Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Прикладные задачи математического анализа

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

Мобильная игра на платформе Android в жанре auto battler

БГУИР КП 1-40 04 01

Студент:
гр.253503
Телего Е.А.

Руководитель:
ассистент каф. информ.
Рогов М.Г.

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире игры становятся все более популярными и востребованными развлечениями, предоставляющими пользователям возможность погрузиться в увлекательные виртуальные миры с помощью своих устройств. В том числе довольно популярными являются именно игры на мобильных платформах, таких как Android. Одним из актуальных жанров в сфере мобильных игр является так называемый auto battler. Игры этого жанра позволяют игрокам создавать команды персонажей и участвовать в автоматизированных битвах с оппонентами или компьютерным искусственным интеллектом.

Цель данной курсовой работы заключается в разработке и реализации мобильной игры с помощью системы Unity для устройств под управлением операционной системы Android в жанре auto battler с элементами стратегии.

Для достижения поставленной цели необходимо решить следующие задачи:

− Исследование существующих игр в жанре auto battler и анализ их особенностей.

− Проектирование архитектуры игры с учетом основных механик и возможностей платформы Unity.

− Реализация базового игрового функционала, включающего в себя систему боев, управления игровой армией и взаимодействия с игровым миром.

− Создание игровых уровней, персонажей и игровых элементов, а также балансировка игрового процесса.

− Тестирование игры на различных устройствах Android и устранение возможных неполадок.

Игра будет выполнена на движке Unity и языке C#. В проекте будет присутствовать регистрация, система сохранений, настройки, главное меню, меню выбора локаций, игровые уровни. Каждый из этих компонентов представляет собой отдельную игровую сцену.

В ходе разработки программного средства будут использованы принципы ООП и SOLID. Система сохранений, загрузки и регистрации будет использовать реляционную базу данных SQLite совместно с ORM подходом для взаимодействия с ней.

Игра будет иметь однопользовательскую геймплейную составляющую. Игровым оппонентом будут выступать специально прописанные автоматизированные скрипты.

Пользователь сможет рассчитывать на сохранение своего игрового процесса между сессиями благодаря ручным и автоматизированным сохранениям.

Разработанное программное средство будет представлять функционирующее объектно-ориентированное приложение. Оно должно решать все ранее поставленные задачи и обеспечивать удобство и эффективность работы для пользователей.

ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1 Структура и архитектура платформы

Unity является кросс-платформенным игровым движком, предоставляющим разработчикам возможность создавать игры для различных платформ, включая мобильные устройства, компьютеры, игровые консоли и виртуальную реальность. Архитектура Unity состоит из ряда модулей, включающих движок для отображения графики, физический движок, звук, анимацию, сценарии и многое другое.

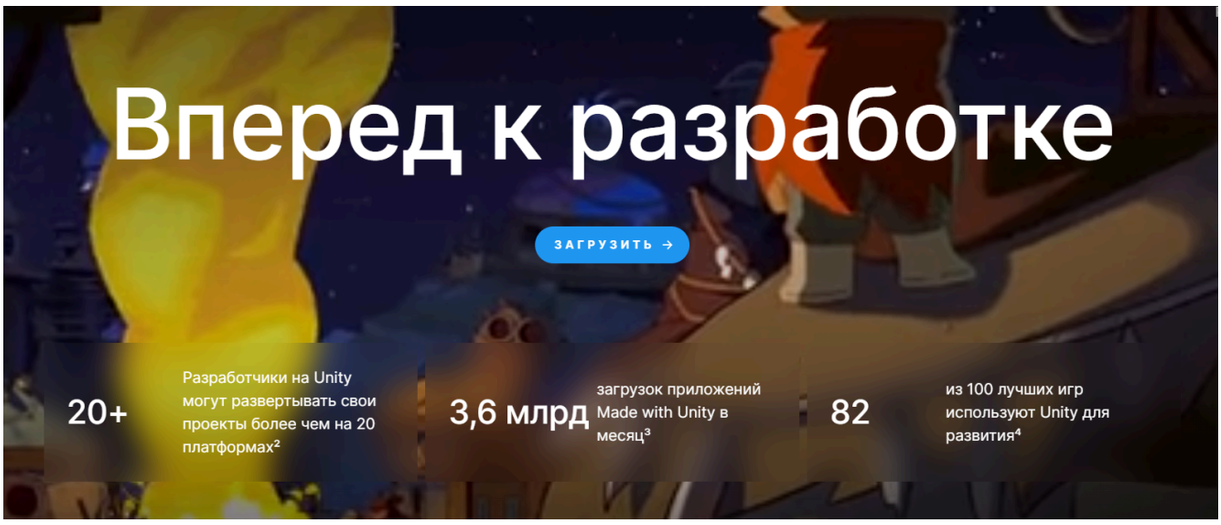

Рисунок 1 − Изображение с официального сайта Unity

1.2 История, версии и достоинства

Unity была выпущена в 2005 году Unity Technologies и с тех пор претерпела значительные изменения и улучшения. Каждая новая версия Unity вносит улучшения в производительность, графику, инструменты разработки и поддержку новых технологий. Достоинства Unity включают в себя:

− Кросс-платформенность, позволяющая разрабатывать игры для различных устройств с минимальными изменениями.

− Богатая библиотека ресурсов и инструментов для разработки игр.

− Простота в использовании и поддержка различных языков программирования, таких как C# и JavaScript.

− Активное сообщество разработчиков и поддержка со стороны компании Unity Technologies.

1.3 Обоснование выбора платформы

Выбор Unity для разработки мобильной игры обоснован его широкими возможностями, удобством в использовании и поддержкой различных платформ. Благодаря кроссплатформенности Unity, разработчики могут сосредоточиться на

создании игрового контента, минимизируя затраты времени и ресурсов на адаптацию игры под различные устройства.

1.4 Анализ существующих аналогов

Среди разнообразных жанров в последнее время стал популярным жанр auto battler, особенностью которого является автоматизация части игрового процесса. Как правило, игрок отвечает за стратегическую составляющую, в то время как непосредственно за боевую систему отвечает искусственный интеллект. Представителями такого жанра являются Dota Underlords, Hearthstone Battlegrounds, Gladiator Guild Manager и многие другие. Данные игры предоставляют разнообразный геймплей со стратегическими элементами в реальном времени.

Рисунок 2 − Изображение игрового процесса Gladiator Guild Manager

ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

2.1 Обоснование необходимости разработки

Мобильные игры, особенно в жанре auto battler с элементами стратегии, представляют собой популярное и востребованное развлечение среди пользователей мобильных устройств. Разработка такого проекта имеет ряд преимуществ:

− Удовлетворение спроса на игровые проекты на рынке мобильных приложений.

