**Міністерство оствіти та науки України**

**Дніпропетровський національний університет залізничного транспорту**

**імені академіка В.Лазаряна**

**КУРСОВА РОБОТА**

Розробка програми з використанням

об’єктно-орієнтирного програмування

з навчальної дисципліни «Основи програмування»

Виконав:

студент групи 921

Шульга Є.О

Науковий керівник:

доцент

Андрющенко В.О.

м.Дніпро

2017

Оглавление

[1 Описание предметной области и разработки спецификаций (требования к программе)… 2](#_Toc483648858)

[1.1 Постановка задачи 2](#_Toc483648859)

[1.2 Требования к программе 3](#_Toc483648860)

[1.3 Среда и язык программирования 3](#_Toc483648861)

[1.3 Методология программирования 3](#_Toc483648862)

[1.4 Требования к входным данным 4](#_Toc483648863)

[1.5 Требования к выходным данным 4](#_Toc483648864)

[1.6 Требования к функциональности 4](#_Toc483648865)

[2 Разработка объектно-ориентированной модели 5](#_Toc483648866)

[2.1 Описание ответственности классов 5](#_Toc483648867)

[2.2 Описание отношений между классами 5](#_Toc483648868)

[2.3 Диаграмма классов и спецификация классов 6](#_Toc483648869)

[2.4 Диаграмма последовательностей 10](#_Toc483648870)

[3 Разработка и описание интерфейсной части классов 12](#_Toc483648871)

[4 Разработка программы 21](#_Toc483648872)

[5 Текст программы 22](#_Toc483648873)

[6 Тестирование программы черным и белым ящиком 65](#_Toc483648874)

[7 Пример работы программы 66](#_Toc483648875)

[8 Анализ результатов и выводы 68](#_Toc483648876)

# Описание предметной области и разработки спецификаций (требования к программе)

# Постановка задачи

Написать на компьютерном языке стратегическую игру «Монополия».

«Монополия» – [игра](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%B0) для двух и более участников, в которой игроки по очереди бросают кубик, и делают ходы. В зависимости от его положения на поле может покупать, продавать, или обменивать свое имущество. Противником может быть, как человек, так и компьютер. Если у игрока закончились деньги, и имущество он считается банкротом . Цель игрока – не допустить своего банкротства.

# Требования к программе

Программа должна иметь визуальное отображение основных действий пользователя:перемещение игрока, покупка недвижимости, получение карточки ШАНС.

*Требования к надежности*

 Аварийное прерывание работы программы допускается 1 раз из 100 запусков. Допустимое количество ошибок – 1 на 1000 операций.

# 1.3 Среда и язык программирования

Программный продукт был разработан с помощью игрового дивжка Unity3D, интегрированной среды разработки – Microsoft Visual Studio 2017 с использованием языка С#.

**Unity** — это инструмент для разработки двух- и трёхмерных приложений и игр, работающий под операционными системами Windows, Linux и OS X. Созданные с помощью Unity приложения работают под операционными системами Windows, OS X, Windows Phone, Android, Apple iOS, Linux, а также на игровых приставках Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One и MotionParallax3D дисплеях (устройства для воспроизведения виртуальных голограмм), например, Nettlebox. Есть возможность создавать приложения для запуска в браузерах с помощью специального подключаемого модуля Unity (Unity Web Player), а также с помощью реализации технологии WebGL. Ранее была экспериментальная поддержка реализации проектов в рамках модуля Adobe Flash Player, но позже команда разработчиков Unity приняла сложное решение по отказу от этого.

C#-объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и, в особенности, Java — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

# Методология программирования

При написании программы использовалось методология объектно-ориентированного программирования.

**Объе́ктно-ориенти́рованное программи́рование (ООП)** — методология программирования, основанная на представлении программы в виде совокупности [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), каждый из которых является экземпляром определенного [класса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), а классы образуют иерархию наследования.

Существует три основных принципа ООП:

* *инкапсуляция* – механизм языка, позволяющий ограничить доступ одних компонентов программы к другим, также позволяет связать данные с методами, предназначенными для обработки этих данных;
* *наследование*– свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью;
* *полиморфизм* – свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Программа разбита на классы для лучшей читабельности и понимания кода.

Класс — универсальный [тип данных](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), состоящий из тематически единого набора «полей» (переменных более элементарных типов) и «методов» (функций для работы с этими полями), то есть он является моделью информационной сущности с внутренним и внешним [интерфейсами](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) для оперирования своим содержимым (значениями полей).

# Требования к входным данным

Основные входные данные, которые понадобятся при создании приложения:

* Количество игроков;
* Имя для каждого игрока;
* Цвет фишки для каждого игрока;

# Требования к выходным данным

Основные выходные данные после игры:

* результат игры (имя победителя);

# Требования к функциональности

В данном программном проекте в техническом задании выделены следующие функциональные требования:

1. Выбор пункта меню.

Игроку предоставляется четыре варианта пункта меню: «Новая игра», «Настройки», «Инструкция» «Выход». Игроку необходимо выбрать один из пунктов для следующих действий приложения.

1. Выбор типа.

Игрок должен иметь право выбрать себе тип игры. Существует 3 типа игры («Онлайн», «Против компьютера», и « Игра с друзьями») на даном этапе реализованно только тип «Игра с друзьями»

1. Выбор количества игроков

Минимальное количество 2, максимальное количество 6

1. Заполнение информации про игроков.

Для каждого игрока пользователь должен ввести его имя, и выбрать цвет его фишек.

1. Возможность выхода из игры.

Игрок имеет право выйти из игры до ее завершения.

# Разработка объектно-ориентированной модели

# Описание ответственности классов

Проект состоит из 16 основных сущностей (классов):

* Game;
* Player;
* Interface;
* BoardElements;
* BoardProperty
* BoardCard
* BoardPayOnly
* BoardSingleProperty
* ButtonClickSound;
* MenuMusic.
* ControlerVariantOfGame
* ControlerSetting
* ControlerMutliplayerGame
* ControlerMainMenu
* ControlerInstruction
* MainGameControler

Класс **Game** – это класс, который содержит в себе массив игроков, количество игроков, имеет основные функции для поддержания игры. Класс реализован как патерн Singleton, для того чтобы гарантировать присутствие одного экземпляра этого класса в игре.

Этот класс ответствен:

* за создание игроков;
* за хранение в себе игроков;
* за бросание кубика;
* за проверку на банкротство определение победителся;

Класс **Player** –основной класс, который отвечает за действия пользователя.

Основные функции:

* Покупка;
* Продажа;
* Оплата;
* Повышение уровня;
* Сделать ход;
* Выполнить операцию на карточке;
* Продать имущество

Player – это абстратный класс, который имеет наследника:**RealPlayer.** Сделанно это для того чтобы в дальнейшем была возможность добавить еще 2 типа игры.

**RealPlayer-**класс в котором реализованы вышеописанные действия для реального пользователя.

Класс **Interface** –класс который отвечает за динамическое создание кнопочек, всплывающих окошек, для визуального отображения действий пользователя. Класс реализован как патерн Singleton, для того чтобы гарантировать присутствие одного экземпляра этого класса в игре.

Класс **BoardElements** – хранит в себе общие данные про элементы карты.

Основные функции:

* Расстановка в себе играков;
* Установка своих размеров.

Класс **BoardProperty** этот представляет собой клетку на карте которую можно купить, и повышать ее уровень, хранит в себе дополнительные данные о клетках.

Класс **BoardPayOnly** этот представляет собой клетку на карте в которую можно только оплатить , хранит в себе необходимую к оплате суму.

Класс **BoardSingleProperty** этот представляет собой клетку на карте которую можно купить, и нельзя повышать ее уровень, хранит в себе дополнительные данные о клетках.

Класс **BoardCars** этот представляет собой клетку на карте которую Chance или Community, и вам необходимо тянуть карту и выполнить действие которое там написано,хранит в себе масив таких карточек, и тип поля Chance или Community

Класс **ButtonClickSound** этот отвечает за воспроизведение звуков при нажании на кнопки. Класс реализован как патерн Singleton, для того чтобы гарантировать присутствие одного экземпляра этого класса в игре.

Класс **MenuMusic** этот отвечает за воспроизведение главной музыки. Класс реализован как патерн Singleton, для того чтобы гарантировать присутствие одного экземпляра этого класса в игре.

Класс **ControlerVariantOfGame** класс обработчик нажатий клавишь для сцены выбора типа игры.Отвечает за обработку нажатий клавиш при выбре типа игры

Класс **ControlerSetting** класс обработчик нажатий клавишь для сцены настроек .Отвечает за обработку нажатий клавиш при настройках.

Класс **ControlerMutliplayerGame** класс обработчик нажатий клавишь для сцены настроек многопользовательской игры .Отвечает за обработку нажатий клавиш настройках многопользовательской игры

Класс **ControlerMainMenu** класс обработчик нажатий клавишь для сцены главного меню .Отвечает за обработку нажатий клавиш в главном меню

Класс **ControlerInstruction** класс обработчик нажатий клавишь для сцены инструкции.Отвечает за обработку нажатий клавиш при прочтении инструкции

Класс **MainGameControler** класс обработчик нажатий клавишь для сценыглавной игры.Отвечает за обработку нажатий клавиш при игре

# Описание отношений между классами

Класс **Game** имеет связь с классом Player.

Класс Game хранит в себе масив Player.Связь между ними называется агрегацией, потому что класс Game пользуется методами класса Player.

Класс **Player** имеет связь с классами RealPlayer и BoardElements.

Класс RealPlayer является наследником класса Player, поэтому связь между этими классами является обобщенной.Каждый игрок хранит в себе масив его имущества,тоесть масив BoardElements.Такая связь засывается агрегацией.

Класс **Interface**

Класс **BoardElements** имеет связь с классами BoardProperty, BoardSingleProperty, BoardPayOnly, BoardCard. Класс **BoardElements** хранит в себе все вышеперечисленные классы как поля. Такая связб называется агрегацией.

# Диаграмма классов и спецификация классов

Класс **Game**

*Атрибуты:*

* Количество игроков;
* Масив экземпляров игроков;
* Выпавшее число на кубике;
* Флаг для проверки брошен ли кубик;
* объект игрока-победителя;

*Методы:*

* статическая функция для получения экземпляра класса Game;
* функция для бросания кубика;
* функция для задания количества игроков;
* функция для увеличения количества игроков;
* функция для уменьшения количества игроков;
* функция поиска игрока по имени;
* функция для проверки игрока на банкротство;
* функция определения победителя;

Класс**Interface**:

*Атрибуты:*

* вспомогательный массив координат поля;
* массив фишек игрока;

*Методы:*

* функция изменения размера выбраной фишки(при выборе цвета игрока) ;
* функция для проверки размера фишки
* функция для сброса размера фишек
* функция для поворота фишек
* функия для показа окна об ошибке
* функция для динамического добавления игроков на карту
* функция для загрузки изображения фишки
* функция для создания окна информации об игроков
* функия для показания понели информации об игроков
* функция для закрытия панели информации об игроков
* функция для показа и создания панели информации про элемент на карте
* функия для закрытия панели информациипро элемент на карте
* функция для создания панели для недвижемости
* функция для удаления панели для недвижемости
* фунция для создание панели только для оплаты
* функция для удаления панели только для оплаты
* функия для создания панели одиночной недвижемости
* функция для удаления панели одиночной недвижемости
* функция для создания панели с дополнительными картами
* функция для удаления панели с дополнительными картами
* функия для показания рандомной карты Шанс
* функция для удалениярандомной карты Шанс
* фунция для изменения текста нынешнего игрока

Класс **BoardElement**

*Атрибуты:*

* обьект «Недвижемости»
* обьект «Только оплаты»
* обьект «Одиночной недвижемости»
* обьект «Обьект дполнительной карты»
* имя обьекта
* координаты обьекта
* количество игроков на элементе
* картинка элемента
* массив фишек игроков

*Методы:*

* функция для расстановки игроков на своей ячейке;
* функция для установки своих конечных точек

Класс **BoardProperty**

*Атрибуты:*

* Цена
* Уровень недвижемости;
* Флаг для проверки купленна ли недвижемость
* Имя владельца

Класс **BoardSingleProperty**

*Атрибуты:*

* Цена
* Флаг для проверки купленна ли недвижемость
* Имя владельца

Класс **BoardPayOnly**

*Атрибуты:*

* Цена

Класс **BoardCard**

*Атрибуты:*

* Массив пар с текстурой карты и ее номером

Класс **RealPlayer**

*Атрибуты:*

* Имя игрока;
* Цвет фишек игрока
* Количество денег игрока
* Картинка фишки игрока
* Масив с его недвижемостью
* Позиция игрока на карте

*Методы:*

* Функция для произвидения хода;
* Функция для покупки недвижеости
* Функция для оплаты
* Функция для повышения уровня недвижемости
* Функция для оплаты оренды
* Функия выполенния задания которое написанно на карте
* Функция для продажи
* Функия для проверки играет ли игрок дальше

Класс **MenuMusic**

*Атрибуты:*

* Флаг для проверки включен ли звук

*Методы:*

* Функия для включения музыки
* Функция для выключения музыки
* Функция для измениния громкости музыки
* Функция для состояния музыки
* Функия для включения безвучного режима
* Функия для выключения безвучного режима
* Функция для проверки включен ли безвучный режим
* Функия для возвращения уровня громкости

Класс **ButtonClickSoubd**

*Атрибуты:*

* Флаг для проверки включен ли звук

*Методы:*

* Функия для включения звука
* Функция для выключения звука
* Функция для измениния громкости звука
* Функция для состояния звука
* Функия для включения безвучного режима
* Функия для выключения безвучного режима
* Функция для проверки включен ли безвучный режим
* Функия для возвращения уровня громкости звука

Класс **ControlerInstruction**

*Методы:*

* Функция начала работы класса
* Функия для обработки нажатия назад

Класс **ControlerMainMenu**

*Методы:*

* Функция начала работы класса
* Функия для обработки нажатия Играть
* Функция для обработки нажания Настройки
* Функция для обработки нажатия Инструкция
* Функция для обработки нажатия Выход

Класс **ControlerSetting**

*Методы:*

* Функция начала работы класса
* Функия для обработки изменения громкости музыки
* Функция для обработки изменения громкости звука
* Функция для обработки нажатия Назад
* Функция для обработки нажатия Вкл/выкл бесшумного режима

Класс **ControlerVariantOfGame**

*Методы:*

* Функция начала работы класса
* Функия для обработки нажатия на «Многопользовательская игра»

Класс **MainGameControler**

*Атрибуты:*

* Индекс выбраного персонажа
* Массив элементов карты.

*Методы:*

* Функция начала работы класса
* Функия для инициализации всех полей на карте
* Функция для инициализации положения полей на карте
* Функия для вызова на экран панели информации про игроков
* Функция для выполнения хода
* Функция для проверки не выходид ли индекс за пределы
* Функция для первой расстоновки играков
* Функуия для инициализации фишек всех игроков
* Функция для обрабоки «Покупки»
* Функция для обрабоки «Продажи»
* Функция для обрабоки «Заплатить»
* Функция для обрабоки «Увелечение уровня недвижмости»
* Функция для обрабоки «Оплата оренды»
* Функция для обрабоки «Пропуск действия»
* Функция для обрабоки «Выполнения действия»
* Функция для обрабоки «Продажи»

На рисунке 2 представлена диаграмма классов и связи между ними.

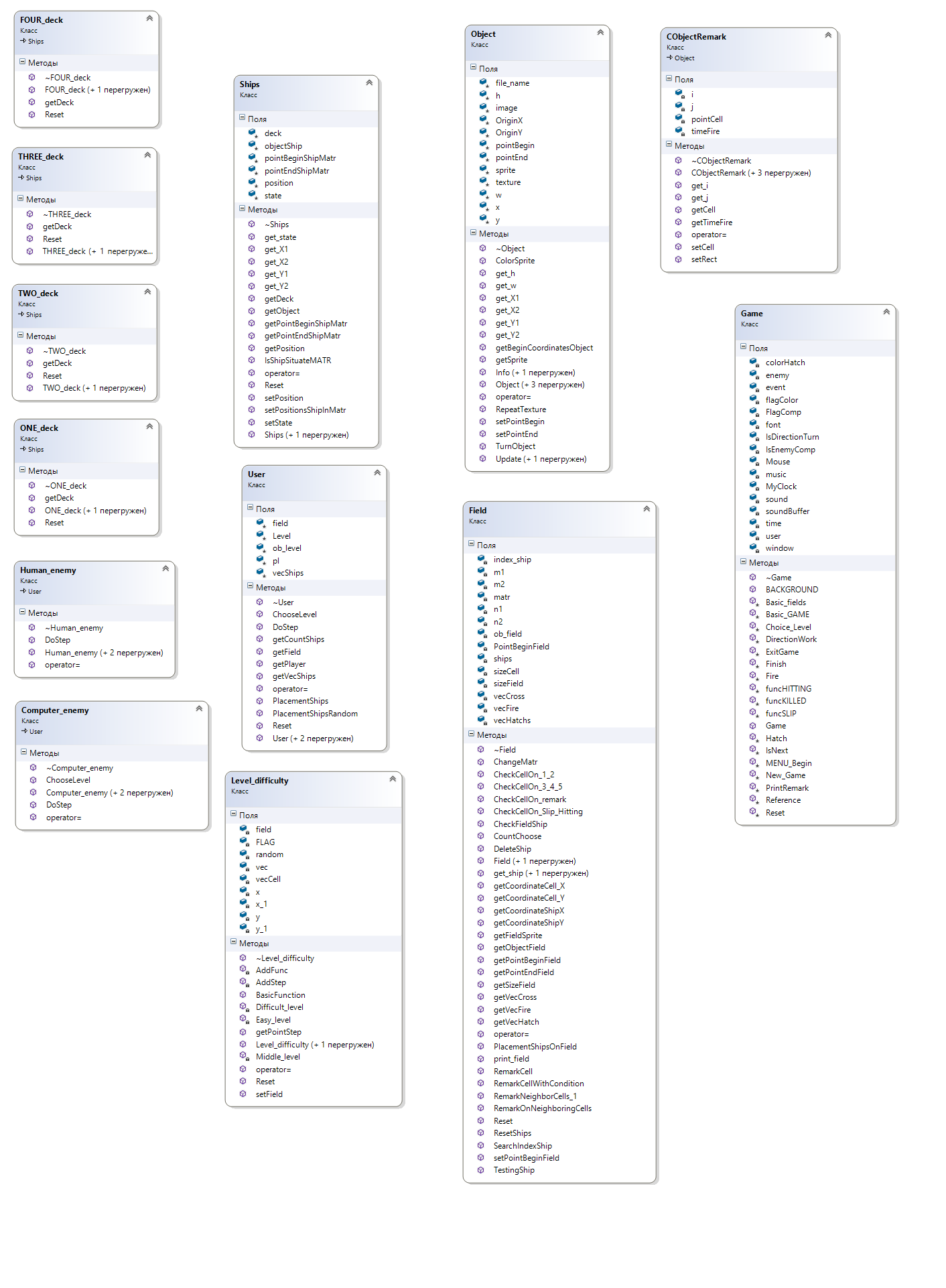
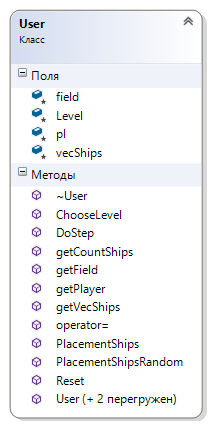
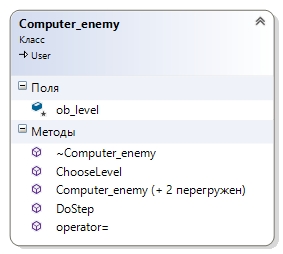


