Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Дисциплина «Объектно-ориентированное программирование и проектирование»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  ассистент кафедры ПИКС  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Листратенко |
|  | \_\_\_.\_\_\_\_.2022 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«ИГРОВОЕ ПРОГРАММНОЕ СРЕДСТВО “КАРТОЧНАЯ ИГРА” ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ»**

БГУИР КП 1-40 05 01-10 027 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 114301  Тюленев Юрий-Франтишек Александрович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2022  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2022

**РЕФЕРАТ**

БГУИР КР 1-40 05 01-10 027 ПЗ

**Тюленев** **Ю-Ф.А.** Игровое программное средство “Карточная игра” для мобильных устройств /Ю-Ф.А. Тюленев– Минск: БГУИР, 2022. – 47 с.

Пояснительная записка 47 с., 22 рисунка, 5 источников, 3 приложения.

ИГРОВОЕ ПРОГРАММНОЕ СРЕДСТВО “КАРТОЧНАЯ ИГРА” ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ.

Предмет: средство для игры по сети.

Объект: игровое взаимодействие через удаленный сервер.

Цель: разработка игрового программного средства, в котором пользователи смогут играть против друг друга удаленно.

Задачи: изучение литературных источников и существующих программных средств по теме, проектирование пользовательского интерфейса, реализация функций для выполнения задания курсового проекта, оценка результатов.

Методология проведения работы: в процессе решения поставленных задач использованы принципы клиент серверного взаимодействия.

Результаты работы: выполнен анализ литературных источников и существующих программных средств по теме; реализованы алгоритмы игрового процесса и клиент-серверного взаимодействия.

Программа предназначена для игры в карточкую игру “Дурак”.

Область применения результатов: личное использование.

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc121956946)

[1 Анализ исходных данных на курсовое проектирование 7](#_Toc121956947)

[1.1 Анализ исходных данных к курсовому проекту 7](#_Toc121956948)

[1.2 Обоснование и описание выбора языка программирования, средств разработки, используемых технологий и сторонних библиотек 7](#_Toc121956949)

[2 Проектирование и разработка программного средства 9](#_Toc121956950)

[2.1 Проектирование архитектуры и описание состояний программного средства 9](#_Toc121956951)

[2.2 Проектирование и разработка графического интерфейса 12](#_Toc121956952)

[3 Эксплуатация программного средства 19](#_Toc121956953)

[3.1 Ввод в эксплуатацию и обоснование минимальных технических требований к оборудованию 19](#_Toc121956954)

[3.2 Руководство по эксплуатации программного средства 20](#_Toc121956955)

[список использованных источников 25](#_Toc121956956)

[Приложение А (обязательное) Проверка на антиплагиат 26](#_Toc121956957)

[Приложение Б (обязательное) Листинг программного кода 27](#_Toc121956958)

[Приложение В (обязательное) Ведомость курсового проекта 47](#_Toc121956959)

# **ВВЕДЕНИЕ**

Сегодня мобильные устройства занимают не последнее место в нашей жизни. В частности мобильные игры. Благодаря мобильности и мощности современных телефонов, они дают возможность играть в игры, по возможностям не сильно уступающим играм на персональных компьютерах, причем играть в совершенно любых местах.

Люди играют пока едут в метро, чтобы скоротать время, в перерывах на работе, чтобы расслабится, на парах в университете, когда им скучно.

Много мобильных игр позволяет играть людям вместе, совместно или против друг друга, находясь на тысячи километров друг от друга.

В данной работе рассматривается разработка игрового программного средства “Карточная игра”.

Цель курсового проекта – дать возможность интересно и просто играть людям вместе, через интернет.

Для достижения цели курсового проекта необходимо решить следующие задачи:

– изучить предметную область;

– построить алгоритм программы;

– разработать интерфейс программы;

– построить архитектуру программы;

– наладить взаимодействие клиента и сервера

–протестировать программу, исправить обнаруженные ошибки.

1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ

* 1. Анализ исходных данных к курсовому проекту

Тема курсовой работы: разработка игрового программного средства “Карточная игра” для пользователей под ОС Android.

Cреда разработки Unity

Язык программирования С#.

Парадигма программирования – объектно-ориентированная.

Способ организации данных – классы.

Способ хранения данных – Player Prefs

К защите курсовой работы представляются:

1. мобильное программное средство;
2. схема алгоритма (формат А2/А3);
3. UML диаграмма классов (плакат, формат А2/А3);
4. диаграмма состояний (плакат, формат А2/А3);
5. структура графического пользовательского интерфейса (плакат, формат А2/А3) и пояснительная записка.

Текст пояснительной записки оформляется в соответствии со стандартом предприятия СТП 01–2017.

1.2 Обоснование и описание выбора языка программирования, средств разработки, используемых технологий и сторонних библиотек

В качестве средства разработки я выбрал Unity. Это межплатформенная среда разработки компьютерных игр, которая позволяет создавать приложения, работающие под более чем 20 различными операционными системами, включающими персональные компьютеры, игровые консоли, мобильные устройства и интернет-приложения.

Преимущества Unity:

* кроссплатформенность;
* удобный и функциональный интерфейс;
* компонентно-ориентированный подход разработки;
* много встроенных библиотек, плагинов и поддерживаемых API;
* наличие отладчика;
* большое сообщество разработчиков.

В Unity уже реализованы многие игровые модули: Физический,

звуковой, графический, менеджер ресурсов, пользовательский ввод и многие другие. Это позволяет быстро и качественно разрабатывать приложение, сосредотачиваться только на постройке игровой архитектуры и логике.

Язык программирования я выбрал исходя из средства разработки. В новых версиях Unity поддерживает только C#.

С# («Си Шарп») – один из наиболее быстро растущих, востребованных и при этом «удобных» языков программирования. Это модификация фундаментального языка С от компании Microsoft, призванная создать наиболее универсальное средство для разработки программного обеспечения для большого количества устройств и операционных систем.

Язык входит в семью С-подобных языков. Синтаксис приближен к Java и C++. Его особенности:

* статистическая типизация;
* поддерживается полиморфизм;
* поддерживается перегрузка операторов;
* доступна делегация, атрибуты, события, обобщенные типы и анонимные функции.

В рамках Unity язык C# отвечает за игровую логику. В Unity представлены базовые классы для взаимодействие с игровым окружением. В частности MonoBehavior, который имеет базовые события. Основные из них:

* Start() – Вызывается при старте сцены;
* Update() – Вызывается при каждом обновлении кадра;
* FixedUpdate() – Вызывается каждые N секунд, не зависит от частоты кадров.

В качестве клиент-серверной части я использую библиотеку Photon Unity Networking. Она позволяет быстро и просто создавать клиент-серверное взаимодействие между игроками. Преимущества Photon Unity Networking:

* Простой и многофункциональный API;
* Встроенный подбор оппонентов;
* Понятная документация;
* Условно бесплатный.

# **2** **ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

**2.1 Проектирование архитектуры и описание состояний программного средства**

Проектирование игрового программного средства состоит из разработки отдельных игровых модулей, разделенных логически между собой. Каждый игровой модуль представляет из себя отдельную сборку – Assembly Defenition.

Каждая сборка компилируется отдельно в .dll файл, что сильно ускоряет время компиляции проекта, так как не нужно компилировать весь код проекта, а только отдельный модуль. Между сборками не может быть циклических связей. Проект состоит из 9 модулей:

* модуль TouchInput;
* модуль UI;
* модуль Audio;
* модуль PhotonUnityNetworking;
* модуль Extention;
* модуль Data;
* модуль Lobby;
* модуль Cards;
* модуль Managament.

Модуль TouchInput содержит в себе класс TouchInput, который отвечает за считываение жестов пользователя и преобразование их в события. Класс CardMovement считывает события класса TouchInput и отвечает за перемещение карт пользователем.

Модуль UI отвечает за элементы пользовательского интерфейса. Содержит в себе классы, отвечающие за анимации элементов, считывание нажатия, отображение списков и прочие. Так же содержит класс PlayerUI, который отвечает за интерфейс игрока.

Модуль Audio содержит класс SoundManager, в котором хранится массив звуков, доступных для проигрывания из любого места программы.

Модуль PhotonUnityNetworking представляет собой импортированную библиотеку для клиент-серверного взаимодействия.

Модуль Extention содержит в себе класс Extention, который содержит методы-расширения для встроенных классов. Например метод Shuffle для IList, который случайно перемешивает лист.

Модуль Data содержиь в себе класс DataBase который отвечает за сохранение пользовательской информцации: имя пользователя, имя комнаты, количество матчей и информацию о каждом сыгранном матче. Для сохранения информации используется встроенный в Unity класс PlayerPrefs, который хранит информацию в виде ключ-значение.

Модуль Lobby представляет собой класс Lobby, который отвечает за работу главное меню, создание и присоединение к комнате, просмотр игровой статистики, изменении псевдонима игрока.