− Возможность получения опыта в разработке приложений для мобильных устройств.

− Улучшение знаний в области разработки с использованием принципов ООП.

2.2 Технологии программирования, используемые для решения поставленных задач

Для разработки мобильной игры на платформе Unity используются следующие технологии программирования:

− Язык программирования C#, который является основным языком программирования в Unity и обеспечивает широкие возможности для разработки игровой логики, управления интерфейсом пользователя и взаимодействия с игровым миром.

− Unity Scripting API, который предоставляет разработчикам доступ к различным функциям и возможностям движка Unity для создания игровых объектов, управления анимациями, физикой и другими аспектами игры.

− SQLite, являющаяся легковесной реляционной базой данных. Используется для хранения и управления игровыми данными, такими как прогресс игроков и настройки. SQLite обеспечивает быструю и эффективную работу с данными, что важно для мобильных приложений.

2.3 Связь архитектуры ООП с разрабатываемым программным обеспечением

Архитектура объектно-ориентированного программирования (ООП) тесно связана с разработкой мобильных с помощью Unity. Основные принципы ООП, такие как инкапсуляция, наследование и полиморфизм, позволяют создавать модульный и расширяемый код, что облегчает разработку, поддержку и масштабирование игрового проекта. Эффективная разработка игр с помощью Unity без знаний ООП практически невозможна, поскольку все игровые объекты Unity должны быть обязательно представлены в виде классов. В проекте также реализуются принципы SOLID, такие как принцип единственной ответственности

(SRP), где каждый класс обладает определенной задачей и отвечает только за свою функциональность, и принцип открытости/закрытости (OCP), который позволяет добавлять новую функциональность без изменения существующего кода.

ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ ПРОГРАММЫ

3.1 Описание функций программного обеспечения (программы, приложения) с учетом выбранной темы курсового проекта

Игровые скрипты выполнены как классы, которые наследуются от MonoBehaviour. Игровые уровни представлены в виде “сцен” Unity. Игровые единицы имеют название “юниты”. Навигация по приложению представлена ниже расположенной схемой.

Рисунок 3 − Навигация по приложению

Далее представлены скрипты боевой системы. Класс Health определяет здоровье игровых юнитов (приложение А). Он содержит публичный метод GetHit, через который идет передача урона юниту. После чего информация об уроне обрабатывается и передается в приватный метод DamageScript, который в свою очередь обрабатывает возможные шансы на уворот и возможность получения эффекта, после чего, для подсчета урона с учетом резистов, передает информацию об уроне в ResistScript. После обратного получения информации DamageScript обрабатывает последствия урона. Этот скрипт также содержит метод RestoreHP для корректного восстановления здоровья и SacrificeHp для штрафа в виде убавления здоровья, без соответствующих подсчетов. Класс AttackScript привязан к оружию юнита. Как только родительский объект замечает вхождение коллайдера в зону триггера, вызывается метод Attack, который вызывает анимацию атаки. В ходе анимации, через встроенный в неё UnityEvent, вызывается либо метод

DetectColliders, который после обнаружения противников в радиусе вызывает у них GetHit, либо выпускается снаряд (проджектайл) с помощью метода SendProjectile. В ходе создания снаряда в него закладываются характеристики отправителя и снаряд, имеющий классы ProjectileScript и MovementScript, летит до столкновения с противников, финишем или пока не истечет время его жизни. В слцчае проджектайла используется метод Unity из MonoBehaviour – OnTriggerEnter2D. Юниты и проджектайлы имеют класс передвижения MovementScript, который на каждый Update (каждый фрейм) производит движение объекта в определенном направлении. Для обнаружения вражеского объекта используется метод DetectActivity, который вызывается при вхождении в триггер объекта с определенными критериями.