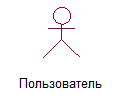
Рисунок 2 – Диаграмма классов



# Диаграмма последовательностей

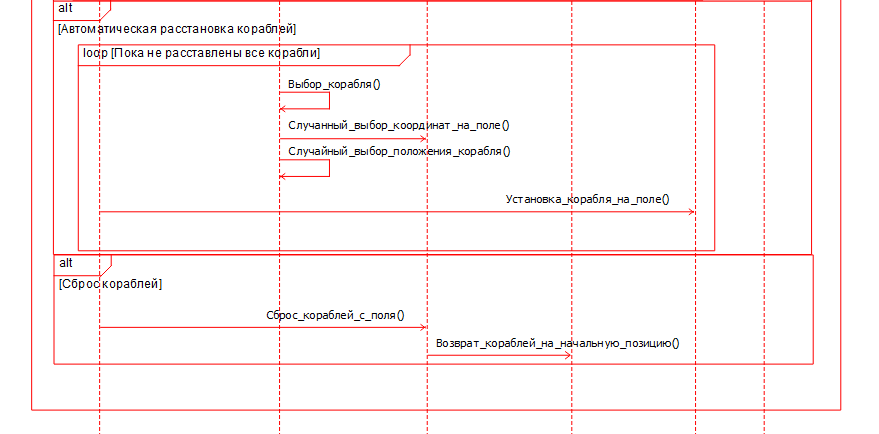
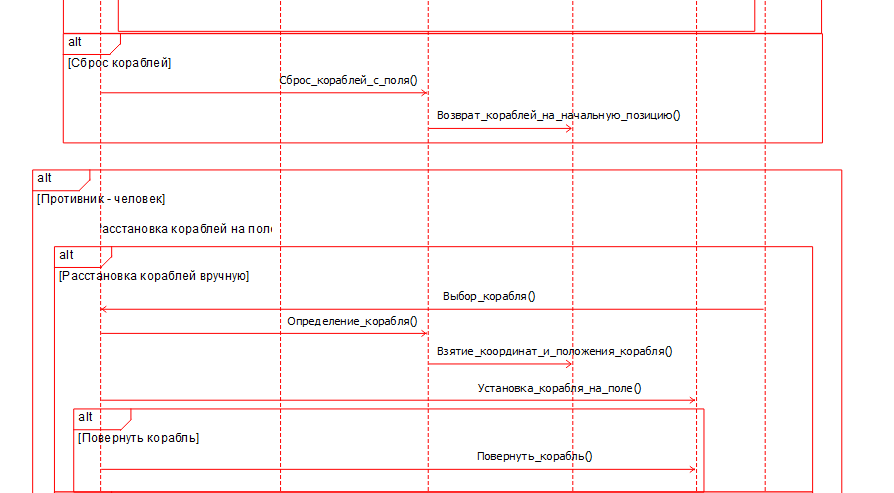
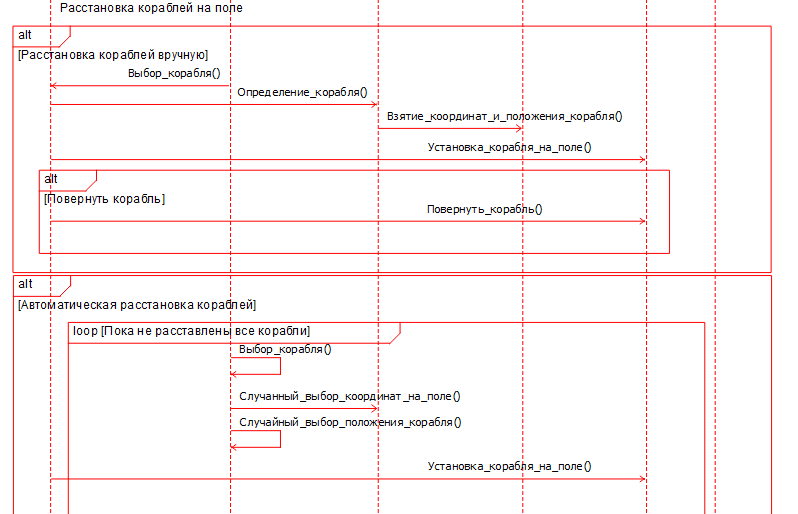
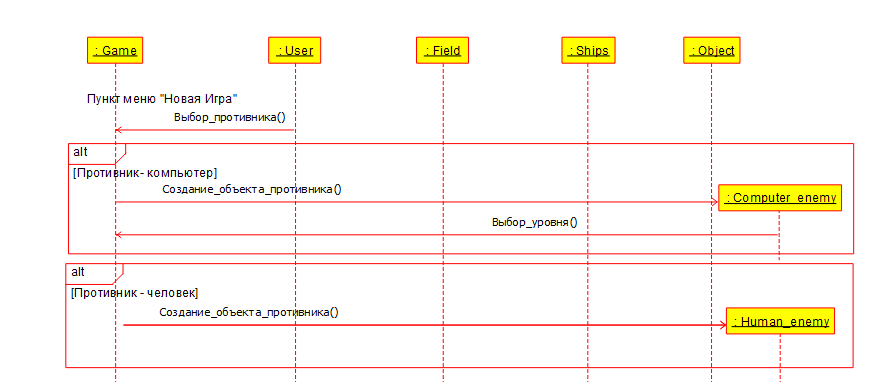
Для реализации игры «Морской Бой» было определено несколько сущностей – объектов, которые принимают и передают сообщения во время их взаимодействия. Каждый объект может отправлять сообщение другому объекту или себе. На рисунке 3 представлена диаграмма последовательности игры «Морской Бой».

За действия актера «Пользователь» отвечает объект User



За действия актера «Игрок №2» отвечает объект Human\_enemy

или объект Computer\_enemy





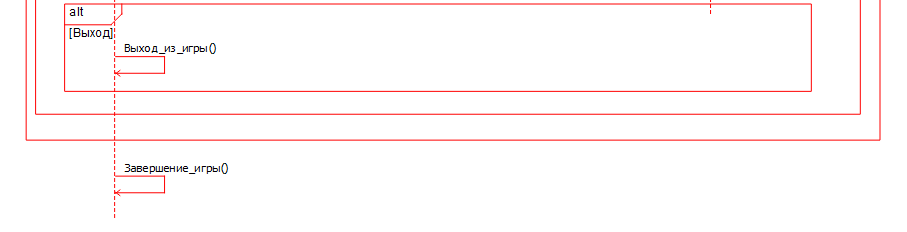
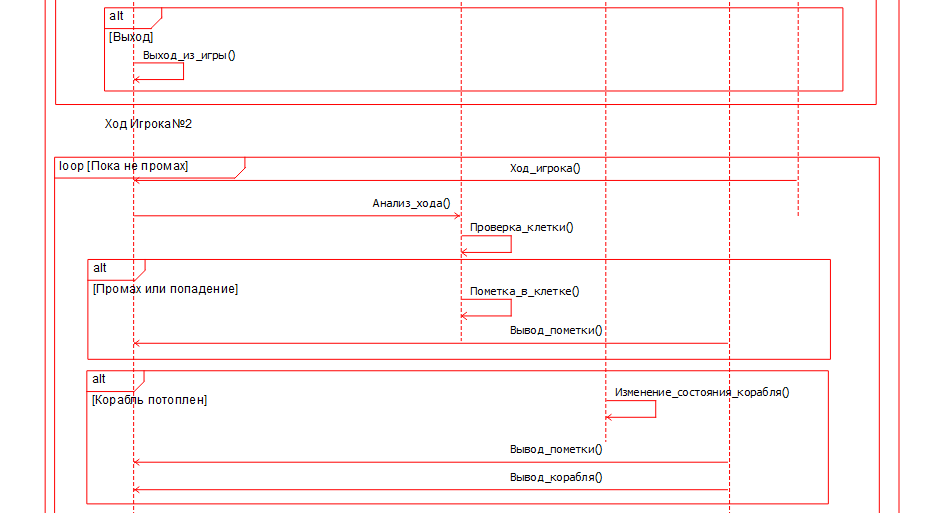
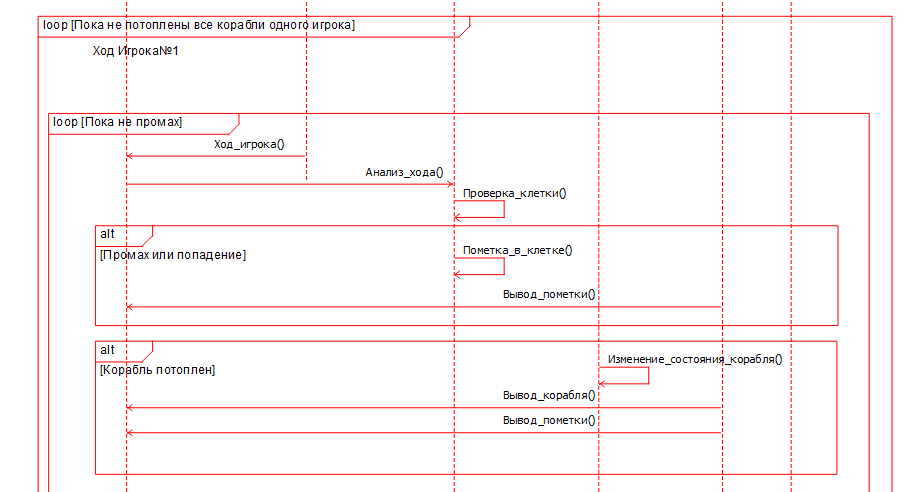


Рисунок 3 – Диаграмма последовательности

# 3 Разработка и описание классов

Классом, который отвечает за создание и хранения игроков,бросание кубира и т.д, является класс **Game**.

Интерфейс класса Game содержит:

*Атрибуты:*

* numberOfPlayers количество игроков;
* players– массив игроков;
* Step – число на кубике;
* chekTrow- флаг для проверки брошен ли кубик
* winPlayer-игрок победитель
* game- ебинственный экземпляр этого класса

*Методы:*

* static public Game getGame()

Назначение: Инициализация единственного экземпляра класса .

Входные данные: поля класса Game.

Выходные данные: инициализированный экземпляр класса Game.

* public void changeNumberOfPlayersMore();

Назначение: Функция для увелечения количества играков.

Входные данные:

Выходные данные: возвращаент дальнейшие действия программы.

* public void changeNumberOfPlayersLess();

Назначение: Функция для уменьшения количества играков.

Входные данные:

Выходные данные: возвращаент дальнейшие действия программы.

* public Player findByName(string name);

Назначение: Функция для поиска игрока по имени

Входные данные:имя игрока

Выходные данные: найденный игрок, или null.

* public void setNumberOfPlayer();

Назначение: Функция для задания количества игроков и создания их.

Входные данные:количество игроков

Выходные данные:массив игроков

* public void throwCube();

Назначение: Бросания кубика.

Входные данные:

Выходные данные: Число выпавшее на кубире.

* public void chekBanclotsPlayer();

Назначение: Функция для проверки всех игроков на банкротство.

Входные данные: массив игроков.

Выходные данные: возвращает измененный массив игроков.

* public bool isGameOver()

Назначение: Функция для определения победителя.

Входные данные:Массив игроков

Выходные данные:Победитель

Класс, визуализацию дествий игроков, является класс **Interface.**

Интерфейс класса Interface содержит:

*Атрибуты:*

* massAchors- массив из 4 элементов обозначающщие края поля;
* \_interface-единственный экземпляр класса
* PlayersLogo- массив фишек игроков

*Методы:*

* public void changeScaleOfButton(UnityEngine.UI.Button button, bool flag)\

Назначение: Для изменения размеров кнопки при выборе пользователем цвета фишек

Входные данные:

Button button- кнопка которая нуждается в изменении размера.

Flag-если true увеличиваем кнопку,если false уменьшаем.

Выходные данные: Кнопка с измененным размеров.

* public bool chekScaleColorBatton(UnityEngine.UI.Button button);

Назначение:Для проверки изменен ли размер кнопки

Входные данные:

Button button -кнопка которая нуждается в проверке

Выходные данные: возвращает false - если не изменен ,true – иначе.

* public void resetScaleButton(List<UnityEngine.UI.Button> list)

Назначение: Для сброса размера всех кнопок до исходного.

Входные данные:

List<Button> list -массив всех кнопок

Выходные данные:Кнопки с исходным размером.

* public void rotationButton(string button, bool flag);

Назначение: Функция для поворота кнопки если пользователь ее окончательно выбрал.

Входные данные:

String button – имя клавиши,

Bool flag – если true-поворот влево,иначе- вправо.

Выходные данные: повернутая кнопка.

* public void ShowCloseMassageBox(bool flag, string str = " ");

Назначение: Функция для показа или скрытия сообщения об ошибке.

Входные данные:

Bool flag-если true то показать,иначе скрыть.

String str-сообщение об ошибке

Выходные данные: действие с панелью ошибок.

* public void addPlayersElementsToTheBoad(GameObject parent);

Назначение: Функция для динамического добавления игроков на сцену.

Входные данные:

GameObject parent- родительский обьект.

Выходные данные: добавленные игроки на сцену.

* public void addInformationAboutPlayer();

Назначение: Добавления понели с информацией о игроках.

Входные данные:

Выходные данные:Добавленные панели с информацией.

* public void showPanelInformation(Player player);

Назначение: Функция для показа панели с информацией о игроке.

Входные данные:

Player player игрок о котором нужно показать информацию.

Выходные данные:

* public void showInformationAboutBoardElemets(BoardElements elements)\;

Назначение: Функция для показа информации про ячейку на которую стал игрок.

Входные данные:

BoardElement element-ячейка о которой нужно показать информацию

Выходные данные:

* public void closeInformationAboutBoardElements();

Назначение: Функция для скрытия панели с информацие о ячейке.

Входные данные:

Выходные данные:

* public void createPanelForProperty(BoardElements element);

Назначение: Функция для поиска создание дополнительных кнопок которых требует «Недвижемсть».

Входные данные:

BoardElement element- ячейка для которой нужно добавить кновки

Выходные данные: добавленные кнопки на панель.

* public void createPanelForPayOnly(BoardElements element);

Назначение: Функция для поиска создание дополнительных кнопок которых требует «Только заплати».

Входные данные:

BoardElement element- ячейка для которой нужно добавить кновки

Выходные данные: добавленные кнопки на панель.

* public void createPanelForSingleProperty(BoardElements element);

Назначение: Функция для поиска создание дополнительных кнопок которых требует «Одиночная недвижемость».

Входные данные:

BoardElement element- ячейка для которой нужно добавить кновки

Выходные данные: добавленные кнопки на панель.

* public void createPanelForBoardCard(BoardElements element);

Назначение: Функция для поиска создание дополнительных кнопок которых требует «Дополнительные карточки».

Входные данные:

BoardElement element- ячейка для которой нужно добавить кновки

Выходные данные: добавленные кнопки на панель.

* public void showRandCard(BoardElements element);

Назначение: Функция для определения и показания рандоиной карты с дполнительным действием.

Входные данные:

BoardElement element- елемент из которого будут выниматься карточки

Выходные данные: карточка с дополнительным действием.

* public void returnAnchors();

Назначение: Функция для возвращения на свои места краев панели с информацией о ячейке.

Входные данные: ячейка в которой надо поменять края

Выходные данные: ячейка с измененными краями.

* public void changeTextPlayer(int index);

Назначение: Функция для выделения курсиво игрока который делает ход.

Входные данные:

Int index- индекс игрока которого нужно выделить

Выходные данные: измененный шрифт игрока.

Класс, овечающий за все ячейки на карте, является класс **BoardElements.**

Интерфейс класса BoardElements содержит:

*Атрибуты:*

* name – имя ячейки;
* positions – расположение ячейки на карте;
* Count- количество игроков на этой ячейки
* playersInTheCase-массив игроков в этой ячейке
* boardProperty- обьект «Недвижемость» инициализируеться только тогда когда ячейка имеет тип «Недвижемость»
* boardSingleProperty- обьект «Одиночная недвижемость» инициализируеться только тогда когда ячейка имеет тип «Одиночная недвижемость»
* boardPayOnly- обьект «Заплати» инициализируеться только тогда когда ячейка имеет тип «Заплати»
* boardCard- обьект «Дополнительная карточка» инициализируеться только тогда когда ячейка имеет тип «Дополнительная карточка»
* sprite- картинка этой ячейки

*Методы:*

* public void setAnchors(float MaxX,float MaxY,float MinX,float MinY)

Назначение: Функция для задание размеров ячейки.