Модуль Cards представляет классы игровых объектов. Класс Card представляет собой сущность карты. Класс Hand представляет собой сущность руки игрока, держащая карты. Класс Deck представляет собой сущность колоды карт. Класс DiscardPile представляет собой сущность отбоя. Класс Table представляет собой сущность игрового стола, который отвечает за прием карты, проверки может ли одна карта побить другую, закончен ли ход, кто победил в данном ходу, кто победил в матче.

Модуль Managament представляет классы, отвечающие за управление игровым процессом. Класс GameContext содержит в себе ссылки на все классы, отвечающие за игровой процесс и связывает их в единое целое. Он подписывается на все события игровых классов и отвечает за их жизненный цикл. Класс HostController отвечает за клиент-серверное взаимодействие, он подписывается на события всех игровых объектов и дублирует нужные событие на другом клиенте.

Диаграмма классов представлена на рисунке 2.2.1

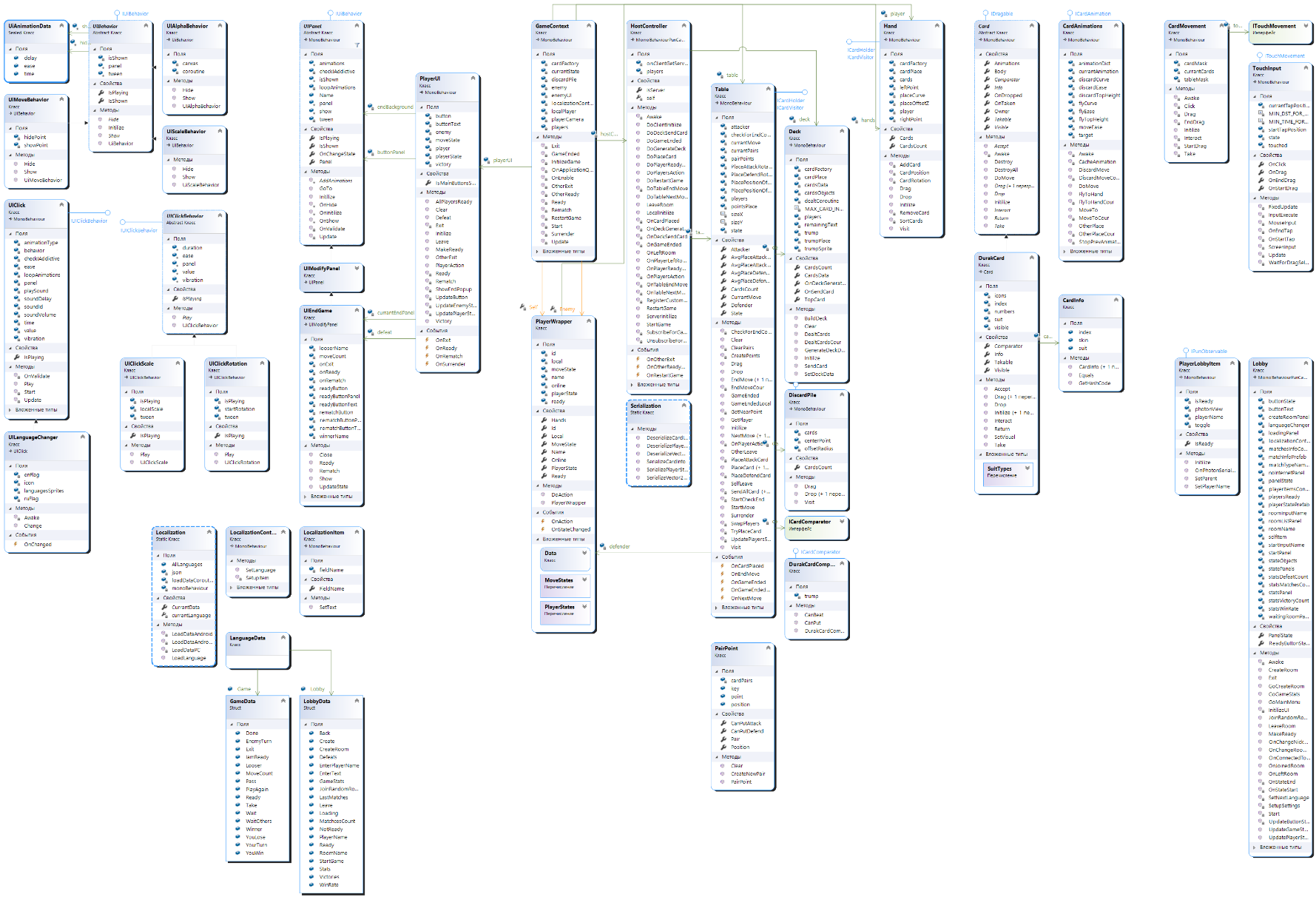


Рисунок 2.1 – Диаграмма классов

Диаграмма состояний — поведенческая диаграмма, отображающая состояния, в которых может находится программное средство, система или объекты этой системы. На диаграмме состояний программного средства отображены следующие ключевые состояния:

* отображение экрана загрузки;
* отображение экрана главного меню;
* отображение экрана создания комнаты;
* отображение экрана присоединения к комнате;
* отображение экрана игровой статистики;
* отображение игрового процесса;
* отображение панели конца игры.

Диаграмма состояний изображена на рисунке 2.2.

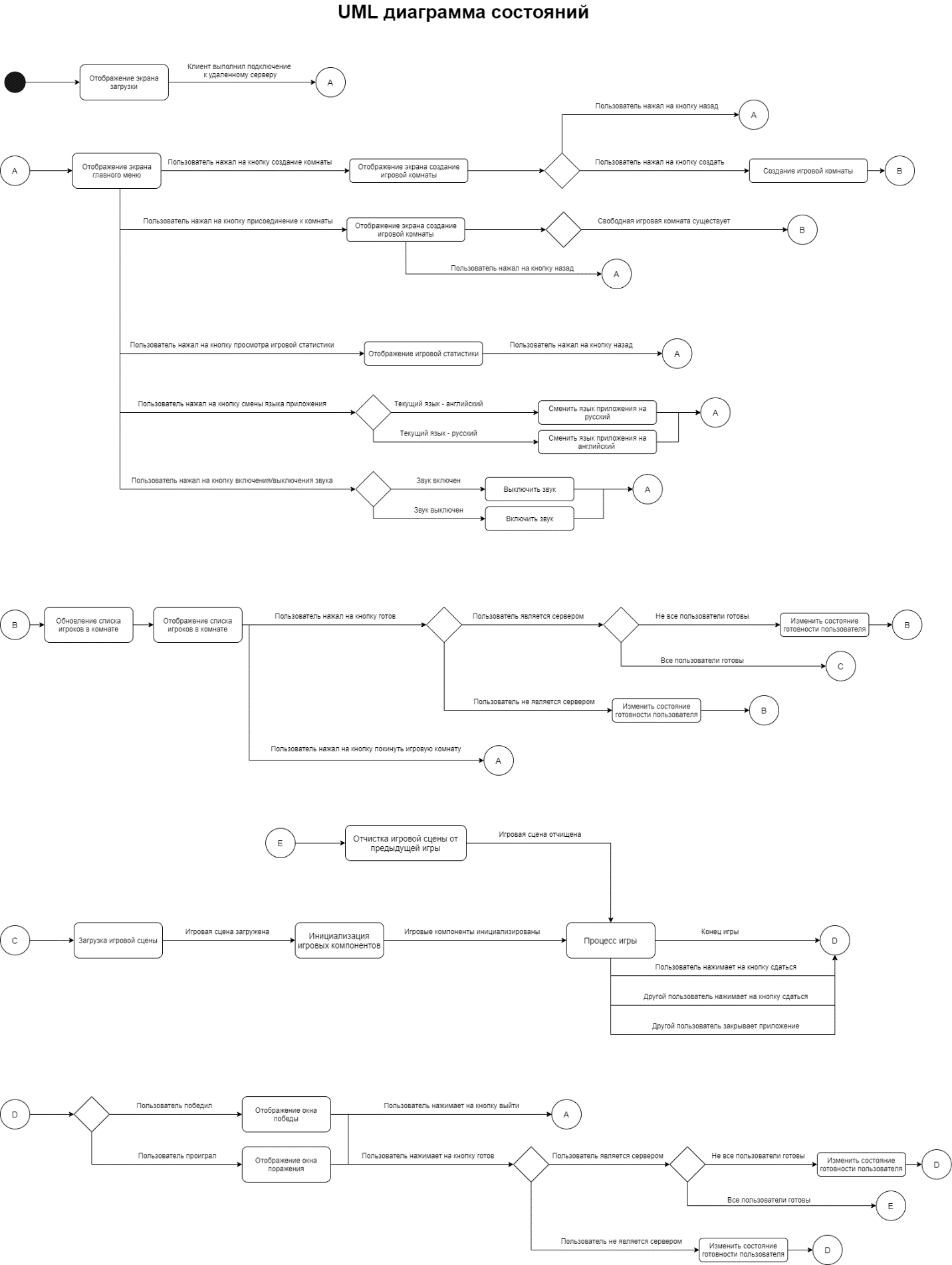


Рисунок 2.2 – UML диаграмма состояний

2.2 Проектирование и разработка графического интерфейса

GUI (Graphical User Interface, графический интерфейс пользователя) –

это форма пользовательского интерфейса , которая позволяет пользователям взаимодействовать с приложением используя графические элементы.

