, что происходит отправка запроса в BattlePunRPC.cs, где происходит проверка на наличие соперника и принятие решения о способе отправки данных – через сервер с помощью RPC либо локально сразу в обратный метод. Кастомный метод продемонстрирован в листинге 7.

Листинг 7. Вызов Pun RPC

```
public void Call(Action<string> action, RpcTarget rpcTarget, string data)
    {
        Debug.Log("PhotonRPC - " + action.Method.Name);
        if (BattleHelper.WithPlayer)
        {
            PhotonView photonView = PhotonView.Get(this);
            photonView.RPC(action.Method.Name, rpcTarget, data);
        }
        else
        {
            if (action == SendBattleData)
            {
                BattleHelper.OpponentReady = true;
                return;
            }
            action(data);
        }
    }
```

Возвращаемся к описанию хода битвы. После успешной отработки асинхронного NextRound() из BattleManager.cs начинается ожидание от интерфейса на тему выбора игроком карты, с помощью которой будет проходить

раунд. Если игрок не укладывается вовремя, то карта выбирается автоматически случайным образом.

После выбора того, с чем пользователь будет в этом раунде, идет ожидание аналогичного выбора от соперника. Если сражение идет с ботом, либо же карта соперника была определена раньше, то это ожидание мгновенное. Если на момент выбора соперник еще не определился, то появляется попап (всплывающие уведомление), информирующее об этом.

Листинг 8. Выбор боевой карты

```
private void CardSelected(bool isUser, CardData selectedCardData)
    {
        StopAllCoroutines();
        Debug.Log(CardDataManager.ParseCardData(selectedCardData));
        cardTable.MoveToTable(isUser, selectedCardData);
        if (isUser) userBattleData.CardSelected(selectedCardData);
        else enemyBattleData.CardSelected(selectedCardData);
        cardTable.Fighting();
        if (isUser)
BattlePunRPC.instance.Call(BattlePunRPC.instance.IncreaseReadyCount,
RpcTarget.All, null);
    }
```

Описанный и приведенный выше метод останавливает все запущенные корутины (сопрограммы; например, таймер на выбор карты), визуализацию перемещение карты на стартовое положение на столе и информирование противника о готовности к бою. Данный функционал реализуется через количество готовых игроков к бою (листинг 9):

Листинг 9. Количество готовых игроков

```
public static int ReadyCount
    {
        get => readyCount;
        set {
            readyCount = value;
            if (value > 0)
            {
                Debug.Log("ReadyCount: " + value + "; WithPlayer: " + WithPlayer);
                if (value == 1 && !WithPlayer)
                {
                    CardData                                cardData                                =
BattleManager.Instance.enemyBattleData.GetRandomCardData();

BattlePunRPC.instance.Call(BattlePunRPC.instance.SendSelectedCardData,
RpcTarget.Others, JsonUtility.ToJson(cardData));
                }
                if (value == 2 && WithPlayer)
                {
                    CardData                                cardData                                =
BattleManager.Instance.userBattleData.SelectedCard.GetCardData();

BattlePunRPC.instance.Call(BattlePunRPC.instance.SendSelectedCardData,
RpcTarget.Others, JsonUtility.ToJson(cardData));
                }
            }
        }
    }
```

После готовности обоих игроков к сражению начинает своё функционирование CardTable.cs, отвечающий за визуализацию происходящего на игровом столе. Методы внутри данного класса занимаются перемещением выбранных карт на стол, а также запуск и управление анимированным сражением с отправлением запросов на совершение различного рода действий (отнятие хп (здоровья) у карт в соответствии с количеством атаки у противоположной). Анимация сражения происходит за счет использования Unity Animation у полученных из Unity Asset Store моделей различных персонажей с уже готовыми сценариями анимаций. Мною были лишь отобраны необходимые модели и перестроены аниматоры по необходимым мне целям. Реализуется запуск процесса через компоненты 3D визуализации карты (листинг 10).

Листинг 10. Запуск карты в бой

```
public async void Fight()
    {
        CardStateType = CardStateType.OnFight;
        AnimationClip idleClip = character.animator.GetCurrentAnimatorClipInfo(0)[0].clip;
        character.animator.SetTrigger("Fight");
        await UniTask.WaitUntil(() => character.animator.GetCurrentAnimatorClipInfo(0)[0].clip == idleClip);
        CardStateType = CardStateType.CompleteFight;
    }
```

Завершение раунда контролируется из BattleManager посредством OnCompleteFight(), проверяющем состояние боевой колоды:

- Следующий раунд – в колоде каждого игрока остались живые карты (те карты, у которых здоровье больше 0);
- Победа – только у пользователя остались живые карты;

- Поражение – только у соперника остались живые карты;
- Ничья – у обоих игроков пустые колоды.

В первом случае – снова запускается NextRound(), а в иных случаях вызывается всплывающие уведомление с итогами сражения (победа, ничья, поражение) и производится возврат на основную сцену. На этом игровой цикл текущей версии проекта заканчивается.

2.6 Сборка проекта

Перед тем, как провести тестирование финальной версии игры и произвести цифровую дистрибьюцию, необходимо собрать проект под необходимую платформу. В данном случае целевой платформой является Android, но написанное приложение можно собрать под любую платформу – практически безошибочно это получится сделать для PC (Windows) и Mobile (Android, iOS).

Компиляция приложения под платформу Android для дальнейшей публикации в Google Play Market делается благодаря внутренним системам Unity, где можно настроить все названия пакетов/версий, включить нужные параметры и т.д. Для того, чтобы попасть в данное окно, необходимо пройти по пути «File > Build Settings» (рисунок 31), выбрать сцены, которые будут включены в сборку и переключиться на необходимую платформу Android (она будет доступа в случае наличия модулей Android SDK, NDK и JDK устанавливаемых в версию Unity через Unity Hub).

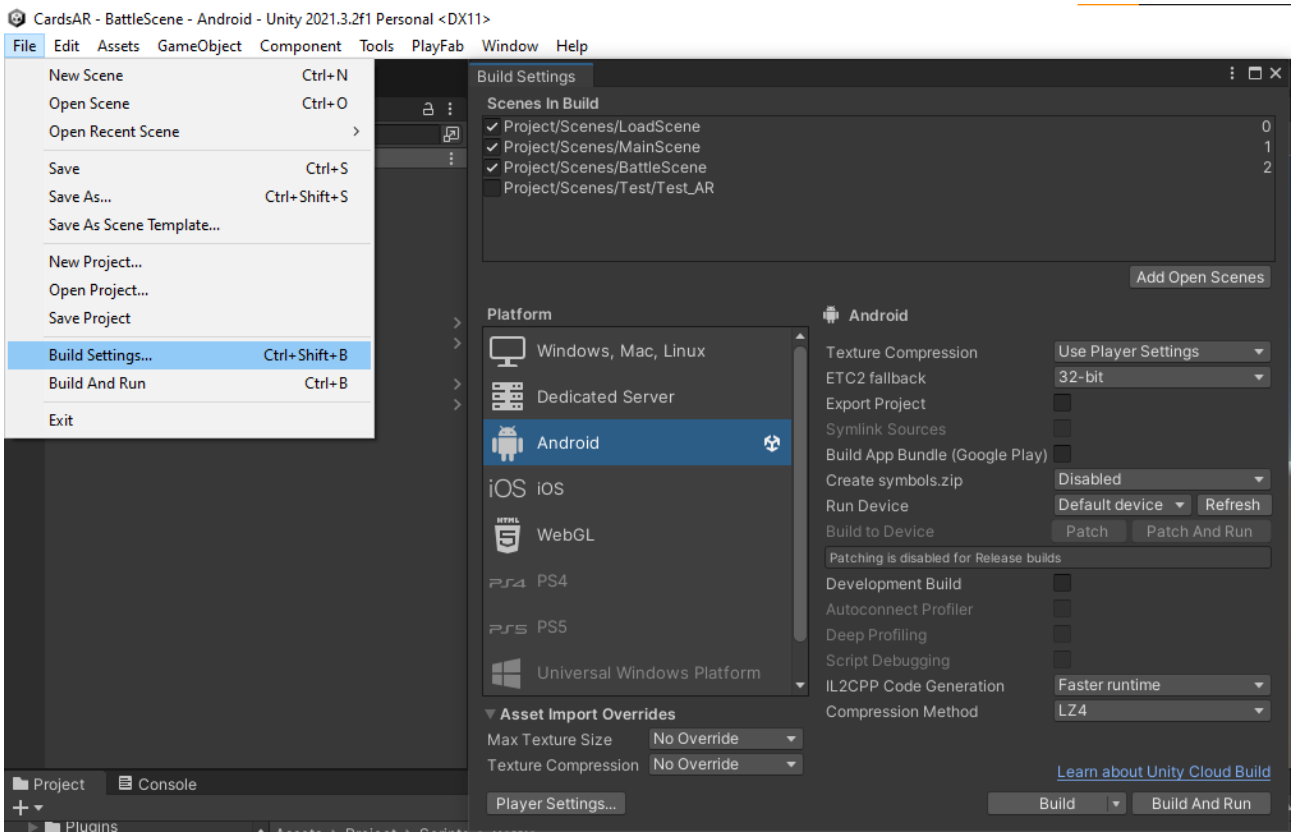


Рисунок 31. Build Settings

Более обширные настройки находятся по кнопке «Player Settings» (рисунок 32) - там производится настройка билда (сборки) по 5 направлениям – визуальная часть (Icon), разрешение и ориентация приложения (Resolution and Presentation), экран запуска (Splash Image), технические настройки (Other Settings), публикация (Publishing Settings). Основные важные параметры находятся в последних двух вкладках – например, первая содержит в себе выбор диапазона API уровней Android и настройку версии, а вторая – ключ публикации для предотвращения возможности обновления приложения без его наличия.

