МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Новосибирский национальный исследовательский государственный университет»
(Новосибирский государственный университет, НГУ)
Структурное подразделение Новосибирского государственного университета – Высший колледж информатики Университета (ВКИ НГУ)
КАФЕДРА ИНФОРМАТИКИ

ОТЧЕТ КУРСОВОГО ПРОЕКТА

Основы алгоритмизации и программирования

РАЗРАБОТКА МОБИЛЬНОЙ ИГРЫ ДЛЯ ОС ANDROID В ЖАНРЕ «ТРИ В РЯД» НА ПЛАТФОРМЕ UNITY

Преподаватель доцент ВКИ НГУ	Голкова Н.В. «_14_»__июня__2022 г.
Студент 1 курса гр. 107 сб2	Санникова А.С. «_14_»__июня__2022 г.

Новосибирск

2022

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ

Мобильная игра – игровая программа для мобильных устройств.

Двумерное пространство (2D) – геометрическая модель плоской проекции физического мира.

UML – язык графического описания для объектного моделирования в области разработки программного обеспечения.

Спрайт – графический объект в компьютерной графике.

Скрипт – последовательность действий, описанных с помощью скриптового языка программирования для автоматического выполнения определенных задач.

Пользовательский интерфейс (UI) – интерфейс, обеспечивающий передачу информации между пользователем и программно-аппаратным компонентами компьютерной системы.

RectTransform – прямоугольник, внутри которого может быть размещен элемент. Содержит информацию о положении, размере, привязке и повороте относительно родительского прямоугольника.

Raycast – выпускаемый луч вдоль программного луча из точки испускания до встречи коллайдера на сцене.

Prefab – это особый тип ассетов, позволяющий хранить игровые объекты со всеми компонентами и значениями свойств. Выступает в роли шаблона для создания экземпляров хранимого объекта в сцене.

PlayerPrefs – функция, которая способна сохранить в реестр переменные стокового и числового типа.

UI.Slider – плагин, позволяющий создать элемент интерфейса в виде ползунка.

UI.Toggle – стандартный переключатель, который имеет состояние включить/выключить.

ВВЕДЕНИЕ

На фоне технического прогресса, улучшения качества и доступности мобильных устройств наблюдается стремительный рост популярности мобильных игр во всем мире.

На сегодняшний момент выручка за I квартал 2022 г. составляет 22 млрд долларов, что на 42 % больше, чем два года назад. А число загрузок мобильных игр в сравнении с 2019 г. увеличилось на 45% и достигло 14 млрд. Именно на игры приходится больше половины всей прибыли App Store — 65% и Google Play — 73%. [1].

Такие темпы роста вовлеченности пользователей не удивляют, ведь качественная графика и игровой процесс на мобильных устройствах ничем не уступают играм на консоли. Нельзя не отметить и другие доступные возможности такие, как кроссплатформенные, соревновательные и социальные игровые функции.

Что касается жанров мобильных игр, их огромное количество и разнообразие оставляет пользователю широкий выбор игровых приложений на любой потребительский вкус. С ростом популярности мобильных игр многие игровые механики, которые, казалось бы, безнадежно устарели для игр на ПК и консолях, «воскресли» и вновь обрели былую популярность. Особенно это коснулось различных головоломок, которые в ПК-гейминге уже давно были вытеснены в формат мини-игр для более серьезных проектов или же изживали себя в виде казуальных и браузерных игр. Что удивительно, именно такой вид мобильных игр наиболее популярны во всем мире. По результатам исследований больше половины пользователей смартфонов имеют на своих устройствах хотя бы одну головоломку. Такая тенденция не обошла стороной и российский рынок мобильных игр: результаты опроса американской компании Google и исследовательской компании Savanta показали, что российский рейтинг топ-3 составляют игры «три в ряд», игры с картами и казино.

Целью курсовой работы является разработка 2D игры-головоломки для ОС Android на платформе Unity.

Для достижения поставленной цели необходимо решить следующие задачи:

- произвести анализ предметной области;
- провести обзор программных средств разработки игр для мобильных ОС;
- изучить существующие аналоги игровых приложений на рынке, произвести их сравнения и выделить особенности разрабатываемой игры;
- спроектировать и разработать архитектуру мобильного приложения;
- реализация игрового приложения;
- тестирование разработанного игрового приложения.

Способ решения данных задач зависит от выбора рабочей среды и языка программирования. Для разработки программного средства будут использоваться следующие технологии: межплатформенная среда разработки Unity и язык программирования C#.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Одним из самых популярных игровых направлений на любых устройствах являются казуальные игры, которые предназначены для широкого круга пользователей. Казуальные игры отличаются простыми правилами и не требуют от пользователя затрат времени на обучение или каких-либо особых навыков [2]. Обычно суть игры лежит на поверхности и понятна без каких-либо инструкций.

Головоломки считаются поджанром казуальных игр, охватывают широкую аудиторию, держат прочные позиции и приносят наибольшее количество доходов на рынке мобильных игр. В центре внимания игры оказывается необходимость решения различных логических задач. Цель головоломок понятна, но при этом достичь ее не так легко, приходится задействовать логику и смекалку, чтобы победить или набрать наибольшее количество очков. Данный поджанр казуальных игр подходит для людей из разных возрастных и социальных групп.

Три в ряд — тип игровой механики казуальных головоломок, получивший свою популярность за счет своей простоты и гибкости. Цель игры данного типа заключается в передвижении игрового элемента для составления цепочки или линии из трех и более одинаковых элементов. Игровые элементы располагаются в пространстве, что чаще всего представляет собой квадратное поле. Для победы игрок должен удовлетворить неким условиям, например — набрать нужное количество очков, поместить определённые элементы в нужное положение, собрать тройки над выделенными ячейками поля и т.д.

Способы построения цепочек из игровых элементов в своём большинстве делятся на три категории:

- Перемена мест. Игрок выделяет две фишки на игровом поле, которые меняются местами друг с другом. Как правило, перемена возможна лишь в том случае, когда хотя бы одна из перемещённых фишек войдёт в состав новообразованной цепочки.

- Вращение. Игрок прокручивает строку или столбец игрового поля таким образом, чтобы получить цепочку или линейку из игровых элементов.
- Выделение. Игрок находит уже готовые группы или цепочки на игровом поле и выделяет один из элементов, после чего выбранные фишки исчезают.

Для того, чтобы усложнить процесс составления линий из одинаковых элементов, в игре создаются препятствия. Самой распространённой преградой является изменение формата игрового поля, таким образом, в определённых точках весьма проблематично построить цепочку. Другой вид преграды — зафиксированные игровые элементы. Чаще всего это замороженная или оплетённая цепями фишка. Такой элемент нельзя сдвинуть с места, однако в большинстве случаев он может участвовать в построении цепочек и групп, после чего либо исчезает, либо превращается в рядовую фишку.

В игре также может присутствовать система бонусов, которой игрок может воспользоваться в какой-то определенный момент, чтобы уничтожить один или несколько элементов, разрушить приятстве и т.д. Такие бонусы существенно упрощают достижение цели игры. Воспользоваться бонусом можно набрав нужное количество игровых баллов, найдя соответствующий символ на поле или при каком-либо другом выполненном условии [3].

Во время проектирования игр жанра «три в ряд» обращают внимание на следующие особенности:

- Короткие игровые сессии. Большинство казуальных игр характеризуются короткими игровые сессиями, и при этом в них можно как быстро начинать играть, так и без осложнений прерывать игру для того, чтобы потом в неё вернуться. Игроки могут играть в игру долго, но игровой процесс строится таким образом, чтобы позволять себя легко прерывать.
- Автоматическое сохранение. Казуальные, игры сами сохраняют состояние игрового мира даже если игрок сам закрывает приложение.
- Очень простые правила.

- Умеренная степень инновации. Данная особенность диктуется целевой аудиторией, которая уже знакома с казуальными играми жанра «три в ряд» и ищут что-то похожее, и при этом не хотят осваивать слишком сложные новые элементы.
- Если локальная задача решается более сложным способом, то игрок вознаграждается. Например, если игрок делает комбо или за раз сопоставляет большее количество элементов, чем требуется.
- Множество наград. У игрока есть возможность достичь успеха в самом начале.
- Небольшие наказания игроков. Игры жанра «три в ряд» стараются избегать наказывать игроков если они допускают ошибки.

Помимо этого, такие простые концепты могут комбинироваться и разбиваться на множество уровней, позволяя добавить историю в игру и поддерживать разнообразие графикой[4].

Таким образом, предметом разработки является мобильное 2D игровое приложение в жанре «Три в ряд» на движке Unity.

2 ПОСТАНОВКА ЗАДАЧИ

Цель работы – изучение игрового движка Unity, языка программирования C# и библиотеки C# для Unity. А именно, разработать и реализовать игровую механику мобильного приложения в жанре «Три в ряд», позволяющий игроку задействовать логику и смекалку для достижения цели игры, на платформе Unity, которая поддерживает скрипты на языке C#. Игровое приложение должно отвечать поставленным функциональным требованиям.

Наименование игры – «Сказочный лес» («Fairy Forest»).

Игра является однопользовательским мобильным игровым приложением на платформе Android.