Реализация загрузочного экрана на рисунке 2.3.



Рисунок 2.3 – загрузочный экран

Реализация главного меню на рисунке 2.4.

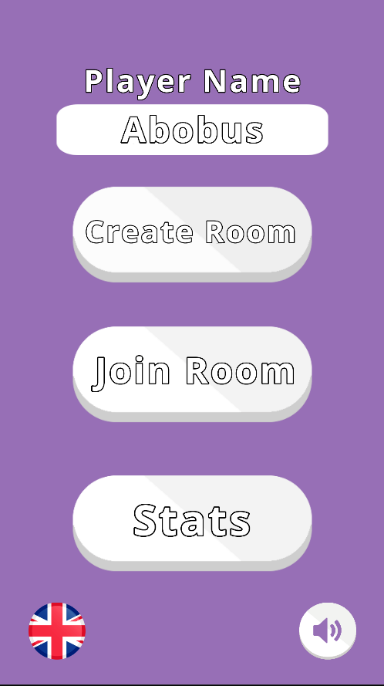


Рисунок 2.4 – главное меню

Реализация меню создания комнаты на рисунке 2.5.



Рисунок 2.5 – меню создания комнаты

Реализация меню присоединения к комнате на рисунке 2.6.



Рисунок 2.6 – меню присоединения к комнате

Реализация меню ожидания игроков на рисунке 2.7.

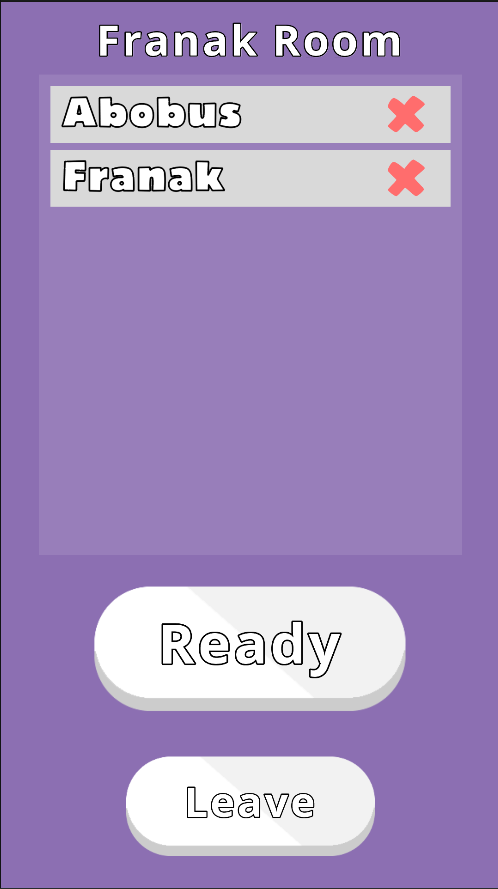


Рисунок 2.7 – меню ожидания игроков

Реализация меню игровой статичтики на рисунке 2.8.



Рисунок 2.8 – меню игровой статистики

Реализация игрового интерфейса на рисунке 2.9.

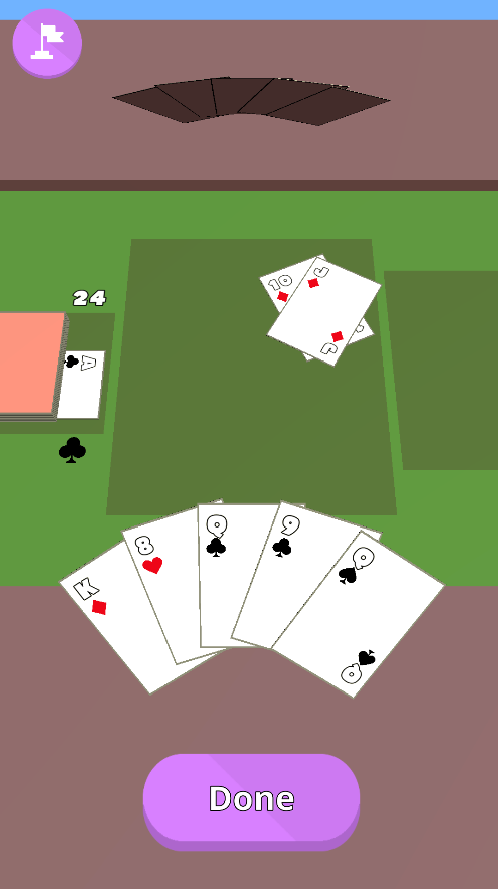


Рисунок 2.10 – игровой интерфейс

Реализация панели конца матча на рисунке 2.11.



Рисунок 2.11 – панель конца матча

**2.3 Описание и реализация используемых в программном средстве**

**алгоритмов.**

Основные аглоритмы, которые используются в данном игровом программном средстве это алгоритны, связаные с игровым процессом.

Схема алгоритма создания игрового поля показана на рисунке 2.12.

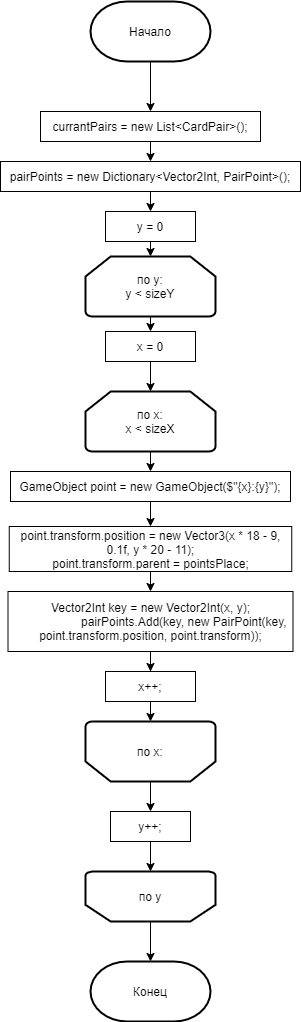


Рисунок 2.12 – блок-схема алгоритма создания игрового поля

Схема алгоритма проверки конца матча показана на рисунке 2.13. Данный метод асинхронный, реализован с помощью Coroutine. Метод вызывается только на клиенте-сервере.

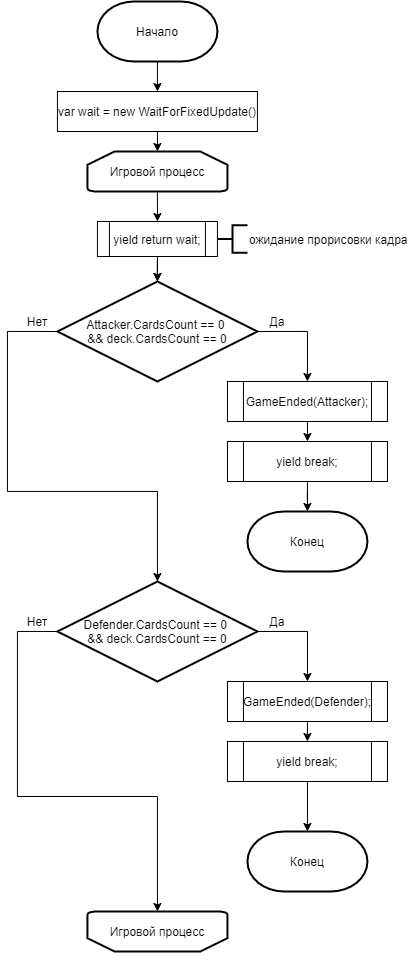


Рисунок 2.13 – блок-схема алгоритма проверки конца матча

Схема алгоритма раздачи карт показана на рисунке 2.14. Данный метод асинхронный, реализован с помощью Coroutine. Метод вызывается только на клиенте-сервере.