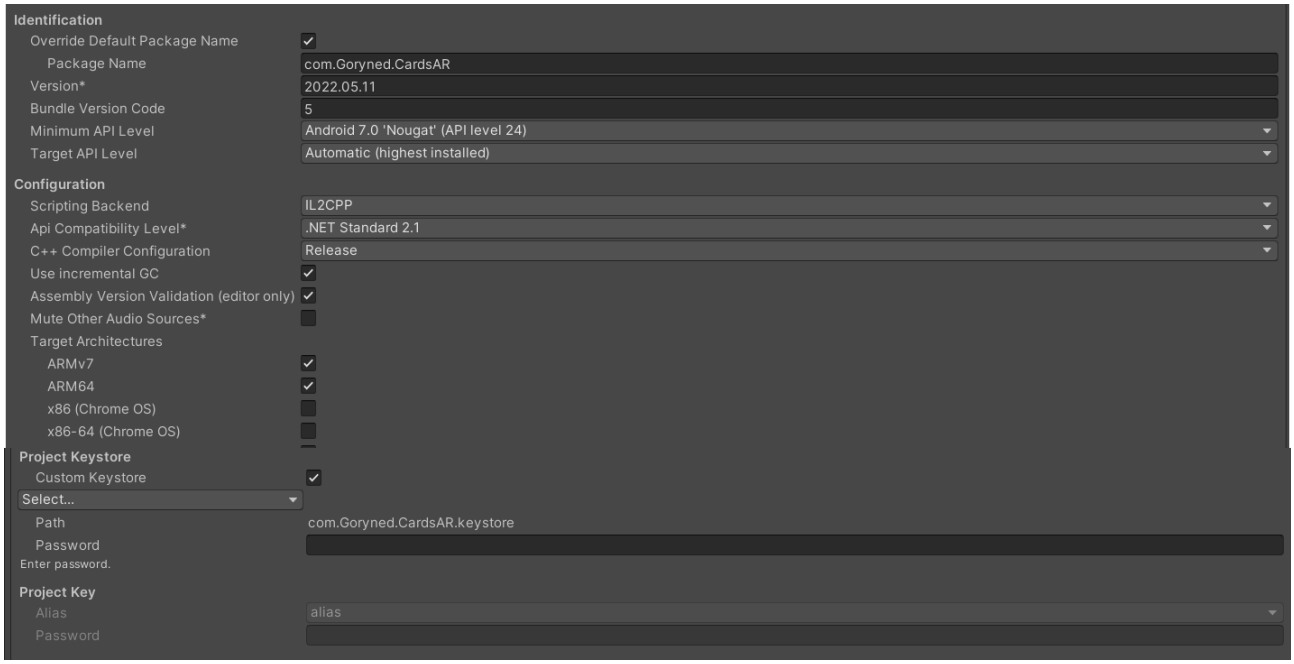


Рисунок 32. Player Settings

После корректной настройки всех параметров можно начинать сборку путем нажатия кнопки Build в изначальном окне. Обычно первая сборка занимает продолжительное время в связи с отсутствием в буфере проекта Unity необходимых файлов, но последующие компиляции происходят моментально.

2.7 Тестирование и цифровая дистрибьюция

Перед полноценным релизом игрового проекта необходимо провести тестирование проекта на целевой аудитории. Раньше для этого требовалось, в основном, рассылать APK сборку по клиентам лично либо посредством публикации в облачное хранилище, но в данный момент это можно реализоваться путем закрытых и открытых тестирований через Google Play Console в Google Play Market без необходимости предоставления «голого» устанавливаемого файла.

Для создания проекта в Google Play Console необходимо наличие подписки разработчика Google Play, которая оформляется на официальном сайте консоли. После успешной регистрации себя либо студии как разработчика открывается

доступ к необходимому для размещения приложения функционалу. Чтобы начать ведение проекта, необходимо нажать «Создать приложение» (рисунок 33), ввести запрашиваемую информацию (название приложения, язык, тип), согласиться с условиями и продолжить. Дальнейшие шаги совершаются путем специального чек-листа на главной странице созданного проекта – например, настройка политики конфиденциальности, выбор целевой аудитории, установка текстовой и медиа информации.

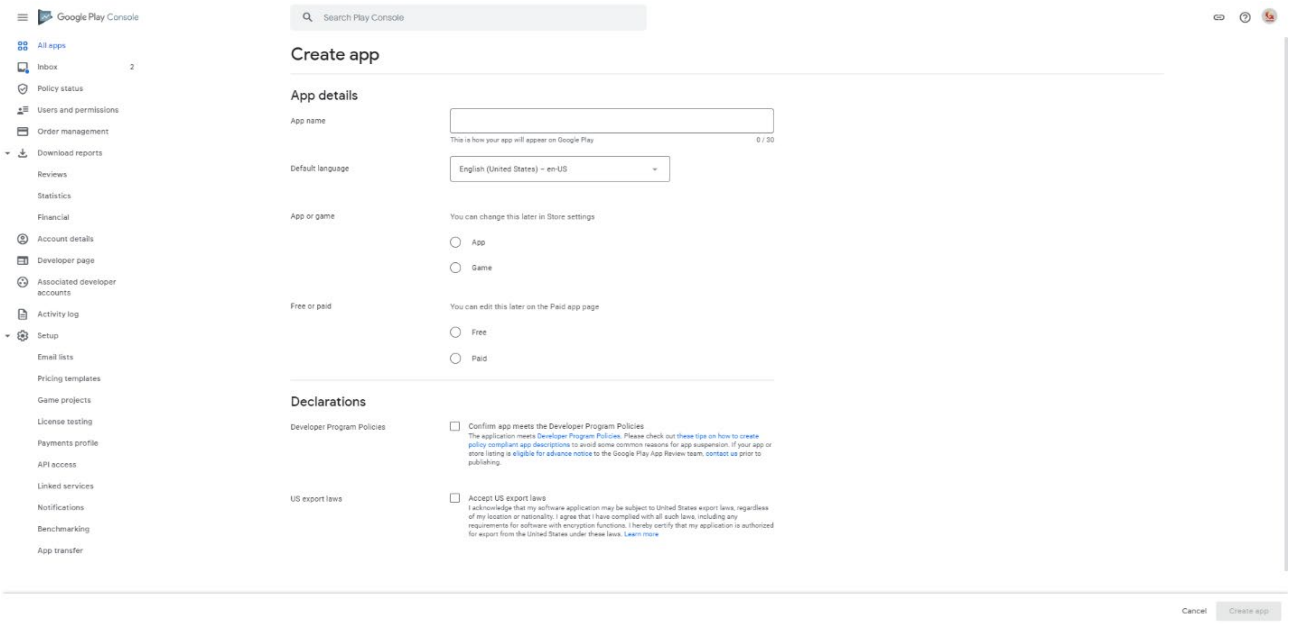


Рисунок 33. Создание приложения (английский интерфейс)

После совершения всех запрашиваемых настроек можно переходить в раздел «Релиз» и вкладку «Тестирование». Я производил внутреннее тестирование, позволяющее добавлять необходимых людей в список тестировщиков. Это помогло пригласить на тесты тех, кто уже ранее делал подобное с другими релизнутыми проектами. Создание такого выпуска довольное простое – необходимо загрузить APK либо AAB файл приложения и указать номер выпуска и изменения. Полноценный релиз осуществляется аналогичным способом, но так же, в случае предварительного тестирования, есть возможность переноса версии в релиз без повторной отправки установочного файла.

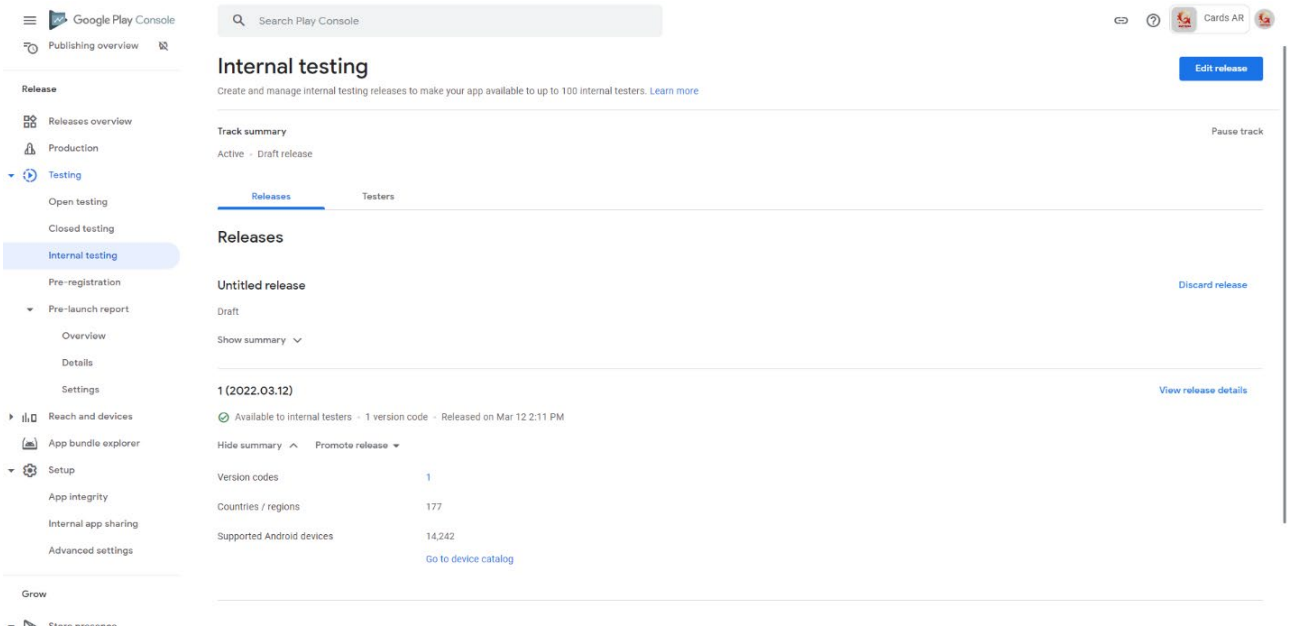


Рисунок 34. Внутреннее тестирование (английский интерфейс)

После успешной публикации проекта во внутреннее тестирование была произведена рассылка по тестировщикам следующей информации: приглашение на присоединение к тестам, ссылка на закрытую страницу приложения в Google Play и анкета для сбора отзывов. Анкета включает в себя 6 (шесть) основных вопросов, необходимых для анализа, с оценочным видом ответа (от 1 до 5 баллов/звезд). В итоге, было получено 29 анонимных отзывов (рисунки 35-40) за короткий промежуток времени. Эти отзывы дали понимание, что приложение устраивает большинство по большинству параметров, но, само собой, имеются недочеты, обусловленные урезанным функционал и инди-разработкой проекта.