Цель разрабатываемой игры – набрать наибольшее число очков за ограниченное количество ходов.

Правила игры должны быть просты и понятны, приложение не должно вызывать затруднения у пользователя. Мобильное приложение подходит для людей любой возрастной и социальной категории.

Управление в игре реализовывается посредством касания экрана.

С помощью перемены места выбранного игрового элемента с соседними будет строиться линия из одинаковых элементов. Обмен возможен только в том случае, если один из элементов войдет в состав цепочки. Каждый раз, когда игрок выстраивает линию из трех и более одинаковых элементов, игровые фишки, входящие в цепочку, будут сгорать, а общий счет будет увеличиваться на 100 очков за каждую сгоревшую фишку.

Как только игрок истратит все свои ходы, игра закончится.

С каждым новым уровнем игра будет усложняться с помощью изменения конфигурации игрового поля, чтобы игроку было сложнее выстроить цепочку и набрать необходимые очки.

Система бонусов в игре будет отсутствовать.

3 АНАЛОГИ

В ходе анализа предметной области, были изучены популярные и схожие по тематике мобильные игровые приложения в жанре «Три в ряд». Рассмотрим некоторые из них:

1. Candy Crush Saga – бесплатная игра-головоломка, выпущенная компанией King. На рисунке 1 представлен скриншот игровой панели мобильного приложения.

Рисунок 1 – Скриншот игрового поля игры «Candy Crush Saga»

Достоинства игры:

- Простые и четкие инструкции.
- Проработанный пользовательский интерфейс и графика.
- Наличие системы бонусов в игре.
- Наличие преград при прохождении уровней.

Недостатки:

- В игре встречаются сбои, из-за которых уровень невозможно пройти.

2. Town Harvest: Match 3 – бесплатная игра-головоломка, выпущенная компанией Higgs Studio. На рисунке 2 представлен скриншот игровой панели мобильного приложения.

Рисунок 2 – Скриншот игрового поля «Town Harvest: Match 3»

Достоинства игры:

- Приятный дизайн для восприятия пользователя.
- Простые инструкции и четко прописанные цели.
- Игра усложняется препятствиями.
- Присутствует система бонусов.

Недостатки:

- Нет возможности переиграть пройденные уровни.
- Неудобный пользовательский интерфейс.

Выполним сравнение представленных выше мобильных приложений. Данные игровые приложения будут оцениваться по трех бальной шкале, где: 3-отлично, 2-нормально, 1- плохо. Эта шкала позволит продемонстрировать, в каких областях тот или иной сайт выигрывает у других сайтов, или наоборот уступает им.

Критерии для оценки:

- Проработанная графика и дизайн.
- Удобство взаимодействия с пользовательским интерфейсом.
- Изменение сложности в ходе прохождения игры.
- Разнообразие геймплея и наличие системы бонусов.
- Функциональность.

В таблице 1 приведены оценки по вышеперечисленным критериям и произведено сравнение мобильных приложений.

Таблица 1 – Сравнение аналогов

Критерии	Candy Crash Saga	Town Harvest: Match 3	Разрабатываемое мобильное приложение «Fairy Forest»
Графика и дизайн.	3/3	3/3	3/3
Пользовательский интерфейс	3/3	1/3	3/3
Сложность	3/3	3/3	2/3
Геймплей	3/3	3/3	2/3
Функциональность	1/3	3/3	3/3

Проанализировав данные таблицы 1, можно сделать вывод, что, представленные игры очень популярны. С другой стороны, имеются проблемы с функциональностью и пользовательским интерфейсом.

Выполненный сравнительный анализ приложений-аналогов и разрабатываемого программного средства позволил выявить достоинства и недостатки игр-аналогов, а также оценить конкурентное преимущество разрабатываемого игрового приложения.

4 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К ПРОГРАММНОМУ ПРОДУКТУ

4.1 Классы пользователей

Разрабатываемое игровое приложение должно быть простым в использовании и доступен широкому кругу пользователей. В качестве пользователей можно выделить только один класс пользователя:

1. Игрок – неавторизованный пользователь, обладает правами:

- Просмотр игровых сцен.
- Просмотр карты уровней.
- Запуск выбранного уровня.
- Изменения параметров звука (уменьшить или увеличить громкость, отключить звук или мелодию).
- Возможность прервать игру в любой момент.
- Просмотреть результаты.
- Выйти из приложения.

Исходя из данных характеристик была разработана UML-диаграмма прецедентов для класса пользователя – игрок. На рисунке 3 представлена диаграмма вариантов использования игроком игрового приложения.

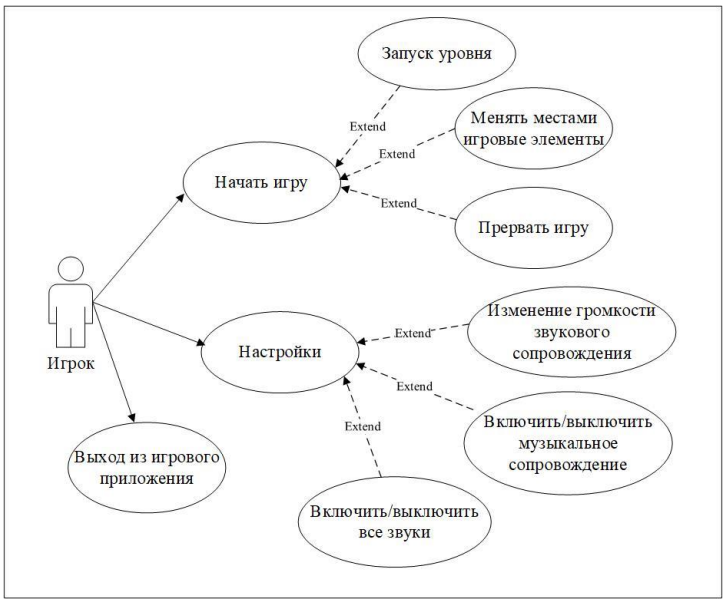

Рисунок 3 – Диаграмма вариантов использования

4.2 Требования к представлению игры

Мобильное приложение будет состоять из четырех сцен.

Язык локализации приложения – английский.

Первая сцена является главным меню игры и должна содержать навигационные кнопки и название игры «Fairy Forest».

Графическая оболочка для главного меню должна делиться на следующие разделы:

- Графический заголовок с названием игры.
- Навигационные кнопки, обеспечивающие переход к пунктам меню.
- Поле для отображения панели содержимого выбранной кнопки.

Структура главного меню должна иметь следующий вид:

1. «Start» - кнопка перехода к выбору уровня и началу игры. Уровни с четвертого по пятнадцатый в стадии разработки.

- Уровень 1 – кнопка перехода на вторую сцену.
- Уровень 2 – кнопка перехода на третью сцену.
- Уровень 3 – кнопка перехода на четвертую сцену.
- Уровень 4.
- Уровень 5.
- Уровень 6.
- Уровень 7.
- Уровень 8.
- Уровень 9.
- Уровень 10.
- Уровень 11.
- Уровень 12.
- Уровень 13.
- Уровень 14.
- Уровень 15.

2. «Settings» - кнопка перехода в панель настроек.

- «Sound volume» - ползунок громкости звукового сопровождения.
- «Melody on/off» - кнопка включения и выключения музыкального сопровождения.
- «Sound on/off» - кнопка включения и выключения всего звукового сопровождения.

3. «Quit» - кнопка выхода из игрового приложения.

Сцены со второй и далее являются игровыми и должны содержать игровое поле, где непосредственно происходит сама игра.

Графическая оболочка для всех игровых сцен должна делиться на следующие разделы:

- Кнопка выхода в главное меню.
- Поле для отображения игрового поля.
- Поле для отображения панели счетчика ходов.
- Поле для отображения панели счетчика очков.
- Поле для отображения результата игровой сессии.
- Поле для отображения системы звезд:
  - загорается одна звезда, если игрок набрал более 5000 очков;
  - загораются две звезды, если игрок набрал более 7500 очков;
  - загораются три звезды, если игрок набрал более 12000 очков.

Структура игровых сцен имеет следующий вид:

1. Кнопка выхода в главное меню.

- «Yes» - кнопка перехода на сцену главного меню.
- «No» - кнопка, возвращающая к игровому процессу.

2. Панель результатов

- «Menu» - кнопка перехода на сцену главного меню.
- «Restart» - кнопка перезапуска текущей игровой сцены.
- «Next» - кнопка перехода на сцену следующего уровня.

5 НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К ПРОГРАММНОМУ ПРОДУКТУ

5.1 Требования к программному обеспечению

1. Приложение должно быть написано на языке C# под платформу Android с помощью платформы Unity.
2. Приложение должно функционировать на операционной системе Android с версией не ниже 4.4.
3. Разрешение экрана должно быть не меньше 1920x1080.
4. Изображение должно адаптироваться под все виды разрешений экрана.
5. При запуске игры должна быть автоматическая горизонтальная ориентация.
6. Корректное отображение всего содержимого на экране (объектов и текста), чтобы у пользователя не возникали трудности и раздражение при просмотре.

