|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ** | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»** | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Институт информационных технологий и управления в технических системах | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| *(название института полностью)* | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Кафедра/департамент | | | | | | «Информационные системы» | | | | | | | | | | | | | | | | | | | | | | | | |
| *( наименование кафедры/департамента полностью)*  09.03.03 Прикладная информатика | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| *(код и наименование направления подготовки/специальности)*  Геоинформационные системы | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| *(наименование профиля/специальности)* | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **КУРСОВАЯ РАБОТА / КУРСОВОЙ ПРОЕКТ** | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| по дисциплине | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Объектно-ориентированное програмиирование** | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| *(наименование дисциплины)* | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| на тему | | | |  | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Выполнил: обучающийся | | | | | | | | | | | | | | |
| группы: | | | | | | ПИ/б-18-1-о | | | | | | | | |
|  | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | |
| *(инициалы, фамилия)* | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | |
| « |  | | » | |  | | | 20 | | 20 | | г. | | | |
|  | | | | | | | | | | | | | | | |
| Научный руководитель: | | | | | | | | | | | | | | |
| **д** | | | | | | | | | | | | | | |
| *(инициалы, фамилия)* | | | | | | | | | | | | | | |
| « |  | | | » | | |  | | 20 | | 20 | | г. | | | | |
|  | | | | | | | | | | | | | | | |
| Оценка | | | |  | | | | | | |  | | | | | | | | | | | | | | | |
|  | | | | | | | | | | |  | | | | | | | | | | | | | | | |
| « |  | » | |  | | 20 | 20 | г. |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Севастополь | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | | | | | | | | | | | 20 | | |  | | | | | | | | | | | |

# АННОТАЦИЯ

В данном документе описывается разработка программы, выполненная в соответствии с постановкой задачи на курсовое проектирование по дисциплине «Объектно-ориентированное программирование». Пояснительная записка содержит описание этапов разработки проекта, выполненного в соответствии с техническим заданием. Разработка модели была выполнена с использованием объектно-ориентированного подхода. Также документ содержит программную реализацию построенной модели: описание алгоритмов, реализации, интерфейса пользователя и текст программы в приложении.

СОДЕРЖАНИЕ

[АННОТАЦИЯ 1](#_Toc43396853)

[ВВЕДЕНИЕ 3](#_Toc43396854)

[1 СПИСОК ИСПОЛНИТЕЛЕЙ 5](#_Toc43396855)

[2 ПОСТАНОВКА ЗАДАЧИ 6](#_Toc43396856)

[2.1 Цель разработки 6](#_Toc43396857)

[2.2 Описательная постановка задачи 6](#_Toc43396858)

[2.3 Ограничения, условия выполнения и функционирования 7](#_Toc43396859)

[3 РАЗРАБОТКА МОДЕЛИ 8](#_Toc43396860)

[3.1 Построение иерархии классов 8](#_Toc43396861)

[3.3 Построение информационной модели 9](#_Toc43396862)

[3.4 Описание жизненного цикла программы и одного из объектов 10](#_Toc43396863)

[3.5 Диаграмма переходов состояний 11](#_Toc43396864)

[3.6 Диаграмма потоков данных и действий 13](#_Toc43396865)

[4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 15](#_Toc43396866)

[4.1 Обоснование выбора языка программирования 15](#_Toc43396867)

[4.2 Общее описание взаимодействия программных модулей 16](#_Toc43396868)

[4.3 Описание реализации отдельных классов и их методов 16](#_Toc43396869)

[4.3.1 Класс Tower 16](#_Toc43396870)

[4.3.2 Класс Mobs 17](#_Toc43396871)

[4.3.3 Класс Enemy 17](#_Toc43396872)

[4.3.4 Класс Heroes 18](#_Toc43396873)

[4.4 Интерфейс пользователя 18](#_Toc43396874)

[4.5 Критерии качества программной системы 26](#_Toc43396875)

[ВЫВОДЫ 28](#_Toc43396876)

[ПЕРЕЧЕНЬ ССЫЛОК 30](#_Toc43396877)

[ПРИЛОЖЕНИЕ А 31](#_Toc43396878)

# ВВЕДЕНИЕ

Целью курсового проекта является закрепление знаний в области ООП программирования на примере создания игры.

В жизни современного человека игры занимают неотъемлемую часть, особенно в детский и юношеский этапы. В детстве каждый хотел быть каким-то сверхъестественным существом, рыцарем, королём или просто хотел обладать каким-то необычным даром.

Но реальность сурова и все эти наши мечты мы можем воплотить только в игре, можем окунуться в эту желанную атмосферу. Во взрослой жизни игры это один из вариантов отдыха, чтобы абстрагироваться от рутины. Каждому человеку необходимо уметь расслабиться и игры как никто могут в это помочь.

Пояснительная записка содержит описание программы «Охотник», его разработки и программного кода. Представлено графическое отображение функциональности программы в виде UML схем.

Для достижения цели в ходе проектирования должны быть выполнены следующие задачи:

* постановка задачи;
* проектное решение;
* программная реализация.

В первом разделе указана цель разработки, описание постановки задачи, ограничения и условия выполнения программы. Второй раздел посвящён выбору объектов игры, построения иерархии классов и информационной модели, описание жизненного цикла игры, содержит диаграммы. Третий раздел содержит выбор языка программирования, общее взаимодействия программных модулей, описание реализации основных классов и их методов, описание интерфейса пользователя. Необходимо указать заключения, которое содержит умозаключения о проделанной работе по разработке компьютерной игры и её применении.

# 1 СПИСОК ИСПОЛНИТЕЛЕЙ

Данный курсовой проект был выполнен студентами Севастопольского государственного университета Джинория Д.И., Шведенко А.С. и Николенко В.В..

Вместе был выбран жанр игры, её цель и путь достижения этой цели. Также было выполнено описание игры, построение иерархии классов, на основе чего было сделано разделение заданий для исполнителей проекта.