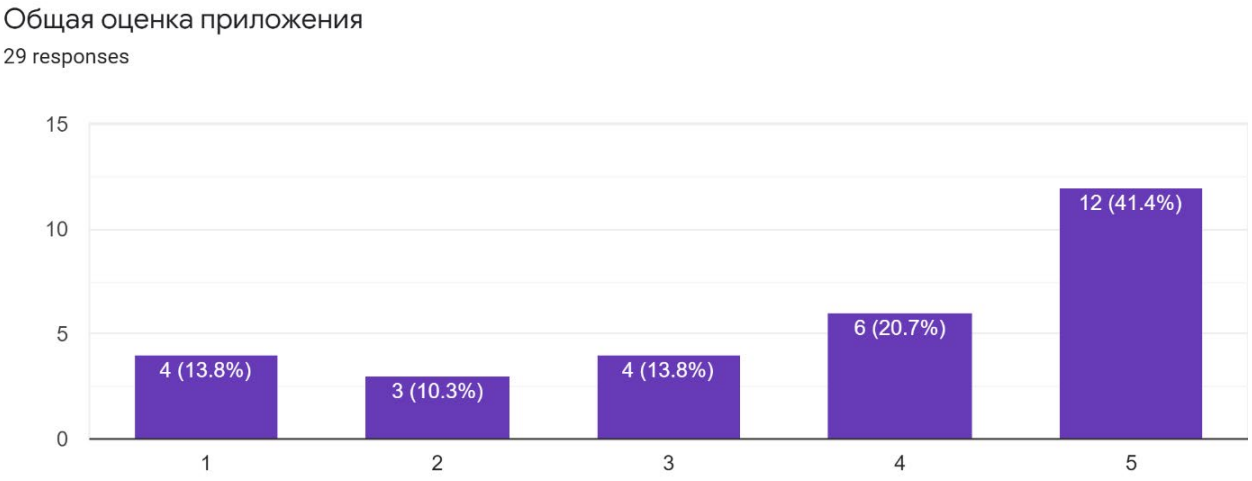


Рисунок 35. Анкета – Общая оценка приложения

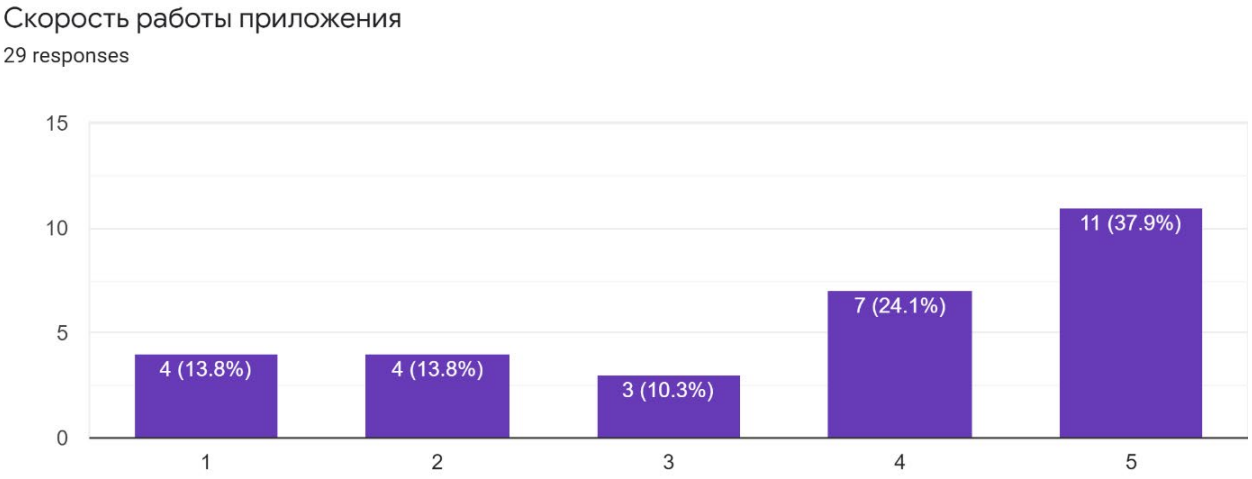


Рисунок 36. Анкета – Скорость работы приложения

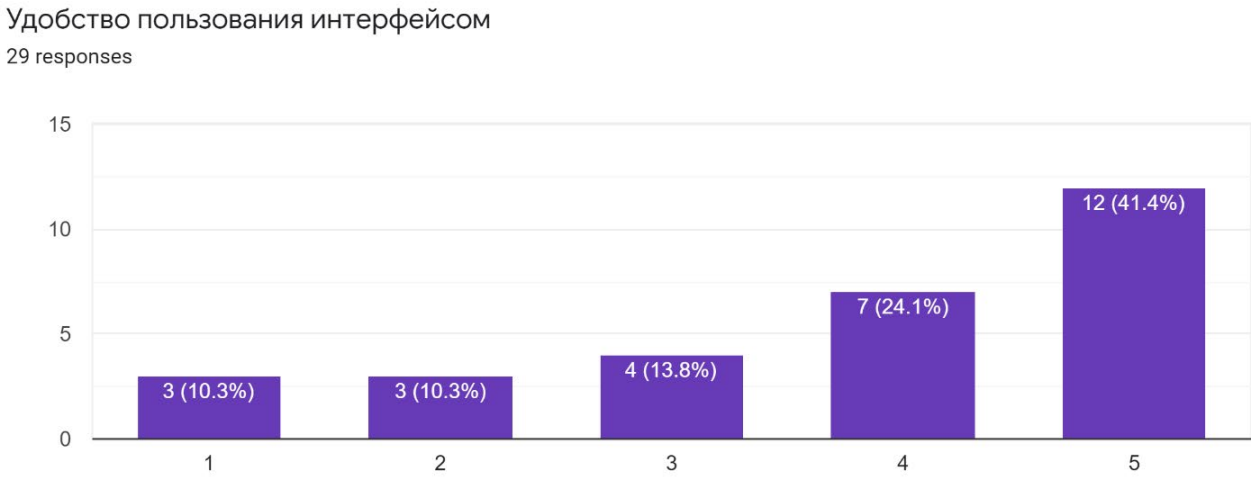


Рисунок 37. Анкета – Удобство пользования интерфейсом

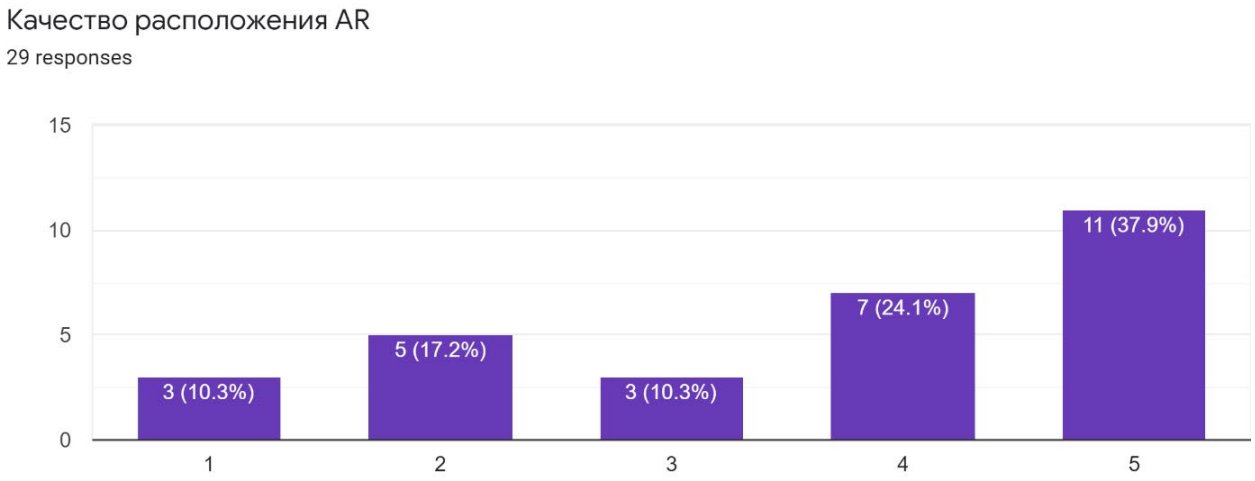


Рисунок 38. Анкета – Качество расположения AR

Рисунок 39. Анкета – Наличие соревновательного эффекта

Рисунок 40. Анкета – Погружаемость в карточное сражение

2.8. Вывод

В данной главе были описаны все этапы разработки от проектирования до полноценного релиза проекта. Ход разработки включал проектирование приложения на основе описанной в 1 главе задумки, установка выбранного

программного обеспечения, разработка проекта, публикацию и тестирование конечного продукта.

Разработка проекта включала в себе приведение блок-схем и листингов описываемых действий для полного понимания логики движения по игровому проекту. Соответственно, был понятен процесс создания и написания скриптов. Помимо этого, показана настройка и сборка установочного файла для дальнейшего его релиза.

Заключение

В рамках выполнения дипломного проекта была поставлена цель: создание мобильного игрового приложения с применением технологии дополненной реальности в виде коллекционной карточной игры.

Для реализации данной цели был выполнен ряд задач:

- Изучение предметной области;
- Анализ существующих аналогов приложения;
- Выбор необходимых технологий;
- Разработка мобильной AR-игры.

Реализация данной цели потребовала содержательного и методического решения следующих задач:

Первая глава посвящена изучению предметной области. В этой части подробно описаны истории появления и развития коллекционных карточных игр и дополненной реальности. Рассказана идея осуществленного проекта с приведением референсов и сравнением с существующими аналогами. Произведен выбор необходимых технологий и программного обеспечения, нужного для комфортной разработки проекта.

Вторая глава посвящена описанию всех этапов разработки приложения, начиная с установки выбранного ПО (Unity, ARFoundation, PlayFab, Photon и т.д.) и заканчивая тестированием и публикацией конечного продукта. Описание хода включает в себя приведение действующих выдержек из игрового проекта через скриншоты, блок-схемы и листинги того, что описывалось в тексте.