5.2 Требования к аппаратному обеспечению

1. Любая модель смартфона, соответствующая требованиям к ПО.
2. Оперативная память 2 GB и более.
3. Процессор с частотой 2.2 ГГЦ и более

5.3 Требования к надёжности

Уязвимость приложения через так называемые «лаги», а также уязвимость вирусами должны быть сведены к нулю. В противном случае, приложение не будет в состоянии выполнять возложенные на него функции и придет в негодность. Приложение не должно требовать подключения к сети Интернет и всегда быть доступным пользователю.

6 ХАРАКТЕРИСТИКА ВЫБРАННЫХ ПРОГРАММНЫХ СРЕД И СРЕДСТВ

В качестве основных критериев при выборе языков программирования и среды разработки были выбраны: простота реализации и удобство. Исходя из этого, была выбрана платформа Unity, а также язык программирования C#.

6.1 Unity

Unity — межплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие [5].

Особенность этого редактора в том, что всё это можно делать и настраивать прямо во время запуска или тестирования сюжета.

Преимущества Unity:

- Наличие визуальной среды разработки. Этот фактор включает не только инструментарий визуального моделирования, но и интегрированную среду, цепочку сборки, что направлено на повышение производительности разработчиков, в частности, этапов создания прототипов и тестирования
- Межплатформенная поддержка. Под межплатформенной поддержкой предоставляется не только места развёртывания (установка на персональном компьютере, на мобильном устройстве, консоли и т.д.), но и наличие инструментария разработки (интегрированная среда может использоваться под Windows и Mac OS).
- Модульная система компонентов Unity, с помощью которой происходит конструирование игровых объектов, когда последние представляют собой комбинируемые пакеты функциональных элементов. В отличие от механизмов наследования, объекты в Unity создаются посредством объединения функциональных блоков, а не помещения в узлы дерева

наследования. Такой подход облегчает создание прототипов, что актуально при разработке игр.

- Доступность. Начать разработку и выпускать свои первые проекты можно бесплатно.
- Обучение. Для новичков создали подробные обучающие материалы в разделе Learn, где объясняется, как создать проект, разместить персонажа, создать для него окружение, научить взаимодействовать с предметами, создавать разные уровни сложности и в итоге собрать первый проект. Такое доступное и бесплатное обучение — важная особенность Unity.
- Большое комьюнити.

Недостатки:

- Ограничение визуального редактора при работе с многокомпонентными схемами, когда в сложных сценах визуальная работа затрудняется.
- Отсутствие поддержки Unity ссылок на внешние библиотеки, работу с которыми программистам приходится настраивать самостоятельно, и это также затрудняет командную работу
- Производительность. Нужно хорошо знать тонкости разработки пользовательского интерфейса, чтобы сделать его производительным.
- Оптимизация. Кроссплатформенные и кроссжанровые движки имеют меньшую производительность по сравнению с узконаправленными движками.
- Нет шаблонов. Как только игра становится чуть сложнее, нужна хорошо продуманная архитектура, иначе ее не получится выпустить.

6.2 Язык C#

C# — объектно-ориентированный язык программирования, разработан как язык разработки приложений для платформы Microsoft .NET Framework и .NET Core.

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, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников C#, 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 множественная реализация интерфейсов) [6].

Преимущества:

- C# – это объектно-ориентированный, который позволяет разработчикам создавать многофункциональные приложения.
- Поддержка подавляющего большинства продуктов Microsoft
- Типы данных имеют фиксированный размер (32-битный int и 64-битный long), что повышает «мобильность» языка и упрощает программирование.
- Автоматическая «сборка мусора» Это значит, что в большинстве случаев не придётся заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.
- Большое количество «синтаксического «сахара» — специальных конструкций, разработанных для понимания и написания кода. Они не имеют значения при компиляции.
- Низкий порог вхождения. Синтаксис C# имеет много схожего с другими языками программирования, благодаря чему облегчается переход для программистов. Язык C# часто признают наиболее понятным и подходящим для новичков.

- С помощью Xamarin на C# можно писать программы и приложения для таких операционных систем, как iOS, Android, MacOS и Linux.
- Наличие большого количества библиотек и шаблонов.

Недостатки C#:

- Приоритетная ориентированность на платформу Windows.
- C# очень легко дизассемблируется. Это означает, что с большой долей вероятности твой код будет получен и изучен конкурентами.
- .NET использует концепцию JIT-компиляции. Это означает, что программа будет скомпилирована в машинные коды по мере необходимости прямо во время работы приложения.
- C# не является повсеместно распространенным языком.

7 АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ

7.1 Файловая структура приложения

В результате разработки была создана файловая структура игрового приложения, которая состоит из набора каталогов, содержащих файлы звукового сопровождения, спрайты, шрифты, игровые скрипты и готовые компоненты. На рисунке 4 представлена файловая структура приложения.

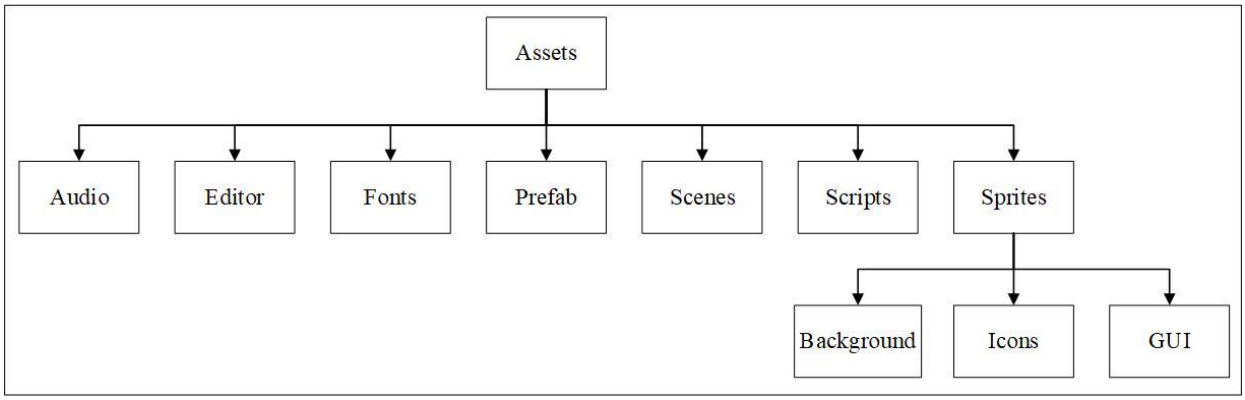

Рисунок 4 – Файловая структура

В каталоге Audio содержатся звуковые композиции.

В каталоге Editor содержится скрипт для изменения редактора компонента.

В каталоге Fonts содержится текстовый шрифт, используемый в игровом приложении.

В каталоге Prefab содержится шаблон игрового объекта.

В каталоге Scenes содержатся игровые сцены: Menu, Lvl1, Lvl2, Lvl3.

В каталоге Scripts содержатся скрипты, используемые в игровом приложении.

В каталоге Sprites содержатся папки с графическими файлами, используемые в игровом приложении. В папке Background хранятся фоновые изображения игры. В папке Icons хранятся спрайты игровых элементов. В папке GUI содержатся графические элементы, используемые в пользовательском интерфейсе.

7.2 Алгоритмы реализации программного средства

Все основные действия игрового приложения происходят на сценах уровней. Прецедент «Запуск уровня» является наиболее значимым и загруженным, поэтому для большей ясности поведения на игровой сцене было решено построить диаграмму деятельности, которая представлена на рисунке 5.