Джинория Д.И. отвечал за создание класса Enemy, Heroes и Mobs, а

также за создание всех классов наследников. За время разработки сделал систему нанесения урона.

Шведенко А.С. создал класс Tower, а также разрабатывал класс Heroes и Mobs.Занимался разработкой диаграмм классов прецедента.

Николенко В.В. выполнила разработку класса Map, сделала систему появления врагов (WaveSpawner). Кроме этого занималась разработкой графического интерфейса и созданием моделей.

Совместно была написана пояснительная записка.

# 2 ПОСТАНОВКА ЗАДАЧИ

# 2.1 Цель разработки

Цель выполнения курсовой работы по дисциплине "Объектно-ориентированное программирование" – заключается в проектировании и программной реализации игры, в которой в соответствии с вариантом задания реализуются иерархия классов, процессы создания и взаимодействия объектов, что позволит закрепить на практике теоритические знания объектно ориентированного программирования.

Тематика данной курсовой работы – игра в жанре «Tower Defence».

# 2.2 Описательная постановка задачи

Взаимодействие пользователя с игрой происходит с помощью мыши и клавиатуры. При этом игра обрабатывает взаимодействия и предоставляет все соответствующие им возможности.

Цель игры представляет собой защиту своей территории от врагов.

В игре пользователю предоставляется возможность строить башни, управлять героем, вызывать метеоритный дождь и подмогу.

Существует несколько типов для башен и врагов.

Для башен: лучники (быстро атакуют, но наносят мало урона), маги (медленно атакуют, наносят много урона), бомбардировщики (очень медленно атакуют, наносят много урона в радиусе падения снаряда) и рыцари (наземные войска, сдерживают врагов).

Для врагов: орки (средняя скорость, маленький урон), рыцари (медленные, бронированные, средний урон) и дракон (летающий, много здоровья, большой урон)

# 2.3 Ограничения, условия выполнения и функционирования

Игра состоит из нескольких уровней. На каждом уровне будет своё расположение башен. Здоровье уровня отнимается, когда враг доходит до конца карты.

За убийство врагов начисляются монеты. Герой может передвигаться в пределах всей карты.

Враги ограничены в передвижении по карте, передвигаться должны только по тропинке.

# 3 РАЗРАБОТКА МОДЕЛИ

# 3.1 Построение иерархии классов

Данная диаграмма классов была построена на этапе проектирование программы и отражает общий взгляд. Диаграмма классов представлена на рисунке 3.1.