Список литературы

1. Карты игральные // Энциклопедический словарь Брокгауза и Ефрона: в 86 т. (82 т. и 4 доп.). — СПб., 1890—1907. (дата обращения: 21.05.2022)
2. Комиссаренко С. С., Игра в карты как культурная традиция русского общества XVIII—XIX веков. (дата обращения: 21.05.2022)
3. Карточная игра / Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Карточная_игра (дата обращения: 21.05.2022)
4. Коллекционная карточная игра / Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Коллекционная_карточная_игра (дата обращения: 21.05.2022)
5. 100 млн человек играют в Microsoft Solitaire Collection / Windows Blogs [Электронный ресурс]. URL: https://blogs.windows.com/russia/2016/09/01/100-mln-chelovek-igrajut-v-microsoft-solitaire-collection/ (дата обращения: 21.05.2022)
6. Hearthstone / Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Hearthstone (дата обращения: 21.05.2022)
7. Hearthstone / Официальный сайт проекта [Электронный ресурс]. URL: https://playhearthstone.com/ru-ru (дата обращения: 21.05.2022)
8. A Brief History of Augmented Reality (+Future Trends & Impact) / G2.com [Электронный ресурс]. URL: https://www.g2.com/articles/history-of-augmented-reality (дата обращения: 29.05.2022)
9. The Mainstreaming of Augmented Reality: A Brief History / Harvard Business Publishing [Электронный ресурс]. URL: https://hbr.org/2016/10/the-mainstreaming-of-augmented-reality-a-brief-history (дата обращения: 29.05.2022)
10. Unity vs Unreal Engine: Which Game Engine Should You Choose? / Hackr.io [Электронный ресурс]. URL: https://hackr.io/blog/unity-vs-unreal-engine (дата обращения: 29.05.2022)

11. Ultimate AR Comparison Guide: ARKit vs ARCore vs Vuforia vs AR Foundation / Circuit Stream [Электронный ресурс]. URL: https://circuitstream.com/blog/augmented-reality-guide/ (дата обращения: 29.05.2022)
12. What is PlayFab? / Документация Microsoft [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/gaming/playfab/what-is-playfab (дата обращения: 29.05.2022)
13. Photon Engine / Официальный сайт [Электронный ресурс]. URL: https://www.photonengine.com/pun (дата обращения: 02.06.2022)
14. Clash Royale / Официальный сайт [Электронный ресурс]. URL: https://clashroyale.com/ (дата обращения: 02.06.2022)
15. Pokémon Go / Официальный сайт [Электронный ресурс]. URL: https://pokemongo.nianticlabs.com/ru/ (дата обращения: 02.06.2022)
16. Argy Bargy Craft / Официальный сайт [Электронный ресурс]. URL: https://twobsoft.com/argy-bargy-craft/ (дата обращения: 02.06.2022)
17. В США выпущена карточная игра с элементами дополненной реальности / Digital Media [Электронный ресурс]. URL: http://digimedia.ru/game-news/12452/ (дата обращения: 02.06.2022)
18. 3 настольные игры дополненной реальности, которые открывают будущее домашних развлечений / Голографика [Электронный ресурс]. URL: https://holographica.space/articles/3-best-ar-board-games-11286/ (дата обращения: 02.06.2022)
19. Unity / Официальный сайт [Электронный ресурс]. URL: https://unity3d.com/get-unity/download (дата обращения: 02.06.2022)

Приложение А – Программный код

Листинг 1. Класс по работе с AR

```
using Goryned.AR;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.XR.ARFoundation;

public class ARManager : MonoBehaviour
{
    [Header("Fields")]
    [SerializeField] private ARRaycastManager raycastManager;
    [SerializeField] private Transform placementObject;
    [SerializeField] private Button fixButton;

    [Header("State")]
    [SerializeField] private bool isObjectFixed;
    [SerializeField] private ARPlacementData placementData;

    public bool IsObjectFixed
    {
        get => isObjectFixed;
        set
        {
            isObjectFixed = value;
            fixButton.GetComponentInChildren<TMP_Text>().text = value ? "Открепить" : "Зафиксировать";
        }
    }
```

```
private void Awake()
    {
        IsObjectFixed = false;
    }

    private void Update()
    {
        if (!IsObjectFixed)
        {
            placementData = ARPlacementManager.GetPlacementData(raycastManager);
            placementObject.position = placementData.SpawnPosition == Vector3.zero ? placementObject.position : placementData.SpawnPosition;
            placementObject.rotation = Quaternion.Euler(Vector3.zero);
            fixButton.interactable = placementData.IsReadyToSpawn;
        }
    }

    public void OnFixButtonPressed()
    {
        IsObjectFixed = !IsObjectFixed;
    }
    private void OnEnable()
    {
        placementObject.position = Vector3.zero;
        placementObject.rotation = Quaternion.Euler(Vector3.zero);
    }
    private void OnDisable()
    {
        placementObject.position = Vector3.zero;
        placementObject.rotation = Quaternion.Euler(Vector3.zero);
    }
}
```

Листинг 2. Структура боевых данных

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

[Serializable]
public struct BattleData
{
   public CardsZoneFiller DeckHolder;
   public List<CardData> CardDatas;
   [SerializeField] private Card selectedCard;
   public Card SelectedCard
   {
      get => selectedCard;
      set
      {
         selectedCard = value;
         if (selectedCard != null)
            selectedCard.transform.localScale *= 1.1f;
      }
   }
   public void CardSelected(CardData selectedCardData)
   {
      int index = CardDatas.FindIndex(d => d.UID == selectedCardData.UID);
      SelectedCard = DeckHolder.transform.GetChild(index).GetComponent<Card>();
      for (int i = 0; i < DeckHolder.transform.childCount; i++)
      {
         if (DeckHolder.transform.GetChild(i).GetComponent<BattleableCard>())

GameObject.Destroy(DeckHolder.transform.GetChild(i).GetComponent<BattleableCard>());
      }
   }
```

Продолжение. Листинг 2. Структура боевых данных

```
public void UpdateSelectedCard(CardData newCardData)
    {
        int index = CardDatas.FindIndex(d => d.UID == newCardData.UID);
        CardDatas[index] = newCardData;
        SelectedCard.SetCardData(newCardData);
        SelectedCard.healthText.transform.DOPunchScale(Vector3.one * 0.3f, 0.5f);
    }

    public void Refill(List<CardData> cardDatas, CardsZoneFillerType fillerType)
    {
        DeckHolder.ReFill(cardDatas, fillerType);
        CardDatas = cardDatas;
        SelectedCard = null;
    }

    public CardData GetRandomCardData()
    {
        return CardDatas[UnityEngine.Random.Range(0, CardDatas.Count)];
    }
    public void RemoveDeadCards()
    {
        CardDatas.RemoveAll(d => d.Health <= 0);
    }
    public int GetCardsCount()
    {
        return CardDatas.Count;
    }
}
```

Листинг 3. Класс по упрощению работы с боем

```
using Cysharp.Threading.Tasks;
using Photon.Pun;
using UnityEngine;

public class BattleHelper : MonoBehaviour
{
    public static bool StatsUpdated = false;
    public static bool WithPlayer
    {
        get
        {
            bool withPlayer = PhotonNetwork.PlayerList.Length == 2;
            if (!withPlayer && PhotonNetwork.InRoom)
            {
                PhotonNetwork.LeaveRoom();
            }
            return withPlayer;
        }
        set
        {
            if (!value && PhotonNetwork.InRoom)
            {
                PhotonNetwork.LeaveRoom();
            }
        }
    }

    private static bool opponentReady;
    public static bool OpponentReady { get => opponentReady; set => opponentReady = value; }
```

Продолжение. Листинг 3. Класс по упрощению работы с боем

```
private static int readyCount;
    public static int ReadyCount
    {
        get => readyCount;
        set
        {
            readyCount = value;
            if (value > 0)
            {
                Debug.Log("ReadyCount: " + value + "; WithPlayer: " + WithPlayer);
                if (value == 1 && !WithPlayer)
                {
                    CardData cardData = BattleManager.Instance.enemyBattleData.GetRandomCardData();
                    BattlePunRPC.instance.Call(BattlePunRPC.instance.SendSelectedCardData, RpcTarget.Others, JsonUtility.ToJson(cardData));
                }
                if (value == 2 && WithPlayer)
                {
                    CardData cardData = BattleManager.Instance.userBattleData.SelectedCard.GetCardData();
                    BattlePunRPC.instance.Call(BattlePunRPC.instance.SendSelectedCardData, RpcTarget.Others, JsonUtility.ToJson(cardData));
                }
            }
        }
    }
```

Продолжение. Листинг 3. Класс по упрощению работы с боем

```
public static async void ExitBattle()
    {
        await UniTask.WaitUntil(() => StatsUpdated);
        if (PhotonNetwork.InRoom)
        {
            PhotonNetwork.LeaveRoom();
            await UniTask.WaitUntil(() => !PhotonNetwork.InRoom);
        }
        if (PhotonNetwork.IsConnected)
        {
            PhotonNetwork.Disconnect();
            await UniTask.WaitUntil(() => !PhotonNetwork.IsConnected);
        }
    }
    public async static UniTask SendReady()
    {
        OpponentReady = false;
        do
        {
            BattlePunRPC.instance.Call(BattlePunRPC.instance.SendReady, RpcTarget.Others, null);
            await UniTask.Delay(100);
        } while (!OpponentReady);
    }
    public async static UniTask SendBattleData()
    {
        OpponentReady = false;
        BattleData battleData = BattleManager.Instance.userBattleData;
        BattlePunRPC.instance.Call(BattlePunRPC.instance.SendBattleData,        RpcTarget.Others,
JsonUtility.ToJson(battleData));
        await UniTask.WaitUntil(() => OpponentReady);
    }
}
```

Листинг 4. Класс боевого менеджера

```
using Photon.Pun;
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

public class BattleManager : MonoBehaviour
{
    public static BattleManager Instance;
    public static Action<bool, CardData> OnCardSelected;
    public static Action OnFightStarted, onFightCompleted;
    [Header("Header Zone")]
    [SerializeField] private Goryned.UI.UIElement timerElement;
    [SerializeField] private TMP_Text userNameField, enemyNameField;
    [Header("Battle Decks")]
    [SerializeField] public BattleData userBattleData;
    [SerializeField] public BattleData enemyBattleData;
    [Header("Table")]
    [SerializeField] private CardTable cardTable;
    private void Awake()
    {
        Instance = this;
    }
    private void OnEnable()
    {
        OnCardSelected += CardSelected;
        OnFightStarted += OnStartFight;
        onFightCompleted += OnCompleteFight;
    }
```

Продолжение. Листинг 4. Класс боевого менеджера