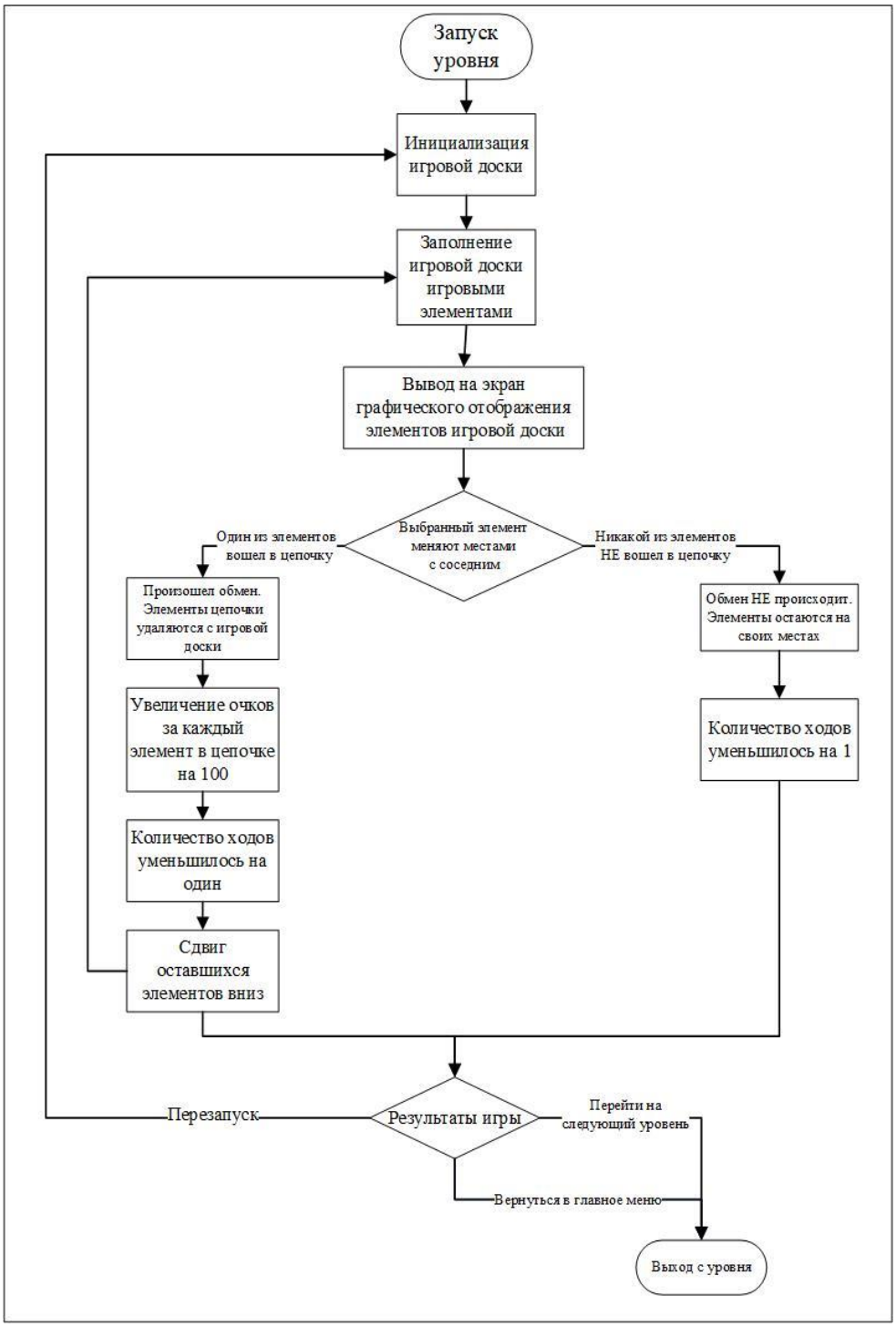

Рисунок 5 – Диаграмма деятельности прецедента «Запуск уровня»

Исходя из диаграммы деятельности прецедента «Запуск уровня» можно увидеть, что все действия игрока происходят на игровой доске. Игровая доска является двумерным массивом, представленным в плоской проекции. Каждый объект доски— это спрайт с размерами 100x100 пикселей. Плитки имеют квадратную форму. На рисунке 6 показан алгоритм, с помощью которого спрайты укладываются в плиточную структуру.

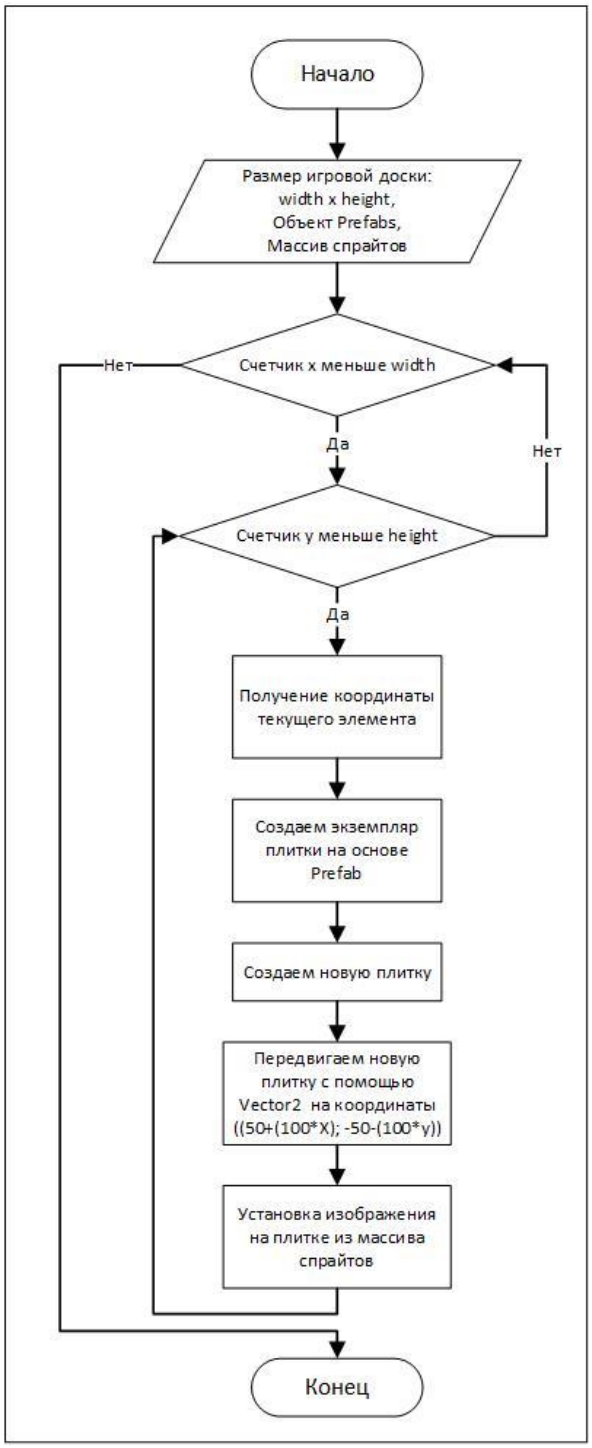

Рисунок 6 – Алгоритм заполнения игровой доски

Анимация изменения положения игрового элемента и обмена его с соседним элементом возможны благодаря классу RectTransform и подключенной системе обработки событий EventSystem.

Event System – способ отправки событий к объектам в приложении, основанный на вводе с клавиатуры или мыши; с помощью касаний или персональных устройств. Система состоит из нескольких компонентов, работающих вместе. Эта система с помощью Raycast определяет, что находится под указателем. Но для ее работы требуется два условия: на сцене должен присутствовать объект Event System, а на объекте, с которым мы хотим взаимодействовать с помощью этой системы, класс с интерфейсами событий [7].

EventSystem использовалась для обработки события нажатия на объект. Для этого использовались, представленные на рисунке 7, метод, отслеживающий событие нажатой игровой плитки и метод, отслеживающий, когда эту плитку отпустили. Для каждого из них требуется добавить свой интерфейс: IPointerDownHandler и IPointerUpHandler.

```
public void OnPointerDown(PointerEventData eventData)
    {
        if (update) return;
        MovePiece.instance.MovingPiece(this);
    }

public void OnPointerUp(PointerEventData eventData)
    {
        MovePiece.instance.DropPiece();
    }
```

Рисунок 7 – Методы для обработки события нажатия игровой плитки

Сама анимация движения игрового элемента реализована с помощью методов MovePosition и MovePositionTo, описанные на рисунке 8 и использующие объект класса RectTransform.

```
public RectTransform rect;

public void MovePosition(Vector2 pos)
    {
        rect.anchoredPosition += pos * Time.deltaTime * 25f;
    }

public void MovePositionTo(Vector2 pos)
    {
        rect.anchoredPosition = Vector2.Lerp(rect.anchoredPosition, pos, Time.deltaTime * 25f);
    }
```

Рисунок 8 – Методы для реализации анимации движения плиток

Для реализации загрузки и сохранения данных о лучшем набранном счете в игровом приложении был создан скрипт UI. При первом запуске игры происходит инициализация переменных в хранилище PlayerPrefs, а при последующих запусках игрового происходит их считывание из того же хранилища.

На рисунке 9 приведена часть метода, где происходит обновление текущего результата на актуальный при изменении рекорда.

```
public void LevelScore() //подсчет очков
{
     if (score > PlayerPrefs.GetInt("Score1"))
            {
                PlayerPrefs.SetInt("Score1", score);
                gTextBestScore.text = "New Best: " + score.ToString();
            }

}
```

Рисунок 9 – Обновление рекорда набранных очков

7.3 Схема функционирования программного средства

В разработке мобильного игрового приложения основной задачей стоит разработка пользовательских классов, которые подключаются к игровым объектам как компоненты. Все такие классы должны наследоваться от класса MonoBehaviour. Указание этого отношения значительно перегрузило бы диаграмму, поэтому данный класс не включается в схему.

На рисунке 10 представлена диаграмма классов игрового приложения.