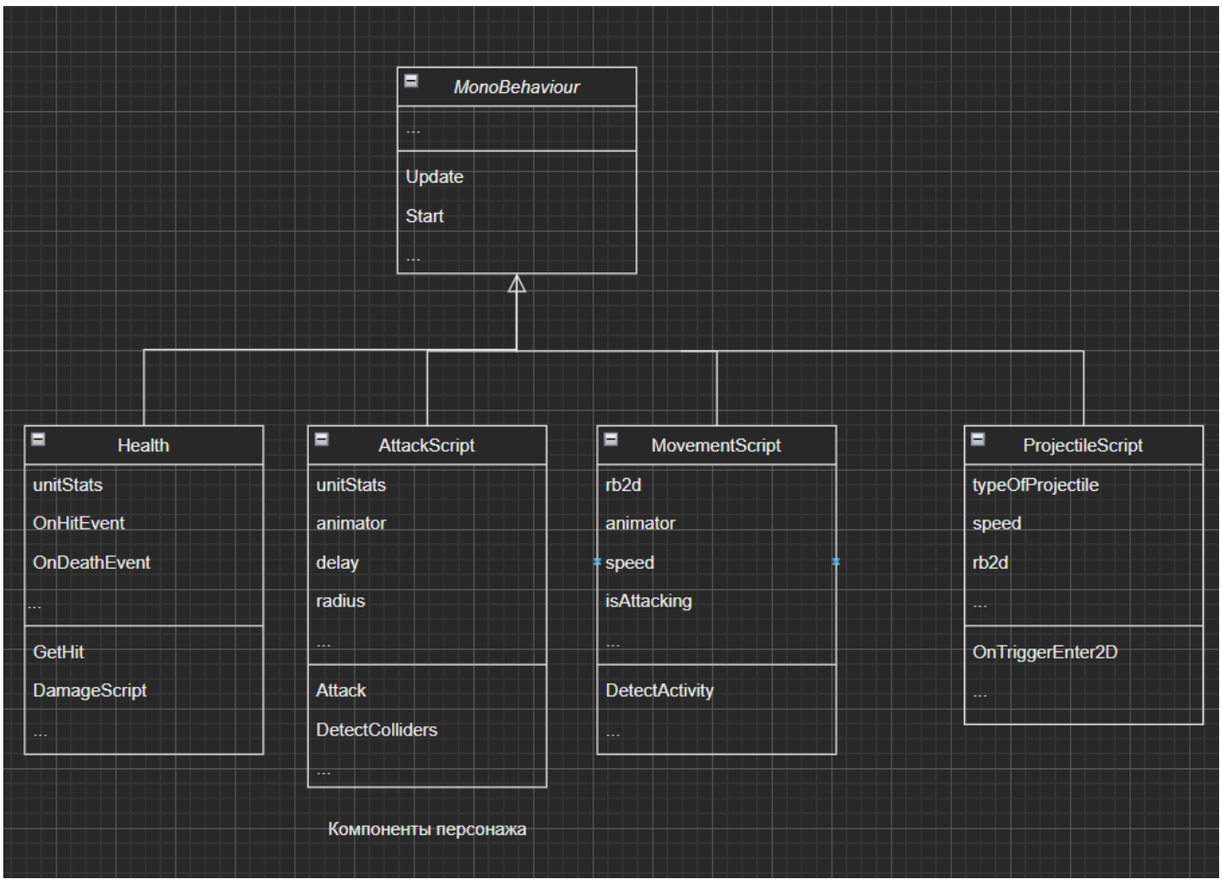

Рисунок 4 – Блок-схема компонентов персонажа

Главным управляющим классом боя является ArenaManager. Он содержит методы загрузки локации и её инициализации LoadLoc, а также функции остановки игры (PauseGame) и ее продолжения (ContinueGame). Также этот класс контролирует набор очков сторонами с помощью методов OnRightDeath и OnLeftDeath. Также идет контроль за поведением в случае победы (Victory) или поражения (Defeat). Для контроля за кулдауном игрока используется класс BattleController. Он использует методы SetCooldown для установления кулдауна (восстановления), CheckCooldown для его проверки и метод StopBattle в случае завершения боя после победы или поражения. Для контроля за поведением противника используется класс EnemyBattleAI. Он выбирает случайный ряд для выставления юнитас помощью SetRandomRow, случайную способность с

помощью SetAbility и случайного юнита с помощью SetRandomUnit. Класс UI_Control нужен для управления элементами пользовательского интерфейса, в том числе и панелью юнитов, использование которой ограничивается BattleController. Передвижение осуществляется с помощью метода SetRow. Метод SpawnButton вызывает появление юнита, обращаясь к классу SpawnScript. Класс SpawnScript отвечает за появление юнитов с обеих сторон. Контроль за прохождением юнитом финиша осуществляет класс FinishScript, содержащий метод OnCollisionEnter2D, который активируется при вхождении объектом вколлайдер финиша. За музыкальное сопровождение отвечает класс MusicManager, который имеет метод SetRandomMusic, включающий случайную звуковую дорожку из доступного ему списка и метод SetMusic, отвечающий за включение и выключение музыки. Класс MobileCamera позволяет на мобильном устройстве android двигать камеру методом Drag и масштабировать размер методом Zoom.