Входные данные:

Float MaxX-максимальное значение х;

Float MaxY-максимальное значение y;

Float MinX-минимальное значение x;

Float MinY-минимальное значение y;

Выходные данные:расположенная клетка.

Класс, овечающий за все ячейки «Недвижемости» на карте, является класс **BoardProperty.**

Интерфейс класса BoardProperty содержит:

*Атрибуты:*

* UpdateLevel-уровень улучшения недвижемости
* Ownername-имя владельца
* isOwner-флаг для проверки есть ли владелец у недвижемости
* Price- цена покупки и улучшения недвижеости

Класс, овечающий за все ячейки «Дополнительная карта» на карте, является класс **BoardCard.**

Интерфейс класса BoardCard содержит:

*Атрибуты:*

* Type-тип карточки Шанг или громадська касса
* Images-масив пар и изображением и номером карточки

Класс, овечающий за все ячейки «Заплати» на карте, является класс **BoardPayOnly.**

Интерфейс класса BoardPayOnly содержит:

*Атрибуты:*

* Price-неодходимая цена к оплате

Класс, овечающий за все ячейки «Одиночная недвижемость» на карте, является класс **BoardSingleProperty.**

Интерфейс класса BoardSingleProperty содержит:

*Атрибуты:*

* Price-цена
* isOwner-флаг для проверки есть ли владелец
* OwnerName-имя владельца

Класс, овечающий за действия игроков, является класс **RealPlayer.**

Интерфейс класса RealPlayer содержит:

*Атрибуты:*

* namePlayer-имя игрока;
* colorPlayer- цвет игрока
* moneyPlayer-кличество денег у игрока
* LogoPlayer- картинка фишки игрока
* RealTimePosition- нынешняя позиция игрока на карте
* PlayersElement-масив недвижемости у игрока

*Методы:*

* public int makeMove(int step);;

Назначение: Функция для произвиденея вдижения игрока по карте.

Входные данные:

Int step-число выпавшее на кубике, на сколько позиций нужно сделать ход

Выходные данные: новое положение на карте.

* public bool buy(BoardElements boardElements);

Назначение: Функция для покупки какой-то недвижемости.

Входные данные:

BoardElements boardElements-элемент который предлагается к покупке

Выходные данные: true- если удалось,false иначе.

* public bool payOnly(BoardElements boardElements);

Назначение: Функция для оплаты.

Входные данные:

BoardElements boardElements-элемент за кторый нужно заплатить

Выходные данные:true- если удалось,false иначе.

* public bool updateLevel(BoardElements boardElements);

Назначение: Функция для повышения уровня недвижемости.

Входные данные:

BoardElements boardElements-элемент который можно улучшить

Выходные данные: true- если удалось,false иначе.

* public bool payRent(BoardElements boardElements)

Назначение: Функция для оплаты оренды.

Входные данные:

BoardElements boardElements-элемент за оторый нужно заплатить оренду

Выходные данные: true- если удалось,false иначе.

* public int doAction(int ID, BoardElements boardElements)

Назначение: Функция для выполнения действия описанного на дополнительной карточке.

Входные данные:

BoardElements boardElements-элемент за оторый нужно заплатить оренду

ID- номер карты

Выходные данные: ID карты для визуального отображения дейтвия.

* public void sell(BoardElements elements);

Назначение: Функция для продажи недвижемости.

Входные данные:

BoardElements boardElements-элемент который нужно продать

Выходные данные: измененный масив с недвижемостью

Класс, овечающий за воспроизвидения звука при нажатии на клавишу, является класс **ButtonClickSound.**

Интерфейс класса ButtonClickSound содержит:

*Атрибуты:*

* sound- единственный екземпляр этого класса
* ButtonClick-хранит в себе звук.

*Методы:*

* static public ButtonClickSound GetSounds()

Назначение: Функция для получения единственного экземпляра этого класса.

Входные данные:

Выходные данные: эдинственный экземпляр этого класса

* public void play()

Назначение: Функция для востроизвидения звука.

Входные данные:

Выходные данные:

* public void stop()

Назначение: Функция для остановки звука.

Входные данные:

Выходные данные:

* public void setMute()

Назначение: Функция для включения безвучного режима.

Входные данные:

Выходные данные:

* public void uтsetMute()

Назначение: Функция для выключения безвучного режима.

Входные данные:

Выходные данные:

* public void changeVolume(float level)

Назначение: Функция для изменения громкости.

Входные данные:

Float level-уровень громкости

Выходные данные: измененный уровень громкости

* public float getVolumeLevel()

Назначение: Функция для возврпщения уровня громкости.

Входные данные:

Выходные данные: уровень громкости

Класс, овечающий за воспроизвидения музыки, является класс **MenuMusic.**

Интерфейс класса MenuMusic содержит:

*Атрибуты:*

* \_inctence- единственный екземпляр этого класса
* MainMusic-хранит в себе музыку.

*Методы:*

* static public MenuMusic GetMainMusic()

Назначение: Функция для получения единственного экземпляра этого класса.

Входные данные:

Выходные данные: эдинственный экземпляр этого класса

* public void play()

Назначение: Функция для востроизвидения музыки.

Входные данные:

Выходные данные:

* public void stop()

Назначение: Функция для остановки музыки.

Входные данные:

Выходные данные:

* public void setMute()

Назначение: Функция для включения безвучного режима.

Входные данные:

Выходные данные:

* public void uтsetMute()

Назначение: Функция для выключения безвучного режима.

Входные данные:

Выходные данные:

* public void changeVolume(float level)

Назначение: Функция для изменения громкости.

Входные данные:

Float level-уровень громкости

Выходные данные: измененный уровень громкости

* public float getVolumeLevel()

Назначение: Функция для возврпщения уровня громкости.

Входные данные:

Выходные данные: уровень громкости

Класс, овечающий за обработку действий на сцене «Инструкция», является класс **ControlerInstruction.**

Интерфейс класса ControlerInstruction содержит:

*Методы:*

* void Start ()

Назначение: Функция которая выполняется при первой загрузке сцены, аналог конструктора.

Входные данные:

Выходные данные:

* private void PressReturn()

Назначение: Функция которая обрабатывает нажание на клавишу «Назад».

Входные данные:

Выходные данные:

Класс, овечающий за обработку действий на сцене «Главное меню», является класс **ControlerMainMenu.**

Интерфейс класса ControlerMainMenu содержит:

*Методы:*

* void Start ()

Назначение: Функция которая выполняется при первой загрузке сцены, аналог конструктора.

Входные данные:

Выходные данные:

* private void PressPlay()

Назначение: Функция которая обрабатывает нажание на клавишу «Играть».

Входные данные:

Выходные данные:

* private void PressSetting()

Назначение: Функция которая обрабатывает нажание на клавишу «Настройки».

Входные данные:

Выходные данные:

* private void PressInstruction()

Назначение: Функция которая обрабатывает нажание на клавишу «Инструкция».

Входные данные:

Выходные данные:

* private void PressExit)

Назначение: Функция которая обрабатывает нажание на клавишу «Выход».

Входные данные:

Выходные данные:

Класс, овечающий за обработку действий на сцене «Настройка многопользовательской игры», является класс **ControlerMultiplayerGame.**

Интерфейс класса ControlerMultiplayerGame содержит:

*Атрибуты:*

* chek-флаг для определения введенны ли данные
* Index-переменная которая отвечает за индекс персонажа о котором вводим информацию
* Name-временная переменная для хранения имя
* Color-временная переменная для хранения цвета игрока
* flagGo-флаг для определения можно ли начать игру
* ButtonList-масив кнопок(фишек)

*Методы:*

* void Start ()

Назначение: Функция которая выполняется при первой загрузке сцены, аналог конструктора.

Входные данные:

Выходные данные:

* private void initialiseButtonList()

Назначение: Функция которая выполняет инициализацию масива кнопок.

Входные данные:

Выходные данные:

* private void InitialiseColors(string name)

Назначение: Функция которая выполняет размера кнопки в зависимости от выбран ли цвет.

Входные данные:

String name -имя игрока

Выходные данные: измененный размер кнопки

* private void redColor()

Назначение: Функция которая отвечает за обработку нажатия на клавишу «Крассный цвет».

Входные данные:

Выходные данные:

* private void greenColor()

Назначение: Функция которая отвечает за обработку нажатия на клавишу «Зеленый цвет».

Входные данные:

Выходные данные:

* private void greenBlueColor()

Назначение: Функция которая отвечает за обработку нажатия на клавишу «СинеЗеленый цвет».

Входные данные:

Выходные данные:

* private void blueColor()

Назначение: Функция которая отвечает за обработку нажатия на клавишу «Синий цвет».

Входные данные:

Выходные данные:

* private void violetColor()

Назначение: Функция которая отвечает за обработку нажатия на клавишу «Фиолетовый цвет».

Входные данные:

Выходные данные:

* private void yellowColor()

Назначение: Функция которая отвечает за обработку нажатия на клавишу «Желтый цвет».

Входные данные:

Выходные данные:

* private void indexMore()

Назначение: Функция которая отвечает за обработку нажатия на клавишу «Следуйщий игрок».

Входные данные:

Выходные данные:

* private void indexLess()

Назначение: Функция которая отвечает за обработку нажатия на клавишу «Предыдущий игрок».

Входные данные:

Выходные данные:

* private void PressOk()

Назначение: Функция которая отвечает за обработку нажатия на клавишу «OK».

Входные данные:

Выходные данные:

* private void PressErrorButton()

Назначение: Функция которая отвечает за обработку нажатия на клавишу «OK на сообщении с ошибкой».

Входные данные:

Выходные данные:

* private void setRotateButton()

Назначение: Функция которая отвечает за поворот всех кнопок.

Входные данные:

Выходные данные:

* private bool cheksetColor(string name)

Назначение: Функция которая проверяет выбрал ли уже такой цвет.

Входные данные:

String name-имя цвета

Выходные данные:true- если цет свободен,false иначе

* private bool cheksetName(string name)

Назначение: Функция которая проверяет выбрал ли уже такое имя.

Входные данные:

String name-имя

Выходные данные:true- если имя свободен,false иначе

* private void seachAndChangeScaleOfColorButon(string colors, bool flAG)

Назначение: Функция которая отвечает изменения размеров всех кнопочек.

Входные данные:

String colors-имя текущего цвета

Bool flag- отвечает за увечечение или уменьшение размера

Выходные данные:

* private void GO()

Назначение: Функция которая отвечает вылет кнопочки GO.

Входные данные:

Выходные данные:

Класс, овечающий за обработку действий на сцене «Игра», является класс **ControlerMainGame.**

Интерфейс класса ControlerMainGame содержит:

*Атрибуты:*

* Index-переменная которая отвечает за индекс персонажа
* BoardElements-маисв всех елементов на карте

*Методы:*

* void Start ()

Назначение: Функция которая выполняется при первой загрузке сцены, аналог конструктора.

Входные данные:

Выходные данные:

* private void initBoardElements()

Назначение: Функция которая инициализирует все ячейки на карте.

Входные данные:

Выходные данные:

* private void initBoardElementsPosition()

Назначение: Функция которая инициализирует полиции ячейек на карте.

Входные данные:

Выходные данные:

* private void addEventForInformation()

Назначение: Функция которая открывает информацию про игрока

Входные данные:

Выходные данные:

* private void makeMove()

Назначение: Функция которая отвечает за обработку нажатий на клавишу «Сделать ход»

Входные данные:

Выходные данные:

* private void PayOnly()

Назначение: Функция которая отвечает за обработку нажатий на клавишу «Заплати»

Входные данные:

Выходные данные:

* private void UPD()

Назначение: Функция которая отвечает за обработку нажатий на клавишу «улучшить»

Входные данные:

Выходные данные:

* private chekIndex

Назначение: Функция которая отвечает за проверку индекса

Входные данные:

Выходные данные:

* private void Buy()

Назначение: Функция которая отвечает за обработку нажатий на клавишу «Купить»

Входные данные:

Выходные данные:

* private void payRent()

Назначение: Функция которая отвечает за обработку нажатий на клавишу «Заплати оренду»

Входные данные:

Выходные данные:

* private void skip()

Назначение: Функция которая отвечает за обработку нажатий на клавишу «Пропустить»

Входные данные:

Выходные данные:

* private void doAction()

Назначение: Функция которая отвечает за обработку нажатий на клавишу «Сделаь дополнительное действие»

Входные данные:

Выходные данные:

# Разработка программы

Программный продукт должен представлять собой реализацию классической игры «Монополия».

Начальным этапом разработки программы было изучения предметной области:

* основных принципов объектно-ориентированного программирования (ООП), их особенностей;
* использования ООП в языке С#;
* строение принципов работы графического движка Unity.

Также был проведен обзор аналогов программы-игры «Монополия». Было рассмотрено более 5 версий игры и в каждой программе были свои достоинства и недостатки. После обзора были определены основные функциональные требования к программе, а именно:

* *Выбор типа игры.*

Существует три типа игры,онлайн,с компьютером, и с друзьями

*Выбор количества игроков*

В игре можно выбрать количество игроков которые участвуют в игре, их может быть от 2 до 6, каждый из них имеет свой индивидульный цвет, и свое имя.

Следующим этапом разработки было проектирование классов и их связей. Далее были определены поля и методы каждого из классов.

Только после всего этого был написан код программы.

# Текст программы

public enum CARDTYPE { CHANCE,COMUNITY};

public struct Anchors

{

public float MinX { set; get; }

public float MinY { set; get; }

public float MaxX { set; get; }

public float MaxY { set; get; }

public Anchors(float MaxX, float MaxY, float MinX, float MinY)

{

this.MaxX = MaxX;

this.MaxY = MaxY;

this.MinX = MinX;

this.MinY = MinY;

}

}

public class BoardElements

{

//name of player

public string Name { get; set; }

//position player on the board

public Anchors Positions { set; get; }

//object image

//public Image Images { set; get; }

//count players of the element

public int Count { set; get; }

public List<GameObject> playersInTheCase { set; get; }

public BoardProperty boardProterty { get; set; }

public BoardCard boardCard { set; get; }

public BoardPayOnly boardPayOnly { set; get; }

public BoardSingleProperty boardSingleProperty { set; get; }

public Sprite sprite { set; get; }

public BoardElements(string name,string path,BoardProperty boardProterty = null,BoardCard boardCard = null,

BoardPayOnly boardPayOnly = null,BoardSingleProperty boardSingleProperty = null)

{

Name = name;

this.boardCard = boardCard;

this.boardPayOnly = boardPayOnly;

this.boardProterty = boardProterty;

this.boardSingleProperty = boardSingleProperty;

this.sprite = new Sprite();

this.sprite= Resources.Load<Sprite>(path);

playersInTheCase = new List<GameObject>();

Count = 0;

}

public void setAnchors(float MaxX,float MaxY,float MinX,float MinY)

{

Positions = new Anchors(MaxX, MaxY, MinX, MinY);

}

public void setPositionsInTheCase(List<GameObject> playersLogo=null)

{

Count = playersInTheCase.Count;

//if odd count of players

if(Count%2!=0)Count++;

if(Count>2)

Count /= 2;

float length = (Positions.MaxX - Positions.MinX) / Count;

float width = (Positions.MaxY - Positions.MinY) / Count;

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

{

//first element

if(i==0)

{

playersInTheCase[i].GetComponent<RectTransform>().anchorMax = new Vector2(Positions.MaxX,Positions.MinY + width);

playersInTheCase[i].GetComponent<RectTransform>().anchorMin = new Vector2(Positions.MaxX - length, Positions.MinY);

playersInTheCase[i].GetComponent<RectTransform>().transform.localPosition = Vector3.zero;

playersInTheCase[i].GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

playersInTheCase[i].GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

}

//even element

else if(i%2==0)

{

playersInTheCase[i].GetComponent<RectTransform>().anchorMax = new Vector2(playersInTheCase[i - 2].GetComponent<RectTransform>().anchorMin.x-0.005f, playersInTheCase[i - 2].GetComponent<RectTransform>().anchorMax.y);

playersInTheCase[i].GetComponent<RectTransform>().anchorMin = new Vector2(playersInTheCase[i - 2].GetComponent<RectTransform>().anchorMin.x-length, playersInTheCase[i - 2].GetComponent<RectTransform>().anchorMin.y);

playersInTheCase[i].GetComponent<RectTransform>().transform.localPosition = Vector3.zero;

playersInTheCase[i].GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

playersInTheCase[i].GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

}

//odd element

else if(i%2!=0)

{

playersInTheCase[i].GetComponent<RectTransform>().anchorMax = new Vector2(playersInTheCase[i-1].GetComponent<RectTransform>().anchorMax.x, playersInTheCase[i - 1].GetComponent<RectTransform>().anchorMax.y+width);

playersInTheCase[i].GetComponent<RectTransform>().anchorMin = new Vector2(playersInTheCase[i - 1].GetComponent<RectTransform>().anchorMin.x, playersInTheCase[i - 1].GetComponent<RectTransform>().anchorMax.y);

playersInTheCase[i].GetComponent<RectTransform>().transform.localPosition = Vector3.zero;

playersInTheCase[i].GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

playersInTheCase[i].GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

}

}

}

}

public class BoardProperty

{

public int UpdateLevel { set; get; }

public string OwnerName { set; get; }

public bool isOwner { set; get; }

public int Price { set; get; }

public BoardProperty(int price)

{

Price = price;

OwnerName = " ";

isOwner = false;

UpdateLevel = 0;

}

}

public class BoardCard

{

//Chanse or CommunityChest

public CARDTYPE Type { set; get; }

//for save Image and ID card

public KeyValuePair<int,Sprite>[] Images { set; get; }

public BoardCard(CARDTYPE type)

{

Type = type;

if (Type== CARDTYPE.CHANCE)

{

string path = @"CHANSE\";

Sprite[] tmpList = Resources.LoadAll<Sprite>(path);

this.Images = new KeyValuePair<int, Sprite>[tmpList.Length];

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

{

Images[i] = new KeyValuePair<int, Sprite>(System.Convert.ToInt32(tmpList[i].name.Replace("CHANSE", "")), tmpList[i]);

//Images.Add(i + 1, tmpList[i]);

}

}

if(Type== CARDTYPE.COMUNITY)