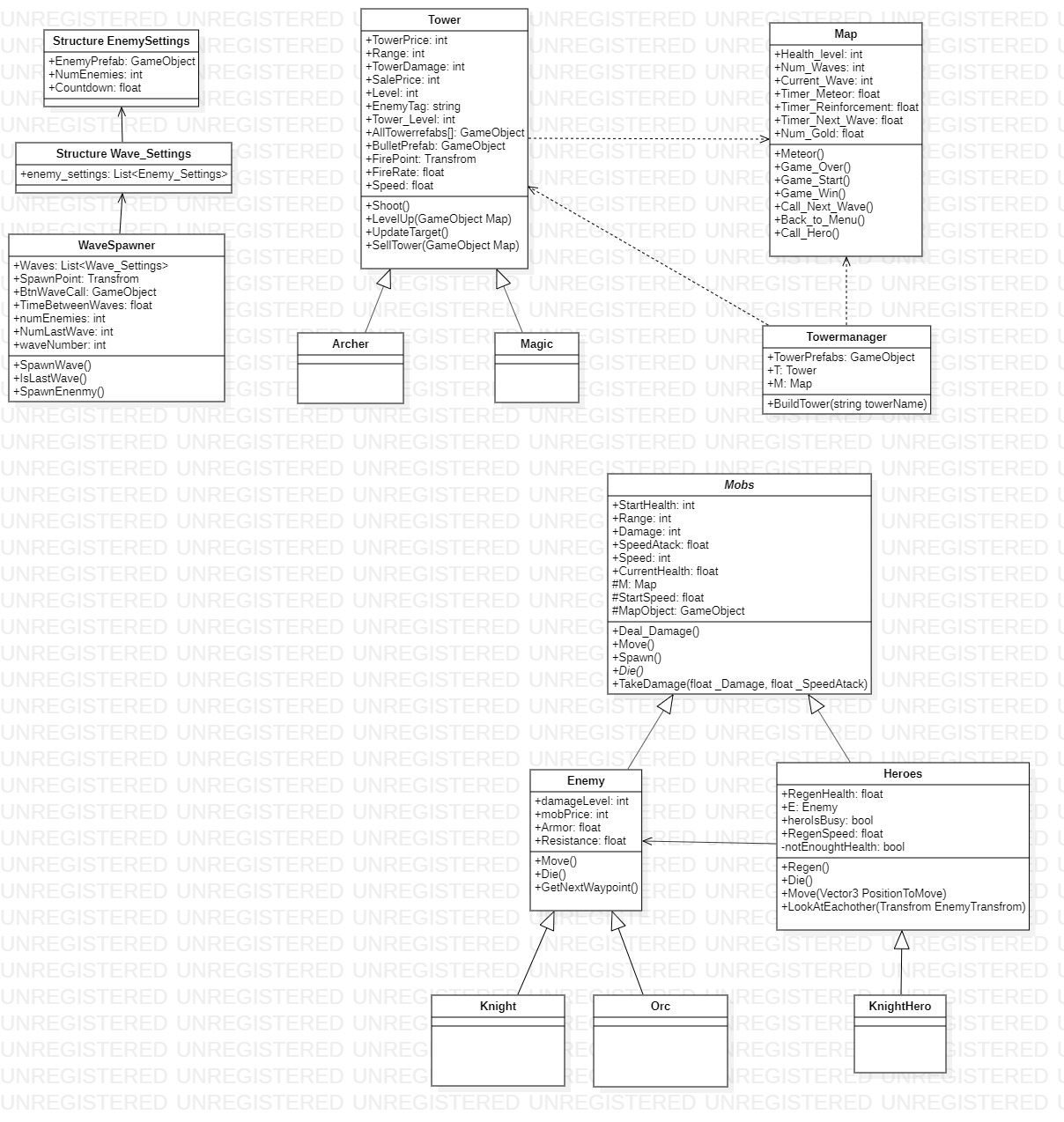


Рисунок 3.1 – Диаграмма классов

Существует два основных класса Tower и Mobs. Класс Tower отвечает за постройку, улучшение и продажу башен, с помощью которых, игрок защищается от противников. От класса Tower наследуется два класса: Magic и Archer. Классы Enemy и Heroes наследуются от класса Mobs. E\_Knight и Orc наследуются от класса Enemy, а KnightHero является наследников класса Heroes.

# 3.3 Построение информационной модели

Построение информационной модели основывается на анализе иерархии классов и предметной области. Представлена на рисунке 3.2.

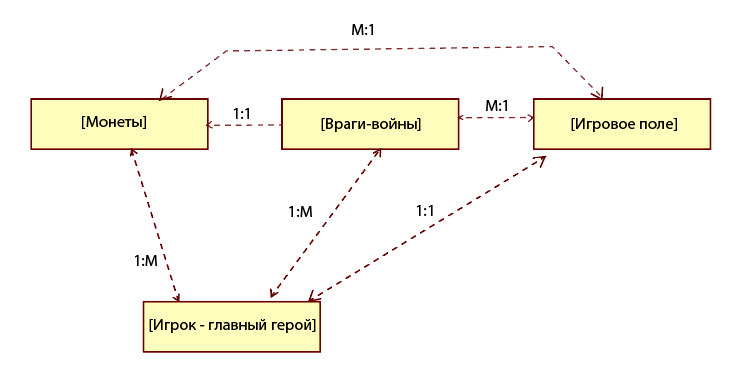


Рисунок 3.2 – Информационная модель

Описание информационной модели:

Игровое поле, существует в единственном числе и знает о всех объектах, которые располагаются на нём: один – персонаж главного героя, много – врагов двух видов, монеты, которые либо выпадают из убитого врага, либо лежат на игровом поле.

Персонаж главного героя один на игровом поле, взаимодействует со многими врагами- воинами и многими монетами, которые может подбирать.

Враги воины, существуют на карте, в множественном количестве и влияют на одного персонажа главного героя, из каждого врага может выпасть одна монета после его смерти.

Монеты, лежат на карте в множественном количестве, либо одна монета появляется на карте после смерти одного врага. С ними может взаимодействовать один персонаж главного героя.

# 3.4 Описание жизненного цикла программы и одного из объектов

Описание жизненного цикла программы в целом основано на созданной программе, изображено на рисунке 3.3. Описание жизненного цикла персонажа главного героя изображено на рисунке 3.4.

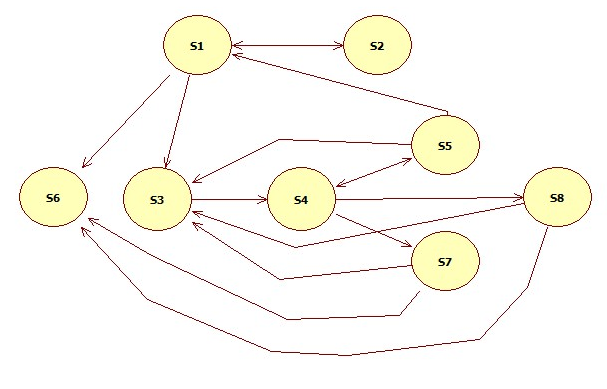


Рисунок 3.3 – Жизненный цикл программы

Описание жизненного цикла программы:

S1 - вход в программу, вызов главного меню;

S2 - выбор уровня;

S3 - начало игрового процесса;

S4 - игровой процесс;

S5 - меню паузы, остановка игрового процесса;

S6 - выход из игры;

S7 - меню, вызываемое при поражении игрока в игре;

S8 - меню, вызываемое при победе игрока в игре.

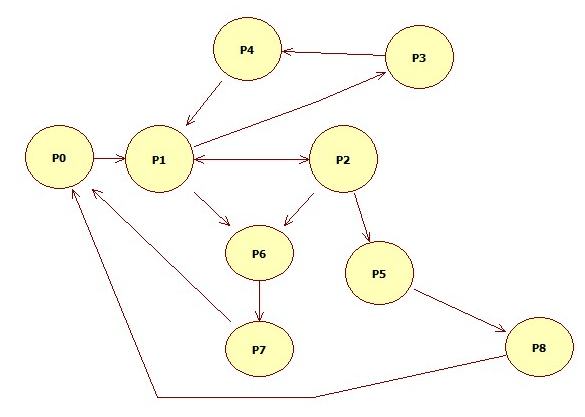


Рисунок 3.4 – Жизненный цикл объекта персонажа главного героя

Описание жизненного цикла объекта героя:

P0 - начальное состояние объекта, ожидание действий игрока;

P1 - персонаж стоит на месте;

P2 – передвижение по карте персонажа;

P3 - выбор цели;

P4 - атака врага, нанесение ему урона;

P5 - при убийстве врага увеличивается счётчик собранных монет;

P6 – объект героя игрока принимает урон от врагов;

P7 – смерть героя в игре от урона врагов;

P8 – победа игрока в игре, когда игрок убил последнего врага.

# 3.5 Диаграмма переходов состояний

Диаграмма переходов состояний отражает объект класса врага и изображена на рисунке 3.5.