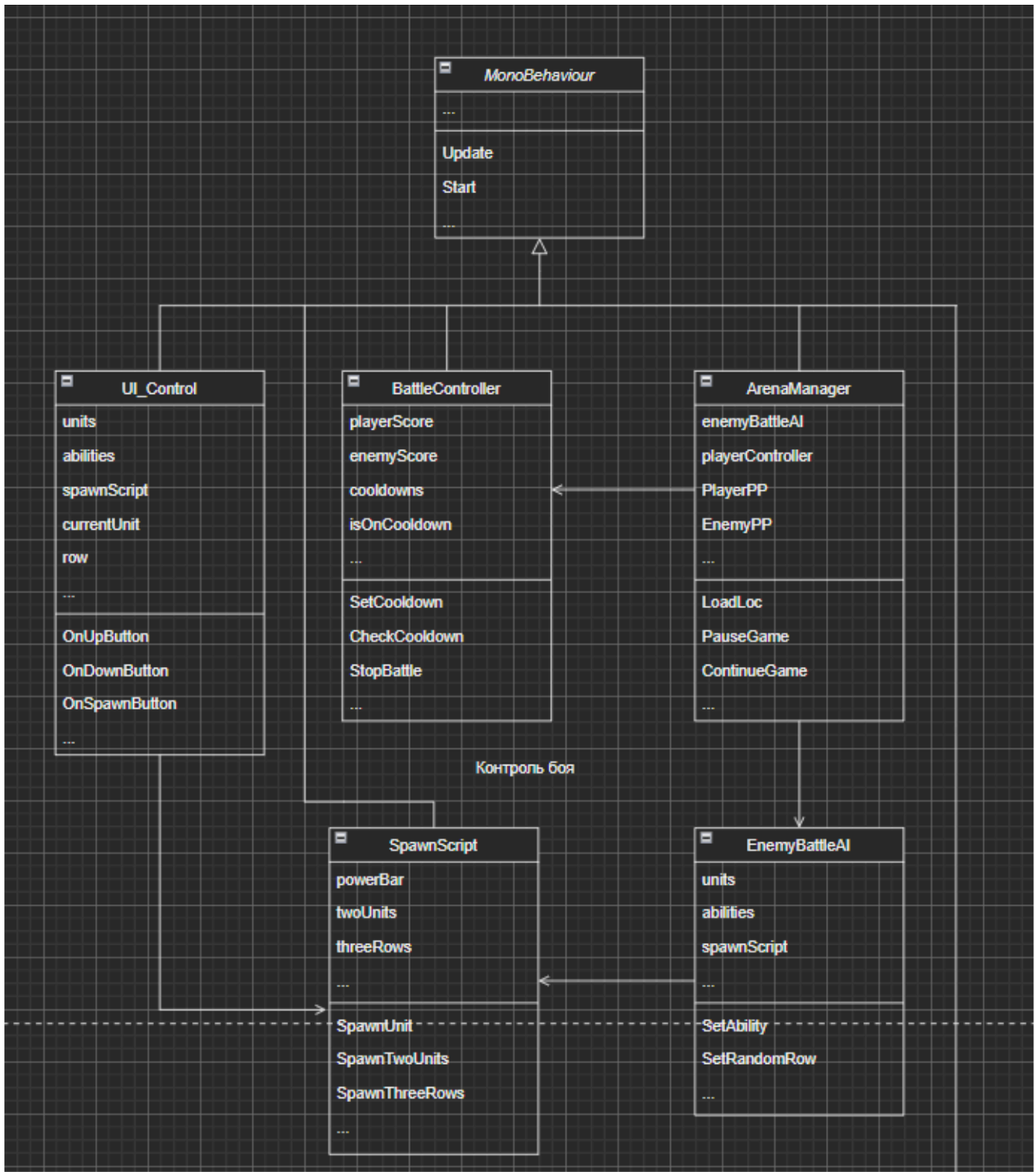

Рисунок 5 − Блок-схема элементов контроля боя

Далее представлены скрипты сцены выбора локаций. Главным управляющии классом является WorldManager. При помощи метода SetMenu он контролирует элементы пользовательского интерфейса, такие как возможность приобрести юнитов, настройки и информацию о локации. Также этот метод осуществляет переход на боевую сцену методом StartBattle и выход в главное меню методом MainMenu. Класс WorldLocationData контролирует локации и их распределение между игровыми фракциями. Метод SetLocation производит стартовую инициализацию локаций. Метод ChangeLoc осуществляет смену локации между фракциями.Метод AddLocToPlayer добавляет определённую локацию к списку локаций игрока. Информация о конкретной локации хранится в классе LocationData. Из методов этого класса можно выделить SetInfo, устанавливающий информацию в классе TransitionData для последующего использования на боевой сцене, метод SetIcons устанавливает изображения кнопки в виде изображений юнитов противника для возможности получения информации о них игроком. Метод CheckNeighbours проверяет, есть ли среди соседних локаций локация игрока. Класс MerchantStoreController производит симуляцию магазина в игре. Класс содержит метод ShowInfo для просмотра информации о юните, BuyUnit для осуществления непосредственного приобретения юнита и SetPlayerUnits для инициализации магазина юнитами соответствующей фракции. Класс BattleScript нужен для отображения меню информации о противнике на выбранной локации. Класс содержит методы Initialize, для стартовой инициализации иконок и информации о противнике, ShowInfo для показа информации, OnChooseButton для обработки события нажатия выбора информации о противнике.

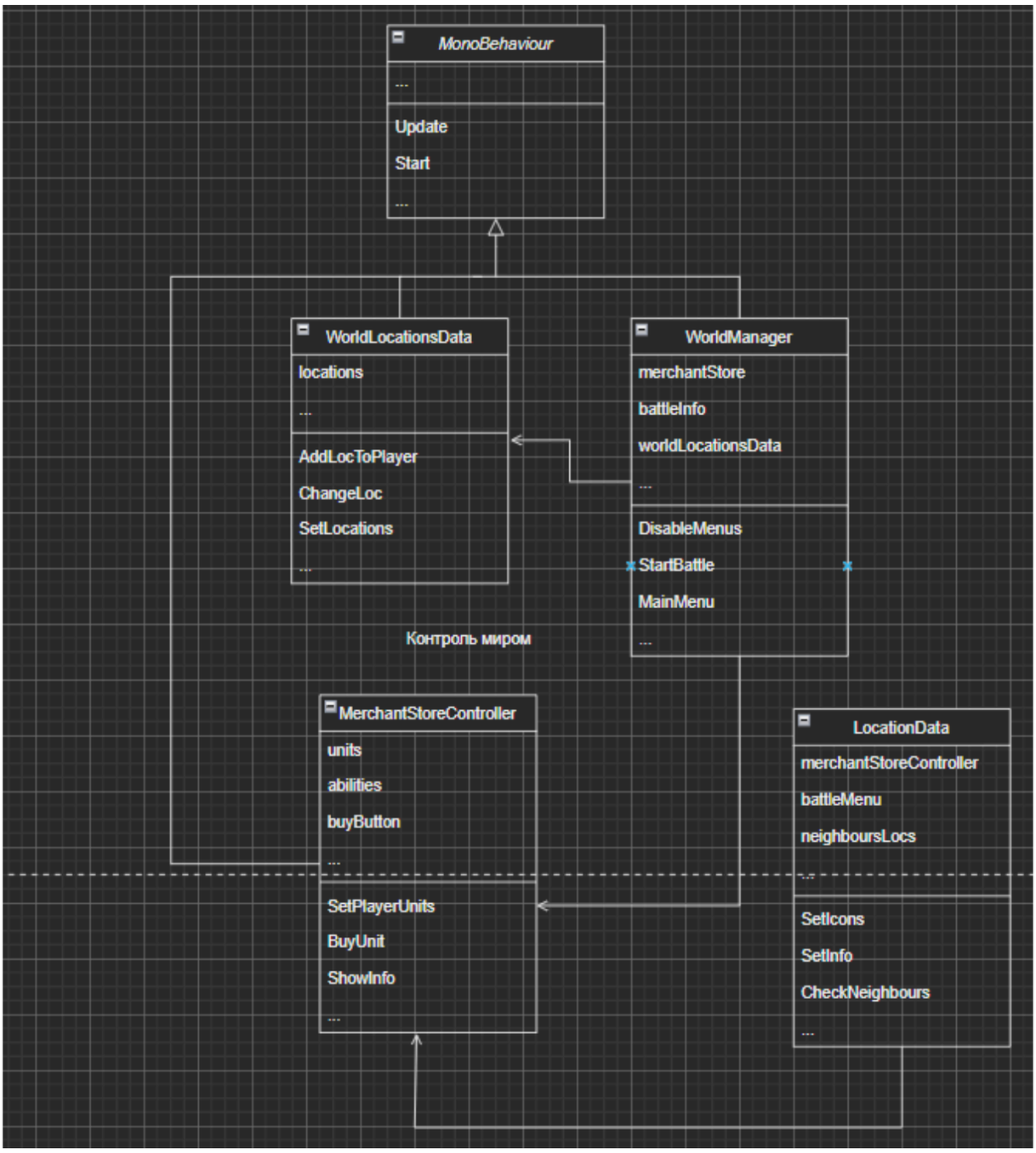

Рисунок 6 − Блок-схема элементов контроля локациями

ОПИСАНИЕ ИСПОЛЬЗОВАНИЯ ПРИЛОЖЕНИЯ

4.1 Меню регистрации

Запуская игру, первым делом игрок попадает в меню регистрации. Оно состоит из полей для ввода имени пользователя, почты и пароля. Меню регистрации связано с базой данной SQLite, позволяющей сохранять и загружать игровой процесс. Меню имеет кнопки регистрации, входа в аккаунт и выхода из игры. Под полем для ввода пароля располагается подсказка с требованиями к паролю. В случае некорректного ввода выводится соответствующая подсказка и сообщение об ошибке. После корректного ввода игрок попадает в главное меню.