```
private void OnDisable()
    {
        OnCardSelected -= CardSelected;
        OnFightStarted -= OnStartFight;
        onFightCompleted -= OnCompleteFight;
    }

    public void InitializeBattle()
    {
        if (DataManager.Instance.PlayerDatas.UserPlayerData.Cards == null || !DataManager.Instance.PlayerDatas.UserPlayerData.Cards.ContainsKey(CardListType.Deck))
        {
            DataManager.Instance.PlayerDatas.UserPlayerData.Cards = new Dictionary<CardListType, List<CardData>>();
            DataManager.Instance.PlayerDatas.UserPlayerData.Cards.Add(CardListType.Deck, CardDataManager.GetRandomCardDatas(3, string.Empty));
        }
        userBattleData.CardDatas = DataManager.Instance.PlayerDatas.UserPlayerData.Cards[CardListType.Deck];

        enemyBattleData.CardDatas = CardDataManager.GetRandomCardDatas(3, string.Empty);
        DataManager.Instance.PlayerDatas.EnemyPlayerData.Cards = new Dictionary<CardListType, List<CardData>>();
        DataManager.Instance.PlayerDatas.EnemyPlayerData.Cards.Add(CardListType.Deck, enemyBattleData.CardDatas);

        NextRound();
    }
```

Продолжение. Листинг 4. Класс боевого менеджера

```
public async void NextRound()
    {
        BattleHelper.ReadyCount = 0;
        await BattleHelper.SendReady();
        await BattleHelper.SendBattleData();
        CheckUsers();
        RespawnBattleDecks();
        cardTable.ResetCards();
        StartCoroutine(TimerCounter());
    }
    public void SendEnemyCard(CardData cardData)
    {
        CardSelected(false, cardData);
    }
    public void RespawnBattleDecks()
    {
        userBattleData.Refill(userBattleData.CardDatas, CardsZoneFillerType.Battleable);
        enemyBattleData.Refill(enemyBattleData.CardDatas, CardsZoneFillerType.Simple);
    }
    private IEnumerator TimerCounter()
    {
        int waitingTime = 10;
        while (waitingTime > 0)
        {
            waitingTime--;
            timerElement.SetText(waitingTime.ToString(), Color.white);
            timerElement.ColorImage(waitingTime > 3 ? Color.green : Color.red);
            yield return new WaitForSeconds(1f);
        }
        CardSelected(true, userBattleData.GetRandomCardData());
    }
```

Продолжение. Листинг 4. Класс боевого менеджера

```
private void CardSelected(bool isUser, CardData selectedCardData)
    {
        StopAllCoroutines();
        Debug.Log(CardDataManager.ParseCardData(selectedCardData));
        cardTable.MoveToTable(isUser, selectedCardData);
        if (isUser) userBattleData.CardSelected(selectedCardData);
        else enemyBattleData.CardSelected(selectedCardData);
        cardTable.Fighting();
        if (isUser) BattlePunRPC.instance.Call(BattlePunRPC.instance.IncreaseReadyCount,
RpcTarget.All, null);
    }

    private void OnStartFight()
    {
        CardData userCardData = userBattleData.SelectedCard.GetCardData();
        CardData enemyCardData = enemyBattleData.SelectedCard.GetCardData();
        userCardData.Health -= enemyCardData.Attack;
        enemyCardData.Health -= userCardData.Attack;
        userBattleData.UpdateSelectedCard(userCardData);
    }

    private void OnCompleteFight()
    {
        userBattleData.RemoveDeadCards();
        enemyBattleData.RemoveDeadCards();

        BattleResultType battleResultType = GetBattleResultType();
        Debug.Log(battleResultType);
        if (battleResultType == BattleResultType.None)
        {
            NextRound();
        }
```

Продолжение. Листинг 4. Класс боевого менеджера

```
        else
        {
            List<PopupButtonData> buttons = new List<PopupButtonData>();
            PopupButtonData buttonData = new PopupButtonData();
            buttonData.Text = "Завершить";
            buttonData.TextColor = Color.white;
            buttonData.ButtonColor = Color.green;
            buttonData.Action = delegate { SceneManager.LoadScene("MainScene"); };
            buttons.Add(buttonData);
            string resultText = battleResultType == BattleResultType.Victory ? "Победа" :
                battleResultType == BattleResultType.Defeat ? "Поражение" : "Ничья";
            Popup.Instance.Open(resultText, resultText, buttons);
        }
    }
    public BattleResultType GetBattleResultType()    {
        int userCount = userBattleData.GetCardsCount();
        int enemyCount = enemyBattleData.GetCardsCount();
        if (userCount > 0 && enemyCount > 0) return BattleResultType.None;
        else
        {
            if (userCount <= 0 && enemyCount <= 0) return BattleResultType.Draw;
            else
            {
                if (userCount <= 0) return BattleResultType.Defeat;
                else return BattleResultType.Victory;
            }
        }    }
    private void CheckUsers()    {
        userNameField.text = PhotonNetwork.NickName;
        enemyNameField.text = BattleHelper.WithPlayer ? PhotonNetwork.PlayerList[1].NickName
: "Случайный игрок";
    }}
```

Листинг 5. Класс по работе с Photon Pun RPC

```
using Photon.Pun;
using Photon.Realtime;
using System;
using UnityEngine;

public class BattlePunRPC : MonoBehaviourPunCallbacks
{
    public static BattlePunRPC instance;
    private void Awake()
    {
        instance = this;
    }
    public void Call(Action<string> action, RpcTarget rpcTarget, string data)
    {
        Debug.Log("PhotonRPC - " + action.Method.Name);
        if (BattleHelper.WithPlayer)
        {
            PhotonView photonView = PhotonView.Get(this);
            photonView.RPC(action.Method.Name, rpcTarget, data);
        }
        else
        {
            if (action == SendBattleData)
            {
                BattleHelper.OpponentReady = true;
                return;
            }
            action(data);
        }
    }
```

Продолжение. Листинг 5. Класс по работе с Photon Pun RPC

```
[PunRPC]
    public void SendBattleData(string data)
    {
        BattleData battleData = JsonUtility.FromJson<BattleData>(data);
        BattleManager.Instance.enemyBattleData.CardDatas = battleData.CardDatas;
        Call(SendReady, RpcTarget.All, null);
    }
    [PunRPC]
    public void SendReady(string data)
    {
        BattleHelper.OpponentReady = true;
    }
    [PunRPC]
    public void IncreaseReadyCount(string data)
    {
        BattleHelper.ReadyCount++;
    }
    [PunRPC]
    public void SendSelectedCardData(string data)
    {
        CardData cardData = JsonUtility.FromJson<CardData>(data);
        BattleManager.OnCardSelected?.Invoke(false, cardData);
    }



    public override void OnPlayerLeftRoom(Player otherPlayer)
    {
        base.OnPlayerLeftRoom(otherPlayer);
        BattleHelper.ReadyCount = BattleHelper.ReadyCount;
    }
}
```

Листинг 6. Класс по управлению боевой сценой

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

public class BattleScene : MonoBehaviour
{
    public static BattleScene Instance;

    [Header("Components")]
    [SerializeField] private BattleManager battleManager;
    private void Awake()
    {
        if (DataManager.Instance == null) SceneManager.LoadScene(0);
        Instance = this;
        if (!battleManager) battleManager = FindObjectOfType<BattleManager>();
    }

    private void Start()
    {
        battleManager.InitializeBattle();
    }


}
```

Листинг 7. Переключение типа битвы

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using TMPro;
using UnityEngine;
public class BattleTypeManager : MonoBehaviour{
    [Header("Objects")]
    [SerializeField] private List<GameObject> objects3D;
    [SerializeField] private List<GameObject> objectsAR;
    [Header("Fields")]
    [SerializeField] private TMP_Text typeTextField;
    [Header("State")]
    [SerializeField] private bool isArActive;
    public bool IsArActive
    {
        get => isArActive;
        set
        {
            isArActive = value;
            typeTextField.text = value ? "AR" : "3D";
            objects3D.ForEach(x => x.SetActive(!value));
            objectsAR.ForEach(x => x.SetActive(value));
        }
    }
    private void Awake()
    {
        IsArActive = false;
    }
    public void OnChangeTypeButtonPressed()
    {
        IsArActive = !IsArActive;
    }}
```

Листинг 8. Кнопка старта битвы

```
using Goryned.Photon;
using System.Collections.Generic;
using UnityEngine;
[RequireComponent(typeof(PhotonConnecter))]
public class StartBattleButton : MonoBehaviour{
    [SerializeField] public PhotonConnecter photonConnecter;
    public void OnButtonPressed()    {
        List<PopupButtonData> buttons = new List<PopupButtonData>();
        PopupButtonData buttonData = new PopupButtonData();
        buttonData.Text = "Отмена";
        buttonData.TextColor = Color.black;
        buttonData.ButtonColor = Color.white;
        buttonData.Action = delegate { photonConnecter.CancelSearch(); };
        buttons.Add(buttonData);
        Popup.Instance.Open("Поиск соперника", "Состояние", buttons);
        photonConnecter.ConnectToPhoton();    }
    private void OnEnable()    {
        PhotonConnecter.OnSendMessage += ShowMessage;    }
    private void OnDisable()    {
        PhotonConnecter.OnSendMessage -= ShowMessage;    }
    private void ShowMessage(string message)
    {
        Popup.Instance.ChangeDescription(message);
    }
}
```

Листинг 9. Класс боевой карты

```
using UnityEngine;
using UnityEngine.EventSystems;

public class BattleableCard : MonoBehaviour, IPointerClickHandler
{
   public void OnPointerClick(PointerEventData eventData)
   {
      Debug.Log("OnPointerClick");
      BattleManager.OnCardSelected?.Invoke(true, GetComponent<Card>().GetCardData());
   }
}
```

Листинг 10. Класс карты

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class Card : MonoBehaviour
{
   [Header("Card Data")]
   [SerializeField] private CardData cardData;

   [Header("Fields")]
   [SerializeField] private TMP_Text characterName;
   [SerializeField] private TMP_Text characterDescription;
   [SerializeField] public Image characterImage;
   [SerializeField] private TMP_Text attackText;
   [SerializeField] public TMP_Text healthText;
   private CardData CardData    {
      get => cardData;
      set       {
         cardData = value;
```