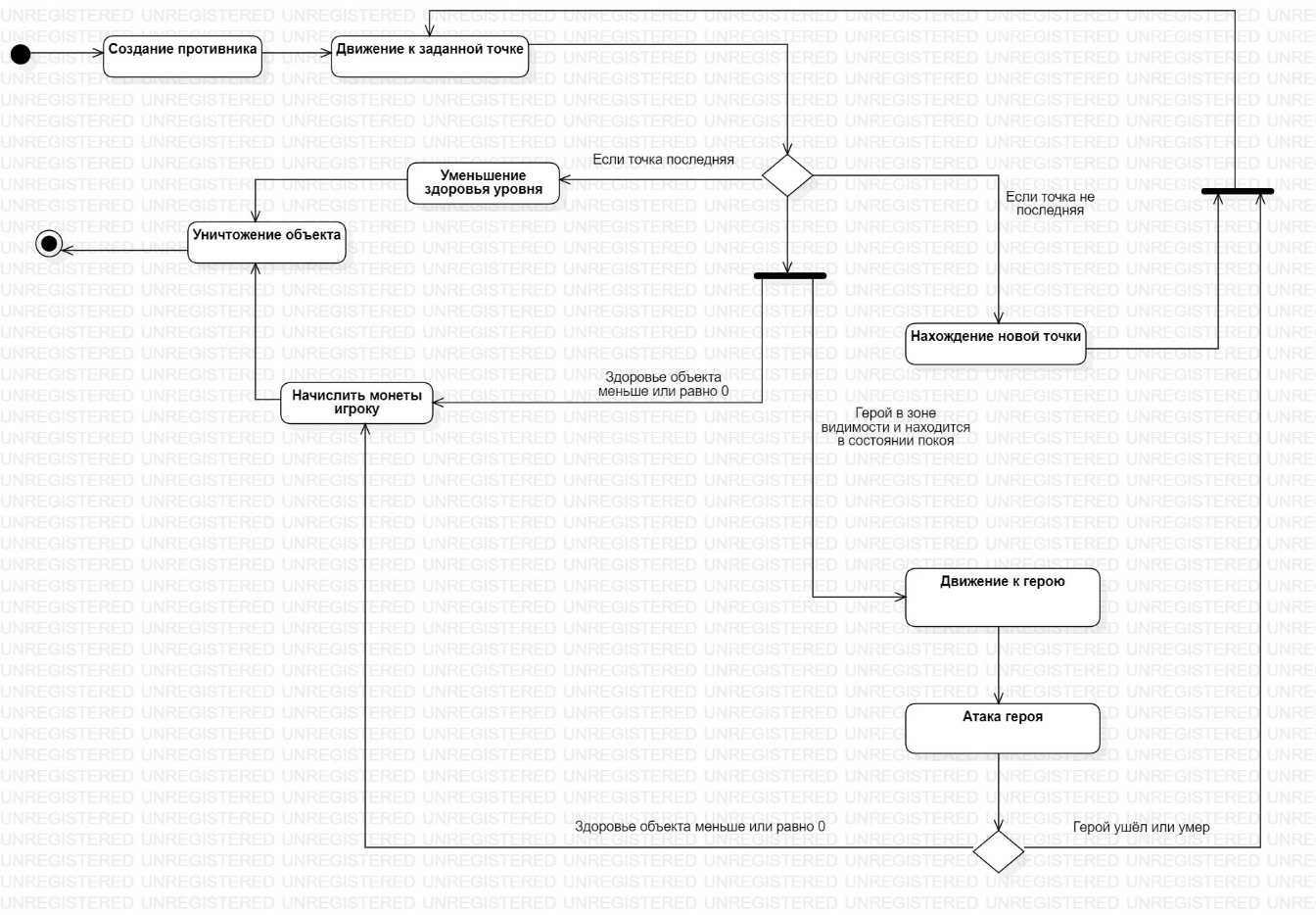


Рисунок 3.5 – Диаграмма переходов состояний объекта врага

Описание диаграммы: Начальное состояние объекта характеризуется его инициализации на игровом поле, сразу после этого объект переходит в состояние движения. Объект ищет ближайшую контрольную точку и движется к ней. При приближении врага к заданной точке:

1. если точка не последняя, он движется к ней, а затем опять ищет новую;
2. если точка последняя и враг к ней приблизился, то он умирает и отнимает жизнь уровня, после чего, объект уничтожается.

Когда здоровье противника становится меньше, либо равно 0, игроку начисляются монеты и объект удаляется с игрового поля.

При встрече врага с героем, который находится в состоянии покоя, враг подойдёт к нему и начнёт его атаковать:

1. если полоска здоровья у врага станет равна 0, игрок получит монеты, а противник умрёт;
2. если герой начнёт движение или умрёт, объект продолжит движение по своему маршруту.

# 3.6 Диаграмма потоков данных и действий

Диаграмма потоков данных и действий представлена на рисунке 3.6.

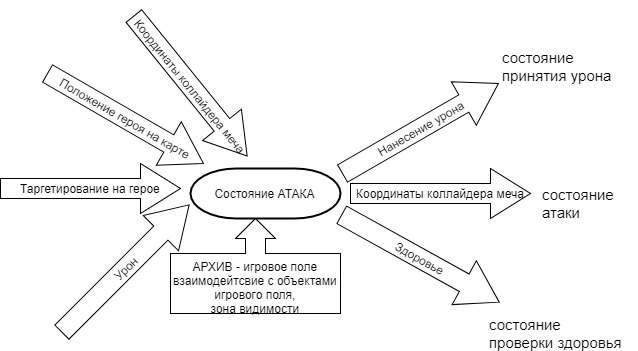


Рисунок 3.6 – Диаграмма потоков данных и действий

Описание:

Рассматривается состояния атаки объекта класса врага воина. Урон наносится при соприкосновении коллайдера меча объекта с коллайдером объекта персонажа главного героя, поэтому необходимо принять параметры коллайдера меча, его координаты и затем передать измененные параметры, для дальнейшей обработки нанесения урона. Так что для этого состояния необходимо знать положение объекта класса героя на карте, потому что атака возможно только при сближении объектов. Необходимо сообщить данному состоянию цель, то есть объект персонажа главного героя и количество наносимого урона. А передать из этого состояние параметры текущего здоровья объекта – в состояние проверки здоровья и урон, который поступает в состояние принятие урона, который он нанёс объекту главного героя. Архивом выступает сам игровой мир, и зона видимости объекта, в которой он может переходить в состояние атаки.