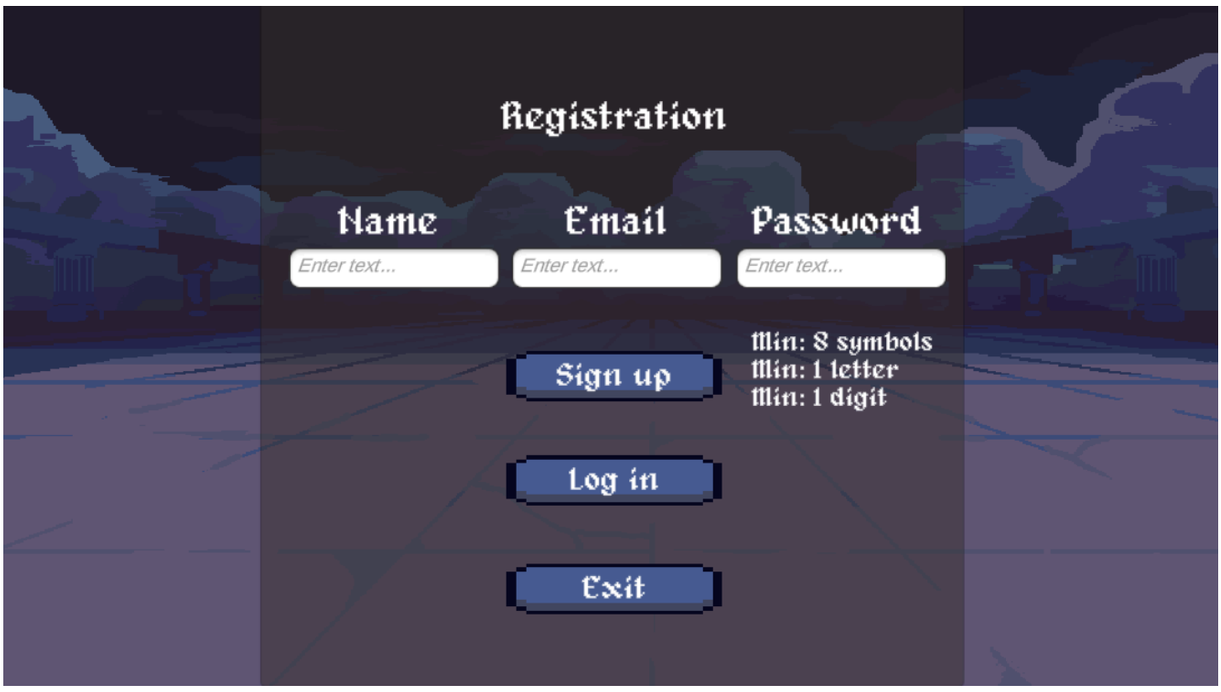

Рисунок 7 − Изображение меню регистрации

4.2 Главное меню

Главное меню содержит панель, содержащую 5 кнопок. Кнопка Exit производит выход из игры. Во время загрузки сцены главного меню при переходе в него через любую другую сцену происходит автосохранение игровых данных.

Рисунок 8 − Изображение главного меню

Кнопка New Game открывает меню выбора стартовой фракции. При нажатии на кнопку фракции появляется описание и изображение соответствующей фракции. Кнопка Close закрывает данное меню. После нажатия

на кнопку Choose происходит сохранение выбора игрока и переход на уровень выбора локации.

Рисунок 9 − Изображение меню выбора фракции

Кнопка Continue необходима для продолжения игровой сессии. При нажатии происходит переход на уровень выбора локации, если уже есть зарегистрированная игровая сессия.

Кнопка Tutorial открывает меню игрового обучения. Оно содержит текстовое описание игровых механик и соответствующее изображение. Навигация происходит с помощью кнопок Next и Back. Кнопка Close закрывает данное меню.

Рисунок 10 − Изображение меню обучения с экрана мобильного телефона Honor X8

Кнопка Settings производит переход на сцену игровых настроек. Данная сцена содержит кнопку включения и отключения музыки, кнопка переключения сложности игрового процесса и кнопка возврата из меню настроек в главное меню.

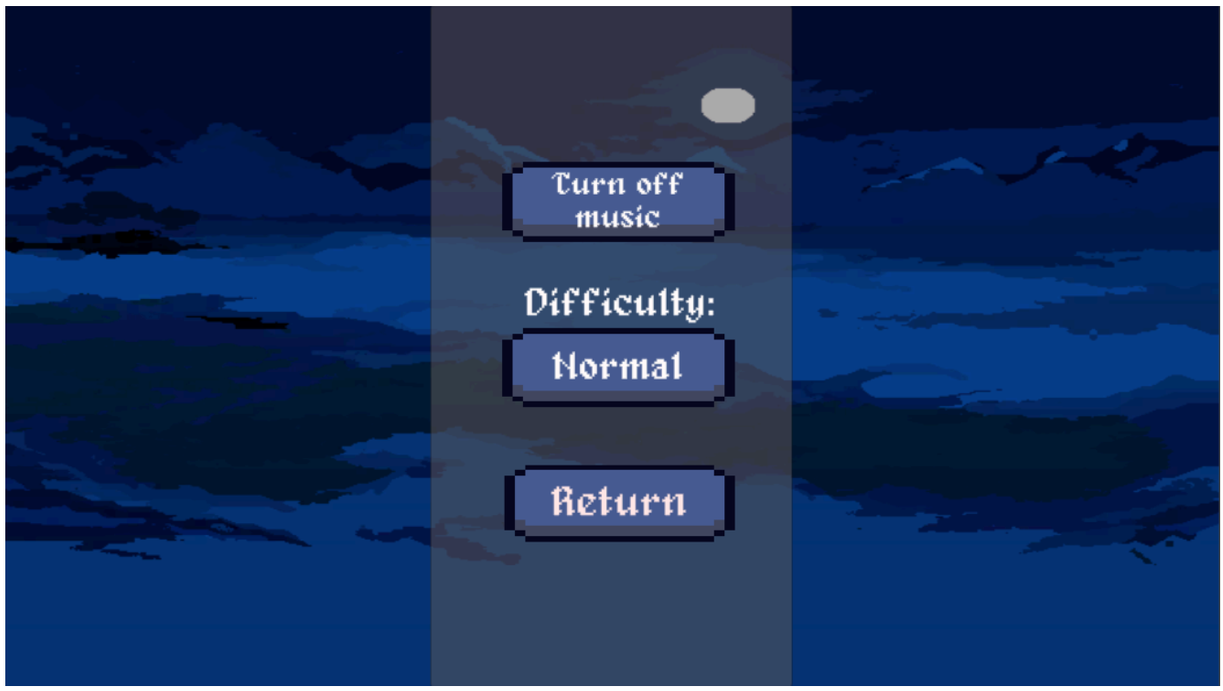

Рисунок 11 − Изображение меню настроек

4.3 Меню выбора локации

Меню выбора локации является одним из стратегических элементов игры. Данная сцена представляет собой карту всех игровых локаций, имеющих цвет соответствующей фракции, являющуюся ее владельцем.