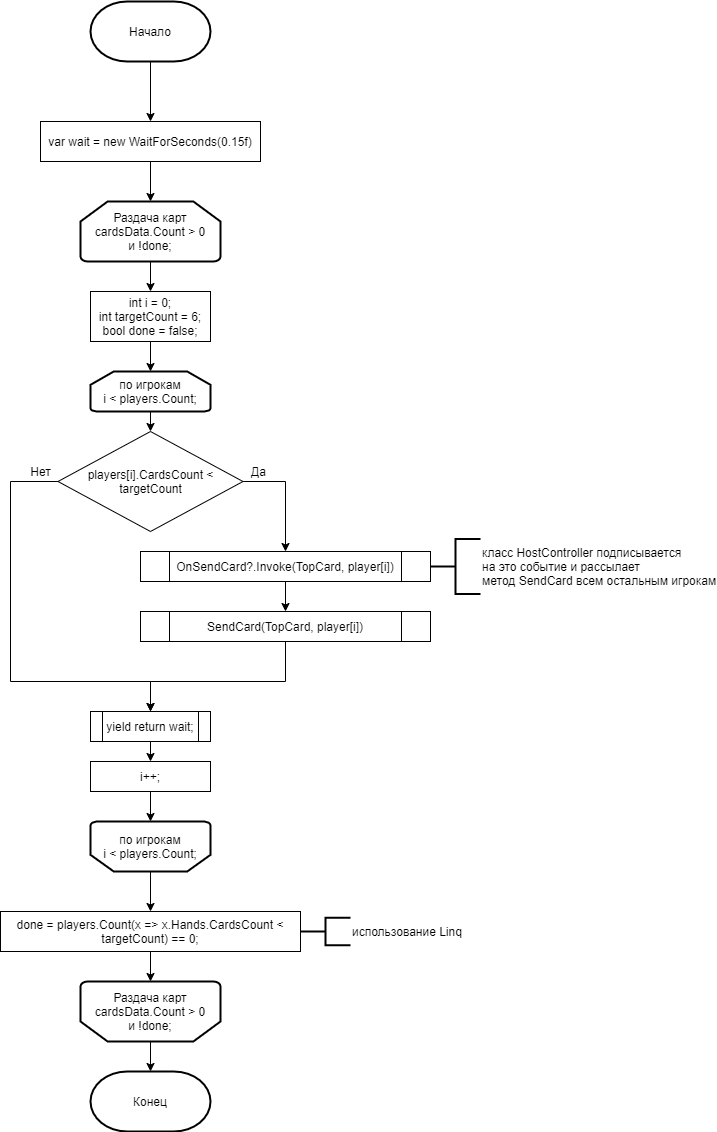


Рисунок 2.14 – блок-схема алгоритма проверки конца матча

3 ЭКСПЛУАТАЦИЯ ПРОГРАММНОГО СРЕДСТВА

* 1. **Ввод в эксплуатацию и обоснование минимальных технических требований к оборудованию**

Для распостранение игры среди пользователей, использующих мобильные устройства на операционной системе Android нужно собрать проект в .apk файл. При установке Unity, так же устанавливаются необходимые SDK, JDK и Gradle для сборки приложения под операционную систему Android.

При сборке проекта Unity сгенерирует дистрибутив приложения (.apk-файл) со всеми необходимыми библиотеками и сериализованными ассетами.

В операционной системе Android минимальные технические требования представлены двумя атрибутами: android:minSdkVersion и android:targetSdkVersion.

Android:minSdkVersion – это минимальное значение версии платформы Android, которое поддерживает приложение.

Android:targetSdkVersion – это значение версии платформы Android, для которого разработано приложение.

Минимальные требования к оборудовнию изображены в таблице 1.

Таблица 1 – минимальные требования к мобильному устройству

|  |  |
| --- | --- |
| **Требование** | **Значение** |
| Android:minSdkVersion | API LEVEL 19 |
| Android:targetSdkVersion | API LEVEL 29 |
| Свободная память | 51мб |
| Подключение к интернету | Да |

3.2 Руководство по эксплуатации программного средства

После установки .apk файла, нужно запустить приложение “Card Game” после чего запуститься загзузочный экран представленный на рисунке 3.1.



Рисунок 3.1 – загрузочный экран

Для работы приложение нужно подключение к интернету для взаимодействия с серверами Photon Networking.

После загрузки приложения игрок попадает в главное меню, представленное на рисунке 3.2.

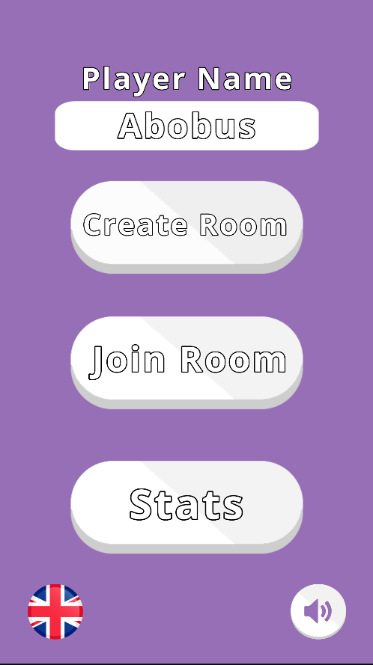


Рисунок 3.2 – главное меню

В главном меню пользователь может изменить свой игровой псевдоним, при первом запуске будет сгенерировано случайное имя. Игрок может изменить язык приложения нажав на круг слева снизу, так же может включить/выключить звук приложения.

Для проведения матчка необходимо два устройства. Первый игрок нажимает на кнопку создания комнаты, затем указывает ее имя –индентификатор, по которому второй игрок сможет к ней присоединится (игрок создавший комнату является клинтом-сервером), после чего нажимает на кнопку создать комнату, изображенную на рисунке 3.3.



Рисунок 3.3 – меню создания комнаты

Второй игрок нажимает на кнопку присоединения к комнате, вписывает нужное имя комнаты(имя комнаты сохраняется и при последующих присоединениях к комнате одно и тоже имя комнаты вписывать не придется) и нажимает на кнопку присоединиться, изображенной на рисунке 3.4.



Рисунок 3.4 – меню присоединения к комнате

После того как игрок создал либо присоединился к комнате откроется меню ожидания игроков, изображенный на рисунке 3.5.

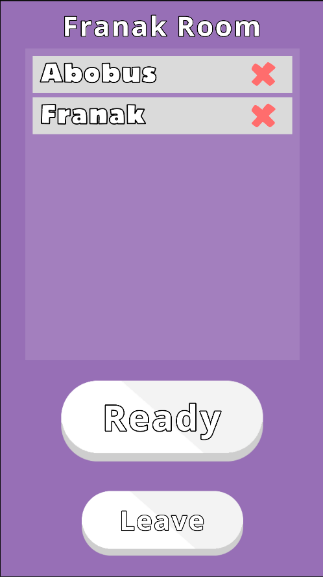


Рисунок 3.5 – меню ожидания игроков

Игроки нажимают на кнопку готовности и после того, как все игроки в комнате готовы, игрок, создавший комнату может запустить матч.

После запуска матча запуститься процесс игры, изображенный на рисунке 3.6.

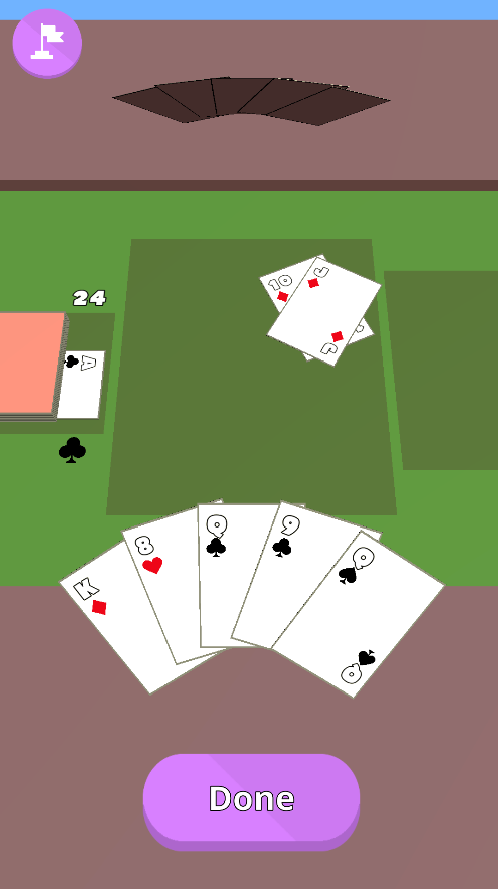


Рисунок 3.6 – игровой процесс

Игроки ходят по очереди в соответствии правил в игре “Дурак”. Атакующий игрок подкидывает карты, а защищающийся отбивается. После конца хода карты сбрасываются в отбой, если все карты отбиты, в противном случае идут в руку отбивающеегося игрока.

Если игрок нажмет кнопку “Сдаться”, или закрывает приложение, то он проигрывает, а его оппонент побеждает, матч заканчивается.

Матч заканчивается как только у одного игрока в руке нету карт и в колоде тоже.

После завершения матча, у игроков появляется панель конца матча, изображенная на рисунке 3.7.



Рисунок 3.7 – панель конца матча

Результат матча записывается в базу данных каждого игрока. У игрока есть два варинта: выйти из игры и играть сново, как только все игроки готовы и игрок, создавший комнату нажимает на кнопку “Играть сново”, игра перезапускается.