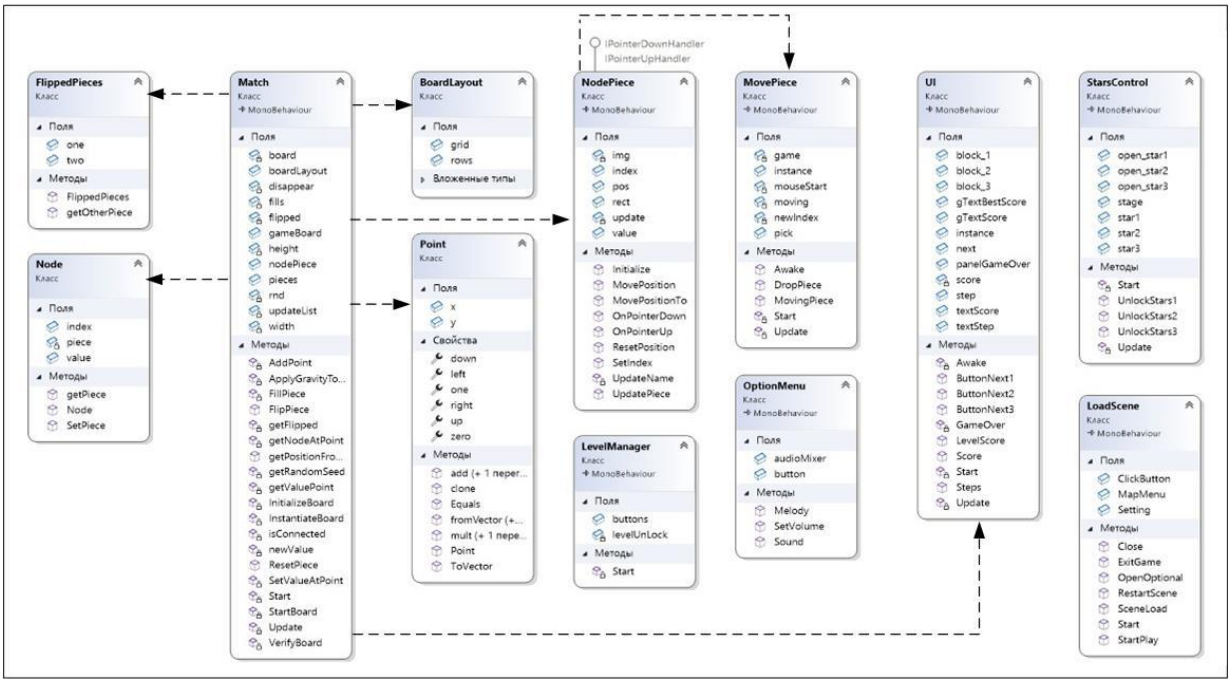

Рисунок 10 - Диаграмма классов

Класс Match реализует основную логику игрового приложения.

Класс Node предназначен для описания характеристик игрового элемента.

Класс FlippedPieces отвечает за элементы, которые обменивают.

Класс BoardLayout предназначен для работы с игровой доской.

Класс Point предназначен для управления координатами местоположения игровых элементов.

Класс NodePiece предназначен для управления спрайтами игровых плиток.

Класс MovePiece отвечает за движение игровых элементов.

Класс UI реализует игровой пользовательский интерфейс.

Класс LevelMeneger отвечает за сохранение активности и включение кнопок уровней.

Класс OptionMenu отвечает за изменения настроек в игре (уменьшить или увеличить громкость, отключить звук или мелодию).

Класс StarsControl отвечает за сохранение и включения системы звезд.

Класс LoadScene отвечает за смену сцен в игровом приложении.

7.4 Схема экранов приложения

На рисунке 11 изображена схема связи экранов игрового приложения.

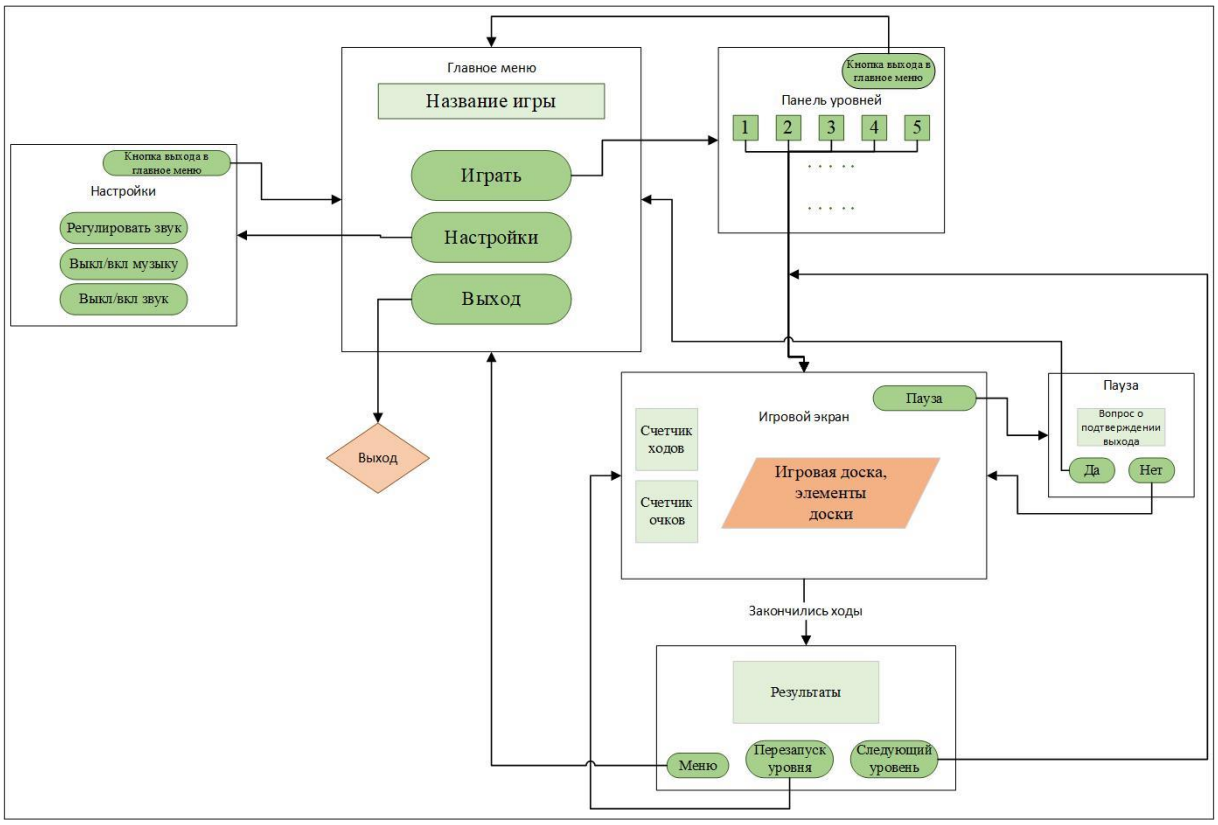

Рисунок 11 – Схема экранов приложения

При запуске игрового приложения появляется главное меню, которое позволяет начать игру («Start»), регулировать настройки(«Settings»), и выйти из приложения(«Quit»). Название игры «Fairy Forest» отображается вверху сцены меню.

Панель уровней включает в себя кнопки, позволяющую начать игровой процесс, на выбранном доступном уровне. А также отображают прогресс с помощью системы звезд. Обновление системы звезд происходит с помощью скрипта Stars Control, а разблокировка уровней с помощью скрипта Level Manager.

В панели настроек конфигурации игры можно изменить громкость фоновой музыки игры с помощью ползунка громкости, реализованного с помощью UI.Slider. Также можно совсем отключить все звуки, нажав на

кнопку-переключатель «Sound On/Off» или же отключить только фоновую музыку с помощью кнопки-переключателя «Melody On/Off». Обе кнопки представляют UI.Toggle. Все элементы выполняют методы, содержащиеся в скрипте OptionMenu.

Во время игрового процесса можно нажать на кнопку в верхнем правом углу игровой панели, представляющая собой стрелку, что поставит игру на паузу. Во время паузы высветиться панель с сообщением о прерывании игры и двумя кнопками. Игра возобновляется после нажатия на кнопку «No».

Панель результатов содержит кнопки «Menu», «Restart» и «Next» и отображает количество набранных очков текущей игровой сессии и лучший счет за все время.

Панель счетчика ходов отображает количество ходов на данный момент игры. Обновление счетчика происходит при помощи метода из скрипта UI, который берет данные из скрипта Match.

Панель счетчика очков отображает количество набранных очков и систему звезд. Обновление счетчика происходит при помощи метода из скрипта UI, который берет данные из скрипта Match. Обновление звезд происходит с помощью метода из скрипта Stars Control.

Все кнопки в мобильном приложении зеленого цвета и выполняют методы, описанные в скрипте LoadScene. При наведении на кнопки они приобретают более темный оттенок.

8 ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Входные данные – величины, которые задаются до начала работы алгоритма или определяются динамически во время его работы. Входные данные берутся из определенного набора объектов [8].

Входными данными игрового приложения являются:

- двумерный массив, описанный на языке C#, для описания игровой доски;
- массив спрайтов для заполнения игровой доски;
- аудио компоненты (объекты класса AudioSource) для сопровождения игрового процесса.

Выходные данные – это данные, получаемые в результате решения какой-либо задачи, алгоритма.

Выходными данными являются:

- графическое изображение игровой доски с элементами, формируемое в процессе выполнения программы;
- аудио эффекты и фоновая музыка, воспроизводимые в процессе выполнения программы.

9 ТЕСТИРОВАНИЕ И ОТЛАДКА

Чтобы установить соответствие разработанного программного обеспечения (ПО) исходным функциональным требованиям необходимо провести функциональное тестирование. Функциональное тестирование — это тестирование ПО в целях проверки реализуемости функциональных требований, позволяющий проверить способность системы работать в определенных условиях и решать задачи, нужные пользователям [9].

В таблице 2 приведены результаты функциональное тестирования мобильного приложения.