Продолжение. Листинг 10. Класс карты

```
            Sprite characterIcon = CharacterDataManager.GetCharacterIcon(cardData.Name);
            characterName.text = characterIcon.name;
            characterDescription.text = "";
            characterImage.sprite = characterIcon;

            attackText.text = cardData.Attack.ToString();
            healthText.text = cardData.Health.ToString();
        }
    }

    public void SetCardData(CardData data)
    {
        CardData = data;
    }
    public CardData GetCardData()
    {
        return new CardData(CardData);
    }
}
```

Листинг 11. Структура карты

```
public struct CardData
{
    [Header("Information")]
    public string UID; // Уникальный идентификатор для различения карт
    public string Name; // Имя персонажа (состоит из номеров туловища и головы персонажа)

    [Header("Parameters")]
    public int Level; // Уровень (в будущем, для активации дополнительных возможностей)
    public int Attack; // Атака = количество, которое отнимается от защиты за 1 ход
    public int Health; // Здоровье = максимальное количество защиты

    [Header("System")]
    public string PlayFabID; // Идентификатор первого владельца карты
    public JsonDateTime DateTime; // Дата и время первого появления
    public int RandomNumber; // Случайное число для уникальности UID

    public CardData(CardData cardData)
    {
        UID = cardData.UID;
        Name = cardData.Name;

        Level = cardData.Level;
        Attack = cardData.Attack;
        Health = cardData.Health;

        PlayFabID = cardData.PlayFabID;
        DateTime = cardData.DateTime;
        RandomNumber = cardData.RandomNumber;
    }
}
```

Листинг 12. Помощник по работе с CardData

```
public static class CardDataManager
{
    public static char separator = ';';
    public static CardData GetRandomCardData(string playFabID)
    {
        CardData cardData = new CardData();
        cardData.Name = CharacterDataManager.GetRandomCharacterIcon().name;
        cardData.Level = 0;
        cardData.Attack = Random.Range(1, 5);
        cardData.Health = Random.Range(5, 10);
        cardData.PlayFabID = playFabID;
        DateTime dateTime = DateTime.UtcNow;
        cardData.DateTime = dateTime;
        cardData.RandomNumber = Random.Range(0, 1000);
        cardData.UID = cardData.Name
            + "-" + cardData.PlayFabID
            + "-" + dateTime.ToString("G") + "." + dateTime.ToString("fff")
            + "-" + cardData.RandomNumber;

        return cardData;
    }
    public static List<CardData> GetRandomCardDatas(int count, string playFabID)
    {
        List<CardData> cardDatas = new List<CardData>();
        for (int i = 0; i < count; i++)
        {
            CardData cardData = GetRandomCardData(playFabID);
            cardDatas.Add(cardData);
        }
        return cardDatas;
    }
```

Продолжение. Листинг 12. Помощник по работе с CardData

```
public static string ParseCardData(CardData cardData)
    {
        string data = JsonUtility.ToJson(cardData);
        return data;
    }
    public static CardData ParseCardData(string data)
    {
        CardData cardData = JsonUtility.FromJson<CardData>(data);
        return cardData;
    }
    public static string ParseCardDatas(List<CardData> cardDatas)
    {
        string datas = "";
        foreach (var cardData in cardDatas)
        {
            string data = ParseCardData(cardData);
            string sep = string.IsNullOrEmpty(datas) ? "" : separator.ToString();
            datas += sep + data;
        }
        return datas;
    }
    public static List<CardData> ParseCardDatas(string datas)
    {
        if (string.IsNullOrEmpty(datas)) return new List<CardData>();

        string[] strDatas = datas.Split(separator);
        List<CardData> cardDatas = new List<CardData>();
        foreach (string strData in strDatas) cardDatas.Add(ParseCardData(strData));
        return cardDatas;
    }
```

```
private static CardDataInformation FindCardDataInformation(string UID)
   {
      CardListType cardListType = CardListType.Deck; int index = -1;
      foreach (var key in DataManager.Instance.PlayerDatas.UserPlayerData.Cards.Keys)
      {
         cardListType = key;
         index = DataManager.Instance.PlayerDatas.UserPlayerData.Cards[key].FindIndex(d => d.UID == UID);
         if (index >= 0) break;
      }
      return index >= 0 ? new CardDataInformation(cardListType, UID, index, DataManager.Instance.PlayerDatas.UserPlayerData.Cards[cardListType][index]) : new CardDataInformation();
   }
   public static void AddCardData(CardListType cardListType, CardData cardData, int index = -1)
   {
      if (index < 0) DataManager.Instance.PlayerDatas.UserPlayerData.Cards[cardListType].Add(cardData);
      else DataManager.Instance.PlayerDatas.UserPlayerData.Cards[cardListType].Insert(index, cardData);
   }
   public static void UpdateCardData(string UID, CardData newCardData)
   {
      CardDataInformation data = FindCardDataInformation(UID);
      if (string.IsNullOrEmpty(data.UID)) return;
      DataManager.Instance.PlayerDatas.UserPlayerData.Cards[data.CardListType][data.Index] = newCardData;
   }
```

Продолжение. Листинг 12. Помощник по работе с CardData

```
public static void ReplaceCardDatas(CardData cardData1, CardData cardData2)
   {
      CardDataInformation data1 = FindCardDataInformation(cardData1.UID);
      CardDataInformation data2 = FindCardDataInformation(cardData2.UID);



DataManager.Instance.PlayerDatas.UserPlayerData.Cards[data1.CardListType][data1.Index]     =
cardData2;

DataManager.Instance.PlayerDatas.UserPlayerData.Cards[data2.CardListType][data2.Index]     =
cardData1;


      Debug.Log(string.Format("{0}          <->          {1}",          JsonUtility.ToJson(data1),
JsonUtility.ToJson(data2)));
   }
   public static void RemoveCardData(string UID, CardListType cardListType =
CardListType.Collection)
   {
      if (string.IsNullOrEmpty(UID))
      {
         int                                          index                                          =
DataManager.Instance.PlayerDatas.UserPlayerData.Cards[cardListType].Count - 1;

DataManager.Instance.PlayerDatas.UserPlayerData.Cards[cardListType].RemoveAt(index);
      }
      else
      {
         CardDataInformation data = FindCardDataInformation(UID);

DataManager.Instance.PlayerDatas.UserPlayerData.Cards[data.CardListType].RemoveAt(data.In
dex);
      } }}
```

Продолжение. Листинг 12. Помощник по работе с CardData

```
[Serializable]
public struct CardDataInformation
{
    public CardListType CardListType;
    public string UID;
    public int Index;
    public CardData CardData;

    public CardDataInformation(CardListType cardListType, string uID, int index, CardData cardData)
    {
        CardListType = cardListType;
        UID = uID;
        Index = index;
        CardData = cardData;
    }
}
```

Листинг 13. Тип набора карт

```
public enum CardListType { Deck = 0, Collection = 1 }
```

Листинг 14. Состояние 3D карты

```
public enum CardStateType{
    None = 0,
    MovingToTable = 1,
    OnTable = 2,
    MovingToFight = 3,
    ReadyForFight = 4,
    OnFight = 5,
    CompleteFight = 6,
}
```

Листинг 15. Заполнение набора карт данными

```
public class CardsZoneFiller : MonoBehaviour
{
   [SerializeField] public Transform cardsHolder;

   public void Fill(List<CardData> cardDatas, CardsZoneFillerType type = CardsZoneFillerType.Simple)
   {
      GameObject cardObject = DataManager.Instance.CommonDatas.Card2D;
      Goryned.Object.SpawnManager.ClearChildren(cardsHolder);
      foreach (var cardData in cardDatas)
      {
         Card card = Goryned.Object.SpawnManager.Spawn(cardObject, cardsHolder).GetComponent<Card>();
         card.transform.localScale = Vector3.one;
         card.SetCardData(cardData);

         if (type == CardsZoneFillerType.Moveable) card.gameObject.AddComponent<MoveableCard>();
         if (type == CardsZoneFillerType.Battleable) card.gameObject.AddComponent<BattleableCard>();
      }
   }
}
```

Листинг 16. Класс игрового стола

```
public class CardTable : MonoBehaviour
{
    [Header("Cards")]
    [SerializeField] private Card3D userCard;
    [SerializeField] private Card3D enemyCard;

    [Header("State")]
    [SerializeField] public bool isWaitingForFight;

    public void MoveToTable(bool isUserCard, CardData cardData)
    {
        Card3D card = isUserCard ? userCard : enemyCard;
        card.ActivateCardData(cardData);
        card.MoveToTable(Camera.main.transform.position, 3f);
    }

    public async void Fighting()
    {
        if (isWaitingForFight) return;
        isWaitingForFight = true;

        await UniTask.WaitUntil(() => CheckCommonState(CardStateType.OnTable));
        await UniTask.Delay(TimeSpan.FromSeconds(1f));
        userCard.MoveToFight(enemyCard.tablePosition, 1f);
        enemyCard.MoveToFight(userCard.tablePosition, 1f);
        await UniTask.WaitUntil(() => CheckCommonState(CardStateType.ReadyForFight));
        userCard.Fight(); enemyCard.Fight();
        BattleManager.OnFightStarted?.Invoke();
        await UniTask.WaitUntil(() => CheckCommonState(CardStateType.CompleteFight));
        userCard.CompleteFight(); enemyCard.CompleteFight();
        await UniTask.WaitUntil(() => CheckCommonState(CardStateType.None));
        BattleManager.onFightCompleted?.Invoke();
```

Продолжение. Листинг 16. Класс игрового стола

```
        isWaitingForFight = false;
    }

    public void ResetCards()
    {
        userCard.CardStateType = CardStateType.None;
        userCard.CardStateType = CardStateType.None;
    }
    public bool CheckCommonState(CardStateType stateType)
    {
        return userCard.CardStateType == stateType && enemyCard.CardStateType == stateType;
    }
}
```

Листинг 17. Класс перемещаемой карты

```
public class MoveableCard : MonoBehaviour, IDragHandler, IBeginDragHandler, IEndDragHandler
{
    private CanvasGroup canvasGroup;
    private Transform parent;
    private int siblingIndex;

    void Start()
    {
        parent = transform.parent;
        siblingIndex = -1;

        canvasGroup = gameObject.AddComponent<CanvasGroup>();
    }
    public void OnBeginDrag(PointerEventData eventData)
    {
        siblingIndex = transform.GetSiblingIndex();
```