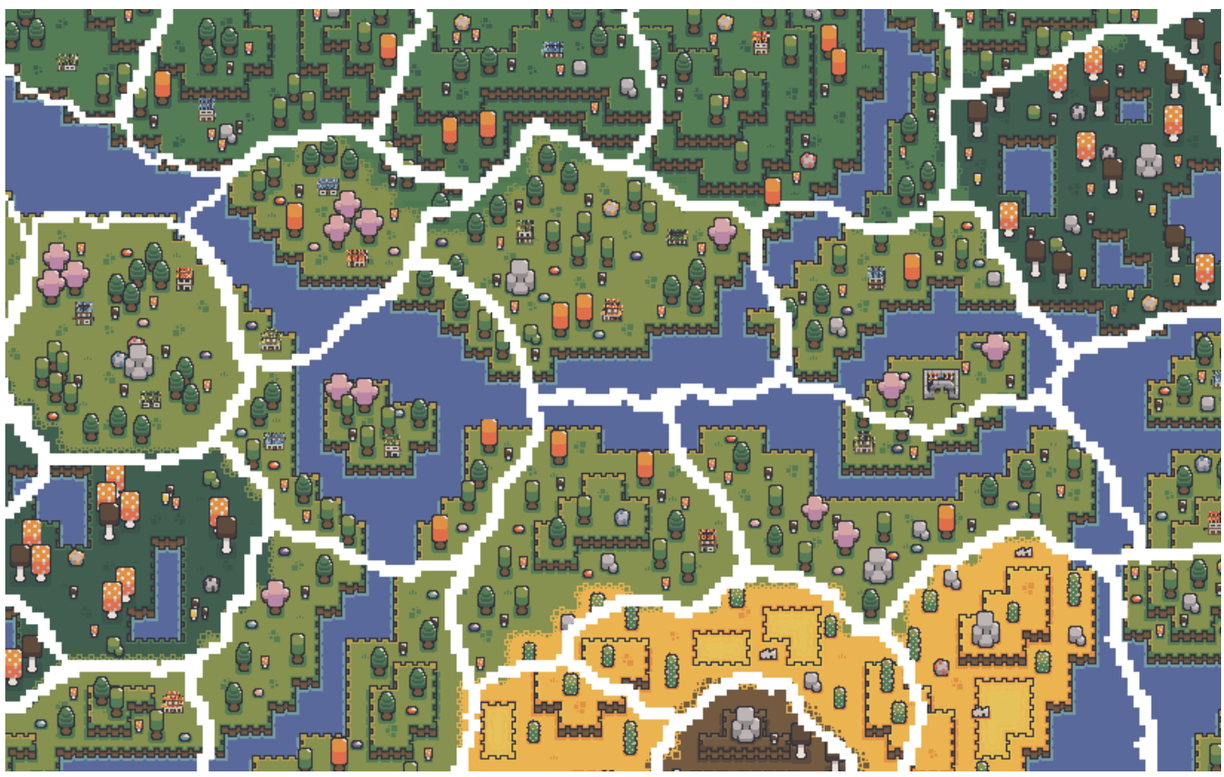

Рисунок 12 − Изображение меню локаций без подсветки

Рисунок 13 − Изображение меню локаций с подсветкой фракций

Перемещение по локации происходит с помощью проведения пальцем по экрану. Также имеется возможность зумирования экрана с помощью касания двух пальцев.

Игрок начинает, имея одну локацию, местоположение и цвет которой зависит от выбранной фракции. Цель игры − взять под контроль все 37 игровых локаций. Чтобы начать бой игрок должен иметь территорию, смежную вражеской локации, на которой игрок хочет начать бой. При двойном нажатии на территорию происходит переход в меню информации о локации. Здесь находится информация о каждом юните противника, о каждой его активной и пассивной способности и награде за победу. Если игрок имеет локацию, смежную вражеской, то появляется кнопка возможности начать бой.

Рисунок 14 − Изображение меню информации о локации

Наверху экрана находиться панель, содержащая кнопки Save, Buy и Main Menu. Кнопка Save отвечает за ручное сохранение игровых данных текущего пользователя. Также при переходе на данную сцену с любой другой происходит автосохранение. Кнопка Buy отвечает за переход к меню торговли, где игрок может потратить игровую валюту на улучшение своей армии. Кнопка Main Menu производит выход в главное меню.

Рисунок 15 − Изображение панели пользовательского интерфейса

Меню торговли содержит ряд кнопок, позволяющих просмотреть информацию о юнитах и способностях, а также купить их. В левом верхнем углу находиться текущее количество игровой валюты в распоряжении игрока. Нажимая

на иконки юнитов или способностей, соответствующая информация появится в правом части экрана. В правой нижней части экрана, рядом с ценой юнита или способности, находиться кнопка приобретения выбранной способности или игровой единицы. При успехе соответствующая иконка появляется среди приобретений игрока.

Рисунок 16 − Изображение меню торговли

4.3 Игровая арена

Игровые сражения происходят на сцене Arena. По левую сторону экрана находиться зона игрока, которую необходимо защищать от оппонента. По правую сторону экрана находиться зона противника, которую необходимо атаковать игроку. Главная цель игрока на этом этапе − провести как можно больше своих юнитов в зону противника и пропустить в свою зону как можно меньше юнитов противника.

Рисунок 17 − Изображение игровой арены

В распоряжении игрока есть приобретенные им юниты и способности. Внизу экрана расположены кнопки, призывающие соответствующих юнитов. После нажатия на кнопку в выбранном ряду появляется игровая единица. После призыва у всех кнопок юнитов начинается перезарядка, пропорциональная игровой силе юнита. Боевая сцена имеет 6 рядов, переключение между которым осуществляется с помощью клика на другой ряд. Текущее положение ряда отображается белыми стрелками.