Таблица 2 – Функциональное тестирование

№	Название теста	Шаги	Ожидаемый результат	Прохождение теста
1	Запуск игрового приложения	Нажать на иконку приложения на экране смартфона	Приложение запустится и будет работать на ОС Android	Да
2	Переход из главного меню на панель уровней	Нажать на кнопку «Start»	Происходит переход на панель уровней	Да
3	Переход из главного меню на панель настроек	Нажать на кнопку «Settings»	Происходит переход на панель настроек	Да
4	Переход из меню уровней на игровую сцену	Нажать на кнопку любого доступного уровня	Переход на игровую сцену. Начало игры	Да
5	Обмен выбранного элемента с соседним при вхождении в состав цепочки.	Передвижение выбранного элемента к соседнему, и их обмен	Один из элементов входит в цепочку, и происходит обмен.	Да
6	Удаление цепочки, и заполнение игрового поля новыми элементами	Элементы, входящие в цепочку, удаляются. Пустые поля заполняются	Образованная цепочка удаляется. Оставшиеся элементы сдвигаются вниз. Освободившиеся сверху поля заполняются новыми элементами	Да

Продолжение таблицы 2

№	Название теста	Шаги	Ожидаемый результат	Прохождение теста
7	Начисление очков	При образовании цепочки начисляются очки	Начисляются очки за каждый элемент входящий в состав цепочки	Да
8	Сокращение количества ходов	При каждой попытке обменять элементы местами количество ходов сокращается	Количество ходов сокращается на единицу	Да
9	Открытие окна с результатами	Когда число ходов равно нулю, открывается окно с результатами	Открывается окно с результатами текущей игровой сессией и лучшим счетом за все время	Да
10	Сохранение лучшего результата	При установке нового рекорда, счетчик лучшего результата обновляется	Обновляется лучший результат	Да
11	Прерывание игры	1.Нажать на кнопку со стрелкой на игровой сцене 2.Нажать на кнопку «Yes» в уточняющем окне	Открывается окно с уточняющим вопросом. Происходит переход на сцену главного мен.	Да
12	Перезапуск уровня	В окне с результатами нажать на кнопку «Restart»	Текущий уровень игры перезапускается. Игра начинается сначала	Да
13	Переход на сцену главного меню	В окне с результатами нажать на кнопку «Menu»	Происходит переход на сцену главного меню	Да
14	Разблокировка следующего уровня	Набрать необходимый минимальный счет на текущем уровне для разблокировки следующего	В окне меню уровней кнопка следующего уровня стала доступной	Да
15	Разблокировка кнопки «Next» в окне результатов уровня	Набрать необходимый минимальный счет на текущем уровне для разблокировки следующего уровня	В окне результатов уровня кнопка «Next» стала активной	Да
16	Переход на следующий уровень	В окне с результатами уровня нажать на кнопку «Next»	Происходит переход на игровую сцену следующего уровня	Да

Окончание таблицы 2

№	Название теста	Шаги	Ожидаемый результат	Прохождение теста
17	Изменение громкости в меню настроек	При передвижении ползунка громкость звукового сопровождения изменяется	Изменяется громкость звукового сопровождения	Да
18	Выключение/ включение музыкального сопровождения в меню настроек	Нажать на кнопку с иконкой ноты	Музыкальное сопровождение выключается/ включаются	Да
19	Выключение/ включение всех звуков в меню настроек	Нажать на кнопку с иконкой динамиков	Все звуковое сопровождение выключается/ включаются	Да
20	Выход из игрового приложения	Нажать на кнопку «Quit»	Игровое приложение закрывается	Да

Все проделанные тесты были успешно пройдены итоговой версией приложения, следовательно, приложение работает правильно.

10 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

После установки приложения, пользователю для запуска необходимо нажать на иконку игры после чего на экране появится главное меню игры.

На рисунке 12 представлено главное меню приложения, откуда осуществляется переход к выбору уровня игры с помощью кнопки «Start» и при выборе кнопки «Settings» к настройкам игры. А также при нажатии на кнопку «Quit» осуществляется выход из игрового приложения.

Рисунок 12 – Главное меню

На рисунке 13 представлена панель настроек конфигурации игры, где можно изменить громкость фоновой музыки игры. Также можно совсем отключить все звуки, нажав на кнопку «Sound On/Off» или же отключить только фоновую музыку с помощью кнопки «Melody On/Off».

Рисунок 13 – Панель настроек

На рисунке 14 изображена панель с выбором уровня игры. Здесь также отображается успех прохождения уровней в виде звезд.

Рисунок 14 – Панель выбора уровня игры

На рисунке 15 представлена игровая панель. Здесь игрок играет в саму игру, меняя местами положение выбранного элемента с элементами по соседству. Слева отображается количество оставшихся ходов и число набранных очков.

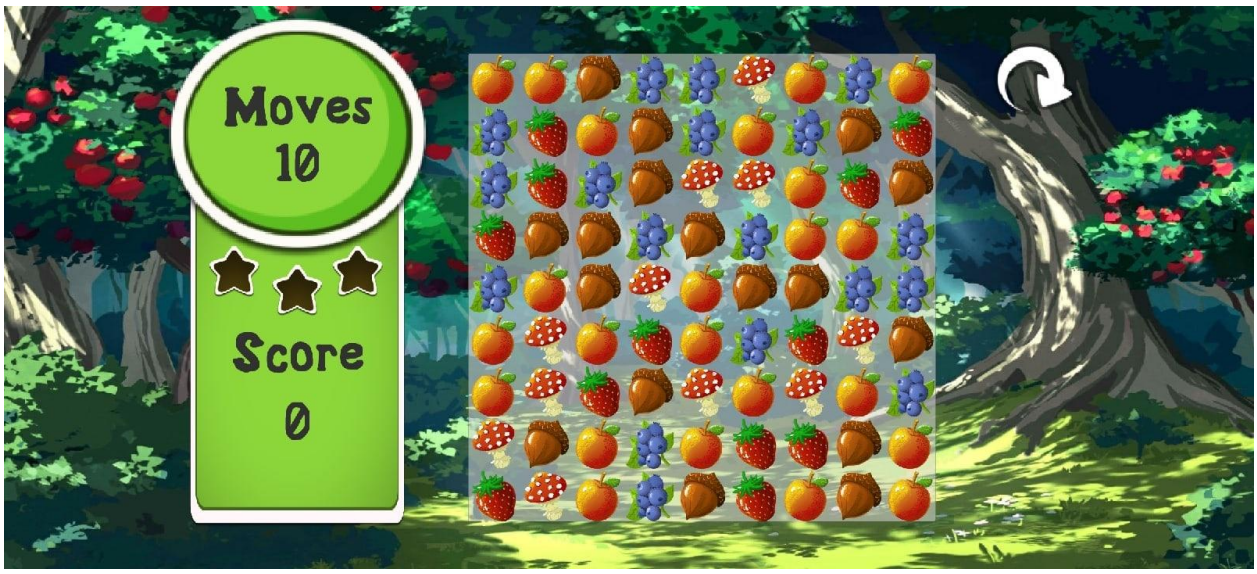

Рисунок 15 – Игровая панель

При нажатии кнопки в верхнем правом углу игровой панели, представляющая собой стрелку, выскакивает панель с сообщением. На рисунке 16 показано окно сообщения, где у пользователя уточняют уверен ли он, что хочет выйти. При выходе результаты игровой сессии не сохраняются.

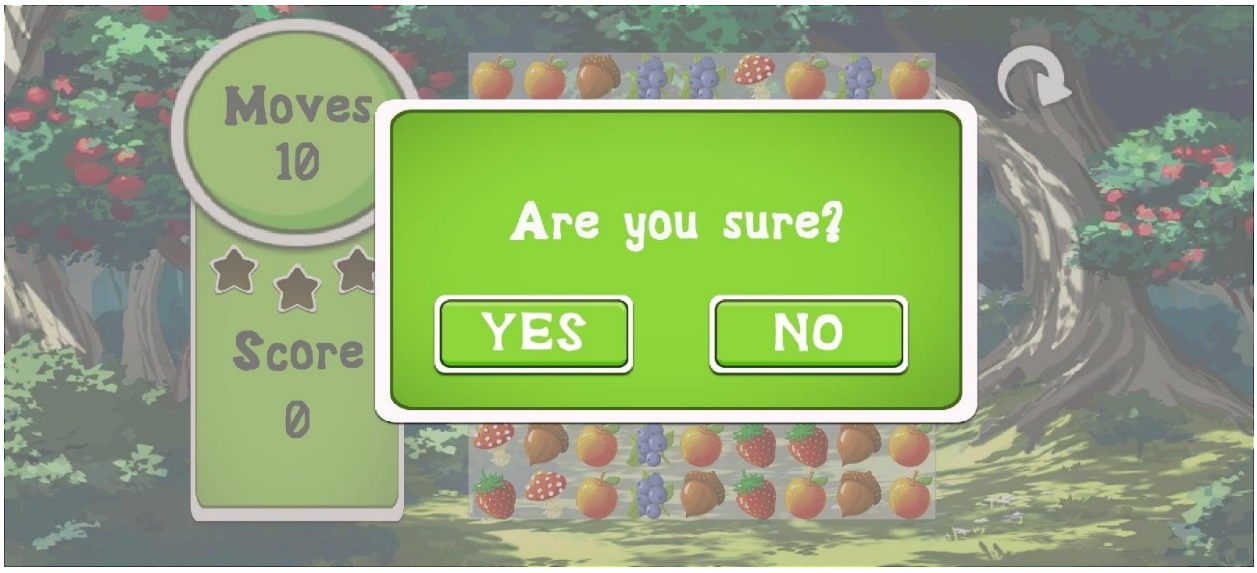

Рисунок 16 – Панель с сообщением о прерывании игры

Когда заканчиваются все ходы, высвечивается панель с результатами игровой сессии, что показано на рисунке 17. Здесь указывается набранный счет за пройденную сессию, а также лучший счет за все время. У игрока есть возможность вернуться в главное меню с помощью кнопки «Menu» и

перезапустить уровень, нажав на кнопку «Restart». Если пользователь набирает необходимо число очков, то следующий уровень разблокируется, как и кнопка перехода «Next». В данном случае кнопка заблокирована – игрок не может перейти на следующий уровень.