Продолжение. Листинг 17. Класс перемещаемой карты

```
        transform.SetParent(MainScene.Instance.mainPanel);
        canvasGroup.blocksRaycasts = false;
    }

    public void OnDrag(PointerEventData eventData)
    {
        transform.position = eventData.position;
    }

    public void OnEndDrag(PointerEventData eventData)
    {
        //Debug.Log("OnEndDrag");
        transform.SetParent(parent);
        transform.SetSiblingIndex(siblingIndex);
        canvasGroup.blocksRaycasts = true;

        GameObject pointerObject = eventData.pointerCurrentRaycast.gameObject;
        if (pointerObject != null && pointerObject.GetComponent<Card>())
        {
            CardData collectionCardData = transform.GetComponent<Card>().GetCardData();
            CardData deckCardData = pointerObject.GetComponent<Card>().GetCardData();

            int collectionIndex =
DataManager.Instance.PlayerDatas.UserPlayerData.Cards[CardListType.Collection].IndexOf(col
lectionCardData);
            int deckIndex =
DataManager.Instance.PlayerDatas.UserPlayerData.Cards[CardListType.Deck].IndexOf(deckCar
dData);


DataManager.Instance.PlayerDatas.UserPlayerData.Cards[CardListType.Deck][deckIndex] =
collectionCardData;
```

Продолжение. Листинг 17. Класс перемещаемой карты

```
DataManager.Instance.PlayerDatas.UserPlayerData.Cards[CardListType.Collection][collectionIn
dex] = deckCardData;

            PlayerDataManager.UpdateUserData();

            MainScene.Instance.RefillCardsZones();

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

Листинг 18. Класс персонажа

```
public class Character : MonoBehaviour
{
    public Animator animator;

    [Header("Head")]
    [SerializeField] private List<GameObject> heads;
    [SerializeField] private GameObject activeHead;

    [Header("Body")]
    [SerializeField] private List<GameObject> bodies;
    [SerializeField] private GameObject activeBody;

    private void Awake()
    {
        animator = GetComponent<Animator>();
    }
    public void ResetCharacter()
    {
        heads.ForEach(g => g.SetActive(false));
```

```
    bodies.ForEach(g => g.SetActive(false));
}

public void Fight()
{
    animator.SetTrigger("Fight");
}

public void SetCharacter(string name)
{
    ResetCharacter();

    if (string.IsNullOrEmpty(name) || !name.Contains('-'))
        name = CharacterDataManager.GetRandomCharacterIcon().name;

    string[] nameContent = name.Split('-');
    activeHead = heads.Find(g => g.name == nameContent[0]); activeHead.SetActive(true);
    activeBody = bodies.Find(g => g.name == nameContent[1]); activeBody.SetActive(true);
}

public void SetCharacter(int headIndex, int bodyIndex)
{
    ResetCharacter();

    if (activeHead) activeHead.SetActive(false);
    activeHead = heads[headIndex];
    activeHead.SetActive(true);

    if (activeBody) activeBody.SetActive(false);
    activeBody = bodies[bodyIndex];
    activeBody.SetActive(true);
}   }
```

Листинг 19. Структура персонажа

```
public struct CharacterData
{
   public string Name;
   public string Description;
   public Sprite Icon;
   public GameObject Prefab;
}
```

Листинг 20. Ассет общих данных

```
[CreateAssetMenu(fileName = "CommonDatas", menuName = "CardsAR/CommonDatas")]
public class CommonDatas : ScriptableObjectInstaller<CommonDatas>
{
   public override void InstallBindings()
   {
      Container.BindInstance(this).AsSingle();
   }

   [Header("Card")]
   public GameObject Card2D;
   public GameObject Card3D;

   [Header("Characters")]
   public List<Sprite> CharacterIcons;
}
```

Листинг 21. Ассет данных игроков

```
[CreateAssetMenu(fileName = "PlayerDatas", menuName = "CardsAR/PlayerDatas")]
public class PlayerDatas : ScriptableObjectInstaller<PlayerDatas>
{
   public override void InstallBindings()   {      Container.BindInstance(this).AsSingle();   }
   public PlayerData UserPlayerData;
   public PlayerData EnemyPlayerData;
}
```

Листинг 22. Класс по загрузке сцены

```
public class LoadScene : MonoBehaviour
{
   public Goryned.UI.UIElement progressSlider;
   private void Start()  {      StartLoading();   }
   private              void              Update()                                                    {
progressSlider.SetProgress(Goryned.Scene.SceneManager.GetSceneProgress().Progress);   }
   public async void StartLoading()
   {
      float loadTime = 5f;
      Goryned.Load.LoadManager.Reset();
      Goryned.Scene.SceneManager.LoadScene("MainScene", loadTime, false);
      Login();
      await UniTask.Delay(TimeSpan.FromSeconds(loadTime));
      await UniTask.WaitUntil(() => Goryned.Load.LoadManager.IsAllComplete());
      Goryned.Scene.SceneManager.AllowSceneActivation();
   }
   public async void Login()
   {
      Goryned.Load.LoadManager.AddStep("LoginPlayFab");
      PlayFabManager.LoginPlayFab();
      await UniTask.WaitUntil(() => PlayFabTempData.loginResult != null);

      LoginResult loginResult = PlayFabTempData.loginResult;
      PlayFabTempData.loginResult = null;

      PlayerDataManager.GetPlayFabProfile(loginResult.PlayFabId);
      PlayerDataManager.GetPlayerStatistics(loginResult.PlayFabId);
      PlayerDataManager.GetUserCards(loginResult.PlayFabId);

      Goryned.Load.LoadManager.CompleteStep("LoginPlayFab");
   }
}
```

Листинг 23. Класс по работе с основной сценой

```
public class MainScene : MonoBehaviour{
    public static MainScene Instance;
    public Transform mainPanel;
    private void Awake()    {
        if (DataManager.Instance == null) SceneManager.LoadScene(0);
        Instance = this;    }
    [Header("Cards Zone Fillers")]
    [SerializeField] private CardsZoneFiller deckZone;
    [SerializeField] private CardsZoneFiller collectionZone;
    private void Start()    {       RefillCardsZones();    }
    public void RefillCardsZones()
    {
        PlayerData playerData = DataManager.Instance.PlayerDatas.UserPlayerData;
        deckZone.ReFill(playerData.Cards[CardListType.Deck]);
        collectionZone.ReFill(playerData.Cards[CardListType.Collection],
CardsZoneFillerType.Moveable);
    }
    public void AddCardToCollection(bool add)
    {
        CardData                                    cardData                                    =
CardDataManager.GetRandomCardData(DataManager.Instance.PlayerDatas.UserPlayerData.Pla
yFabID);

        if (add) CardDataManager.AddCardData(CardListType.Collection, cardData);
        else CardDataManager.RemoveCardData(string.Empty, CardListType.Collection);

        PlayerDataManager.UpdateUserData();

        RefillCardsZones();
    }
}
```

Листинг 24. Структура пользовательских данных

```
public struct PlayerData
{
    [Header("PlayFab Profile")]
    public string PlayFabID;
    public string DisplayName;
    public string DeviceID;

    [Header("Statistics")]
    public Dictionary<StatisticsType, int> Statistics;

    [Header("UserData")]
    public Dictionary<CardListType, List<CardData>> Cards;
}
```

Листинг 25. Класс по работе с пользовательскими данными

```
public static class PlayerDataManager
{
    public static async void GetPlayFabProfile(string playFabID, bool isPlayerData = true)
    {
        Goryned.Load.LoadManager.AddStep("GetPlayFabProfile");

        PlayFabManager.GetPlayerProfile(playFabID);
        await UniTask.WaitUntil(() => PlayFabTempData.playerProfileModel != null);

        PlayerData playerData = isPlayerData ? DataManager.Instance.PlayerDatas.UserPlayerData :
DataManager.Instance.PlayerDatas.EnemyPlayerData;
        playerData.PlayFabID = PlayFabTempData.playerProfileModel.PlayerId;
        playerData.DisplayName = PlayFabTempData.playerProfileModel.DisplayName;
        playerData.DeviceID = Goryned.Core.SystemManager.GetDeviceID();
        if (isPlayerData) DataManager.Instance.PlayerDatas.UserPlayerData = playerData;
        else DataManager.Instance.PlayerDatas.EnemyPlayerData = playerData;

        PlayFabTempData.playerProfileModel = null;
```

Продолжение. Листинг 25. Класс по работе с пользовательскими данными

```
        Goryned.Load.LoadManager.CompleteStep("GetPlayFabProfile");
    }

    public static async void GetPlayerStatistics(string playFabID, bool isPlayerData = true)
    {
        Goryned.Load.LoadManager.AddStep("GetPlayerStatistics");

        PlayFabManager.GetPlayerStatistics();
        await UniTask.WaitUntil(() => PlayFabTempData.playerStatistics != null);

        PlayerData playerData = isPlayerData ? DataManager.Instance.PlayerDatas.UserPlayerData :
DataManager.Instance.PlayerDatas.EnemyPlayerData;
        playerData.Statistics = GetDefaultStatistics();
        foreach (var item in PlayFabTempData.playerStatistics)
        {
            StatisticsType statisticsType = (StatisticsType)Enum.Parse(typeof(StatisticsType),
item.Key);
            if (playerData.Statistics.ContainsKey(statisticsType))
            {
                playerData.Statistics[statisticsType] = item.Value;
            }
        }
        if (isPlayerData) DataManager.Instance.PlayerDatas.UserPlayerData = playerData;
        else DataManager.Instance.PlayerDatas.EnemyPlayerData = playerData;

        PlayFabTempData.playerStatistics = null;

        UpdatePlayerStatistics();

        Goryned.Load.LoadManager.CompleteStep("GetPlayerStatistics");
    }
```

Продолжение. Листинг 25. Класс по работе с пользовательскими данными

```
  public static async void GetUserCards(string playFabID, bool isPlayerData = true)
  {
    Goryned.Load.LoadManager.AddStep("GetUserCards");

    PlayFabManager.GetUserData(playFabID);
    await UniTask.WaitUntil(() => PlayFabTempData.userData != null);