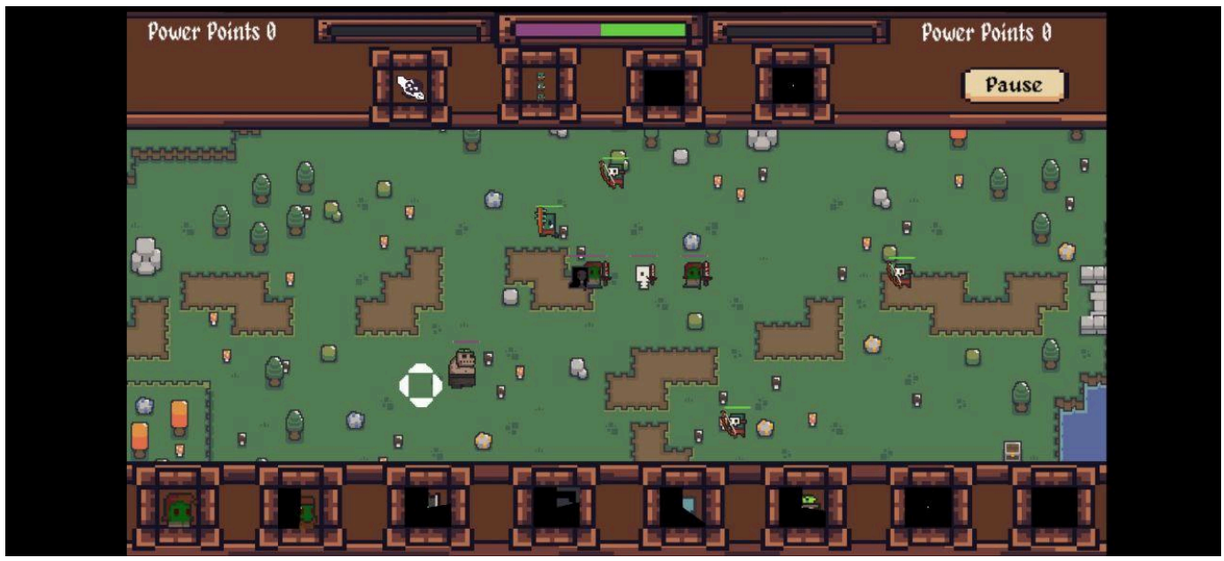

Рисунок 18 − Изображение игрового сражения с мобильного телефона Honor X8

В связи с особенностью жанра auto battler, юниты действуют сами, без помощи игрока. После появления они постепенно перемещаются в сторону зоны оппонента. При столкновении юнитов противоположный сторон они начинают сражение сами, без участия игрока. Каждый тип юнита имеет свою уникальную внешность, анимации, способности и характеристики.

Рисунок 19 − Изображение игрового сражения

Также в распоряжении игрока имеются способности, располагающиеся на верхней панели. Если таковая приобретена, то ее икона появится в одной из рамок. Каждая способность требует определенного количества игровых очков для ее активации. Количество игровых очков игрока представлено левой желтой полоской, а противника − правой желтой полоской. При использовании способностей, как противником, так и игроком, тратится определенная часть их очков и активируется соответствующий эффект. Наверху по центру находятся две полосы цветов фракции игрока и его оппонента. Соотношение этих полос определяет текущий результат сражения. При прохождении юнитом вражеской границы этот юнит исчезает и увеличивает полоску своей фракции пропорционально своей силе. При полном заполнении полоски одной фракции, соответствующая фракция выигрывает, а противоположная − проигрывает. Штрафов за поражение игроком не предусмотрено. В качестве награды за победу игрок получает заявленное количество игровой валюты и соответствующую территорию. После победы или поражения, игрок возвращается на уровень выбора локации.

Рисунок 20 − Изображение верхней панели пользовательского интерфейса

В качестве оппонента выступает искусственный интеллект. В определенные моменты времени он меняет свой текущий ряд, призывает своих юнитов и, если возможно, использует способности. В меню настроек можно выбрать сложность оппонента, которая замедляет или ускоряет его действия.

Рисунок 21 − Изображение игрового сражения

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы по теме “Мобильная игра на платформе Android в жанре auto battler” были достигнуты следующие цели:

1 Проанализированы особенности выбранной платформы программного обеспечения (Unity) и операционной системы (Android) для разработки игры.

2 Обоснована необходимость разработки мобильной игры, учитывая ее популярность среди пользователей и потенциальные возможности для дальнейшего развития.

3 Применены на практике принципы ООП при использовании языка программирования C# и интегрированных инструментов разработки Unity, для создания игрового контента и реализации игровой логики.

В результате работы был создан интересный и увлекательный игровой продукт, который сочетает в себе элементы стратегии и динамичных сражений, предоставляя пользователям возможность погрузиться в захватывающий виртуальный мир.

Разработка мобильной игры на Unity не только позволила овладеть навыками программирования и дизайна игровых элементов, но и предоставила ценный опыт в области проектирования программного обеспечения для мобильных устройств. В целом, выполнение курсовой работы подтвердило важность умения применять знания и навыки в практических проектах и расширило кругозор в области разработки мобильных приложений и использования ООП.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Unity Official Site [Электронный ресурс]. – Режим доступа : https://unity.com/ru – Дата доступа: 01.05.2024.

[2] Unity 2D Game Assets [Электронный ресурс]. – Режим доступа : https://itch.io/game-assets – Дата доступа: 01.05.2024.

[3] ORM SQLite для Unity [Электронный ресурс]. – Режим доступа : https://github.com/robertohuertasm/SQLite4Unity3d – Дата доступа: 01.05.2024.

[4] Использованные игровые ассеты [Электронный ресурс]. – Режим доступа : https://analogstudios.itch.io/fantasy – Дата доступа: 01.05.2024.

[5] Unity Tutorial [Электронный ресурс]. – Режим доступа : https://www.youtube.com/c/sunnyvalleystudio – Дата доступа: 01.05.2024.

[6] Документация Unity[Электронный ресурс]. – Режим доступа : https://docs.unity3d.com/ru/530/Manual – Дата доступа: 01.05.2024.

ПРИЛОЖЕНИЕ А

(справочное)