Рисунок 17 – Панель с результатами

На рисунке 18 представлено окно с новыми результатами и разблокированной кнопкой на следующий уровень.

Рисунок 18 – Панель с новыми результатами

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано программное мобильное игровое приложение на ОС Android в жанре «Три в ряд». Мобильное приложение реализовано средствами мультиплатформенного движка Unity, языка программирования C# и библиотеки C# для Unity.

В результате выполнения данного проекта были решены задачи, поставленные в начале работы:

- Был произведен анализ предметной области.
- Был произведен обзор изученных программных средств разработки игрового приложения для мобильных ОС.
- Был произведен сравнительный анализ существующих игровых приложений, схожих с разрабатываемым.
- Было произведено проектирование, позволившее определить окончательную структуру игрового приложения.
- Был спроектирован простой и удобный пользовательский интерфейс.
- Было проведено функциональное тестирование на этапе контроля качества программного обеспечения.

На основе анализа результатов функционального тестирования можно сделать вывод о том, что разработанное игровое приложение и все входящие в его состав объекты работают корректно.

Таким образом, в результате выполнения курсовой работы было разработано игровое программное средство, отвечающие всем функциональным требованиям, но теперь стоит задача сделать его более совершенным.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1 Рынок мобильных игр в I квартале 2022 года [Электронный ресурс] / LogrusIT — URL: https://games.logrusit.com/ru/news/game-market-q1-2022/ (Дата обращ. 21.05.22).

2 Казуальная игра [Электронный ресурс] / Википедия – свободная энциклопедия — URL: https://ru.wikipedia.org/wiki/Казуальная_игра (Дата обращ. 27.05.22).

3 Три в ряд (жанр игры) [Электронный ресурс] / Академик – словари и энциклопедии — URL: https://dic.academic.ru/dic.nsf/ruwiki/691627#.D0.9F.D1.80.D0.B5.D0.B4.D1.81.D1.82.D0.B0.D0.B2.D0.B8.D1.82.D0.B5.D0.BB.D0.B8_.D0.B6.D0.B0.D0.BD.D1.80.D0.B0 (Дата обращ. 27.05.22).

4 Три в ряд [Электронный ресурс] / Википедия – свободная энциклопедия — URL: https://ru.wikipedia.org/wiki/Три_в_ряд (Дата обращ. 27.05.22).

5 Unity (игровой движок) [Электронный ресурс] / Википедия – свободная энциклопедия — URL: https://ru.wikipedia.org/wiki/Unity_(игровой_движок) (Дата обращ. 29.05.22).

6 C Sharp [Электронный ресурс] / Википедия – свободная энциклопедия — URL: https://ru.wikipedia.org/wiki/C_Sharp (Дата обращ. 29.05.22).

7 EventSystem [Электронный ресурс] /Unity User Manual — URL: https://docs.unity3d.com/ru/2019.4/Manual/EventSystem.html (Дата обращ. 13.05.22).

8 Входные данные [Электронный ресурс] / Академик – словари и энциклопедии — URL: https://dic.academic.ru/dic.nsf/ruwiki/473941 (Дата обращ. 07.06.22).

9 Анализ требований [Электронный ресурс] / Википедия – свободная энциклопедия — URL: https://ru.wikipedia.org/wiki/Анализ_требований (Дата обращ. 30.05.22).