{

string path = @"Community\";

Sprite[] tmpList = Resources.LoadAll<Sprite>(path);

this.Images = new KeyValuePair<int, Sprite>[tmpList.Length];

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

{

Images[i] = new KeyValuePair<int, Sprite>(System.Convert.ToInt32(tmpList[i].name.Replace("CHANSE", "")), tmpList[i]);

//Images.Add(i + 1, tmpList[i]);

}

}

}

}

public class BoardPayOnly

{

public int Price { set; get; }

public BoardPayOnly(int price)

{

Price = price;

}

}

public class BoardSingleProperty

{

public int Price { set; get; }

public string OwnerName { set; get; }

public bool isOwner { set; get; }

public BoardSingleProperty(int price)

{

OwnerName = " ";

isOwner = false;

Price = price;

}

}

//this is a singletone class,he save number of players and other setting of our game

public class Game

{

//for save number of player in Game

public int numberOfPlayers { set; get; }

private List<Player> players;

public int Step { get; set; }

public bool chekTrow { get; set; }

Player winPlayer;

static private Game game;

static public Game getGame()

{

if (game == null)

game = new Game();

return game;

}

//default constructor

private Game()

{

chekTrow = true;

numberOfPlayers = 2;

}

//Methods for change number of players

//set more players

public void changeNumberOfPlayersMore()

{

if(numberOfPlayers<6)

numberOfPlayers++;

}

public void chamgeNumberOfPlayersLess()

{

if(numberOfPlayers>2)

numberOfPlayers--;

}

public Player findByName(string name)

{

return players.Find(x => x.namePlayer == name);

}

public void setNumberOfPlayer()

{

numberOfPlayers = 3;

players = new List<Player>(numberOfPlayers);

for (int i = 0; i < numberOfPlayers; i++)

{

players.Add(new RealPlayer());

}

Debug.Log("Count of players: " + numberOfPlayers);

Debug.Log("ListLengs: " + players.Count);

}

public Player this[int index]

{

set

{

// Debug.Log("Imput Index" + index);

players[index] = value;

}

get

{

// Debug.Log("Imput Index" + index);

return players[index];

}

}

/\*create and add camera to GameObj

GameObject obj- parent gameobject

\*/

public void CreateCameraForBoard(GameObject obj)

{

GameObject camera = new GameObject(obj.name + "Camera");

camera.AddComponent<RectTransform>();

camera.AddComponent<Camera>();

camera.GetComponent<Camera>().orthographic = false;

camera.transform.SetParent(obj.transform);

camera.GetComponent<Camera>().enabled = false;

}

/\*For change Camera\*/

public void changeMainCamera(GameObject camera1,GameObject camera2)

{

//camera1.GetComponent<Camera>().enabled = false;

// camera2.GetComponent<Camera>().enabled = true;

// GameObject.Find("Canvas").GetComponent<Canvas>().worldCamera = camera2.GetComponent<Camera>();

// camera2.transform.position = new Vector3(Game.getGame()[0].position.position.x, Game.getGame()[0].position.position.y, Game.getGame()[0].position.position.z);

// camera2.GetComponent<Camera>().farClipPlane = 10;

// camera2.GetComponent<Camera>().backgroundColor = camera1.GetComponent<Camera>().backgroundColor;

}

public void setFocusPlayerCamera(Player player)

{

//GameObject.Find("BoardCamera").GetComponent<RectTransform>().localPosition = new Vector3(player.position.localPosition.x, player.position.localPosition.y, -50);

}

public void throwCube()

{

if (chekTrow)

{

System.Random rand = new System.Random();

Step = rand.Next(1, 6);

GameObject.Find("CubeNumber").GetComponent<Text>().text = System.Convert.ToString(Step);

chekTrow = false;

}

}

public void chekBanclotsPlayer()

{

foreach(Player p in this.players)

{

if(p.moneyPlayer<0)

{

players.Remove(p);

}

}

}

public bool isGameOver()

{

if(this.players.Count==1)

{

winPlayer = players[0];

return true;

}

return false;

}

}

public class Interface

{

private static Interface \_interface;

public static Interface GetIterface()

{

if (\_interface == null)

\_interface = new Interface();

return \_interface;

}

public Interface()

{

chekCard = true;

}

float[] massAchors=new float[4];

public List<GameObject> PlayersLogo = new List<GameObject>();

/\*This variable for:

\* speed-for a set speed moving objects(Animation)

\* ChekPos-for a set position when object will stop(Animation)

\* PositionScrollPointerMusic-for save pointer position in ScrollBarMusic

\* PositionScrollPointerSounds- for save pointer position in ScrollBarSound

\*/

//private RectTransform rec;

//private const float MinPositionScrollPointer = -0.5f;

//private const float MaxPositionScrollPointer = 4f;

//private float speed;

//private float chekPos;

public bool chekCard { set; get; }

/\*This method for make animation when key pressed\*/

/\*This methods for animation selected other colors and names \*/

public void changeScaleOfButton(UnityEngine.UI.Button button, bool flag)

{

if (flag)

button.transform.localScale = new Vector2(button.transform.localScale.x + 0.5f, button.transform.localScale.y + 0.5f);

else

button.transform.localScale = new Vector2(button.transform.localScale.x - 0.5f, button.transform.localScale.y - 0.5f);

}

//for chek scale of button

//return true if scale is increased

//if scale is no increased return false

public bool chekScaleColorBatton(UnityEngine.UI.Button button)

{

if (button.transform.localScale.x > 1 && button.transform.localScale.y > 1)

return true;

return false;

}

//reset scale button to the standart

public void resetScaleButton(List<UnityEngine.UI.Button> list)

{

foreach (UnityEngine.UI.Button button in list)

{

if (chekScaleColorBatton(button))

{

changeScaleOfButton(button, false);

}

}

}

/\*Rotation Button

\*string button-name of button

\*Flag

-true rotate

-false reset rotate

\*/

public void rotationButton(string button, bool flag)

{

if (flag)

GameObject.Find(button).GetComponent<RectTransform>().transform.rotation = Quaternion.Euler(0, 0, 90);

else

GameObject.Find(button).GetComponent<RectTransform>().transform.rotation = Quaternion.Euler(0, 0, 0);

}

/\*for show or close MessageBox

\* Flag

true-show massage box

false-close massage box

\* str- error text

\*/

public void ShowCloseMassageBox(bool flag, string str = " ")

{

if (flag)

{

GameObject.Find("ErrorPanel").GetComponent<UnityEngine.UI.Image>().enabled = true;

GameObject.Find("ButtonForError").GetComponent<UnityEngine.UI.Image>().enabled = true;

GameObject.Find("ButtonForError").GetComponent<UnityEngine.UI.Button>().enabled = true;

GameObject.Find("TextButtonForError").GetComponent<UnityEngine.UI.Text>().enabled = true;

GameObject.Find("ErrorText").GetComponent<UnityEngine.UI.Text>().enabled = true;

GameObject.Find("ErrorText").GetComponent<UnityEngine.UI.Text>().text = str;

}

else

{

GameObject.Find("ErrorPanel").GetComponent<UnityEngine.UI.Image>().enabled = false;

GameObject.Find("ButtonForError").GetComponent<UnityEngine.UI.Image>().enabled = false;

GameObject.Find("ButtonForError").GetComponent<UnityEngine.UI.Button>().enabled = false;

GameObject.Find("TextButtonForError").GetComponent<UnityEngine.UI.Text>().enabled = false;

GameObject.Find("ErrorText").GetComponent<UnityEngine.UI.Text>().enabled = false;

}

}

/\*Dynamic add players to the map

parent-parent gameobj

\*/

public void addPlayersElementsToTheBoad(GameObject parent)

{

for (int i = 0; i < Game.getGame().numberOfPlayers; i++)

{

GameObject tmpObj = new GameObject("Player" + (i + 1));

tmpObj.AddComponent<RectTransform>();

tmpObj.AddComponent<Image>();

tmpObj.GetComponent<Image>().type = Image.Type.Simple;

tmpObj.GetComponent<Image>().preserveAspect = true;

////if set first playes

//if (i == 0)

//{

// //set min anchor

// tmpObj.GetComponent<RectTransform>().anchorMax = new Vector2(parent.GetComponent<RectTransform>().anchorMax.x + 0.2f,

// parent.GetComponent<RectTransform>().anchorMin.y + 0.02f);

// //set max anchor

// tmpObj.GetComponent<RectTransform>().anchorMin = new Vector2(parent.GetComponent<RectTransform>().anchorMin.x + 0.93f,

// parent.GetComponent<RectTransform>().anchorMin.y - 0.04f);

//}

//else

//{

// //set odd player

// if (i % 2 != 0)

// {

// /\*Change only Y coordinates\*/

// //set max anchor

// tmpObj.GetComponent<RectTransform>().anchorMax = new Vector2(Game.getGame()[i - 1].position.anchorMax.x,

// Game.getGame()[i - 1].position.anchorMax.y + 0.07f);

// //set min anchor

// tmpObj.GetComponent<RectTransform>().anchorMin = new Vector2(Game.getGame()[i - 1].position.anchorMin.x,

// Game.getGame()[i - 1].position.anchorMax.y + 0.01f);

// }

// //set even player

// if (i % 2 == 0)

// {

// /\*Change only X coordinates\*/

// //set max anchor

// tmpObj.GetComponent<RectTransform>().anchorMax = new Vector2(Game.getGame()[i - 2].position.anchorMin.x - 0.005f,

// Game.getGame()[i - 2].position.anchorMax.y);

// //set min anchor

// tmpObj.GetComponent<RectTransform>().anchorMin = new Vector2(Game.getGame()[i - 2].position.anchorMin.x - 0.05f,

// Game.getGame()[i - 2].position.anchorMin.y);

// }

//}

//set patern

tmpObj.GetComponent<RectTransform>().transform.SetParent(parent.transform);

//set scale

tmpObj.GetComponent<RectTransform>().transform.localScale = new Vector3(1, 1, 1);

//move to anchors

tmpObj.GetComponent<RectTransform>().transform.localPosition = Vector3.zero;

tmpObj.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

tmpObj.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

// Game.getGame()[i].position = tmpObj.GetComponent<RectTransform>();

LoadPlayerPicture(tmpObj, Game.getGame()[i]);

//PlayersLogo.Add(tmpObj);

}

}

private void LoadPlayerPicture(GameObject obj, Player player)

{

string path = " ";

if (player.colorplayer == "RedColor") path = "RedPointer";

if (player.colorplayer == "YellowColor") path = "YellowPointer";

if (player.colorplayer == "BlueColor") path = "BluePointer";

if (player.colorplayer == "GreenBlueColor") path = "GreenBluePointer";

if (player.colorplayer == "GreenColor") path = "GreenPointer";

if (player.colorplayer == "VioletColor") path = "VioletPointer";

obj.GetComponent<Image>().sprite = Resources.Load<Sprite>(path);

}

public void addInformationAboutPlayer()

{

for (int i = 0; i < Game.getGame().numberOfPlayers; i++)

{

GameObject tmpobj = new GameObject("InformationAboutPlayer" + (i + 1));

tmpobj.transform.SetParent(GameObject.Find("Canvas").transform);

tmpobj.AddComponent<RectTransform>();

tmpobj.AddComponent<Button>();

if (i == 0)

{

//set min anchor

tmpobj.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find("ThrowCubeButton").GetComponent<RectTransform>().anchorMax.x,

GameObject.Find("ThrowCubeButton").GetComponent<RectTransform>().anchorMin.y - 0.02f);

//set max anchor

tmpobj.GetComponent<RectTransform>().anchorMin = new Vector2(GameObject.Find("ThrowCubeButton").GetComponent<RectTransform>().anchorMin.x,

GameObject.Find("ThrowCubeButton").GetComponent<RectTransform>().anchorMax.y - 0.2f);

}

else

{

//set min anchor

tmpobj.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find("InformationAboutPlayer" + (i)).GetComponent<RectTransform>().anchorMax.x,

GameObject.Find("InformationAboutPlayer" + (i)).GetComponent<RectTransform>().anchorMin.y + 0.02f);

//set max anchor

tmpobj.GetComponent<RectTransform>().anchorMin = new Vector2(GameObject.Find("InformationAboutPlayer" + (i)).GetComponent<RectTransform>().anchorMin.x,

GameObject.Find("InformationAboutPlayer" + (i)).GetComponent<RectTransform>().anchorMax.y - 0.2f);

}

tmpobj.GetComponent<RectTransform>().transform.localPosition = Vector3.zero;

tmpobj.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

tmpobj.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

tmpobj.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

GameObject tmpText = new GameObject("InformationTextPlayer" + (i + 1));

tmpText.transform.SetParent(tmpobj.transform);

tmpText.AddComponent<RectTransform>();

tmpText.transform.SetParent(tmpobj.transform);

tmpText.GetComponent<RectTransform>().anchorMax = new Vector2(1, 1);

tmpText.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

tmpText.GetComponent<RectTransform>().transform.localPosition = Vector3.zero;

tmpText.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

tmpText.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

tmpText.AddComponent<Text>();

tmpText.GetComponent<Text>().text = Game.getGame()[i].namePlayer;

tmpText.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

tmpText.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

tmpText.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

}

}

public void showPanelInformation(Player player)

{

GameObject.Find("PanelNamePlayer").GetComponent<Text>().text = player.namePlayer;

GameObject.Find("PanelPealPlayerMoney").GetComponent<Text>().text = player.moneyPlayer + " $";

GameObject.Find("InformationPanelAboutPlayer").GetComponent<Animations>().speed = 5;

GameObject.Find("InformationPanelAboutPlayer").GetComponent<Animations>().chekPos = 0;

GameObject.Find("InformationPanelAboutPlayer").GetComponent<Animations>().enabled = true;

//createAllElementsOPlayerBoard(player);

GameObject.Find("CloseButton").GetComponent<Button>().onClick.AddListener(closePanelInformation);

}

private void closePanelInformation()

{

GameObject.Find("InformationPanelAboutPlayer").GetComponent<Animations>().speed = -5;

GameObject.Find("InformationPanelAboutPlayer").GetComponent<Animations>().chekPos = -500;

}

public void showInformationAboutBoardElemets(BoardElements elements)

{

GameObject.Find("BoardElementInformation").GetComponent<Animations>().speed = 5;

GameObject.Find("BoardElementInformation").GetComponent<Animations>().chekPos = 0;

GameObject.Find("NameBoardElements").GetComponent<Text>().text = elements.Name;

GameObject.Find("LogoBoardElements").GetComponent<Image>().sprite = elements.sprite;

// GameObject.Find("BoardElementsCloseButton").GetComponent<Button>().onClick.AddListener(closeInformationAboutBoardElements);

}

public void closeInformationAboutBoardElements()

{

GameObject.Find("BoardElementInformation").GetComponent<Animations>().speed = -5;

GameObject.Find("BoardElementInformation").GetComponent<Animations>().chekPos = -715;

try

{

destroyPropertyPanel();

}

catch(System.Exception ex)

{

}

try

{

destroyPayOnlyPanel();

}

catch (System.Exception ex)

{

}

try

{

destroySinglePropertyPanel();

}

catch (System.Exception ex)

{

}

try

{

destroyCardPanel()

;

}

catch (System.Exception ex)

{

}

}

//create and destroy PropertyPanel

public void createPanelForProperty(BoardElements element)

{

showInformationAboutBoardElemets(element);

/\*-----------------CREATE TEXT--------------------------------\*/

//add text for show price propetry

GameObject textPrice = new GameObject("BoardElementsPrice");

textPrice.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

textPrice.AddComponent<RectTransform>();

textPrice.AddComponent<Text>();

//set anchors position

textPrice.GetComponent<RectTransform>().anchorMax = new Vector2(1, GameObject.Find("BoardElementsCloseButton").GetComponent<RectTransform>().anchorMin.y);

textPrice.GetComponent<RectTransform>().anchorMin = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMax.x, GameObject.Find("BoardElementsCloseButton").GetComponent<RectTransform>().anchorMin.y - 0.15f);

textPrice.GetComponent<RectTransform>().localPosition = Vector3.zero;

textPrice.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text settings

textPrice.GetComponent<Text>().resizeTextForBestFit = true;

textPrice.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

textPrice.GetComponent<Text>().font= GameObject.Find("CubeNumber").GetComponent<Text>().font;

textPrice.GetComponent<Text>().text ="Price: "+System.Convert.ToString(element.boardProterty.Price);

/\*NEW OBJECT\*/

//text for ownerName

textPrice = new GameObject("BoardElementUpdateLevel");

textPrice.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

textPrice.AddComponent<RectTransform>();

textPrice.AddComponent<Text>();

//set anchors position

textPrice.GetComponent<RectTransform>().anchorMax = new Vector2(1, GameObject.Find("BoardElementsPrice").GetComponent<RectTransform>().anchorMin.y-0.1f);

textPrice.GetComponent<RectTransform>().anchorMin = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMax.x, GameObject.Find("BoardElementsPrice").GetComponent<RectTransform>().anchorMin.y - 0.25f);

textPrice.GetComponent<RectTransform>().localPosition = Vector3.zero;

textPrice.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text settings

textPrice.GetComponent<Text>().resizeTextForBestFit = true;

textPrice.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

textPrice.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

textPrice.GetComponent<Text>().text = "Update Level: " + System.Convert.ToString(element.boardProterty.UpdateLevel);

//if element have owner

if (element.boardProterty.isOwner == true)

{

//new object text for ownerName

textPrice = new GameObject("BoardElementOwnerName");

textPrice.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

textPrice.AddComponent<RectTransform>();

textPrice.AddComponent<Text>();

textPrice.GetComponent<RectTransform>().anchorMax = new Vector2(1, GameObject.Find("BoardElementUpdateLevel").GetComponent<RectTransform>().anchorMin.y-0.1f);

textPrice.GetComponent<RectTransform>().anchorMin = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMax.x, GameObject.Find("BoardElementUpdateLevel").GetComponent<RectTransform>().anchorMin.y - 0.25f);

textPrice.GetComponent<RectTransform>().localPosition = Vector3.zero;

textPrice.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text settings

textPrice.GetComponent<Text>().resizeTextForBestFit = true;

textPrice.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

textPrice.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

textPrice.GetComponent<Text>().text = "Owner: " + element.boardProterty.OwnerName;