После выхода в главное меню пользователь может нажать на кнопку просмотра игровой статистики, после чего откроется меню просмотра игровой статистики, изображенной на рисунке 3.8.



Рисунок 3.8 – меню игровой статистики

**ЗАКЛЮЧЕНИЕ**

В процессе реализации всех условий курсового проекта были достигнуты все поставленные цели и в результате создано игрового программное, позволяющее играть в карточкую игру “Дурак” против друг друга.

Для реализации клиент серверного взаимодействия, необходимым стало осваивание библиотеки Photon Networking.

Были выполнены все поставленные задачи: изучение литературных источников и аналоговых программных средств по данной теме, проектирование игровых модулей, клиент-серверного взаимодействия и пользовательского интерфейса, тестирование программы и оценка результатов.

В результате выполнения курсового проекта был получен работающие, готовое к использованию приложение, удовлетворяющий всем требованиям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Документация Unity [Электронный ресурс]. – Режим доступа:

https://docs.unity3d.com/2020.3/Documentation/Manual/index.html/. Дата доступа: 06.11.2022.

[2] Документация Photon Unity Networking [Электронный ресурс]. – Режим доступа: https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro/. Дата доступа: 07.11.2022.

[3] Онлайн игра: полный туториал, часть 1 [Unity 3D] [Photon] - матчмейкинг и простая синхронизация [Электронный ресурс]. – Режим доступа:

https://www.youtube.com/watch?v=aqxzrk-pl-8&ab\_channel=EmeraldPowder/. Дата доступа: 13.11.2022.

[4] Онлайн игра: полный туториал, часть 2 [Unity 3D][Photon] - произвольные параметры и игровая карта [Электронный ресурс]. – Режим доступа:

https://www.youtube.com/watch?v=1q2x0IAvLBQ&ab\_channel=EmeraldPowder/. Дата доступа: 19.11.2022.

[5] Онлайн игра: полный туториал, часть 3 [Unity 3D] [Photon] - события и перемещение персонажей [Электронный ресурс]. – Режим доступа:

https://www.youtube.com/watch?v=\_FkeQyW8mtk&t=12s&ab\_channel=EmeraldPowder/. Дата доступа: 25.11.2022.

[6] Онлайн игра: полный туториал, часть 3 [Unity 3D] [Photon] - события и перемещение персонажей [Электронный ресурс]. – Режим доступа:

https://www.youtube.com/watch?v=\_EF8HH2MMEg&t=124s/. Дата доступа: 26.11.2022.

[7] Онлайн игра: полный туториал, часть 3 [Unity 3D] [Photon] - события и перемещение персонажей [Электронный ресурс]. – Режим доступа:

https://www.youtube.com/watch?v=CzRFhoRPE4k&t=64s/. Дата доступа: 27.11.2022.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Проверка на антиплагиат**

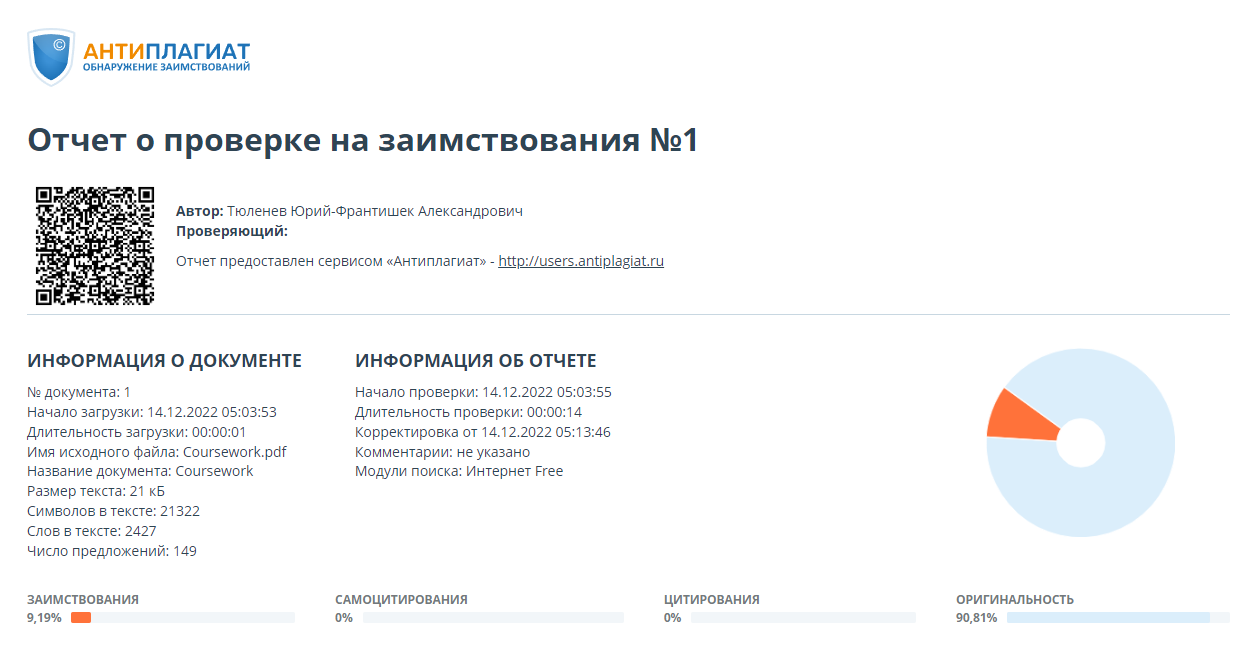
****

Рисунок А.1 – Результат проверки системы "Антиплагиат"

ПРИЛОЖЕНИЕ Б

**(обязательное)**

**Листинг кода**

namespace Managament

{

public class GameContext : MonoBehaviour

{

#region Links

[Header("State")]

[SerializeField] private States currantState;

[Header("Links")]

[SerializeField] private HostController hostController;

[SerializeField] private LocalizationController localizationController;

[Space]

[SerializeField] private Table table;

[SerializeField] private PlayerCamera playerCamera;

[Space]

[SerializeField] private Hand player;

[SerializeField] private PlayerUI playerUI;

[Space]

[SerializeField] private Hand enemy;

[SerializeField] private PlayerUI enemyUI; //Only for tests

[Space]

[SerializeField] private Deck deck;

[SerializeField] private DiscardPile discardPile;

[Space]

[SerializeField] private CardFactory cardFactory;

#endregion

[Header("Players")]

[SerializeField] private List<PlayerWrapper> players;

private bool localPlayer = true;

private PlayerWrapper Self

{

get => players.First(x => x.Local == true);

}

private PlayerWrapper Enemy

{

get => players.First(x => x.Local != true);

}

private void OnEnable()

{

hostController.OnOtherReadyChanged += OtherReady;

hostController.OnOtherExit += OtherExit;

hostController.OnRestartGame += RestartGame;

table.OnGameEndedLocal += GameEnded;

playerUI.OnRematch += Rematch;

playerUI.OnReady += Ready;

playerUI.OnSurrender += Surrender;

playerUI.OnExit += Exit;

localizationController.SetLanguage(DataBase.Language, LocalizationController.Place.Game);

}

private void Start()

{

hostController.ServerInitilize(InitilizeGame);

}

private void InitilizeGame(HostController.Data serverData)

{

DurakCard.SuitTypes trump = serverData.Trump;

players = hostController.LocalInitilize(table, deck, player, enemy);

bool isOffline = players.Count < 2;

if (isOffline)

{

players.Add(new PlayerWrapper(404, enemy, "Local enemy"));

Debug.LogWarning("Test game due to 1 player");

}

table.Initilize(players, deck, discardPile, new DurakCardComparator(trump));

deck.Initilize(players, cardFactory, trump);

playerUI.Initilize(Self, Enemy);

enemyUI.Initilize(Enemy, Self); //Debug

currantState = States.Game;

hostController.StartGame(isOffline);

AudioListener.pause = DataBase.SoundMuted;

}

private void GameEnded(Table.MatchData data)

{

DataBase.MatchData matchData = new DataBase.MatchData

(

data.Winner.Name,

data.Looser.Name,

data.Winner == Self,

data.MoveCount,

(int)data.EndType

);

DataBase.RecordGame(matchData);

currantState = States.Ended;

if (data.Winner.Local)

{

playerUI.Victory(data, hostController.isServer);

}

else

{

playerUI.Defeat(data, hostController.isServer);

}

}

private void RestartGame()

{

Card.DestroyAll();

playerUI.Clear();

enemyUI.Clear();

table.Clear();

deck.Clear();

hostController.ServerInitilize(InitilizeGame);

}

private void OtherExit(PlayerWrapper other)

{

switch (currantState)

{

case States.Ended:

playerUI.OtherExit();

break;

case States.Game:

table.OtherLeave(Self);

break;

}

}

private void Exit()

{

switch (currantState)