ПРИЛОЖЕНИЕ А

- Код класса Match

```
public class Match : MonoBehaviour
{
    public BoardLayout boardLayout;

    [Header("Игровые элементы")]
    public Sprite[] pieces;
    public RectTransform gameBoard;

    [Header("Шаблон")]
    public GameObject nodePiece;



    int width = 9;
    int height = 9;
    int[] fills;
    Node[,] board;

    List<NodePiece> updateList;
    List<FlippedPieces> flipped;
    List<NodePiece> disappear;

    System.Random rnd;

    void Start()
    {
        StartBoard();
    }

    void Update()
    {
        List<NodePiece> finishUpdate = new List<NodePiece>();
        for (int i = 0; i < updateList.Count; i++)
        {
            NodePiece piece = updateList[i];
            if (!piece.UpdatePiece())
            {
                finishUpdate.Add(piece);
            }
```

```
}

for (int i = 0; i < finishUpdate.Count; i++)
{
   NodePiece piece = finishUpdate[i];
   FlippedPieces flip = getFlipped(piece);
   NodePiece flippedPiece = null;

   int x = (int)piece.index.x;
   fills[x] = Mathf.Clamp(fills[x]-1, 0, width);

   List<Point> connected = isConnected(piece.index, true);
   bool wasFlipped = (flip != null);

   if (wasFlipped)
   {
      flippedPiece = flip.getOtherPiece(piece);
      AddPoint(ref connected, isConnected(flippedPiece.index, true));
      UI.instance.Steps(1);
   }

   if (connected.Count == 0) //if match is not exist
   {
      if (wasFlipped)
      {
         FlipPiece(piece.index, flippedPiece.index, false);
      }
   }
   else
   {
      foreach (Point pnt in connected)
      {
         UI.instance.Score(100);

         Node node = getNodeAtPoint(pnt);
         NodePiece nodePiece = node.getPiece();
         if (nodePiece != null)
         {
            nodePiece.gameObject.SetActive(false);
            disappear.Add(nodePiece);
         }
         node.SetPiece(null); }
```

```
            ApplyGravityToBoard();
        }

        flipped.Remove(flip);
        updateList.Remove(piece);
    }
}

void ApplyGravityToBoard()
{
    for (int x = 0; x < width; x++)
    {
        for (int y = (height - 1); y >= 0; y--)
        {
            Point p = new Point(x, y);
            Node node = getNodeAtPoint(p);
            int val = getValuePoint(p);
            if (val != 0) continue;
            for (int ny = (y - 1); ny >= -1; ny--)
            {
                Point next = new Point(x, ny);
                int nextVal = getValuePoint(next);
                if (nextVal == 0) continue;
                if (nextVal != -1)
                {
                    Node gotten = getNodeAtPoint(next);
                    NodePiece piece = gotten.getPiece();

                    node.SetPiece(piece);
                    updateList.Add(piece);

                    gotten.SetPiece(null);
                }
                else
                {
                    int newVal = FillPiece();
                    NodePiece piece;
                    Point fallPoint = new Point(x, (-1 - fills[x]));
                    if (disappear.Count > 0)
                    {
                        NodePiece revived = disappear[0];
```

```
                revived.gameObject.SetActive(true);
                revived.rect.anchoredPosition = getPositionFromPoint(fallPoint);
                piece = revived;

                disappear.RemoveAt(0);
            }
            else
            {
                GameObject obj = Instantiate(nodePiece, gameBoard);
                NodePiece n = obj.GetComponent<NodePiece>();
                RectTransform rect = obj.GetComponent<RectTransform>();
                rect.anchoredPosition = getPositionFromPoint(fallPoint);
                piece = n;

            }
            piece.Initialize(newVal, p, pieces[newVal - 1]);

            Node hole = getNodeAtPoint(p);
            hole.SetPiece(piece);
            ResetPiece(piece);
            fills[x]++;
          }
          break;
        }
      }
    }
}
FlippedPieces getFlipped(NodePiece p)
{
    FlippedPieces flip = null;
    for (int i = 0; i < flipped.Count; i++)
    {
      if (flipped[i].getOtherPiece(p) != null)
      {
        flip = flipped[i];
        break;
      }
    }
    return flip;
}

void StartBoard()
```

```
{
    fills = new int[width];
    string seed = getRandomSeed();
    rnd = new System.Random(seed.GetHashCode());
    updateList = new List<NodePiece>();
    flipped = new List<FlippedPieces>();
    disappear = new List<NodePiece>();

    InitializeBoard();
    VerifyBoard();
    InstantiateBoard();
}

void InitializeBoard()
{
    board = new Node[width, height];
    for (int y=0;y<height;y++)
    {
        for (int x=0;x<width;x++)
        {
            board[x,y] = new Node((boardLayout.rows[y].row[x])? -1 : FillPiece() , new Point(x,y));
        }
    }

}

void VerifyBoard()
{
    List<int> remove;
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            Point p = new Point(x, y);
            int val = getValuePoint(p);
            if (val <= 0) continue;

            remove = new List<int> ();
            while (isConnected(p, true).Count > 0)
            {
                val = getValuePoint(p);
                if (!remove.Contains(val)) remove.Add(val);
```

```
            SetValueAtPoint(p, newValue(ref remove));
        }
    }
  }
}

void InstantiateBoard()
{
  for (int x = 0; x < width; x++)
  {
    for (int y = 0; y < height; y++)
    {
      Node node = getNodeAtPoint(new Point (x,y));

      int val = node.value;
      if (val <= 0) continue;
      GameObject p = Instantiate(nodePiece, gameBoard);
      NodePiece piece = p.GetComponent<NodePiece>();
      RectTransform rect = p.GetComponent<RectTransform>();
      rect.anchoredPosition = new Vector2(50 + (100 * x), -50 - (100 * y));
      piece.Initialize(val, new Point(x, y), pieces[val-1]);
      node.SetPiece(piece);
    }
  }
}

public void ResetPiece(NodePiece piece)
{
  piece.ResetPosition();
  updateList.Add(piece);
}

public void FlipPiece(Point one, Point two, bool main)
{
  if(getValuePoint(one) < 0) return;
  Node nodeOne = getNodeAtPoint(one);
  NodePiece pieceOne = nodeOne.getPiece();
  if (getValuePoint(two) > 0)
  {
    Node nodeTwo = getNodeAtPoint(two);
    NodePiece pieceTwo = nodeTwo.getPiece();
    nodeOne.SetPiece(pieceTwo);
```

```
        nodeTwo.SetPiece(pieceOne); //swap

        if(main) flipped.Add(new FlippedPieces(pieceOne, pieceTwo));

        updateList.Add(pieceOne);
        updateList.Add(pieceTwo);
    }
    else
    {
        ResetPiece(pieceOne);
    }
}

List<Point> isConnected(Point p, bool main)
{
    List<Point> connected = new List<Point>();
    int val = getValuePoint(p);
    Point[] direction =
    {
        Point.up,
        Point.right,
        Point.down,
        Point.left,
    };

    foreach (Point dir in direction)
    {
        List<Point> line = new List<Point>();

        int same = 0;
        for (int i = 1;  i < 3; i++)
        {
            Point check = Point.add(p, Point.mult(dir, i));
            if (getValuePoint(check) == val)
            {
                line.Add(check);
                same++;
            }
        }

        if (same > 1)
        {
```

```
            AddPoint(ref connected, line);
        }
    }

    for (int i = 0; i < 2; i++)
    {
        List<Point> line = new List<Point>();

        int same = 0;
        Point[] check = { Point.add(p, direction[i]), Point.add(p, direction[i + 2]) };
        foreach (Point next in check)
        {
            if (getValuePoint(next) == val)
            {
                line.Add(next);
                same++;
            }
        }

        if (same > 1)
        {
            AddPoint(ref connected, line);
        }
    }

    for (int i = 0; i < 4;  i++)
    {
        List <Point> square = new List<Point>();

        int same = 0;
        int next = i + 1;
        if (next>=4) next -= 4;

        Point[] check =
        {
            Point.add(p, direction[i]),
            Point.add(p, direction[next]),
            Point.add(p, Point.add(direction[i], direction[next]))
        };
        foreach (Point nxt in check)
        {
            if (getValuePoint(nxt) == val)
```

```
            {
                square.Add(nxt);
                same++;
            }
        }

        if (same > 2)
        {
            AddPoint(ref connected, square);
        }
    }

    if (main)
    {
        for (int i = 0; i< connected.Count; i++)
        {
            AddPoint(ref connected, isConnected(connected[i], false));
        }
    }

    return connected;
}

void AddPoint(ref List<Point> points, List<Point> add)
{
    foreach (Point p in add)
    {
        bool doAdd = true;

        for (int i = 0; i < points.Count; i++)
        {
            if (points[i].Equals(p))
            {
                doAdd = false;
                break;
            }
        }

        if (doAdd)
        {
            points.Add(p);
        }
```

```
    }
}

int FillPiece()
{
    int value = 1;
    value = (rnd.Next(0, 100) / (100 / pieces.Length)) + 1;
    return value;
}

int getValuePoint(Point p)
{
    if (p.x < 0 || p.x >= width || p.y < 0 || p.y >= height) return -1;
    return board[p.x, p.y].value;
}

void SetValueAtPoint(Point p, int v)
{
    board[p.x, p.y].value = v;
}

Node getNodeAtPoint(Point p)
{
    return board[p.x, p.y];
}

int newValue (ref List<int> remove)
{
    List<int> available = new List<int>();

    for (int i=0; i < pieces.Length; i++)
    {
        available.Add(i + 1);
    }

    foreach( int i in remove)
    {
        available.Remove(i);
    }

    if (available.Count <= 0) return 0;
    return available[rnd.Next(0, available.Count)];
```

```
    }

    string getRandomSeed()
    {
        string seed = "";
        string acceptableChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstu-
vwxyz1234567890!@#$%^&*()";
        for (int i = 0; i < 20; i++)
        {
            seed += acceptableChars[Random.Range(0,acceptableChars.Length)];
        }
        return seed;
    }

    public Vector2 getPositionFromPoint (Point p)
    {
        return new Vector2(50 + (100 * p.x), -50 - (100 * p.y));
    }

}
```

- Код класса UI

```
public class UI : MonoBehaviour
{
    public static UI instance;

    public Text textScore, textStep;
    private int score = 0;
    public int step;

    public GameObject panelGameOver;
    public Text gTextScore, gTextBestScore;

    public static int block_1, block_2, block_3;
    public Button next;

    private void Awake()
    {
        instance = this;
    }

    void Start()
    {
        block_1 = PlayerPrefs.GetInt("Lvl2", 1);
        block_2 = PlayerPrefs.GetInt("Lvl3", 1);
        block_3 = PlayerPrefs.GetInt("Lvl4", 1);
    }
    void Update()
    {
        if (SceneManager.GetActiveScene().buildIndex == 1) ButtonNext1();
        else if (SceneManager.GetActiveScene().buildIndex == 2) ButtonNext2();
        else if (SceneManager.GetActiveScene().buildIndex == 3) ButtonNext3();
    }
```

```
public void Score (int value)
{
   score += value;
   textScore.text = "Score" + score.ToString();
}

public void Steps (int value)
{
   step -= value;
   textStep.text = step.ToString();
   if (step <= 0) GameOver();
}

private void GameOver()
{
   LevelScore();

   gTextScore.text = "Score: "+ score.ToString();
   panelGameOver.SetActive(true);
}

public void LevelScore()
{
   if (SceneManager.GetActiveScene().buildIndex == 1)
   {
      if (score > PlayerPrefs.GetInt("Score1"))
      {
         PlayerPrefs.SetInt("Score1", score);
         gTextBestScore.text = "New Best: " + score.ToString();
      }
      else
      {
         gTextBestScore.text = "Best: " + PlayerPrefs.GetInt("Score1");
      }
   }
   else if (SceneManager.GetActiveScene().buildIndex == 2)
   {
      if (score > PlayerPrefs.GetInt("Score2"))
      {
         PlayerPrefs.SetInt("Score2", score);
         gTextBestScore.text = "New Best: " + score.ToString();
      }
      else
      {
         gTextBestScore.text = "Best: " + PlayerPrefs.GetInt("Score2");
      }
   }
   else
   {
      if (score > PlayerPrefs.GetInt("Score3"))
      {
         PlayerPrefs.SetInt("Score3", score);
         gTextBestScore.text = "New Best: " + score.ToString();
      }
      else
      {
         gTextBestScore.text = "Best: " + PlayerPrefs.GetInt("Score3");
      }
   }
}
```

```
    public void ButtonNext1()
    {
        if (block_1 == 1)
        {
            next.interactable = false;
        }

        if (block_1 == 2)
        {
            next.interactable = true;
        }

        if (PlayerPrefs.GetInt("Score1") >= 5000)
        {
            block_1 = 2;
            PlayerPrefs.SetInt("Lvl2", block_1);

            int currentLevel = SceneManager.GetActiveScene().buildIndex;
            if (currentLevel >= PlayerPrefs.GetInt("levels"))
            {
                PlayerPrefs.SetInt("levels", currentLevel + 1);
            }
        }
    }

    public void ButtonNext2()
    {
        if (block_2 == 1)
        {
            next.interactable = false;
        }

        if (block_2 == 2)
        {
            next.interactable = true;
        }

        if (PlayerPrefs.GetInt("Score2") >= 5000)
        {
            block_2 = 2;
            PlayerPrefs.SetInt("Lvl3", block_2);

            int currentLevel = SceneManager.GetActiveScene().buildIndex;
            if (currentLevel >= PlayerPrefs.GetInt("levels"))
            {
                PlayerPrefs.SetInt("levels", currentLevel + 1);
            }
        }
    }

    public void ButtonNext3()
    {
        if (block_3 == 1)
        {
            next.interactable = false;
        }

        if (block_3 == 2)
        {
            next.interactable = true;
        }
    }
}
```