}

/\*---------------CREATE BUTTONS------------------\*/

if (element.boardProterty.isOwner == false)

{

GameObject button = new GameObject("BoardElementBuyButton");

button.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

button.AddComponent<RectTransform>();

button.AddComponent<Button>();

//set anchors

button.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin.x, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.15f);

button.GetComponent<RectTransform>().anchorMin = new Vector2(0, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.3f);

button.GetComponent<RectTransform>().localPosition = Vector3.zero;

button.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

button.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

button.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

//create text for button

GameObject buttonText = new GameObject("BoardElementsBuyButtonText");

buttonText.transform.SetParent(GameObject.Find("BoardElementBuyButton").transform);

buttonText.AddComponent<RectTransform>();

buttonText.AddComponent<Text>();

//set achors

buttonText.GetComponent<RectTransform>().anchorMax = new Vector2(1, 1);

buttonText.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localPosition = Vector3.zero;

buttonText.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text

buttonText.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

buttonText.GetComponent<Text>().resizeTextForBestFit = true;

buttonText.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

buttonText.GetComponent<Text>().text = "Buy";

/\*NEW BUTTON\*/

//SkipButton

button = new GameObject("BoardElementSkipButton");

button.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

button.AddComponent<RectTransform>();

button.AddComponent<Button>();

//set anchors

button.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin.x, GameObject.Find("BoardElementBuyButton").GetComponent<RectTransform>().anchorMin.y - 0.15f);

button.GetComponent<RectTransform>().anchorMin = new Vector2(0, GameObject.Find("BoardElementBuyButton").GetComponent<RectTransform>().anchorMin.y - 0.3f);

button.GetComponent<RectTransform>().localPosition = Vector3.zero;

button.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

button.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

button.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

//create text for button

buttonText = new GameObject("BoardElementSkipButton");

buttonText.transform.SetParent(GameObject.Find("BoardElementSkipButton").transform);

buttonText.AddComponent<RectTransform>();

buttonText.AddComponent<Text>();

//set achors

buttonText.GetComponent<RectTransform>().anchorMax = new Vector2(1, 1);

buttonText.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localPosition = Vector3.zero;

buttonText.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text

buttonText.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

buttonText.GetComponent<Text>().resizeTextForBestFit = true;

buttonText.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

buttonText.GetComponent<Text>().text = "Skip";

}

else

{

GameObject button = new GameObject("BoardElementPayorUpdateButton");

button.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

button.AddComponent<RectTransform>();

button.AddComponent<Button>();

//set anchors

button.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin.x, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.15f);

button.GetComponent<RectTransform>().anchorMin = new Vector2(0, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.3f);

button.GetComponent<RectTransform>().localPosition = Vector3.zero;

button.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

button.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

button.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

//create text for button

GameObject buttonText = new GameObject("BoardElementsPayorUpdateButtonText");

buttonText.transform.SetParent(GameObject.Find("BoardElementPayorUpdateButton").transform);

buttonText.AddComponent<RectTransform>();

buttonText.AddComponent<Text>();

//set achors

buttonText.GetComponent<RectTransform>().anchorMax = new Vector2(1, 1);

buttonText.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localPosition = Vector3.zero;

buttonText.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text

buttonText.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

buttonText.GetComponent<Text>().resizeTextForBestFit = true;

buttonText.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

}

}

private void destroyPropertyPanel()

{

MonoBehaviour.Destroy(GameObject.Find("BoardElementsPrice"));

MonoBehaviour.Destroy(GameObject.Find("BoardElementUpdateLevel"));

MonoBehaviour.Destroy(GameObject.Find("BoardElementBuyButton"));

MonoBehaviour.Destroy(GameObject.Find("BoardElementSkipButton"));

try

{

MonoBehaviour.Destroy(GameObject.Find("BoardElementOwnerName"));

MonoBehaviour.Destroy(GameObject.Find("BoardElementPayorUpdateButton"));

}

catch (System.Exception ex)

{

}

}

//create and destroy PayOnlypanel

public void createPanelForPayOnly(BoardElements element)

{

showInformationAboutBoardElemets(element);

/\*-----------------CREATE TEXT--------------------------------\*/

//add text for show price propetry

GameObject textPrice = new GameObject("BoardElementsPrice");

textPrice.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

textPrice.AddComponent<RectTransform>();

textPrice.AddComponent<Text>();

//set anchors position

textPrice.GetComponent<RectTransform>().anchorMax = new Vector2(1, GameObject.Find("BoardElementsCloseButton").GetComponent<RectTransform>().anchorMin.y);

textPrice.GetComponent<RectTransform>().anchorMin = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMax.x, GameObject.Find("BoardElementsCloseButton").GetComponent<RectTransform>().anchorMin.y - 0.15f);

textPrice.GetComponent<RectTransform>().localPosition = Vector3.zero;

textPrice.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text settings

textPrice.GetComponent<Text>().resizeTextForBestFit = true;

textPrice.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

textPrice.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

textPrice.GetComponent<Text>().text = "You need pay: " + System.Convert.ToString(element.boardPayOnly.Price);

/\*---------------CREATE BUTTONS------------------\*/

GameObject button = new GameObject("BoardElementPayButton");

button.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

button.AddComponent<RectTransform>();

button.AddComponent<Button>();

//set anchors

button.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin.x, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.15f);

button.GetComponent<RectTransform>().anchorMin = new Vector2(0, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.3f);

button.GetComponent<RectTransform>().localPosition = Vector3.zero;

button.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

button.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

button.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

//create text for button

GameObject buttonText = new GameObject("BoardElementsPayButtonText");

buttonText.transform.SetParent(GameObject.Find("BoardElementPayButton").transform);

buttonText.AddComponent<RectTransform>();

buttonText.AddComponent<Text>();

//set achors

buttonText.GetComponent<RectTransform>().anchorMax = new Vector2(1, 1);

buttonText.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localPosition = Vector3.zero;

buttonText.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text

buttonText.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

buttonText.GetComponent<Text>().resizeTextForBestFit = true;

buttonText.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

buttonText.GetComponent<Text>().text = "Pay";

}

public void destroyPayOnlyPanel()

{

MonoBehaviour.Destroy(GameObject.Find("BoardElementsPrice"));

MonoBehaviour.Destroy(GameObject.Find("BoardElementPayButton"));

}

//create and destroy SingleProperty Panel

public void createPanelForSingleProperty(BoardElements element)

{

showInformationAboutBoardElemets(element);

/\*-----------------CREATE TEXT--------------------------------\*/

//add text for show price propetry

GameObject textPrice = new GameObject("BoardElementsPrice");

textPrice.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

textPrice.AddComponent<RectTransform>();

textPrice.AddComponent<Text>();

//set anchors position

textPrice.GetComponent<RectTransform>().anchorMax = new Vector2(1, GameObject.Find("BoardElementsCloseButton").GetComponent<RectTransform>().anchorMin.y);

textPrice.GetComponent<RectTransform>().anchorMin = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMax.x, GameObject.Find("BoardElementsCloseButton").GetComponent<RectTransform>().anchorMin.y - 0.15f);

textPrice.GetComponent<RectTransform>().localPosition = Vector3.zero;

textPrice.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text settings

textPrice.GetComponent<Text>().resizeTextForBestFit = true;

textPrice.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

textPrice.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

textPrice.GetComponent<Text>().text = "Price: " + System.Convert.ToString(element.boardSingleProperty.Price);

/\*NEW OBJECT\*/

//if element have owner

if (element.boardSingleProperty.isOwner == true)

{

//new object text for ownerName

textPrice = new GameObject("BoardElementOwnerName");

textPrice.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

textPrice.AddComponent<RectTransform>();

textPrice.AddComponent<Text>();

textPrice.GetComponent<RectTransform>().anchorMax = new Vector2(1, GameObject.Find("BoardElementsPrice").GetComponent<RectTransform>().anchorMin.y);

textPrice.GetComponent<RectTransform>().anchorMin = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMax.x, GameObject.Find("BoardElementsPrice").GetComponent<RectTransform>().anchorMin.y - 0.15f);

textPrice.GetComponent<RectTransform>().localPosition = Vector3.zero;

textPrice.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

textPrice.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text settings

textPrice.GetComponent<Text>().resizeTextForBestFit = true;

textPrice.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

textPrice.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

textPrice.GetComponent<Text>().text = "Owner: " + element.boardSingleProperty.OwnerName;

}

/\*---------------CREATE BUTTONS------------------\*/

if (element.boardSingleProperty.isOwner == false)

{

GameObject button = new GameObject("BoardElementBuyButton");

button.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

button.AddComponent<RectTransform>();

button.AddComponent<Button>();

//set anchors

button.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin.x, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.15f);

button.GetComponent<RectTransform>().anchorMin = new Vector2(0, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.3f);

button.GetComponent<RectTransform>().localPosition = Vector3.zero;

button.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

button.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

button.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

//create text for button

GameObject buttonText = new GameObject("BoardElementsBuyButtonText");

buttonText.transform.SetParent(GameObject.Find("BoardElementBuyButton").transform);

buttonText.AddComponent<RectTransform>();

buttonText.AddComponent<Text>();

//set achors

buttonText.GetComponent<RectTransform>().anchorMax = new Vector2(1, 1);

buttonText.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localPosition = Vector3.zero;

buttonText.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text

buttonText.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

buttonText.GetComponent<Text>().resizeTextForBestFit = true;

buttonText.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

buttonText.GetComponent<Text>().text = "Buy";

/\*NEW BUTTON\*/

//SkipButton

button = new GameObject("BoardElementSkipButton");

button.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

button.AddComponent<RectTransform>();

button.AddComponent<Button>();

//set anchors

button.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin.x, GameObject.Find("BoardElementBuyButton").GetComponent<RectTransform>().anchorMin.y - 0.15f);

button.GetComponent<RectTransform>().anchorMin = new Vector2(0, GameObject.Find("BoardElementBuyButton").GetComponent<RectTransform>().anchorMin.y - 0.3f);

button.GetComponent<RectTransform>().localPosition = Vector3.zero;

button.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

button.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

button.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

//create text for button

buttonText = new GameObject("BoardElementSkipButton");

buttonText.transform.SetParent(GameObject.Find("BoardElementSkipButton").transform);

buttonText.AddComponent<RectTransform>();

buttonText.AddComponent<Text>();

//set achors

buttonText.GetComponent<RectTransform>().anchorMax = new Vector2(1, 1);

buttonText.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localPosition = Vector3.zero;

buttonText.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text

buttonText.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

buttonText.GetComponent<Text>().resizeTextForBestFit = true;

buttonText.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

buttonText.GetComponent<Text>().text = "Skip";

}

else

{

GameObject button = new GameObject("BoardElementBuyButton");

button.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

button.AddComponent<RectTransform>();

button.AddComponent<Button>();

//set anchors

button.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin.x, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.15f);

button.GetComponent<RectTransform>().anchorMin = new Vector2(0, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.3f);

button.GetComponent<RectTransform>().localPosition = Vector3.zero;

button.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

button.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

button.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

//create text for button

GameObject buttonText = new GameObject("BoardElementsBuyButtonText");

buttonText.transform.SetParent(GameObject.Find("BoardElementBuyButton").transform);

buttonText.AddComponent<RectTransform>();

buttonText.AddComponent<Text>();

//set achors

buttonText.GetComponent<RectTransform>().anchorMax = new Vector2(1, 1);

buttonText.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localPosition = Vector3.zero;

buttonText.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text

buttonText.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

buttonText.GetComponent<Text>().resizeTextForBestFit = true;

buttonText.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

/\*---------NEW BUTTON------------------------\*/

GameObject button1 = new GameObject("BoardElementSellButton");

button.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

button.AddComponent<RectTransform>();

button.AddComponent<Button>();

//set anchors

button.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find("BoardElementBuyButton").GetComponent<RectTransform>().anchorMin.x, GameObject.Find("BoardElementBuyButton").GetComponent<RectTransform>().anchorMin.y - 0.15f);

button.GetComponent<RectTransform>().anchorMin = new Vector2(0, GameObject.Find("BoardElementBuyButton").GetComponent<RectTransform>().anchorMin.y - 0.3f);

button.GetComponent<RectTransform>().localPosition = Vector3.zero;

button.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

button.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

button.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

//create text for button

GameObject buttonTexts = new GameObject("BoardElementSellButtonText");

buttonText.transform.SetParent(GameObject.Find("BoardElementSellButton").transform);

buttonText.AddComponent<RectTransform>();

buttonText.AddComponent<Text>();

//set achors

buttonText.GetComponent<RectTransform>().anchorMax = new Vector2(1, 1);

buttonText.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localPosition = Vector3.zero;

buttonText.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text

buttonText.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

buttonText.GetComponent<Text>().resizeTextForBestFit = true;

buttonText.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

buttonTexts.GetComponent<Text>().text = "Sell";

GameObject.Find("BoardElementSellButton").GetComponent<Button>().enabled = false;

GameObject.Find("BoardElementSellButtonText").GetComponent<Text>().enabled = false;

}

}

private void destroySinglePropertyPanel()

{

MonoBehaviour.Destroy(GameObject.Find("BoardElementsPrice"));

MonoBehaviour.Destroy(GameObject.Find("BoardElementBuyButton"));

MonoBehaviour.Destroy(GameObject.Find("BoardElementSkipButton"));

try

{

MonoBehaviour.Destroy(GameObject.Find("BoardElementOwnerName"));

MonoBehaviour.Destroy(GameObject.Find("BoardElementSellButton"));

}

catch (System.Exception ex)

{

}

}

//create and destroy cardPanel

public void createPanelCard(BoardElements element)

{

showInformationAboutBoardElemets(element);

/\*---------------CREATE BUTTONS------------------\*/

GameObject button = new GameObject("BoardElementDoButton");

button.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

button.AddComponent<RectTransform>();

button.AddComponent<Button>();

//set anchors

button.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin.x, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.2f);

button.GetComponent<RectTransform>().anchorMin = new Vector2(0, GameObject.Find("NameBoardElements").GetComponent<RectTransform>().anchorMin.y - 0.35f);

button.GetComponent<RectTransform>().localPosition = Vector3.zero;

button.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

button.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

button.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

//create text for button

GameObject buttonText = new GameObject("BoardElementsTakeCardButtonText");

buttonText.transform.SetParent(GameObject.Find("BoardElementDoButton").transform);

buttonText.AddComponent<RectTransform>();

buttonText.AddComponent<Text>();

//set achors

buttonText.GetComponent<RectTransform>().anchorMax = new Vector2(1, 1);

buttonText.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localPosition = Vector3.zero;

buttonText.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text

buttonText.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

buttonText.GetComponent<Text>().resizeTextForBestFit = true;

buttonText.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

buttonText.GetComponent<Text>().text = "Take a card";

/\*temp\*/

// b.GetComponent<Button>().enabled = false;

///////////////////////////

GameObject.Find("BoardElementDoButton").GetComponent<Button>().onClick.AddListener(delegate { showRandCard(element); });

}

public void showRandCard(BoardElements element)

{

GameObject.Find("BoardElementDoButton").GetComponent<Button>().enabled = false;

GameObject.Find("BoardElementsTakeCardButtonText").GetComponent<Text>().enabled = false;

// destroyCardPanel();

System.Random rand = new System.Random();

int randNumber = rand.Next(1, element.boardCard.Images.Length);

//for save anchors

massAchors[0] = GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMax.x;

massAchors[1] = GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMax.y;

massAchors[2]=GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin.x;

massAchors[3]=GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin.y;

//set new positions

GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMax = new Vector2(1, 1);

GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().localPosition = Vector3.zero;

GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

GameObject.Find("LogoBoardElements").GetComponent<Image>().sprite = element.boardCard.Images[randNumber].Value;

// GameObject.Find("BoardElementDoButton").GetComponent<Button>().enabled = false;

//GameObject b = new GameObject("newDoButton");

//b.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

//b.AddComponent<RectTransform>();

//b.GetComponent<RectTransform>().anchorMax = new Vector2(1, 0.3f);

//b.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0.1f);

//b.GetComponent<RectTransform>().localScale = Vector3.zero;

//b.GetComponent<RectTransform>().localPosition = Vector3.zero;

//b.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

//b.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

//b.AddComponent<Button>();

//b.GetComponent<Button>().onClick.AddListener(closeInformationAboutBoardElements);

//b.GetComponent<Button>().onClick.AddListener(returnAnchors);

//GameObject buttonTexts = new GameObject("BoardElementsDopButtonText");

//buttonTexts.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

//buttonTexts.AddComponent<RectTransform>();

//buttonTexts.AddComponent<Text>();

////set achors

//buttonTexts.GetComponent<RectTransform>().anchorMax = new Vector2(1, 0.3f);

//buttonTexts.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0.1f);

//buttonTexts.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//buttonTexts.GetComponent<RectTransform>().localPosition = Vector3.zero;

//buttonTexts.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

//buttonTexts.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

////set text

//buttonTexts.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

//buttonTexts.GetComponent<Text>().resizeTextForBestFit = true;

//buttonTexts.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

//buttonTexts.GetComponent<Text>().text = "Do Action";

// GameObject.Find("BoardElementDoButton").GetComponent<RectTransform>().anchorMax = new Vector2(1,0.3f);

// GameObject.Find("BoardElementDoButton").GetComponent<RectTransform>().anchorMin = new Vector2(0,0.1f);

// GameObject.Find("BoardElementDoButton").GetComponent<RectTransform>().localPosition = Vector3.zero;

// GameObject.Find("BoardElementDoButton").GetComponent<RectTransform>().offsetMax = new Vector2(0,0);

// GameObject.Find("BoardElementDoButton").GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

//GameObject.Find("BoardElementsTakeCardButtonText").GetComponent<Text>().text = "Do Action";

//GameObject.Find("BoardElementDoButton").GetComponent<Button>().enabled = false;

//GameObject.Find("BoardElementDoButton").GetComponent<Text>().enabled = false;

GameObject buttonText = new GameObject("BoardElementsIDCardText");

buttonText.transform.SetParent(GameObject.Find("BoardElementInformation").transform);

buttonText.AddComponent<RectTransform>();

buttonText.AddComponent<Text>();