{

case States.Game:

table.SelfLeave(Self);

break;

}

hostController.LeaveRoom();

}

private void OtherReady(PlayerWrapper other)

{

if (!hostController.isServer)

return;

Self.Ready = true;

if (players.Count(x => x.Ready != true) == 0)

{

playerUI.AllPlayersReady();

}

}

private void Ready()

{

if (Self.Ready)

return;

Self.Ready = true;

hostController.OnPlayerReadyChanged(Self);

playerUI.MakeReady();

}

private void Surrender()

{

table.Surrender(Self);

}

private void Rematch()

{

if (!hostController.isServer)

return;

Self.Ready = true;

if(players.Count(x => x.Ready != true) == 0)

{

hostController.RestartGame();

}

}

namespace Managament

{

public class HostController : MonoBehaviourPunCallbacks

{

public bool isServer => PhotonNetwork.IsMasterClient;

private Table table;

private Deck deck;

private List<PlayerWrapper> players;

private System.Action<Data> onClientGetServerData;

private PlayerWrapper self

{

get => players.First(x => x.Local);

}

public event System.Action OnRestartGame;

public event System.Action<PlayerWrapper> OnOtherReadyChanged;

public event System.Action<PlayerWrapper> OnOtherExit;

private void Awake()

{

RegisterCustomTypes();

}

public void ServerInitilize(System.Action<Data> onClientGetServerData)

{

this.onClientGetServerData = onClientGetServerData;

if(!PhotonNetwork.IsConnected || !PhotonNetwork.InRoom)

{

onClientGetServerData?.Invoke(new Data(Random.Range(0, 4)));

return;

}

if (PhotonNetwork.IsMasterClient)

{

object data = new object[1]

{

Random.Range(0, 4),

};

photonView.RPC(nameof(DoClientInitilize), RpcTarget.AllBuffered, data);

}

}

public void StartGame(bool offline = false)

{

if (PhotonNetwork.IsMasterClient || offline)

{

deck.GenerateDeckData();

deck.BuildDeck();

table.StartMove();

}

}

private void UnsubscribeForGameActions()

{

deck.OnDeckGenerated -= OnDeckGenerated;

deck.OnSendCard -= OnDeckSendCard;

table.OnNextMove -= OnTableNextMove;

table.OnEndMove -= OnTableEndMove;

table.OnCardPlaced -= OnCardPlaced;

table.OnGameEnded -= OnGameEnded;

}

private void SubscribeForGameActions()

{

self.OnAction += OnPlayersAction;

deck.OnDeckGenerated += OnDeckGenerated;

deck.OnSendCard += OnDeckSendCard;

table.OnNextMove += OnTableNextMove;

table.OnEndMove += OnTableEndMove;

table.OnCardPlaced += OnCardPlaced;

table.OnGameEnded += OnGameEnded;

}

private void RegisterCustomTypes()

{

PhotonPeer.RegisterType(typeof(Vector2Int), 10, Serialization.SerializeVector2Int, Serialization.DeserializeVector2Int);

PhotonPeer.RegisterType(typeof(CardInfo), 11, Serialization.SerializeCardInfo, Serialization.DeserializeCardInfo);

PhotonPeer.RegisterType(typeof(PlayerWrapper.Data), 12, Serialization.SerializePlayerState, Serialization.DeserializePlayerState);

}

#region Lobby-Networking

public void OnPlayerReadyChanged(PlayerWrapper player)

{

photonView.RPC(nameof(DoPlayerReadyChanged), RpcTarget.MasterClient, new object[2] { player.Id, player.Ready });

}

[PunRPC]

public void DoPlayerReadyChanged(object[] data)

{

int id = (int)data[0];

PlayerWrapper player = players.First(x => x.Id == id);

player.Ready = (bool)data[1];

OnOtherReadyChanged?.Invoke(player);

}

public void LeaveRoom()

{

PhotonNetwork.LeaveRoom();

}

public override void OnLeftRoom()

{

SceneManager.LoadScene(0);

}

public override void OnPlayerLeftRoom(Player otherPlayer)

{

PlayerWrapper player = players.First(x => x.Id == otherPlayer.ActorNumber);

player.Online = false;

OnOtherExit?.Invoke(player);

}

public void RestartGame()

{

photonView.RPC(nameof(DoRestartGame), RpcTarget.All);

}

[PunRPC]

public void DoRestartGame()

{

UnsubscribeForGameActions();

OnRestartGame?.Invoke();

}

#endregion

#region Game-Networking

public List<PlayerWrapper> LocalInitilize(Table table, Deck deck, Hand player, Hand enemy)

{

this.table = table;

this.deck = deck;

players = new List<PlayerWrapper>()

{

new PlayerWrapper(PhotonNetwork.LocalPlayer.ActorNumber, player, PhotonNetwork.LocalPlayer.NickName, true),

};

if (PhotonNetwork.PlayerListOthers.Length > 0)

{

Player other = PhotonNetwork.PlayerListOthers[0];

players.Add(new PlayerWrapper(other.ActorNumber, enemy, other.NickName));

}

else

{

PhotonNetwork.OfflineMode = true;

}

players = players.OrderBy(x => x.Id).ToList();

SubscribeForGameActions();

return players;

}

[PunRPC]

public void DoClientInitilize(object[] info)

{

Data data = new Data((int)info[0]);

onClientGetServerData?.Invoke(data);

}

private void OnGameEnded(Table.MatchData data)

{

object[] info = new object[4]

{

data.Winner.Id,

data.Looser.Id,

data.MoveCount,

data.EndType,

};

photonView.RPC(nameof(DoGameEnded), RpcTarget.All, info);

}

[PunRPC]

public void DoGameEnded(object[] data)

{

PlayerWrapper winner = players.First(x => x.Id == (int)data[0]);

PlayerWrapper looser = players.First(x => x.Id == (int)data[1]);

Table.MatchData info = new Table.MatchData(winner, looser, (int)data[2], (Table.MatchData.EndTypes)data[3]);

table.GameEndedLocal(info);

}

#endregion

#region Table-Networking

private void OnCardPlaced(PairPoint point, Card card)

{

object[] data = new object[3]

{

point.key,

card.Info,

card.Owner.Id,

};

photonView.RPC(nameof(DoPlaceCard), RpcTarget.Others, data);

}

[PunRPC]

public void DoPlaceCard(object[] data)

{

Vector2Int key = (Vector2Int)data[0];

CardInfo info = (CardInfo)data[1];

PlayerWrapper player = players.First(x => x.Id == (int)data[2]);

table.PlaceCard(key, info, player);

}

private void OnPlayersAction(PlayerWrapper player)

{

photonView.RPC(nameof(DoPlayersAction), RpcTarget.Others, player.Id);

}

[PunRPC]

public void DoPlayersAction(object data)

{

int id = (int)data;

PlayerWrapper player = players.First(x => x.Id == id);

player.DoAction();

}

private void OnTableNextMove(int move, PlayerWrapper attacker, PlayerWrapper defender)

{

if (PhotonNetwork.IsMasterClient)

{

object[] info = new object[3]

{

move,

new PlayerWrapper.Data(attacker),

new PlayerWrapper.Data(defender),

};

photonView.RPC(nameof(DoTableNextMove), RpcTarget.Others, info);

}

}

[PunRPC]

public void DoTableNextMove(object[] info)

{

table.NextMove((int)info[0], (PlayerWrapper.Data)(info[1]), (PlayerWrapper.Data)(info[2]));

}

private void OnTableEndMove(Table.States endState)

{

if(PhotonNetwork.OfflineMode)

{

table.EndMove(endState);

return;

}

object[] data = new object[2]

{

0,

(int)endState,

};

photonView.RPC(nameof(DoTableEndMove), RpcTarget.All, data);

}

[PunRPC]

private void DoTableEndMove(object[] info)

{

int move = (int)info[0];

Table.States endState = (Table.States)info[1];

if (PhotonNetwork.IsMasterClient)

{

table.EndMove(endState);

}

else

{

table.EndMove(endState, move);

}

}

#endregion

#region Deck-Networking

private void OnDeckSendCard(CardInfo cardInfo, PlayerWrapper player)

{

if (PhotonNetwork.IsMasterClient)

{

object[] info = new object[2]

{

cardInfo,

player.Id

};

photonView.RPC(nameof(DoDeckSendCard), RpcTarget.Others, info);

}

}

[PunRPC]

public void DoDeckSendCard(object[] info)

{

PlayerWrapper player = players.Where(x => x.Id == (int)info[1]).First();

deck.SendCard((CardInfo)info[0], player);

}

private void OnDeckGenerated(List<CardInfo> cards)

{

object[] data = new object[cards.Count];

for (int i = 0; i < cards.Count; i++)

{

data[i] = cards[i];

}

photonView.RPC(nameof(DoGenerateDeck), RpcTarget.Others, data);

}

[PunRPC]

public void DoGenerateDeck(object[] info)