# 4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

# 4.1 Обоснование выбора языка программирования

Для реализации программы была выбрана среда разработки Unity, в связи с тем, что Unity предоставляет разработчику широкий набор инструментов для создание игровых программ, а также существенно упрощает разработку. Unity доступен для освоения, и проектирование учебных, а также не коммерческих программ. Из доступных языков программирования был выбран язык С#, так как является Си подобным, а значит прост в изучении.

C# — язык, основанный на современной объектно-ориентированной методологии проектирования. Этот язык программирования принял в себя много положительных черт своих предшественников - Delphi, C++, Java и т.д. При этом из языка программирования С# изъяты проблемные алгоритмы, которые показали свою нестабильность в перечисленных языках программирования, например, так, C# не поддерживает множественное наследование классов (в отличие от C++). Так как целью курсового проекта является закрепление навыков применения объектно-ориентированного подхода, то был выбран язык программирования C#, который претендует на подлинную объектную ориентированность. В данном языке была унифицирована система типов, теперь каждый тип рассматривается как объект. Также важными особенностями языка С# являются простота и автоматическое управление памятью.

Кроме того, данный язык спроектирован так, чтобы дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы, в этом проявляется его эффективность. Важными критериями для выбора данного языка также являются сборка мусора автоматически освобождает память, занятую уничтоженными и неиспользуемыми объектами; обработка исключений дает структурированный и расширяемый способ выявлять и обрабатывать ошибки; строгая типизация языка не позволяет обращаться к неинициализированным переменным, выходить за пределы массива или выполнять неконтролируемое приведение типов. Это упрощает процесс написания программы, позволяет больше сосредоточиться на реализации модели, чем на устранение ошибок в проблемных алгоритмах.

# 4.2 Общее описание взаимодействия программных модулей

Данная программа состоит из файлов (скриптов), в каждом из которых содержится одноимённый класс со всеми определёнными полями и методами, спрайтов (изображений), сцен, префабов, анимаций. Движок Unity обеспечивает взаимодействие данных модулей при выполнении программы.

Классы, описывающие врагов как отдельный игровой объект, соединяются посредством композиции, посредством интерфейса, каждое состояние врага описывается порождением объекта соответствующего состояния, в которое переходит агрегирующий их объект.

# 4.3 Описание реализации отдельных классов и их методов

# 4.3.1 Класс Tower

Данный класс является одним из основных и отвечает за башни, которые помогают герою убивать врагов. Данный класс наследуется от класса MonoBehaviour. Он содержит общие для наследников переменные и методы.

public class Tower : MonoBehaviour

Класс Tower содержит четыре метода:

private Shoot ();

Данный метод отвечает за снаряд, который вылетает из башни и при столкновении с врагом наносит ему урон.

public LevelUp (GameObject Map);

Эта функция предназначена для улучшения башни, когда игрок повышает уровень башни, её урон повышается.

public UpdateTarget ();

В этом методе происходит отслеживание ближайшего врага.

public SellTower (GameObject Map);

Продажа башни реализуется при помощи данного метода.

# 4.3.2 Класс Mobs

Этот класс также является основным и наследуется от

MonoBehaviour.

public abstract class Mobs : MonoBehaviour

Он содержит три метода:

public Deal\_Damage (float \_Damage, float \_speedAtack);

Метод необходим для нанесения урона.

public Die ();

Когда здоровье противника или героя становится , вызывается функция Die (), при помощи которой объекты исчезают из игры.

public TakeDamage (float Damage);

Если врагу или герою наносится урон, то вызывается данная функция, которая подсчитывает количество нанесённого урона.

# 4.3.3 Класс Enemy

Класс Enemy наследуется от класса Mobs и отвечает за врагов.

public class Enemy : Mobs

Данный класс имеет три метода:

public GetNextWaypoint ();

При помощи данного метода враг строит траекторию, по которой он пытается дойти до заданной конечной точки.

public Move ();

Метод Move () отвечает за перемещение врага по игровому полю.

public Die ();

Если противник умирает, вызывается метод Die (), после чего враг умирает и игроку начисляются монеты, за которые игрок строит и улучшает башни.

# 4.3.4 Класс Heroes

Класс Heroes наследуется от основного класса Mobs и предназначен для героя, который помогает игроку защититься от врагов.

public class Heroes : Mobs

Данный класс содержит четыре метода:

public Die();

Когда здоровье у героя заканчивается,он умирает при помощи метода Die();

public void Move (UnityEngine.Vector3 PositionMove);

Данный метод отвечает за передвижение героя. Игрок может управлять персонажем, при нажатии левой кнопки мыши герой перемещается в заданную точку.

public void LookAtEachOther (Transform EnemyTransform);

Если враг подойдёт близко к герою, вызовется данный метод и они повернутся друг к другу лицом.

protected void Regen ();

Функция предназначена для постепенного восстановления здоровья героя.

# 4.4 Интерфейс пользователя

При запуске игры на экране появляется главное меню (рисунок 4.1). Оно содержит две кнопки: «Играть» и «Выход». Нажав левой кнопкой мыши на первую кнопку, пользователь может запустить игру, на вторую – выйти из игры.