//set achors

buttonText.GetComponent<RectTransform>().anchorMax = new Vector2(0.3f, 0.3f);

buttonText.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localPosition = Vector3.zero;

buttonText.GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

buttonText.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

//set text

buttonText.GetComponent<Text>().font = GameObject.Find("CubeNumber").GetComponent<Text>().font;

buttonText.GetComponent<Text>().resizeTextForBestFit = true;

buttonText.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

buttonText.GetComponent<Text>().text = System.Convert.ToString(element.boardCard.Images[randNumber].Key);

GameObject.Find("newDoButton").GetComponent<Button>().enabled = true;

GameObject.Find("newDoButtonText").GetComponent<Text>().enabled = true;

this.chekCard = false;

}

public void returnAnchors()

{

GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMax = new Vector2(massAchors[0], massAchors[1]);

GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().anchorMin = new Vector2(massAchors[2], massAchors[3]);

GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().offsetMax = new Vector2(0, 0);

GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().offsetMin = new Vector2(0, 0);

GameObject.Find("LogoBoardElements").GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

}

private void destroyCardPanel()

{

MonoBehaviour.Destroy(GameObject.Find("BoardElementDoButton"));

}

public void animationTrowCube()

{

}

public void changeTextPlayer(int index)

{

for (int i = 0; i < Game.getGame().numberOfPlayers; i++)

{

GameObject.Find("InformationTextPlayer" + (i + 1)).GetComponent<Text>().fontStyle = FontStyle.Normal;

}

GameObject.Find("InformationTextPlayer" + (index + 1)).GetComponent<Text>().fontStyle = FontStyle.BoldAndItalic;

}

//public void createAllElementsOPlayerBoard(Player player)

//{

// for (int i = 0; i < player.PlayersElements.Count/3; i++)

// {

// GameObject tmpObj = new GameObject(player.PlayersElements[i].Name + "onBoard");

// tmpObj.AddComponent<RectTransform>();

// tmpObj.transform.SetParent(GameObject.Find("AllPlayersElementBoard").transform);

// tmpObj.AddComponent<Button>();

// //

// for (int j = 0; j < 3; j++)

// {

// if (j == 0)

// {

// tmpObj.GetComponent<RectTransform>().anchorMax = new Vector2(0.3f, 1);

// tmpObj.GetComponent<RectTransform>().anchorMin = new Vector2(0, 0.3f);

// }

// else if (j == 1)

// {

// tmpObj.GetComponent<RectTransform>().anchorMax = new Vector2(0.65f, GameObject.Find(player.PlayersElements[j - 1].Name + "onBoard").GetComponent<RectTransform>().anchorMax.y);

// tmpObj.GetComponent<RectTransform>().anchorMin = new Vector2(0.035f, GameObject.Find(player.PlayersElements[j - 1].Name + "onBoard").GetComponent<RectTransform>().anchorMin.y);

// }

// else if (j== 2)

// {

// tmpObj.GetComponent<RectTransform>().anchorMax = new Vector2(1, GameObject.Find(player.PlayersElements[j - 1].Name + "onBoard").GetComponent<RectTransform>().anchorMax.y);

// tmpObj.GetComponent<RectTransform>().anchorMin = new Vector2(0.75f, GameObject.Find(player.PlayersElements[j - 1].Name + "onBoard").GetComponent<RectTransform>().anchorMin.y);

// //tmpObj.GetComponent<RectTransform>().anchorMax = new Vector2(GameObject.Find(player.PlayersElements[i - 2].Name + "onBoard").GetComponent<RectTransform>().anchorMax.x, GameObject.Find(player.PlayersElements[i - 2].Name + "onBoard").GetComponent<RectTransform>().anchorMin.y + 0.2f);

// //tmpObj.GetComponent<RectTransform>().anchorMin = new Vector2(GameObject.Find(player.PlayersElements[i - 2].Name + "onBoard").GetComponent<RectTransform>().anchorMin.x, GameObject.Find(player.PlayersElements[i - 2].Name + "onBoard").GetComponent<RectTransform>().anchorMin.y + 0.05f);

// }

// tmpObj.GetComponent<RectTransform>().localPosition = Vector3.zero;

// tmpObj.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);

// tmpObj.GetComponent<RectTransform>().offsetMax = Vector2.zero;

// tmpObj.GetComponent<RectTransform>().offsetMin = Vector2.zero;

// tmpObj.AddComponent<Image>();

// tmpObj.GetComponent<Image>().sprite = player.PlayersElements[i].sprite;

// }

// }

// GameObject.Find("PanelButtonSELL").GetComponent<Button>().onClick.AddListener(showAllElementsOPlayerBoard);

//}

public void showAllElementsOPlayerBoard()

{

GameObject.Find("AllPlayersElementBoard").GetComponent<Animations>().speed = -5;

GameObject.Find("AllPlayersElementBoard").GetComponent<Animations>().chekPos = 0;

GameObject.Find("AllPlayersElementBoard").GetComponent<Animations>().enabled = true;

}

}

struct Point

{

float x;

float y;

}

abstract public class Player

{

public Player()

{

namePlayer = " ";

RealTimePosition = 0;

PlayersElements = new List<BoardElements>();

}

//name of player

public string namePlayer { set; get; }

//color of other element player

public string colorplayer { set; get; }

//count money of player

public int moneyPlayer { set; get; }

//positions player

public GameObject LogoPlayer { set; get; }

public int RealTimePosition { set; get; }

public List<BoardElements> PlayersElements;

abstract public int makeMove(int step);

abstract public bool buy(BoardElements boardElements);

abstract public bool payOnly(BoardElements boardElements);

abstract public bool updateLevel(BoardElements boardElements);

abstract public bool payRent(BoardElements boardElements);

abstract public int doAction(int ID, BoardElements boardElements);

abstract public void sell(BoardElements elements);

abstract public bool chekLive();

}

public class RealPlayer : Player

{

public RealPlayer() : base()

{

}

public override int makeMove(int step)

{

this.RealTimePosition += step;

//fix bugs

if(this.RealTimePosition>40)

{

this.RealTimePosition -= 40;

}

return this.RealTimePosition;

}

public override bool buy(BoardElements boardElements)

{

//if you can buy element

if(boardElements.boardProterty!=null)

{

if (boardElements.boardProterty.isOwner == false)

{

if (this.moneyPlayer > boardElements.boardProterty.Price)

{

this.PlayersElements.Add(boardElements);

this.moneyPlayer -= boardElements.boardProterty.Price;

boardElements.boardProterty.isOwner = true;

boardElements.boardProterty.OwnerName = this.namePlayer;

return true;

}

}

else

{

//оплата другому игроку

}

}

//if you can buy element

if (boardElements.boardSingleProperty != null)

{

if (boardElements.boardSingleProperty.isOwner == false)

{

if (this.moneyPlayer > boardElements.boardSingleProperty.Price)

{

this.PlayersElements.Add(boardElements);

this.moneyPlayer -= boardElements.boardSingleProperty.Price;

boardElements.boardSingleProperty.isOwner = true;

boardElements.boardSingleProperty.OwnerName = this.namePlayer;

return true;

}

}

else

{

//оплата другому игроку

}

}

return false;

}

public override bool payOnly(BoardElements boardElements)

{

if(boardElements.boardPayOnly!=null)

{

if(this.moneyPlayer>boardElements.boardPayOnly.Price)

{

this.moneyPlayer -= boardElements.boardPayOnly.Price;

return true;

}

return false;

}

return false;

}

public override bool updateLevel(BoardElements boardElements)

{

if(boardElements.boardProterty!=null)

{

if(this.moneyPlayer>boardElements.boardProterty.Price)

{

this.moneyPlayer -= boardElements.boardProterty.Price;

boardElements.boardProterty.UpdateLevel++;

return true;

}

}

return false;

}

public override bool payRent(BoardElements boardElements)

{

if(boardElements.boardProterty!=null)

{

if(boardElements.boardProterty.isOwner==true)

{

if(this.moneyPlayer>boardElements.boardProterty.Price)

{

this.moneyPlayer -= boardElements.boardProterty.Price;

Game.getGame().findByName(boardElements.boardProterty.OwnerName).moneyPlayer += boardElements.boardProterty.Price;

return true;

}

}

}

if (boardElements.boardSingleProperty != null)

{

if (boardElements.boardSingleProperty.isOwner == true)

{

if (this.moneyPlayer > boardElements.boardSingleProperty.Price)

{

this.moneyPlayer -= boardElements.boardSingleProperty.Price;

Game.getGame().findByName(boardElements.boardSingleProperty.OwnerName).moneyPlayer += boardElements.boardSingleProperty.Price;

return true;

}

}

}

return false;

}

public override int doAction(int ID, BoardElements boardElements)

{

if(boardElements.boardCard.Type== CARDTYPE.CHANCE)

{

/\*

You need go to Start.You take 200

\*/

if(ID==1)

{

this.RealTimePosition = 0;

this.moneyPlayer += 200;

return ID;

}

/\*

Happy Birsday.Today is you birthday.All players give you gift 150

\*/

if (ID == 2)

{

for (int i = 0; i < Game.getGame().numberOfPlayers; i++)

{

Game.getGame()[i].moneyPlayer -= 150;

this.moneyPlayer += 150;

}

return ID;

}

/\*

Certificate 500

\*/

if (ID == 3)

{

this.moneyPlayer += 500;

return ID;

}

/\*

Certificate 100

\*/

if (ID == 4)

{

this.moneyPlayer += 100;

return ID;

}

/\*

Take back one is your property

\*/

if (ID == 5)

{

return ID;

}

/\*

Throw dice again

\*/

if (ID == 6)

{

return ID;

}

/\*

Go to Rector's Office

\*/

if (ID == 7)

{

return ID;

}

/\*

Mulct 150

\*/

if (ID == 8)

{

this.moneyPlayer -= 150;

return ID;

}

/\*

Friend birsday 200

\*/

if (ID == 9)

{

System.Random r = new System.Random();

int ran = r.Next(0, Game.getGame().numberOfPlayers);

this.moneyPlayer -= 200;

Game.getGame()[ran].moneyPlayer += 200;

return ID;

}

/\*

Bank take one is your property

\*/

if (ID == 10)

{

return ID;

}

/\*

Mucl 100

\*/

if (ID == 11)

{

this.moneyPlayer -= 100;

return ID;

}

}

if(boardElements.boardCard.Type== CARDTYPE.COMUNITY)

{

/\*

You need go to Start.You take 200

\*/

if (ID == 1)

{

this.RealTimePosition = 0;

this.moneyPlayer += 200;

return ID;

}

/\*

Happy Birsday.Today is you birthday.All players give you gift 150

\*/

if (ID == 2)

{

for (int i = 0; i < Game.getGame().numberOfPlayers; i++)

{

Game.getGame()[i].moneyPlayer -= 150;

this.moneyPlayer += 150;

}

return ID;

}

/\*

Certificate 500

\*/

if (ID == 3)

{

this.moneyPlayer += 500;

return ID;

}

/\*

Certificate 100

\*/

if (ID == 4)

{

this.moneyPlayer += 100;

return ID;

}

/\*

Take back one is your property

\*/

if (ID == 5)

{

return ID;

}

/\*

Throw dice again

\*/

if (ID == 6)

{

return ID;

}

/\*

Go to Rector's Office

\*/

if (ID == 7)

{

return ID;

}

/\*

Mulct 150

\*/

if (ID == 8)

{

this.moneyPlayer -= 150;

return ID;

}

/\*

Friend birsday 200

\*/

if (ID == 9)

{

System.Random r = new System.Random();

int ran = r.Next(0, Game.getGame().numberOfPlayers);

this.moneyPlayer -= 200;

Game.getGame()[ran].moneyPlayer += 200;

return ID;

}

/\*

Bank take one is your property

\*/

if (ID == 10)

{

return ID;

}

/\*

Mucl 100

\*/

if (ID == 11)

{

this.moneyPlayer -= 100;

return ID;

}

}

return ID;

}

public override bool chekLive()

{

if (this.moneyPlayer > 0) return true;

return false;

}

public override void sell(BoardElements elements )

{

if (elements.boardProterty != null)

{

this.moneyPlayer += elements.boardProterty.Price;

this.PlayersElements.Remove(elements);

elements.boardProterty.isOwner = false;

elements.boardProterty.OwnerName = "";

elements.boardProterty.UpdateLevel = 0;

}

else if(elements.boardSingleProperty!=null)

{

this.moneyPlayer += elements.boardSingleProperty.Price;

this.PlayersElements.Remove(elements);

elements.boardSingleProperty.isOwner = false;

elements.boardSingleProperty.OwnerName = "";

}

}

}

class ButtonClickSound

{

private static ButtonClickSound sound;

private static AudioSource ButtonClick;

static public ButtonClickSound GetSounds()

{

if (sound == null)

sound = new ButtonClickSound();

return sound;

}

private ButtonClickSound()

{

initializationSound();

}

private void initializationSound()

{

ButtonClick = GameObject.Find("ButtonClickSounds").GetComponent<AudioSource>();

Debug.Log("Sound initiasated");

}

public void play()

{

initializationSound();

ButtonClick.Play();

}

public void stop()

{

ButtonClick.Stop();

}

public void setMute()

{

ButtonClick.mute = true;

}

public void unsetMute()

{

ButtonClick.mute = false;

}

public void changeVolume(float level)

{

ButtonClick.volume = level;

}

public float getVolumeLevel()

{

return ButtonClick.volume;

}

}

/\*

\* This class for music which playing in main menu

\* This a SINGLETONE class

\*/

class MenuMusic

{

/\*

\*mute- for check our music enabled or no, and change this value

\* \_instanse-for get singletone object

\* MainNusic- this variable for storage music

\*/

private bool mute;

private static AudioSource MainMusic;

private static MenuMusic \_instanse;

static public MenuMusic GetMainMusic()

{

if (\_instanse==null)

\_instanse = new MenuMusic();

return \_instanse;

}

private MenuMusic()

{

mute = false;

if (!MainMusic)

MainMusic = GameObject.Find("MusicMainMenu").GetComponent<AudioSource>();

Debug.Log("MenuMusic constructor worked");

// initialisationMusic();

}

private void initialisationMusic()

{

}

public void play()

{

initialisationMusic();

MainMusic.Play();

}

public void stop()

{

MainMusic.Stop();

}

public void changeMute()

{

if (mute) mute = false;

else if (!mute) mute = true;

}

public void setMute()

{

MainMusic.mute = true;

}

public void unsetMute()

{

MainMusic.mute = false;

}

public bool is\_mute()

{

if (mute) return true;

else return false;

}

public void changeVolume(float level)

{

MainMusic.volume = level;

}

public float getVolumeLevel()

{

return MainMusic.volume;

}

}

public class ControlerInstruction : MonoBehaviour {

// Use this for initialization

void Start ()

{

GameObject.Find("BReturn").GetComponent<Button>().onClick.AddListener(PressReturn);

}

private void PressReturn()

{

SceneManager.LoadScene("Main Menu");

}

}

public class ControlerMainMenu : MonoBehaviour {

void Start ()

{

GameObject.Find("BPlay").GetComponent<Button>().onClick.AddListener(PressPlay);

GameObject.Find("BSetting").GetComponent<Button>().onClick.AddListener(PressSetting);

GameObject.Find("BInstruction").GetComponent<Button>().onClick.AddListener(PressInstructions);

GameObject.Find("BExit").GetComponent<Button>().onClick.AddListener(PressExit);

}

private void PressPlay()

{

ButtonClickSound.GetSounds().play();

SceneManager.LoadScene("VariantGame");

}

private void PressSetting()

{

ButtonClickSound.GetSounds().play();

SceneManager.LoadScene("Settings");

}

private void PressInstructions()

{

ButtonClickSound.GetSounds().play();

SceneManager.LoadScene("Instruction");

}

private void PressExit()

{

ButtonClickSound.GetSounds().play();

Application.Quit();

}

}

public class ControlerMutliplayerGame : MonoBehaviour {

bool chek = true;

public int Index { set; get; }

string name;

string color;

int flafGO = 0;

List<Button> ButtonList;

void Start()

{

initialiseButtonList();

Index = 1;

GameObject.Find("NumberPlayer").GetComponent<Text>().text = System.Convert.ToString(Index);

GameObject.Find("BRight").GetComponent<Button>().onClick.AddListener(indexMore);

GameObject.Find("BLeft").GetComponent<Button>().onClick.AddListener(indexLess);

GameObject.Find("BOK").GetComponent<Button>().onClick.AddListener(PressOk);

GameObject.Find("RedColor").GetComponent<Button>().onClick.AddListener(redColor);

GameObject.Find("YellowColor").GetComponent<Button>().onClick.AddListener(yellowColor);

GameObject.Find("BlueColor").GetComponent<Button>().onClick.AddListener(blueColor);

GameObject.Find("GreenBlueColor").GetComponent<Button>().onClick.AddListener(greenBlueColor);

GameObject.Find("GreenColor").GetComponent<Button>().onClick.AddListener(greenColor);

GameObject.Find("VioletColor").GetComponent<Button>().onClick.AddListener(violetColor);

GameObject.Find("ButtonForError").GetComponent<Button>().onClick.AddListener(PressErrorButton);

GameObject.Find("Button").GetComponent<Button>().onClick.AddListener(Go);

}

private void Update()

{

//Debug.Log("Update");

//Debug.Log("Name " + name);

//Debug.Log("Index " + Index);

// GameObject.Find("NumberPlayer").GetComponent<Text>().text = System.Convert.ToString(Index + 1);

// name = GameObject.Find("NameText").GetComponent<Text>().text;

// Game.getGame()[Index].namePlayer = name;

// Game.getGame()[Index].colorplayer = color;

// Game.getGame()[Index].moneyPlayer = 20000;

}

private void initialiseButtonList()

{

ButtonList = new List<Button>(6);

//initialisation list

ButtonList.Add(GameObject.Find("YellowColor").GetComponent<Button>());

ButtonList.Add(GameObject.Find("BlueColor").GetComponent<Button>());

ButtonList.Add(GameObject.Find("GreenBlueColor").GetComponent<Button>());