Листинг 1 – класс для контроля за состоянием игровых единиц

```
public class Health : MonoBehaviour
{

    public UnitStats unitStats;
    [SerializeField]
    float currentHealth, maxHealth;
    [SerializeField]
    Transform emptyBar, HPBar;
    [SerializeField]
    int evadeChance = 0;
    public bool bubble = false;
    [SerializeField]
    float physicalResist,magicalResist;



    public UnityEvent OnHitEvent;
    public UnityEvent OnDeathEvent;

    ArenaManager arenaManager;
    AttackScript attackScript;
    MovementScript moveScript;



    [SerializeField]
    bool isDead = false;

    public SpecialEffect currentEffect =
SpecialEffect.None;

    [SerializeField]
    float effectTimer = 0,tempTimer = 0;
    private void Awake()
    {
        Initialization();

    }
    void Initialization()
    {
        currentHealth = unitStats.health;
        maxHealth = unitStats.maxHealth;
        physicalResist = unitStats.physicalResist;
```

```
        magicalResist = unitStats.magicResist;
        attackScript =
GetComponentInChildren<AttackScript>();
        moveScript = GetComponent<MovementScript>();
        InitializeBuffs();
        InitializeLocBuffs();
        isDead = false;
        effectTimer = 0;
        tempTimer = 0;

        GameObject temp =
GameObject.FindGameObjectWithTag("ArenaManager");
        arenaManager = temp.GetComponent<ArenaManager>();
    }
    private void Update()
    {
        EffectScript();
    }


    public void GetHit(float amount,GameObject
sender,SpecialEffect effect = SpecialEffect.None,int
effectChance = 0)
    {
        if (isDead)
        {
            return;
        }
        if (sender.layer == gameObject.layer)
        {
            return;
        }
        TypeOfDamage typeOfDamage = TypeOfDamage.Physical;
        if (sender.GetComponent<Health>())
        {
            typeOfDamage =
sender.GetComponent<Health>().unitStats.typeOfDamage; //
check if it is weapon
        }
        else if (sender.GetComponent<ProjectileScript>())
        {
             typeOfDamage =
sender.GetComponent<ProjectileScript>().typeOfDamage; //
check if it is projectile
        }
```

```
        if(bubble)
        {
            bubble = false;
            return;
        }
        if (evadeChance > 0)
        {
            int chance = Random.Range(0, 100);
            if (chance <= evadeChance)
            {
                return;
            }
        }
        DamageScript(amount,
typeOfDamage,effect,effectChance);

    }
    public void SacrificeHP(int amount)
    {
        if (isDead)
        {
            return;
        }

        DamageScript(amount, TypeOfDamage.Physical);
    }
    private void DamageScript(float amount,TypeOfDamage
typeOfDamage, SpecialEffect effect = SpecialEffect.None,
int effectChance = 0)
    {

        ResistCount(ref currentHealth,
amount,typeOfDamage);
        SetEffect(effect,effectChance);
        UpdateBar();
        if (currentHealth > 0)
        {
            OnHitEvent?.Invoke();
        }
        else
        {
            OnDeathEvent?.Invoke();
            if (gameObject.layer == 6)
            {
```

```
                arenaManager.OnLeftDeath(unitStats.tier);
            }
            else
            {
                arenaManager.OnRightDeath(unitStats.tier);
            }
            isDead = true;
            Destroy(gameObject);
        }
    }
    void ResistCount(ref float currentHealth,float
damage,TypeOfDamage type)
    {
        float resist = 1;
        if(type == TypeOfDamage.Physical)
        {
            resist = physicalResist;
        }
        else if (type == TypeOfDamage.Magic)
        {
            resist = magicalResist;
        }
        resist = 1 - resist;
        currentHealth -= damage * resist;
    }

    public void RestoreHP(int heal)
    {
        currentHealth += heal;
        if(currentHealth > maxHealth)
        {
            currentHealth = maxHealth;
        }
        UpdateBar();
    }
    void UpdateBar()
    {
        HPBar.localScale = new Vector3(currentHealth /
maxHealth, HPBar.localScale.y, HPBar.localScale.z);
    }
    public void IncreaseHP(int hp)
    {
        maxHealth += hp;
        currentHealth += hp;
    }
```

```
    public void IncreaseHP_Percent(float percent)
    {
        float hpBuff = maxHealth * (percent / 100);
        maxHealth += hpBuff;
        currentHealth += hpBuff;
    }
    void IncreaseResist(ref float resist,float amount)
    {
        resist += amount;
        if (resist > 1)
        {
            resist = 1;
        }
    }
    void SetEffect(SpecialEffect effect =
SpecialEffect.None, int effectChance = 0)
    {
        if(effect != SpecialEffect.None)
        {
            int chance = Random.Range(1,100);
            if(chance < effectChance)
            {
                currentEffect = effect;
                Vector3 pos = transform.position;
                pos += new Vector3(0, 1, 0);
                if (currentEffect == SpecialEffect.Poison)
                {

SpecialAbility.SpawnEffect(arenaManager.poison, pos,
0.5f);
                }
                else if (currentEffect ==
SpecialEffect.Fire)
                {

SpecialAbility.SpawnEffect(arenaManager.fire, pos, 0.5f);
                }
                else if (currentEffect ==
SpecialEffect.Frostbite)
                {

SpecialAbility.SpawnEffect(arenaManager.frostbite, pos,
0.5f);
                }
```

```
                effectTimer = 5;

            }
        }

    }

    void EffectScript()
    {
        if(currentEffect == SpecialEffect.None)
        {
            return;
        }
        effectTimer -= Time.deltaTime;
        if(effectTimer <= 0)
        {
            currentEffect = SpecialEffect.None;
            effectTimer = 0;
            moveScript.IsFreezed = false;
            attackScript.IsFreezed = false;
            return;
        }
        if(currentEffect == SpecialEffect.Poison ||
currentEffect == SpecialEffect.Fire)
        {
            tempTimer += Time.deltaTime;
            if (tempTimer >= 1)
            {
                DamageScript(5, TypeOfDamage.Physical);
                tempTimer = 0;
            }
        }
        if(currentEffect == SpecialEffect.Frostbite)
        {
            moveScript.IsFreezed = true;
            attackScript.IsFreezed = true;
            tempTimer += Time.deltaTime;
            if (tempTimer >= 1)
            {
                DamageScript(2, TypeOfDamage.Physical);
                tempTimer = 0;
            }
        }
    }
```

```
    void InitializeBuffs()
    {
        List<SpecialAbilities> abilities;
        if (gameObject.CompareTag("Right"))
        {
            abilities = TransitionData.enPassives;
        }
        else
        {
            abilities = TransitionData.plPassives;
        }


if(abilities.Contains(SpecialAbilities.AdvancedArmor))
        {
            IncreaseHP(20);
            IncreaseResist(ref physicalResist, 0.2f);
        }
        if
(abilities.Contains(SpecialAbilities.LeatherArmor))
        {
            IncreaseHP_Percent(20f);
        }
        if
(abilities.Contains(SpecialAbilities.Insensivity))
        {
            IncreaseHP_Percent(15f);
            IncreaseResist(ref magicalResist, 0.3f);
        }
        if (abilities.Contains(SpecialAbilities.Carapace))
        {
            IncreaseHP(10);
            IncreaseResist(ref physicalResist, 0.2f);
            IncreaseResist(ref magicalResist, 0.2f);
        }

    }

    void InitializeLocBuffs()
    {
        Fraction frac;
        if (gameObject.CompareTag("Right"))
        {
            frac = TransitionData.enemyFraction;
        }
```

```
        else
        {
            frac = TransitionData.playerFraction;
        }
        LocType loc = TransitionData.locType;

        //Humans
        if(frac == Fraction.Human)
        {
            if (loc == LocType.Forest || loc ==
LocType.Mountains)
            {
                IncreaseHP(5);
                IncreaseResist(ref physicalResist, 0.1f);
            }
        }

        //Orcs
        if (frac == Fraction.Orc)
        {
            if (loc == LocType.Forest || loc ==
LocType.Mountains || loc == LocType.Mountains)
            {
                IncreaseResist(ref magicalResist, 0.1f);
            }
            else if (loc == LocType.Desert || loc ==
LocType.Snow || loc == LocType.Hell)
            {
                currentHealth -= 5;
                maxHealth -= 5;
            }
        }

        //Undead
        if (frac == Fraction.Undead)
        {
            if (loc == LocType.Snow)
            {
                IncreaseResist(ref physicalResist, 0.1f);
            }
        }

    }
}
```