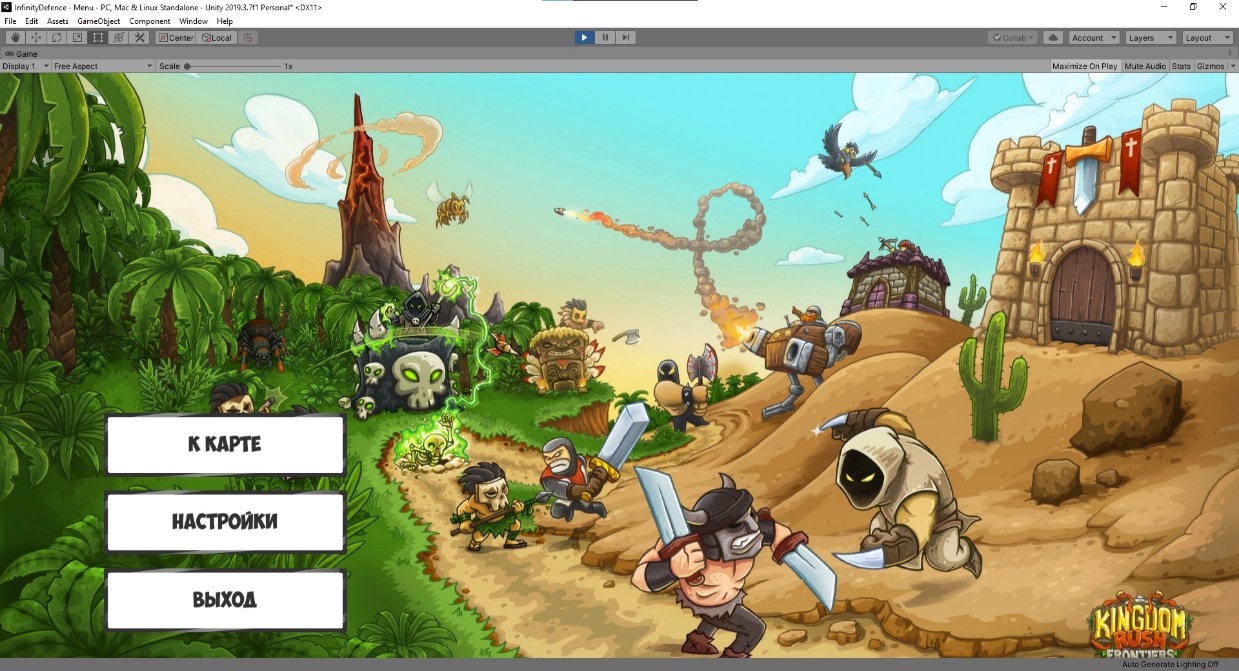


Рисунок 4.1 – Главное меню игры

Когда пользователь нажимает на кнопку «Играть», на экране появляется меню выбора уровня (рисунок 4.2).

В игре пользователю предоставляется возможность сражаться с врагами (рисунок 4.3) и убивать их (рисунок 4.4), получать за убийство монетки (рисунок 4.5), строить (рисунок 4.6), улучшать и продавать башни (рисунок 4.7), а также управлять героем (рисунок 4.8).

На карте есть кнопка, которая создаёт волны врагов (рисунок 4.9). При нажатии на эту кнопку или при истечении таймера вызываются враги, которые идут к концу уровня и если враги дойдут до него, то у игрока отнимется жизнь. Когда герой попадает в зону видимости врага, то он начинает подходить к нему, а если противнику удаётся подойти близко к герою, то он начинает наносить ему урон. При выходе главного персонажа из зоны видимости, враг продолжает к заданной конечной точке.

Во время игры пользователю графически представлена возможность отслеживать количество монет, жизней и волн. В левом верхнем углу показано, сколько монет игрок уже заработал, а рядом – сколько всего жизней осталось и оставшиеся волны врагов. Над всеми персонажами находится информация о здоровье. Она представлена в виде полоски здоровья, в начале игры она полностью заполнена, но если получить ранение, то очки здоровья убавляются и также уменьшается зелёная полоска, характеризующая здоровье. Чтобы победить необходимо убить всех врагов, при этом сохранить как минимум одну жизнь (рисунок 4.10). Поражение наступает, если жизни заканчиваются (рисунок 4.11).

На протяжении всей игры персонаж может ходить по всей карте, его возможность перемещения ограничена пределами карты.

Во время игры можно строить специальные башни, которые будут помогать атаковать врагов. Игрок может строить, улучшать и продавать эти башни за золото, которое он получает за убийство врагов.

Сверху справа экрана, находится кнопка (пауза) (рисунок 4.12) при нажатии на которую появляется меню, и игра приостанавливается. Игроку предоставляется возможность выйти из игры или возобновить её (рисунок 4.13).