{

List<CardInfo> cards = new List<CardInfo>();

for (int i = 0; i < info.Length; i++)

{

cards.Add((CardInfo)info[i]);

}

deck.SetDeckData(cards);

}

#endregion

public class Data

{

public Data(int trump)

{

Trump = (DurakCard.SuitTypes)trump;

}

public readonly DurakCard.SuitTypes Trump;

}

}

}

namespace Cards

{

public class Table : MonoBehaviour, ICardHolder, ICardVisitor

{

private const int sizeX = 2;

private const int sizeY = 2;

[Header("States")]

[SerializeField] private States state;

[SerializeField] private int currantMove = 0;

[SerializeField] private PlayerWrapper attacker;

[SerializeField] private PlayerWrapper defender;

[Header("Placement")]

[SerializeField] private Vector2 PlaceAttackRotationOffset;

[SerializeField] private Vector2 PlaceDefendRotationOffset;

[Space]

[SerializeField] private Vector2 PlacePositionOffsetX;

[SerializeField] private Vector2 PlacePositionOffsetY;

[Header("Components")]

[SerializeField] private Transform pointsPlace;

private Deck deck;

private DiscardPile discardPile;

private List<PlayerWrapper> players;

private Coroutine checkForEndCoroutine;

private Dictionary<Vector2Int, PairPoint> pairPoints;

private List<CardPair> currantPairs;

private ICardComparator comparator;

public States State

{

get => state;

set

{

state = value;

}

}

public int CardsCount => 0;

public int CurrantMove

{

get => currantMove;

set

{

currantMove = value;

attacker = Attacker;

defender = Defender;

}

}

public PlayerWrapper GetPlayer(Card card)

{

return card.Owner;

} //fix

public PlayerWrapper Attacker

{

get => attacker;

private set => attacker = value;

}

public PlayerWrapper Defender

{

get => defender;

private set => defender = value;

}

public float AvgPlaceAttackRotation

{

get => 0;

//get => (PlaceAttackRotationOffset.x + PlaceAttackRotationOffset.y) / 2;

}

public float AvgPlaceDefendRotation

{

get => (PlaceDefendRotationOffset.x + PlaceDefendRotationOffset.y) / 2;

}

public Vector3 AvgPlaceAttackPosition

{

get => new Vector3(-(PlacePositionOffsetX.x + PlacePositionOffsetX.y) / 2, 0, (PlacePositionOffsetY.x + PlacePositionOffsetY.y));

}

public Vector3 AvgPlaceDefendPosition

{

get => new Vector3((PlacePositionOffsetX.x + PlacePositionOffsetX.y) / 2, 0, -(PlacePositionOffsetY.x + PlacePositionOffsetY.y));

}

public event System.Action<int, PlayerWrapper, PlayerWrapper> OnNextMove;

public event System.Action<States> OnEndMove;

public event System.Action<PairPoint, Card> OnCardPlaced;

public event System.Action<MatchData> OnGameEnded;

public event System.Action<MatchData> OnGameEndedLocal;

#region Initilize

public void Initilize(List<PlayerWrapper> players, Deck deck, DiscardPile discardPile, ICardComparator comparator)

{

this.players = players;

this.discardPile = discardPile;

this.comparator = comparator;

this.deck = deck;

foreach(PlayerWrapper player in players)

{

player.OnAction += OnPlayerAction;

player.PlayerState = PlayerWrapper.PlayerStates.Idle;

player.MoveState = PlayerWrapper.MoveStates.Idle;

}

CreatePoints();

}

public void Clear()

{

foreach(CardPair pair in currantPairs)

{

pair.Destroy();

}

}

private void CreatePoints()

{

currantPairs = new List<CardPair>();

pairPoints = new Dictionary<Vector2Int, PairPoint>();

for (int y = 0; y < sizeY; y++)

{

for (int x = 0; x < sizeX; x++)

{

GameObject point = new GameObject($"{x}:{y}");

point.transform.position = new Vector3(x \* 18 - 9, 0.1f, y \* 20 - 11);

point.transform.parent = pointsPlace;

Vector2Int key = new Vector2Int(x, y);

pairPoints.Add(key, new PairPoint(key, point.transform.position, point.transform));

}

}

}

#endregion

#region CardActions

public void Drop(IDragable card, DropCardData data)

{

card.Accept(this, data);

}

public void Visit(Card card, object data = null)

{

try

{

DropCardData info = data as DropCardData;

TryPlaceCard(card);

}

catch { }

}

public void Drag(IDragable card, MoveInfo info)

{

PairPoint min = GetNearPoint(info.position);

Vector3 position = Vector3.Lerp(info.position, min.Position, 0.75f);

position.y += 0.5f;

Quaternion rotation = Quaternion.identity;

if (min.Pair != null && !min.Pair.Done)

{

rotation = Quaternion.Euler(0, AvgPlaceDefendRotation, 0);

position += AvgPlaceDefendPosition;

}

else

{

rotation = Quaternion.Euler(0, AvgPlaceAttackRotation, 0);

position += AvgPlaceAttackPosition;

}

card.Body.position = Vector3.Lerp(card.Body.position, position, 0.3f);

card.Body.rotation = Quaternion.Lerp(card.Body.rotation, rotation, 0.3f);

}

private PairPoint GetNearPoint(Vector3 position)

{

PairPoint min = null;

float lenght = int.MaxValue;

foreach (PairPoint point in pairPoints.Values)

{

if ((position - point.position).magnitude < lenght)

{

min = point;

lenght = (position - point.position).magnitude;

}

}

return min;

}

public void PlaceCard(Vector2Int pointKey, CardInfo info, PlayerWrapper player)//Accept

{

Debug.Log($"C: {player.Hands.Cards.Count} | {info.index} {info.suit}");

Card card = player.Hands.Cards.First(x => x.Info.Equals(info));

pointKey = new Vector2Int(sizeX - 1, sizeY - 1) - pointKey;

PairPoint point = pairPoints[pointKey];

card.OnDropped?.Invoke(card);

PlaceCard(card, point);

UpdatePlayersStates();

}

public void PlaceCard(Card card, PairPoint point)//Accept

{

if (card.Owner == Attacker)

{

bool firstMove = currantPairs.Count == 0;

bool canPut = currantPairs.Count(x => comparator.CanPut(card, x)) > 0;

if (point.CanPutAttack && (firstMove || canPut))

{

PlaceAttackCard(point, card, ICardAnimation.Types.OtherPlace, 0.75f);

}

}

else if (card.Owner == Defender)

{

if (point.CanPutDefend && point.Pair.CanBeat(card))

{

PlaceDefendCard(point, card, ICardAnimation.Types.OtherPlace, 0.75f);

}

else

{

card.Owner.Hands.Drop(card, new DropCardData { Sender = DropCardData.SenderTypes.Self });

}

}

else

{

Debug.LogError("Card have no owner!");

}

UpdatePlayersStates();

}

private void TryPlaceCard(Card card)//Send

{

if(card.Owner == Attacker)

{

PairPoint point = GetNearPoint(card.Body.position);

bool firstMove = currantPairs.Count == 0;

bool canPut = currantPairs.Count(x => comparator.CanPut(card, x)) > 0;

if(point.CanPutAttack && (firstMove || canPut))

{

OnCardPlaced?.Invoke(point, card);

PlaceAttackCard(point, card, ICardAnimation.Types.MoveTo, 0.25f);

}

else

{

card.Owner.Hands.Drop(card, new DropCardData { Sender = DropCardData.SenderTypes.Self });

}

}

else if(card.Owner == Defender)

{

PairPoint point = GetNearPoint(card.Body.position);

if (point.CanPutDefend && point.Pair.CanBeat(card))

{

OnCardPlaced?.Invoke(point, card);

PlaceDefendCard(point, card, ICardAnimation.Types.MoveTo, 0.25f);

}

else

{

card.Owner.Hands.Drop(card, new DropCardData { Sender = DropCardData.SenderTypes.Self });

}

}

else

{

Debug.LogError("Card have no owner!");

}

UpdatePlayersStates();

}

private void PlaceAttackCard(PairPoint point, Card card, ICardAnimation.Types animation, float time)

{

CardPair pair = new CardPair(card, comparator);

point.CreateNewPair(pair);

currantPairs.Add(pair);

Vector3 offset = new Vector3(-Random.Range(PlacePositionOffsetX.x, PlacePositionOffsetX.y), 0f, Random.Range(PlacePositionOffsetY.x, PlacePositionOffsetY.y));

Vector3 position = point.Position + offset;

Quaternion rotation = Quaternion.Euler(0, Random.Range(PlaceAttackRotationOffset.x, PlaceAttackRotationOffset.y), 0);

card.DoMove(animation, position, rotation, time, ICardAnimation.Order.Override);

SoundManagment.PlaySound("place");

}

private void PlaceDefendCard(PairPoint point, Card card, ICardAnimation.Types animation, float time)