    PlayerData playerData = isPlayerData ? DataManager.Instance.PlayerDatas.UserPlayerData :
DataManager.Instance.PlayerDatas.EnemyPlayerData;
    playerData.Cards = GetDefaultCards();
    foreach (CardListType cardListType in Enum.GetValues(typeof(CardListType)))
    {
      if (PlayFabTempData.userData.ContainsKey(cardListType.ToString()))
      {
        string cardsStringData = PlayFabTempData.userData[cardListType.ToString()];
        List<CardData> cardDatas = CardDataManager.ParseCardDatas(cardsStringData);
        if (cardListType == CardListType.Deck && cardDatas.Count != 3) cardDatas =
CardDataManager.GetRandomCardDatas(3, playerData.PlayFabID);
        playerData.Cards[cardListType] = cardDatas;
      }
    }
    if (isPlayerData) DataManager.Instance.PlayerDatas.UserPlayerData = playerData;
    else DataManager.Instance.PlayerDatas.EnemyPlayerData = playerData;

    PlayFabTempData.userData = null;

    UpdateUserData();

    Goryned.Load.LoadManager.CompleteStep("GetUserCards");
  }
```

Продолжение. Листинг 25. Класс по работе с пользовательскими данными

```
private static Dictionary<StatisticsType, int> GetDefaultStatistics()
    {
        Dictionary<StatisticsType, int> statistics = new Dictionary<StatisticsType, int>();
        statistics.Add(StatisticsType.None, 0);
        statistics.Add(StatisticsType.Rating, 0);
        statistics.Add(StatisticsType.SoftCoins, 0);
        statistics.Add(StatisticsType.HardCoins, 0);
        return statistics;
    }
    private static Dictionary<CardListType, List<CardData>> GetDefaultCards()
    {
        Dictionary<CardListType, List<CardData>> cards = new Dictionary<CardListType, List<CardData>>();
        string playFabID = DataManager.Instance.PlayerDatas.UserPlayerData.PlayFabID;
        cards.Add(CardListType.Deck, CardDataManager.GetRandomCardDatas(3, playFabID));
        cards.Add(CardListType.Collection, CardDataManager.GetRandomCardDatas(4, playFabID));
        return cards;
    }

    public static async void UpdatePlayerStatistics()
    {
        PlayFabTempData.updatePlayerStatisticsResult = null;

        Dictionary<string, int> dataDictionary = new Dictionary<string, int>();
        Dictionary<StatisticsType, int> statistics = DataManager.Instance.PlayerDatas.UserPlayerData.Statistics;
        foreach (var item in statistics) dataDictionary.Add(item.Key.ToString(), item.Value);

        PlayFabManager.UpdatePlayerStatistics(dataDictionary);
        await UniTask.WaitUntil(() => PlayFabTempData.updatePlayerStatisticsResult != null);
    }
```

Продолжение. Листинг 25. Класс по работе с пользовательскими данными

```
public static async void UpdateUserData()
    {
        PlayFabTempData.updateUserDataResult = null;

        Dictionary<string, string> dataDictionary = new Dictionary<string, string>();
        Dictionary<CardListType, List<CardData>> cards = DataManager.Instance.PlayerDatas.UserPlayerData.Cards;
        foreach (var item in cards) dataDictionary.Add(item.Key.ToString(), CardDataManager.ParseCardDatas(item.Value));

        PlayFabManager.UpdateUserData(dataDictionary);
        await UniTask.WaitUntil(() => PlayFabTempData.updateUserDataResult != null);
    }
}
```

Листинг 26. Класс всплывающего уведомления

```
public class Popup : MonoBehaviour
{
    public static Popup Instance { get; private set; }

    [Header("Fields")]
    [SerializeField] private GameObject popupPanel;
    [SerializeField] private TMP_Text titleTextField;
    [SerializeField] private TMP_Text descriptionTextField;
    [SerializeField] private Transform buttonsParent;
    [SerializeField] private GameObject buttonPrefab;

    private void Awake()
    {
        Instance = this;
        Close();
    }
```

Продолжение. Листинг 26. Класс всплывающего уведомления

```
    public void Open(string title, string description, List<PopupButtonData> buttonDatas)
    {
        popupPanel.SetActive(true);
        titleTextField.text = title;
        descriptionTextField.text = description;

        SpawnManager.ClearChildren(buttonsParent);
        buttonDatas.ForEach(buttonData
            => SpawnManager.Spawn(buttonPrefab,
buttonsParent).GetComponent<PopupButton>().Initialize(buttonData));
    }

    public void ChangeDescription(string description)
    {
        descriptionTextField.text = description;
    }

    public void Close()
    {
        popupPanel.SetActive(false);
    }
}
```

Листинг 27. Кнопка попапа

```
public class PopupButton : MonoBehaviour
{
    [SerializeField] private Button button;
    [SerializeField] private Image imageField;
    [SerializeField] private TMP_Text textField;

    public void Initialize(PopupButtonData buttonData)
    {
        textField.text = buttonData.Text;
```

Продолжение. Листинг 27. Кнопка попапа

```
textField.color = buttonData.TextColor;


        imageField.color = buttonData.ButtonColor;
        button.onClick.RemoveAllListeners();
        button.onClick.AddListener(buttonData.Action);
    }
}
```

Листинг 28. Данные попапа

```
public struct PopupButtonData
{
    public string Text;
    public Color TextColor;
    public Color ButtonColor;
    public UnityAction Action;
}
```

Приложение Б – Презентация

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий
кафедра Информатики и информационных технологий
направление подготовки 09.03.02 «Информационные системы и технологии»,
профиль «Технологии дополненной и виртуальной реальности в медиаиндустрии»

выпускная квалификационная работа бакалавра на тему:

«Разработка карточной игры с применением дополненной реальности»

Научный руководитель:
Булатников Евгений Владиславович,
к.т.н.

Выполнил: студент
группы 181-724
Томашин Егор

Актуальность

- Игра с соревновательным эффектом;
- На базе коллекционной карточной игры;
- Использование технологии дополненной реальности.

2

Цель и задачи

- Цель работы: создание мобильного игрового приложения с применением технологии дополненной реальности в виде коллекционной карточной игры.

- Задачи:
  1. Изучение предметной области;
  2. Анализ существующих аналогов приложения;
  3. Выбор необходимых технологий;
  4. Разработка мобильной AR-игры.

Референсы

- Hearthstone
- Clash Royale
- Pokémon GO
- Argy Bargy Craft

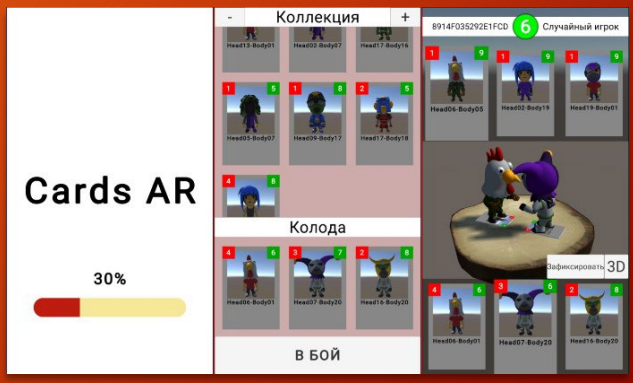
Выбор технологий

- Среда и язык разработки:Unity + C#
- ТехнологияAR: ARFoundation (ARCore + ARKit)
- База данных:Microsoft Azure PlayFab
- Мультиплеер: Photon PUN
- Платформа: ОС Android

Структура приложения

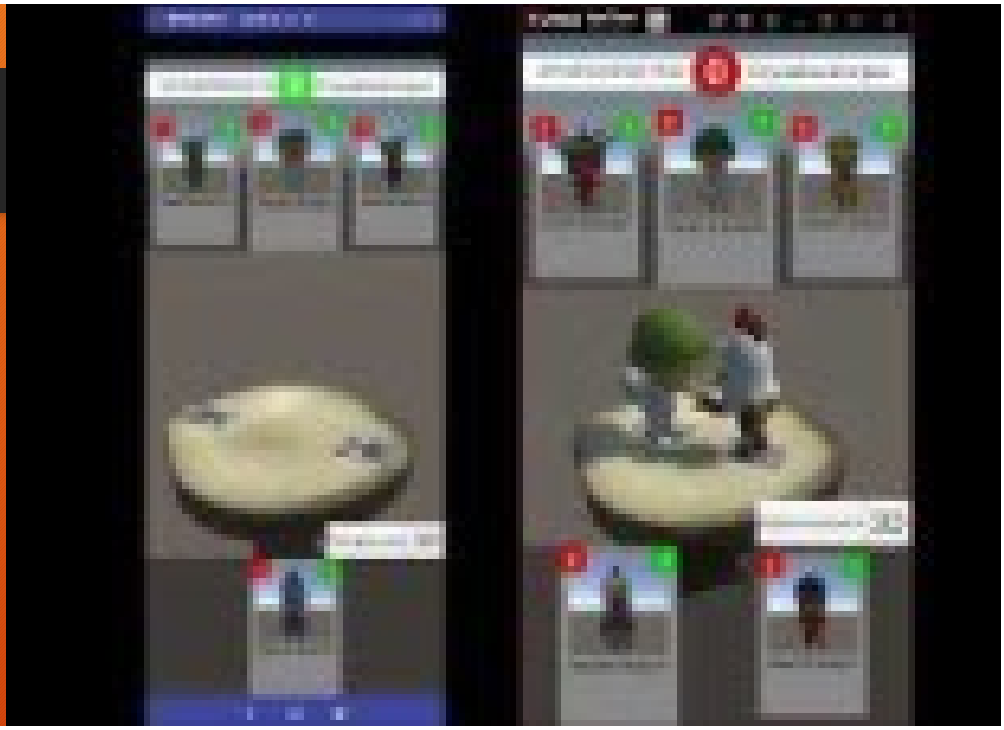
Заключение

Приложение работает корректно и качественно решает поставленные задачи. Есть место масштабированию и коммерческой реализации данного проекта. Все материалы в полной мере удовлетворяют поставленным задачам. Научно-технический уровень работы соответствует уровню, предъявленному к работам бакалавра. Все материалы ВКР могут использоваться для дальнейшего исследования и разработки в данной предметной области.



7

ДОСТУПНО В Google Play



8