Рисунок 4.2 – Меню выбора уровня

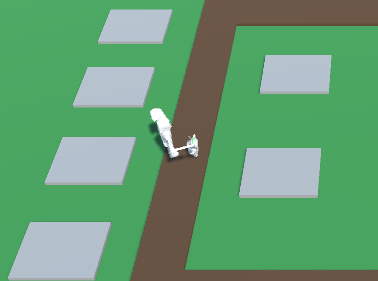


Рисунок 4.3 – Сражение с врагами

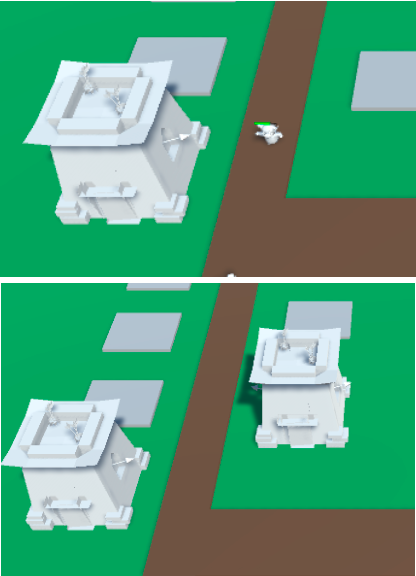


Рисунок 4.4 – Убийство врагов

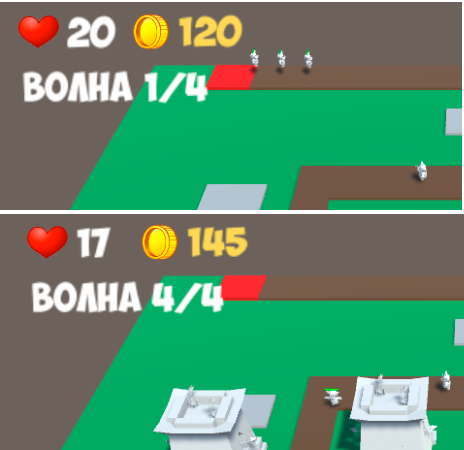


Рисунок 4.5 – Получение монет

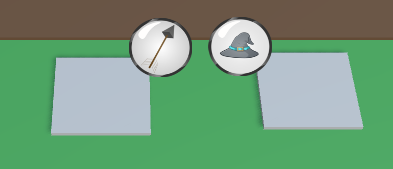


Рисунок 4.6 – Строительство башен

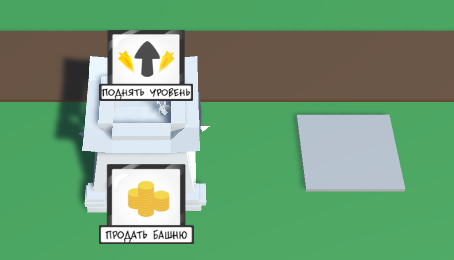


Рисунок 4.7 – Улучшение и продажа башен



Рисунок 4.8 – Управление героем



Рисунок 4.9 – Кнопка вызова волн

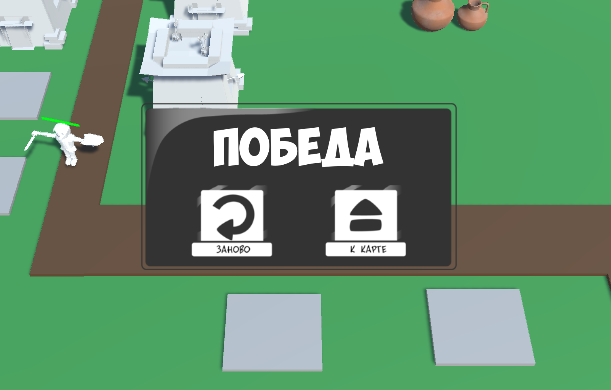


Рисунок 4.10 – Победа

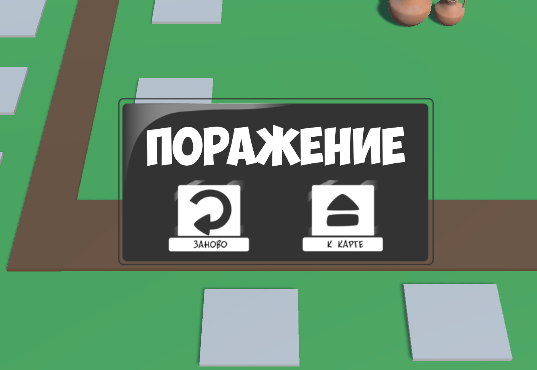


Рисунок 4.11 – Поражение



Рисунок 4.12 – Пауза



Рисунок 4.13 – Меню уровня

# 4.5 Критерии качества программной системы

Созданная программа обладает следующими положительными качествами:

* Модифицируемость – лёгкость внесения изменений, устранения ошибок благодаря тому, что программа написана на основе объектно-ориентированного подхода;
* Удобство применения – наличие дружественного интерфейса, с помощью которого пользователь может использовать программу без дополнительной подготовки;
* Надёжность – устойчивость в работе программы, выполнение предписанных функций обработки, выполнение диагностики возникающих в процессе работы программы ошибок;
* Мобильность – данный программный продукт может быть перенесён из одного окружения в другое и функционировать с такой же эффективностью;
* Сопровождаемость – благодаря использованию объектно-ориентированного подхода улучшена читаемость кода и обеспечивается возможность поддержания программы в актуальном состоянии.

# ВЫВОДЫ

В ходе выполнения курсового проекта были закреплены основы объектно-ориентированного программирования на языке C#. Согласно методическим указаниям по дисциплине «Объектно-ориентированное программирование» была написана программа. В ней реализованы основные принципы объектно-ориентированного программирования: инкапсуляция, полиморфизм и наследование. Инкапсуляция обеспечивает безопасную организацию иерархической управляемости, полиморфизм – сохранение интерфейса почти неизменным для классов, наследование – поэтапное создание сложных классов.

Программа структурирована с точки зрения управляемости, что особенно важно при реализации сложного проекта, так как это улучшает читаемость кода и позволяет легко вносить изменения, поддерживать в актуальном состоянии, а также исправлять ошибки.

Также программа является удобной в использовании, так как содержит дружественный интерфейс. Пользователю не надо проходить дополнительную подготовку для работы с ней, в игре содержится справочная информация, написанная в лёгкой для понимания форме и не содержит лишних данных.

Работоспособность программы была подтверждена тестированием.

Во время выполнения курсового проекта были закреплены основные этапы объектного подхода: объектно-ориентированный анализ (ООА), объектно-ориентированное проектирование (ООD) и объектно-ориентированное программирование (ООР).

Было приобретено умение построения иерархии, диаграмм состояний и жизненных циклов, информационной модели, а также реализации этого для разработки игры с использованием движка Unity.