{

point.Pair.Beat(card);

Vector3 offset = new Vector3(Random.Range(PlacePositionOffsetX.x, PlacePositionOffsetX.y), 0.15f, -Random.Range(PlacePositionOffsetY.x, PlacePositionOffsetY.y));

Vector3 position = point.Position + offset;

Debug.Log(point.Position);

Quaternion rotation = Quaternion.Euler(0, Random.Range(PlaceDefendRotationOffset.x, PlaceDefendRotationOffset.y), 0);

card.DoMove(animation, position, rotation, time, ICardAnimation.Order.Override);

SoundManagment.PlaySound("place");

}

private void SendAllCard(PlayerWrapper player)

{

foreach (CardPair pair in currantPairs)

{

if (pair.Attacker)

{

player.Hands.Drop(pair.Attacker, new DropCardData { Sender = DropCardData.SenderTypes.Table });

}

if (pair.Defender)

{

player.Hands.Drop(pair.Defender, new DropCardData { Sender = DropCardData.SenderTypes.Table });

}

}

SoundManagment.PlaySound("take");

ClearPairs();

}

private void SendAllCard(DiscardPile discardPile)

{

discardPile.Drop(currantPairs);

SoundManagment.PlaySound("discard");

ClearPairs();

}

private void ClearPairs()

{

currantPairs.Clear();

foreach(PairPoint pair in pairPoints.Values)

{

pair.Clear();

}

}

#endregion

#region GameStates

public enum States { Idle, Game, Defended, NotDefended, Ended }

public void StartMove() //Send

{

if(Random.Range(0, 2) == 0)

{

Attacker = players[0];

Defender = players[1];

}

else

{

Attacker = players[1];

Defender = players[0];

}

State = States.Game;

Attacker.PlayerState = PlayerWrapper.PlayerStates.Attacker;

Defender.PlayerState = PlayerWrapper.PlayerStates.Defender;

deck.DealtCards();

UpdatePlayersStates();

StartCheckEnd();

OnNextMove?.Invoke(CurrantMove, Attacker, Defender);

}

public void NextMove() //Send

{

if (State == States.Idle)

{

CurrantMove++;

State = States.Game;

Attacker.PlayerState = PlayerWrapper.PlayerStates.Attacker;

Defender.PlayerState = PlayerWrapper.PlayerStates.Defender;

deck.DealtCards();

UpdatePlayersStates();

OnNextMove?.Invoke(CurrantMove, Attacker, Defender);

}

}

private void StartCheckEnd()

{

if (checkForEndCoroutine != null)

{

StopCoroutine(checkForEndCoroutine);

}

checkForEndCoroutine = StartCoroutine(CheckForEndCour());

}

private IEnumerator CheckForEndCour()

{

var wait = new WaitForSeconds(0.5f);

while(true)

{

yield return wait;

if(Attacker.Hands.CardsCount == 0 && deck.CardsCount == 0)

{

GameEnded(Attacker);

yield break;

}

if (Defender.Hands.CardsCount == 0 && deck.CardsCount == 0)

{

GameEnded(Defender);

yield break;

}

}

}

public void NextMove(int move, PlayerWrapper.Data attacker, PlayerWrapper.Data defender) //Accept

{

CurrantMove = move;

State = States.Game;

Debug.Log(attacker.id + " " + defender.id);

Attacker = players.First(x => x.Id == attacker.id);

Defender = players.First(x => x.Id == defender.id);

Attacker.PlayerState = attacker.playerState;

Attacker.MoveState = attacker.moveState;

Defender.PlayerState = defender.playerState;

Defender.MoveState = defender.moveState;

UpdatePlayersStates();

}

public void EndMove(States endState)

{

if (State == States.Game)

{

State = States.Idle;

StartCoroutine(EndMoveCour(endState));

}

} //Send

private IEnumerator EndMoveCour(States endState)

{

Debug.Log("Next Move: " + CurrantMove);

switch (endState)

{

case States.Defended:

SendAllCard(discardPile);

SwapPlayers();

break;

case States.NotDefended:

SendAllCard(Defender);

break;

}

Attacker.PlayerState = PlayerWrapper.PlayerStates.Idle;

Attacker.MoveState = PlayerWrapper.MoveStates.Idle;

Defender.PlayerState = PlayerWrapper.PlayerStates.Idle;

Defender.MoveState = PlayerWrapper.MoveStates.Idle;

yield return new WaitForSeconds(0.5f);

NextMove();

yield break;

}

public void EndMove(States endState, int move)

{

Debug.Log("End Move");

State = States.Idle;

switch (endState)

{

case States.Defended:

SendAllCard(discardPile);

break;

case States.NotDefended:

SendAllCard(Defender);

break;

}

Attacker.PlayerState = PlayerWrapper.PlayerStates.Idle;

Attacker.MoveState = PlayerWrapper.MoveStates.Idle;

Defender.PlayerState = PlayerWrapper.PlayerStates.Idle;

Defender.MoveState = PlayerWrapper.MoveStates.Idle;

} //Accept

private void UpdatePlayersStates() //Local only

{

if(pairPoints.Values.Count(x => x.Pair != null) == 0)

{

Attacker.MoveState = PlayerWrapper.MoveStates.FirstMove;

Defender.MoveState = PlayerWrapper.MoveStates.FirstMove;

return;

}

if (Attacker.MoveState == PlayerWrapper.MoveStates.Pass || Defender.MoveState == PlayerWrapper.MoveStates.Pass)

return;

Attacker.MoveState = PlayerWrapper.MoveStates.Playing;

if (pairPoints.Values.Count(x => x.CanPutDefend) > 0)

{

Defender.MoveState = PlayerWrapper.MoveStates.Playing;

}

else

{

Defender.MoveState = PlayerWrapper.MoveStates.EnemyMove;

}

}

private void SwapPlayers()

{

PlayerWrapper temp = Attacker;

Attacker = Defender;

Defender = temp;

}

#endregion

#region PlayerActions

private void OnPlayerAction(PlayerWrapper player)

{

if (State != States.Game)

return;

PlayerWrapper other;

switch (player.PlayerState)

{

case PlayerWrapper.PlayerStates.Attacker:

other = Defender;

switch(player.MoveState)

{

case PlayerWrapper.MoveStates.Playing: //Done

if (currantPairs.Count(x => !x.Done) == 0)

{

OnEndMove?.Invoke(States.Defended);

}

break;

case PlayerWrapper.MoveStates.Pass:

OnEndMove?.Invoke(States.NotDefended);

break;

}

break;

case PlayerWrapper.PlayerStates.Defender:

other = Attacker;

switch (player.MoveState)

{

case PlayerWrapper.MoveStates.Playing: //Pass

Defender.MoveState = PlayerWrapper.MoveStates.Pass;

Attacker.MoveState = PlayerWrapper.MoveStates.Pass;

break;

case PlayerWrapper.MoveStates.Pass: //Take Cards

if (currantPairs.Count(x => !x.Done) > 0)

{

OnEndMove?.Invoke(States.NotDefended);

}

break;

}

break;

}

}

public void SelfLeave(PlayerWrapper self)//Local

{

State = States.Ended;

PlayerWrapper winner = players.First(x => x.Id != self.Id);

MatchData data = new MatchData(winner, self, currantMove, MatchData.EndTypes.Exit);

GameEndedLocal(data);

}

public void OtherLeave(PlayerWrapper self)//Local

{

State = States.Ended;

PlayerWrapper looser = players.First(x => x.Id != self.Id);

MatchData data = new MatchData(self, looser, currantMove, MatchData.EndTypes.Exit);

GameEndedLocal(data);

}

public void Surrender(PlayerWrapper self) //Send

{

PlayerWrapper winner = players.First(x => x.Id != self.Id);

MatchData data = new MatchData(winner, self, currantMove, MatchData.EndTypes.Surrender);

OnGameEnded?.Invoke(data);

}

public void GameEnded(PlayerWrapper winner) //Send

{

PlayerWrapper looser = players.First(x => x.Id != winner.Id);

MatchData data = new MatchData(winner, looser, currantMove, MatchData.EndTypes.Game);

OnGameEnded?.Invoke(data);

}

public void GameEndedLocal(MatchData data) //Accept

{

State = States.Ended;

OnGameEndedLocal?.Invoke(data);

}

#endregion

public class MatchData

{

public MatchData(PlayerWrapper winner, PlayerWrapper looser, int moveCount, EndTypes endType)

{

Winner = winner;

Looser = looser;

MoveCount = moveCount;

EndType = endType;

}

public enum EndTypes { Game, Surrender, Exit}

public PlayerWrapper Winner { get; }

public PlayerWrapper Looser { get; }

public int MoveCount { get; }

public EndTypes EndType { get; }

}

}

}

ПРИЛОЖЕНИЕ В

(обязательное)

Ведомость курсового проекта