ButtonList.Add(GameObject.Find("GreenColor").GetComponent<Button>());

ButtonList.Add(GameObject.Find("RedColor").GetComponent<Button>());

ButtonList.Add(GameObject.Find("VioletColor").GetComponent<Button>());

}

private void InitialiseColors(string name)

{

foreach (Button b in ButtonList)

{

if (b.name == name)

{

Interface.GetIterface().changeScaleOfButton(b, true);

return;

}

}

}

private void redColor()

{

if (!chekSetColor(ButtonList[4].name))

{

Interface.GetIterface().ShowCloseMassageBox(true, "This color was selected");

return;

}

Interface.GetIterface().resetScaleButton(ButtonList);

if (!Interface.GetIterface().chekScaleColorBatton(ButtonList[4]))

{

color = "RedColor";

ColorBlock cb= ButtonList[4].colors;

seachAndChangeScaleOfColorButon("RedColor",true);

}

else

{

color = " ";

seachAndChangeScaleOfColorButon("RedColor", false);

}

}

private void greenColor()

{

if (!chekSetColor(ButtonList[3].name))

{

Interface.GetIterface().ShowCloseMassageBox(true, "This color was selected");

return;

}

Interface.GetIterface().resetScaleButton(ButtonList);

if (!Interface.GetIterface().chekScaleColorBatton(ButtonList[3]))

{

color = "GreenColor";

seachAndChangeScaleOfColorButon("GreenColor", true);

}

else

{

color = " ";

seachAndChangeScaleOfColorButon("GreenColor", false);

}

}

private void greenBlueColor()

{

if (!chekSetColor(ButtonList[2].name))

{

Interface.GetIterface().ShowCloseMassageBox(true, "This color was selected");

return;

}

Interface.GetIterface().resetScaleButton(ButtonList);

if (!Interface.GetIterface().chekScaleColorBatton(ButtonList[2]))

{

color = "GreenBlueColor";

seachAndChangeScaleOfColorButon("GreenBlueColor", true);

}

else

{

color = " ";

seachAndChangeScaleOfColorButon("GreenBlueColor", false);

}

}

private void blueColor()

{

if (!chekSetColor(ButtonList[1].name))

{

Interface.GetIterface().ShowCloseMassageBox(true, "This color was selected");

return;

}

Interface.GetIterface().resetScaleButton(ButtonList);

if (!Interface.GetIterface().chekScaleColorBatton(ButtonList[1]))

{

color = "BlueColor";

seachAndChangeScaleOfColorButon("BlueColor", true);

}

else

{

color = " ";

seachAndChangeScaleOfColorButon("BlueColor", false);

}

}

private void violetColor()

{

if (!chekSetColor(ButtonList[5].name))

{

Interface.GetIterface().ShowCloseMassageBox(true, "This color was selected");

return;

}

Interface.GetIterface().resetScaleButton(ButtonList);

if (!Interface.GetIterface().chekScaleColorBatton(ButtonList[5]))

{

color = "VioletColor";

seachAndChangeScaleOfColorButon("VioletColor", true);

}

else

{

color = " ";

seachAndChangeScaleOfColorButon("VioletColor", false);

}

}

private void yellowColor()

{

if (!chekSetColor(ButtonList[0].name))

{

Interface.GetIterface().ShowCloseMassageBox(true, "This color was selected");

return;

}

Interface.GetIterface().resetScaleButton(ButtonList);

if (!Interface.GetIterface().chekScaleColorBatton(ButtonList[0]))

{

color = "YellowColor";

seachAndChangeScaleOfColorButon("YellowColor", true);

}

else

{

color = " ";

seachAndChangeScaleOfColorButon("YellowColor", false);

}

}

private void indexMore()

{ //if set count players

if (chek)

{

Game.getGame().changeNumberOfPlayersMore();

GameObject.Find("Number").GetComponent<Text>().text = System.Convert.ToString(Game.getGame().numberOfPlayers);

}

else

if (Index < Game.getGame().numberOfPlayers)

{

Index++;

setRotateButton();

Debug.Log("Index++: " + Index);

Interface.GetIterface().resetScaleButton(ButtonList);

GameObject.Find("NumberPlayer").GetComponent<Text>().text = System.Convert.ToString(Index);

InitialiseColors(Game.getGame()[Index - 1].colorplayer);

GameObject.Find("NameText").GetComponent<Text>().text = Game.getGame()[Index - 1].namePlayer;

GameObject.Find("NameField").GetComponent<InputField>().text = Game.getGame()[Index - 1].namePlayer;

}

}

private void indexLess()

{

if (chek)

{

Game.getGame().chamgeNumberOfPlayersLess();

GameObject.Find("Number").GetComponent<Text>().text = System.Convert.ToString(Game.getGame().numberOfPlayers);

}

else

if (Index > 1)

{

Index--;

Debug.Log("Index--: " + Index);

Interface.GetIterface().resetScaleButton(ButtonList);

GameObject.Find("NumberPlayer").GetComponent<Text>().text = System.Convert.ToString(Index);

setRotateButton();

InitialiseColors(Game.getGame()[Index - 1].colorplayer);

GameObject.Find("NameText").GetComponent<Text>().text = Game.getGame()[Index - 1].namePlayer;

GameObject.Find("NameField").GetComponent<InputField>().text = Game.getGame()[Index - 1].namePlayer;

}

}

private void PressOk()

{

if(chek)

{

GameObject.Find("Number").GetComponent<Animations>().enabled = true;

GameObject.Find("NumberPleyers").GetComponent<ChangeText>().Text = "Enter Information";

GameObject.Find("NumberPleyers").GetComponent<ChangeText>().enabled = true;

GameObject.Find("Part2").GetComponent<Animations>().enabled = true;

chek = false;

Game.getGame().setNumberOfPlayer();

Debug.Log(Game.getGame().numberOfPlayers);

}

else

{

name = GameObject.Find("NameText").GetComponent<Text>().text;

Debug.Log("Player " + (Index - 1));

if(!chekSetName(name))

{

Interface.GetIterface().ShowCloseMassageBox(true, "Wrong name");

return;

}

if(color==null)

{

Interface.GetIterface().ShowCloseMassageBox(true, "Please select color");

return;

}

Game.getGame()[Index - 1].namePlayer = name;

Debug.Log("Name " + Game.getGame()[Index - 1].namePlayer);

Game.getGame()[Index - 1].colorplayer = color;

Debug.Log("Color " + Game.getGame()[Index - 1].colorplayer);

// Interface.GetIterface().rotationButton(Game.getGame()[Index - 1].colorplayer, true);

Game.getGame()[Index - 1].moneyPlayer = 200000;

Debug.Log("Money " + Game.getGame()[Index - 1].moneyPlayer);

setRotateButton();

flafGO++;

if (flafGO == Game.getGame().numberOfPlayers)

GameObject.Find("Button").GetComponent<Animations>().enabled = true;

indexMore();

// EditorUtility.DisplayDialog("Hello", "Worls", "ok");

}

}

private void PressErrorButton()

{

Interface.GetIterface().ShowCloseMassageBox(false);

}

/\*This methods for set rotate buttons when colors set or no\*/

private void setRotateButton()

{

foreach (Button b in ButtonList)

{

if (!chekSetColor(b.name))

{

Interface.GetIterface().rotationButton(b.name, true);

}

else

Interface.GetIterface().rotationButton(b.name, false);

}

}

private bool chekSetColor(string name)

{

for(int i=0;i<Game.getGame().numberOfPlayers;i++)

{

if (Game.getGame()[i].colorplayer == name)

return false;

}

return true;

}

private bool chekSetName(string name)

{

if (name == " ")

return false;

for (int i = 0; i < Game.getGame().numberOfPlayers; i++)

{

if (Game.getGame()[i].namePlayer == name)

return false;

}

return true;

}

/\*

\* if flag- true ShowMassageBox

\* if flag-false closeMessageBox

\*/

private void seachAndChangeScaleOfColorButon(string colors, bool flAG)

{

foreach (Button b in ButtonList)

{

if (b.name == colors)

Interface.GetIterface().changeScaleOfButton(b, flAG);

}

}

private void Go()

{

SceneManager.LoadScene("Game");

}

}

public class ControlerSetting : MonoBehaviour {

void Start ()

{

//if (!MenuMusic.GetMainMusic().is\_mute())

//{

// GameObject.Find("BMuteOn").GetComponent<Image>().enabled = false;

// GameObject.Find("BMuteOff").GetComponent<Image>().enabled = true;

//}

//else if (MenuMusic.GetMainMusic().is\_mute())

//{

// GameObject.Find("BMuteOn").GetComponent<Image>().enabled = true;

// GameObject.Find("BMuteOff").GetComponent<Image>().enabled = false;

//}

GameObject.Find("SliderMusic").GetComponent<Slider>().value = MenuMusic.GetMainMusic().getVolumeLevel();

GameObject.Find("SliderSound").GetComponent<Slider>().value = ButtonClickSound.GetSounds().getVolumeLevel();

GameObject.Find("SliderSound").GetComponent<Slider>().onValueChanged.AddListener(delegate { changeVolumeSound(); });

GameObject.Find("SliderMusic").GetComponent<Slider>().onValueChanged.AddListener(delegate { changeVolumeMusic(); });

GameObject.Find("BReturn").GetComponent<Button>().onClick.AddListener(PressReturn);

GameObject.Find("BMuteOn").GetComponent<Button>().onClick.AddListener(PressMute);

GameObject.Find("BMuteOff").GetComponent<Button>().onClick.AddListener(PressMute);

}

private void changeVolumeSound()

{

ButtonClickSound.GetSounds().changeVolume(GameObject.Find("SliderSound").GetComponent<Slider>().value);

}

private void changeVolumeMusic()

{

MenuMusic.GetMainMusic().changeVolume(GameObject.Find("SliderMusic").GetComponent<Slider>().value);

}

private void PressReturn()

{

SceneManager.LoadScene("Main Menu");

}

private void PressMute()

{

//if no mute

if (!MenuMusic.GetMainMusic().is\_mute())

{

Debug.Log("BMuteOff");

//do mute

MenuMusic.GetMainMusic().changeMute();

GameObject.Find("BMuteOn").GetComponent<Image>().enabled = true;

GameObject.Find("BMuteOff").GetComponent<Image>().enabled = false;

MenuMusic.GetMainMusic().setMute();

ButtonClickSound.GetSounds().setMute();

}

else if (MenuMusic.GetMainMusic().is\_mute())

{

Debug.Log("BMuteOn");

MenuMusic.GetMainMusic().changeMute();

GameObject.Find("BMuteOn").GetComponent<Image>().enabled = false;

GameObject.Find("BMuteOff").GetComponent<Image>().enabled = true;

MenuMusic.GetMainMusic().unsetMute();

ButtonClickSound.GetSounds().unsetMute();

}

}

}

public class ControlerVariantOfGame : MonoBehaviour {

void Start ()

{

GameObject.Find("BMultyPlayer").GetComponent<Button>().onClick.AddListener(PressMultiplay);

}

private void PressMultiplay()

{

SceneManager.LoadScene("MultiplayerGameSetting");

}

}

public class MainGameControler : MonoBehaviour

{

private int index = 0;

bool chek = true;

List<BoardElements> boardElements = new List<BoardElements>();

public void Start()

{

// Game.getGame().setNumberOfPlayer();

Interface.GetIterface().addPlayersElementsToTheBoad(GameObject.Find("Board"));

Interface.GetIterface().addInformationAboutPlayer();

initGameobjPlayer();

addEventForInformation();

GameObject.Find("ThrowCubeButton").GetComponent<Button>().onClick.AddListener(Game.getGame().throwCube);

GameObject.Find("MakeMoveButton").GetComponent<Button>().onClick.AddListener(makeMove);

initBoardElements();

initBoardElementsPosition();

initPlayersInFirsCase();

Interface.GetIterface().changeTextPlayer(index);

}

private void initBoardElements()

{

///1SIDE

//1

boardElements.Add(new BoardElements("Start", @"BoardElements\1",boardPayOnly:new BoardPayOnly(200)));

//2

boardElements.Add(new BoardElements("Wreck Room", @"BoardElements\2", boardProterty: new BoardProperty(60)));

//3

boardElements.Add(new BoardElements("Comunity Chest", @"BoardElements\3", boardCard: new BoardCard(CARDTYPE.COMUNITY)));

//4

boardElements.Add(new BoardElements("Boiler Room", @"BoardElements\4", boardProterty: new BoardProperty(60)));

//5

boardElements.Add(new BoardElements("Formation Free", @"BoardElements\5", boardPayOnly: new BoardPayOnly(200)));

//6

boardElements.Add(new BoardElements("Red Van", @"BoardElements\6", boardSingleProperty: new BoardSingleProperty(200)));

//7

boardElements.Add(new BoardElements("Tennis Courts", @"BoardElements\7", boardProterty: new BoardProperty(100)));

//8

boardElements.Add(new BoardElements("Chance", @"BoardElements\8", boardCard: new BoardCard(CARDTYPE.CHANCE)));

//9

boardElements.Add(new BoardElements("Laundry Room", @"BoardElements\9", boardProterty: new BoardProperty(100)));

//10

boardElements.Add(new BoardElements("Weight Room", @"BoardElements\10", boardProterty: new BoardProperty(120)));

//11

boardElements.Add(new BoardElements("Rector's Office", @"BoardElements\11"));

///2SIDE

//12

boardElements.Add(new BoardElements("The Dorm", @"BoardElements\12", boardProterty: new BoardProperty(140)));

//13

boardElements.Add(new BoardElements("Kenrick Light And Magic", @"BoardElements\13", boardProterty: new BoardProperty(150)));

//14

boardElements.Add(new BoardElements("The Library", @"BoardElements\14", boardProterty: new BoardProperty(140)));

//15

boardElements.Add(new BoardElements("The Auditorium", @"BoardElements\15", boardProterty: new BoardProperty(160)));

//16

boardElements.Add(new BoardElements("Silver Van", @"BoardElements\16", boardSingleProperty: new BoardSingleProperty(200)));

//17

boardElements.Add(new BoardElements("The Gym", @"BoardElements\17", boardProterty: new BoardProperty(160)));

//18

boardElements.Add(new BoardElements("Community Chest", @"BoardElements\18", boardCard: new BoardCard(CARDTYPE.COMUNITY)));

//19

boardElements.Add(new BoardElements("The Heights", @"BoardElements\19", boardProterty: new BoardProperty(160)));

//20

boardElements.Add(new BoardElements("The Refectiry", @"BoardElements\20", boardProterty: new BoardProperty(200)));

//21

boardElements.Add(new BoardElements("Free Parking", @"BoardElements\21"));

///3SIDE

//22

boardElements.Add(new BoardElements("The Lobby", @"BoardElements\22", boardProterty: new BoardProperty(220)));

//23

boardElements.Add(new BoardElements("Chance", @"BoardElements\23", boardCard: new BoardCard(CARDTYPE.CHANCE)));

//24

boardElements.Add(new BoardElements("The Courtyatd", @"BoardElements\24", boardProterty: new BoardProperty(220)));

//25

boardElements.Add(new BoardElements("Priest DiningRoom", @"BoardElements\25", boardProterty: new BoardProperty(240)));

//26

boardElements.Add(new BoardElements("White Van", @"BoardElements\26", boardSingleProperty: new BoardSingleProperty(200)));

//27

boardElements.Add(new BoardElements("Convert Chapel", @"BoardElements\27", boardProterty: new BoardProperty(260)));

//28

boardElements.Add(new BoardElements("Mary Mother Of The World Chapel", @"BoardElements\28", boardProterty: new BoardProperty(260)));

//29

boardElements.Add(new BoardElements("Student Compyter Services", @"BoardElements\29", boardPayOnly: new BoardPayOnly(280)));

//30

boardElements.Add(new BoardElements("St.Charles Chapel", @"BoardElements\30", boardProterty: new BoardProperty(280)));

//31

boardElements.Add(new BoardElements("GoToRector's Office", @"BoardElements\31"));

///4SIDE

//32

boardElements.Add(new BoardElements("Glennon Lounge", @"BoardElements\32", boardProterty: new BoardProperty(300)));

//33

boardElements.Add(new BoardElements("Priest's Louge", @"BoardElements\33", boardProterty: new BoardProperty(300)));

//34

boardElements.Add(new BoardElements("Community Chest", @"BoardElements\34", boardCard: new BoardCard(CARDTYPE.COMUNITY)));

//35

boardElements.Add(new BoardElements("Kenrick Lounge", @"BoardElements\35", boardProterty: new BoardProperty(320)));

//36

boardElements.Add(new BoardElements("Sister's Car", @"BoardElements\36", boardSingleProperty: new BoardSingleProperty(200)));

//37

boardElements.Add(new BoardElements("Chance", @"BoardElements\37", boardCard: new BoardCard(CARDTYPE.CHANCE)));

//38

boardElements.Add(new BoardElements("St.Loseph", @"BoardElements\38", boardProterty: new BoardProperty(350)));

//39

boardElements.Add(new BoardElements("Room And Board", @"BoardElements\39", boardPayOnly: new BoardPayOnly(75)));

//40

boardElements.Add(new BoardElements("The Towel", @"BoardElements\40", boardProterty: new BoardProperty(400)));

}

private void initBoardElementsPosition()

{

int index = 0;

//init first Board element

boardElements[index].setAnchors(1, GameObject.Find("Board").GetComponent<RectTransform>().anchorMin.y + 0.13f,

GameObject.Find("Board").GetComponent<RectTransform>().anchorMax.x + 0.09f, 0);

index++;

//init 1 side little elements

for (int i = 0; i < 9; i++)

{

boardElements[index].setAnchors(

boardElements[index - 1].Positions.MinX,

boardElements[index - 1].Positions.MaxY,

boardElements[index-1].Positions.MinX-0.08f,

0);

index++;

}

//last element on 1 side

boardElements[index].setAnchors(

boardElements[index - 1].Positions.MinX,

boardElements[index - 1].Positions.MaxY,

0,

0);

index++;

//init 2 side little elemets

for (int i = 0; i < 9; i++)

{

boardElements[index].setAnchors(

boardElements[index - 1].Positions.MaxX,

boardElements[index-1].Positions.MaxY+0.08f,

boardElements[index - 1].Positions.MinX,

boardElements[index - 1].Positions.MaxY);

index++;

}

//init last element on 2 side

boardElements[index].setAnchors(boardElements[index - 1].Positions.MaxX, 1,

boardElements[index - 1].Positions.MinX, boardElements[index - 1].Positions.MaxY);

index++;

//intt 3 side lettle elements

for (int i = 0; i < 9; i++)

{

boardElements[index].setAnchors(

boardElements[index-1].Positions.MaxX+0.08f,

1,

boardElements[index - 1].Positions.MaxX,

boardElements[index - 1].Positions.MaxY-0.13f);

index++;

}

//init last element on 3 side

boardElements[index].setAnchors(1, 1,

boardElements[index - 1].Positions.MaxX, boardElements[index - 1].Positions.MaxY - 0.13f);

index++;

//init 4 side litle elements

for (int i = 0; i < 9; i++)

{

boardElements[index].setAnchors(1, boardElements[index - 1].Positions.MinY,

boardElements[index - 1].Positions.MinX, boardElements[index - 1].Positions.MinY - 0.08f);

index++;

}

Destroy(GameObject.Find("NULL"));

}

private void addEventForInformation()

{

for (int i = 0; i < Game.getGame().numberOfPlayers; i++)

{

Player p = Game.getGame()[i];

GameObject.Find("InformationAboutPlayer" + (i + 1)).GetComponent<Button>().onClick.AddListener(()=> Interface.GetIterface().showPanelInformation(p));

}

}

private void makeMove()

{

if (!Game.getGame().isGameOver())

{

//delete players from the current position

boardElements[Game.getGame()[index].RealTimePosition].playersInTheCase.Remove(Game.getGame()[index].LogoPlayer);

int local = Game.getGame()[index].makeMove(Game.getGame().Step);

boardElements[local].playersInTheCase.Add(Game.getGame()[index].LogoPlayer);

boardElements[local].setPositionsInTheCase();

//BOARD PROPERTY

if (boardElements[local].boardProterty != null)

{

Interface.GetIterface().createPanelForProperty(boardElements[local]);

if (boardElements[local].boardProterty.isOwner == false)

{

GameObject.Find("BoardElementBuyButton").GetComponent<Button>().onClick.AddListener(Buy);

GameObject.Find("BoardElementSkipButton").GetComponent<Button>().onClick.AddListener(skip);

}

else

{

if (Game.getGame()[index].namePlayer == boardElements[local].boardProterty.OwnerName)

{

GameObject.Find("BoardElementsPayorUpdateButtonText").GetComponent<Text>().text = "Update";

GameObject.Find("BoardElementPayorUpdateButton").GetComponent<Button>().onClick.AddListener(Upd);

GameObject.Find("BoardElementSellButton").GetComponent<Button>().enabled = true;

GameObject.Find("BoardElementSellButton").GetComponent<Button>().onClick.AddListener(sell);

GameObject.Find("BoardElementSellButtonText").GetComponent<Text>().enabled = true;

}

else

{

GameObject.Find("BoardElementsPayorUpdateButtonText").GetComponent<Text>().text = "Pay rent";

GameObject.Find("BoardElementPayorUpdateButton").GetComponent<Button>().onClick.AddListener(payRent);

}

}

}

//BOARD PAYONLY

else if (boardElements[local].boardPayOnly != null)

{

Interface.GetIterface().createPanelForPayOnly(boardElements[local]);

GameObject.Find("BoardElementPayButton").GetComponent<Button>().onClick.AddListener(PayOnly);

}

//BOARD CARD

else if (boardElements[local].boardCard != null)

{

Interface.GetIterface().createPanelCard(boardElements[local]);

chek = true;

GameObject.Find("newDoButton").GetComponent<Button>().onClick.AddListener(doActions);

// GameObject.Find("BoardElementDoButton").GetComponent<Button>().onClick.AddListener(delegate { Interface.GetIterface().showRandCard(boardElements[local]);});

// doAction();

}

//BOARD SINGLE PROPERTY

else if (boardElements[local].boardSingleProperty != null)

{

Interface.GetIterface().createPanelForSingleProperty(boardElements[local]);

if (boardElements[local].boardSingleProperty.isOwner == false)

{

GameObject.Find("BoardElementBuyButton").GetComponent<Button>().onClick.AddListener(Buy);

GameObject.Find("BoardElementSkipButton").GetComponent<Button>().onClick.AddListener(skip);

}

else

{

if (Game.getGame()[index].namePlayer == boardElements[local].boardSingleProperty.OwnerName)

{

GameObject.Find("BoardElementsBuyButtonText").GetComponent<Text>().text = "Skip";

GameObject.Find("BoardElementBuyButton").GetComponent<Button>().onClick.AddListener(skip);

GameObject.Find("BoardElementSellButton").GetComponent<Button>().enabled = true;

GameObject.Find("BoardElementSellButton").GetComponent<Button>().onClick.AddListener(sell);

GameObject.Find("BoardElementSellButtonText").GetComponent<Text>().enabled = true;

}

else

{

GameObject.Find("BoardElementsBuyButtonText").GetComponent<Text>().text = "Pay rent";

GameObject.Find("BoardElementBuyButton").GetComponent<Button>().onClick.AddListener(payRent);

}

}

}

else

{

Interface.GetIterface().showInformationAboutBoardElemets(boardElements[local]);

skip();

}

Game.getGame().chekBanclotsPlayer();

}

}

private void checkIndex()

{

if (index >= Game.getGame().numberOfPlayers) index = 0;

return;

}

private void initGameobjPlayer()

{

for (int i = 0; i < Game.getGame().numberOfPlayers; i++)

{

Game.getGame()[i].LogoPlayer = GameObject.Find("Player" + (i + 1));

}

}

private void initPlayersInFirsCase()

{

for (int i = 0; i <Game.getGame().numberOfPlayers ; i++)

{

boardElements[0].playersInTheCase.Add(Game.getGame()[i].LogoPlayer);

}

boardElements[0].setPositionsInTheCase();

}

private void Buy()

{

if(!Game.getGame()[index].buy(boardElements[Game.getGame()[index].RealTimePosition]))

{

//обработка

}

index++;

checkIndex();

Interface.GetIterface().closeInformationAboutBoardElements();

Interface.GetIterface().changeTextPlayer(index);

Game.getGame().chekTrow = true;

}

private void PayOnly()

{

if (!Game.getGame()[index].payOnly(boardElements[Game.getGame()[index].RealTimePosition]))

{

//обработка

}

index++;

checkIndex();

Interface.GetIterface().closeInformationAboutBoardElements();

Interface.GetIterface().changeTextPlayer(index);

Game.getGame().chekTrow = true;

}

private void Upd()

{

if (!Game.getGame()[index].updateLevel(boardElements[Game.getGame()[index ].RealTimePosition]))

{

//обработка

}

index++;

checkIndex();

Interface.GetIterface().closeInformationAboutBoardElements();

Interface.GetIterface().changeTextPlayer(index);

Game.getGame().chekTrow = true;

}

private void payRent()

{

if (!Game.getGame()[index].payRent(boardElements[Game.getGame()[index].RealTimePosition]))

{

//обработка

}

index++;

checkIndex();

Interface.GetIterface().closeInformationAboutBoardElements();

Interface.GetIterface().changeTextPlayer(index);

Game.getGame().chekTrow = true;

}

private void skip()

{

index++;

checkIndex();

Interface.GetIterface().closeInformationAboutBoardElements();

Interface.GetIterface().changeTextPlayer(index);

Game.getGame().chekTrow = true;

}

private void doActions()

{

if (chek == true)

{

int ID = System.Convert.ToInt32(GameObject.Find("BoardElementsIDCardText").GetComponent<Text>().text);

int oldPos = Game.getGame()[index].RealTimePosition;

Game.getGame()[index].doAction(ID, boardElements[Game.getGame()[index].RealTimePosition]);

if (ID == 1)

{

boardElements[oldPos].playersInTheCase.Remove(Game.getGame()[index].LogoPlayer);

boardElements[0].playersInTheCase.Add(Game.getGame()[index].LogoPlayer);

Game.getGame()[index].RealTimePosition = 0;

boardElements[0].setPositionsInTheCase();

}

else if (ID == 7)

{

boardElements[oldPos].playersInTheCase.Remove(Game.getGame()[index].LogoPlayer);

boardElements[10].playersInTheCase.Add(Game.getGame()[index].LogoPlayer);

//fix bug

Game.getGame()[index].RealTimePosition = 10;

boardElements[10].setPositionsInTheCase();

}

else if (ID == 6)

{

index--;

}

Interface.GetIterface().closeInformationAboutBoardElements();

Interface.GetIterface().returnAnchors();

GameObject.Find("newDoButton").GetComponent<Button>().enabled = false;

GameObject.Find("newDoButtonText").GetComponent<Text>().enabled = false;

MonoBehaviour.Destroy(GameObject.Find("BoardElementsIDCardText"));

index++;

checkIndex();

Interface.GetIterface().changeTextPlayer(index);

Game.getGame().chekTrow = true;

chek = false;

}

}

private void sell()

{

Game.getGame()[index].sell(boardElements[Game.getGame()[index].RealTimePosition]);

index++;

checkIndex();

Interface.GetIterface().changeTextPlayer(index);

Interface.GetIterface().closeInformationAboutBoardElements();

Game.getGame().chekTrow = true;

}

}

# Тестирование программы черным и белым ящиком

Тестирование черного ящика – это– тестирование как функциональное, так и нефункциональное, не предполагающее знания внутреннего устройства компонента или системы.

Во время тестирования программы методом черного ящика были обнаружены такие ошибки:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Проблема | Причина проблемы | Решение |
| 1 | Возможность выбрать один и тот же цвет для всех персонажей | Не было проверки на уже выбранный цвет | Добавления проверки на уже выбранный цвет |
| 2 | Неправильное поведение игроков при попадание на поле «Дополнительная карточка» | Некоторые карточки требовали дополнительной визуальной обработки действия которой не было | Добавление дополнительной обработки при выпадании некоторых карточек |
| 3 | Неправильное отображение игроков в своей ячейке | Неправильный алгоритм растановки игроков в ячейке | Изменения илгоритма расстановки игроков в ячейке |
| 4 | Вылет при прохождении круга | При прохождении круга не обнулялся индекс элементов | Обнуление индекса элементов |

Тестирование белого ящика – это тестирование, основанное на анализе внутренней структуры компонента или системы.

Во время тестирования программы белым ящиком были использованы такие методы:

* Покрытие операторов. Каждый оператор, который прописанный в коде выполняется, по крайней мере, один раз.
* Покрытие решений. Было проведено некое количество тестов для проверки каждого условия на выполнение.

Все ошибки, которые найдены во время тестирования, были исправлены и устранены.

# Пример работы программы

При запуске приложения появляется окно с меню.Меню содержит 4 пункта «Игра», «Настройки», «Инструкция», «Выход». Представленно на рисунке 1



Рисунок 1-главное меню

Дальше при нажатии на пункт «Настройки» можно увидеть настройки звука и музыки в программе. Представленно на рисунке 2

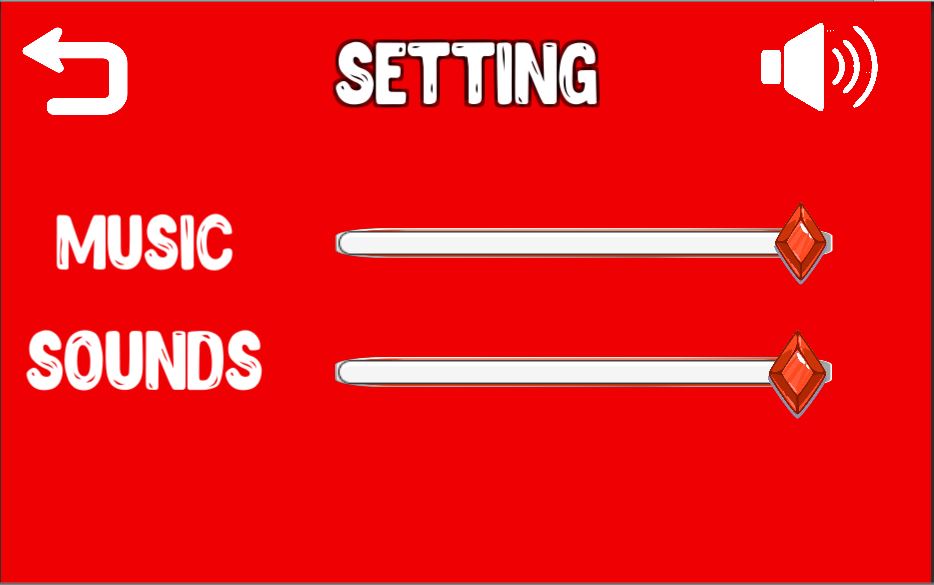


Рисунок 2 –«Настройки»

При выборе пункта Интсрукции можно увидеть текст и инструкцию работы с программой. Показанно на рисунке 3



Рисунко 3-«Инструкция»

При выборе пункта играть можноивидеть выбор типа игры. На данном этапе реализованна только “MultyPlayer”. Показано на риснке 4



Рисунок 4- выбор типа игры

Дальше при выборе пункта “Multyplayer” мы можем увидеть меню с выбором количества игроков, это можно увидеть на рисунке 5.

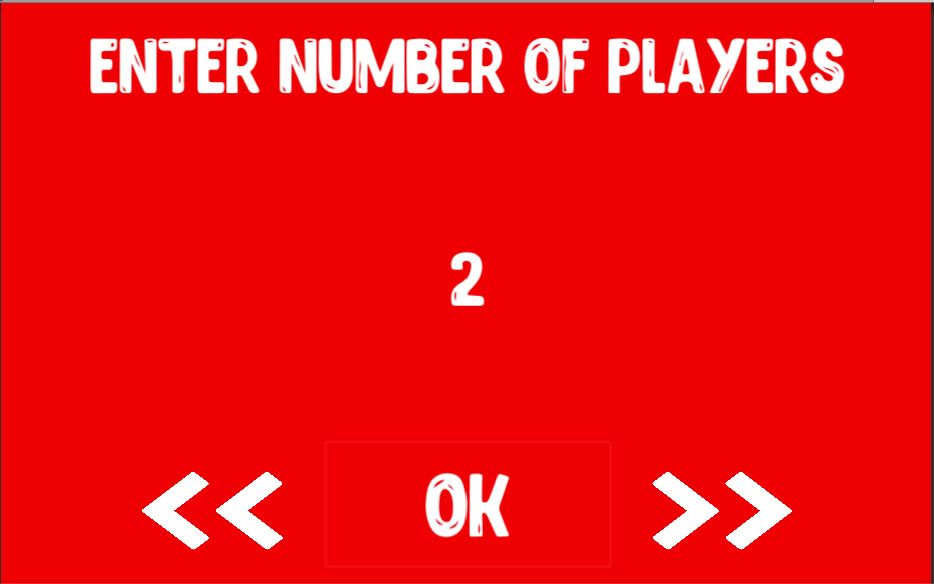


Рисунок 5- выбор количества игроков

После того как выбрали нужное количество игроков, нужно задать их цвет, и ввести имя что позволяет следуйщий пунки, который предсавлен на рисунке 6

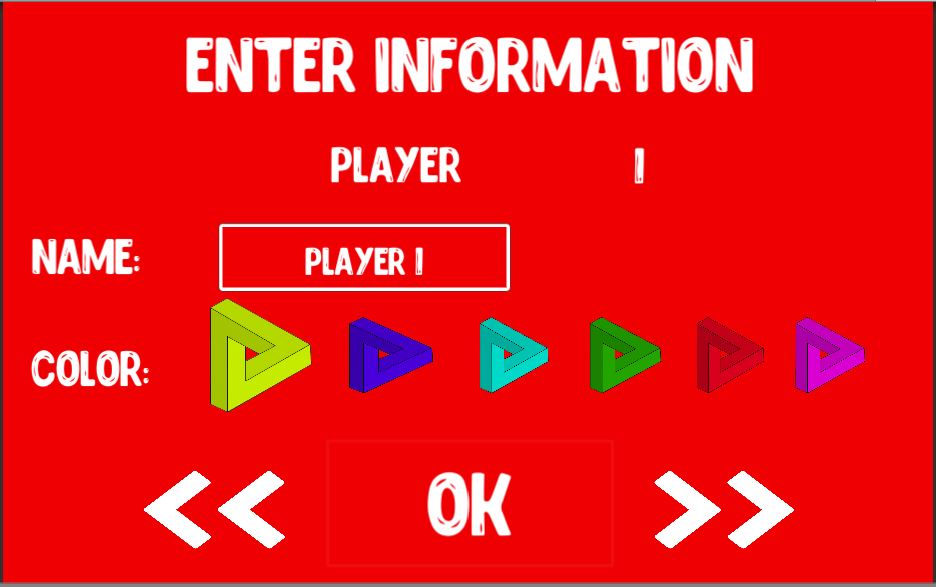


Рисунок 6- настройки данных о персонаже

Дальше пользователь попадает на карту где и будет происходить вся игра. Показано на рисунке 7



Рисункок 7 -главное игрокое поле

При нажатии на имя игрока справа можно увидеть информацию о игроке. Показано на рисунке 8



Рисунок 8-информация о игроке

При соверщении хода и попадении на любое поле можно увидеть всплывающее окно с действиями, на рисунке 9 можно увидеть окно при попадении на «Дополнительная Карта»



Рисунок 9-всплывающее окно при попадании на «Дополнительная карта»

# Анализ результатов и выводы

В результате изучения ООП, было разработано приложение, реализующее игру «Морской Бой». Созданный продукт полностью выполняет все запланированные функциональные требования, такие как:

* различные уровни сложности игры;
* автоматическая расстановка кораблей;
* расстановка кораблей вручную;
* визуализация интерфейса, использование графических элементов, анимации;
* добавление музыки, звуков;
* корректность работы программы.

В ходе выполнения работы были получены дополнительные навыки в программировании на языке C++, использовании графической библиотеки SFML , а также навыки проектирования программного проекта, моделирования требований, проведения тестирования и составления документации.