Были изучены основы работы с Unity, написание скриптов, создание анимации, префабов, спрайтов для создания игры на основе составленных в начале работы над проектом иерархии, диаграмм и моделей.

В результате выполнения проекта было доказано, что применение объектного подхода улучшает структурированность с точки зрения управляемости, улучшает читаемость кода, делает программу надёжной, модифицируемой, сопровождаемой и удобной.

Данная программа может служить в дальнейшем для развлекательных целей.

# ПЕРЕЧЕНЬ ССЫЛОК

1. Шарп Д. Microsoft Visual C#. Подробное руководство/ Д.Шарп – СПб.:Питер, 2017 – 848 с.
2. Буч Г. Объектно-ориентированное проектирование с примерами применения / Г.Буч. — М.: Конкорд, 1992 — 519 с.
3. Албахари Д., Албахари Б. C# 7.0. Карманный справочник/ Д.Албахари, Б.Албахари; пер. с англ. – М.:Вильямс, 2017 – 224 с.
4. Стиллмен Э., Грин Д. Изучаем C#/ Э.Стиллмен, Д.Грин; пер. с англ. – СПб.:Питер, 2014 – 816 с.
5. Хокинг Д. Unity в действии. Мультиплатформенная разработка на C#/ Д.Хокинг – СПб.:Питер, 2016 – 336 с.
6. Нагел К., Ивьен Б., Глинн Д., Уотсон К., Скиннер М. C# 5.0 и платформа .NET 4.5 для профессионалов/ К.Нагел, Б.Ивьен, Д.Глинн, К.Уотсон, М.Скиннер – М.:Вильямс, 2014 – 1440 с.

# ПРИЛОЖЕНИЕ А

**Файл Archer.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Archer : Tower

{

}

**Файл Bullet.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Bullet : Tower

{

private Transform bullet\_target;

private float thisTowerDamage;

public void Seek(Transform \_target, float \_towerDamage)

{

bullet\_target = \_target;

thisTowerDamage = \_towerDamage;

}

public void Update()

{

if(bullet\_target == null)

{

Destroy(gameObject);

return;

}

Vector3 dir = bullet\_target.position - transform.position;

float distanceThisFrame = speed \* Time.deltaTime;

transform.LookAt(bullet\_target);

transform.Translate(dir.normalized \* distanceThisFrame, Space.World);

}

void OnCollisionEnter(Collision col)

{

if (col.gameObject.tag == "Enemy")

{

col.gameObject.GetComponent<Enemy>().TakeDamage(thisTowerDamage \* col.gameObject.GetComponent<Enemy>().Armor);

}

Destroy(gameObject);

}

}

**Файл CastSystem.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CastSystem : MonoBehaviour

{

public GameObject selectCircle = null;

private bool Cast = false;

private Ray p\_ray;

private RaycastHit p\_hit;

private MeteorShower p\_meteorShower;

private void Start()

{

p\_meteorShower = GetComponent<MeteorShower>();

selectCircle.SetActive(false);

}

private void Update()

{

if(Input.GetKey(KeyCode.Q))

{

Cast = true;

}

if(Cast)

{

p\_ray = Camera.main.ScreenPointToRay(Input.mousePosition);

if(Physics.Raycast(p\_ray, out p\_hit))

{

selectCircle.transform.position = new Vector3(p\_hit.point.x, p\_hit.point.y + 0.5f, p\_hit.point.z);

selectCircle.SetActive(true);

}

if (Input.GetMouseButton(1))

{

Cast = false;

selectCircle.SetActive(false);

}

if (Input.GetMouseButton(0))

{

p\_meteorShower.Init(p\_hit.point);

Cast = false;

}

} else {

selectCircle.SetActive(false);

}

}

}

**Файл Enemy.cs**

using UnityEngine;

using System.Collections;

public class Enemy : Mobs

{

private int WavePointIndex = 0;

private Transform target;

private bool EnemyIsBusy = false;

private static bool OneEnemyIsBusy = false;

private static int numOfDeadEnemies;

private int numOfEnemies;

public int damageLevel;

public int mobPrice;

public float Armor;

public float Resistance;

public void GetNextWaypoint()

{

if (WavePointIndex >= WayPoints.points.Length - 1)

{

M.ApplyDamage(damageLevel);

numOfDeadEnemies++;

if (numOfDeadEnemies == numOfEnemies && M.GetComponent<Map>().levelHealth > 0)

{

M.GetComponent<Map>().GameWin();

}

Destroy(gameObject);

return;

}

WavePointIndex++;

target = WayPoints.points[WavePointIndex];

transform.LookAt(target);

}

public void Move()

{

Vector3 dir = target.position - transform.position;

transform.Translate(dir.normalized \* Speed \* Time.deltaTime, Space.World);

if (Vector3.Distance(transform.position, target.position) <= 0.2f)

{

GetNextWaypoint();

}

}

public override void Die()

{

if (EnemyIsBusy)

{

Debug.Log("ok");

StopAllCoroutines();

M.Hero.HeroIsBusy = false;

}

base.Die();

numOfDeadEnemies++;

if ( numOfDeadEnemies == numOfEnemies && M.GetComponent<Map>().levelHealth > 0)

{

M.GetComponent<Map>().GameWin();

}

M.AddMoney(mobPrice);

}

private IEnumerator OnTriggerEnter(Collider other)

{

if (other.tag == "Friendly" && !(M.Hero.GetHeroIsBusy()) &&!(OneEnemyIsBusy))

{

M.Hero.HeroIsBusy = true;

OneEnemyIsBusy = true;

EnemyIsBusy = true;

Speed = 0;

M.Hero.Move(transform.position-new Vector3 ( (float)0.5,0,0));

yield return new WaitForSeconds(1f);

M.Hero.LookAtEachOther(transform);

StartCoroutine(DealDamage(damage, speedAtack));

StartCoroutine(DealDamage(M.Hero.damage, M.Hero.speedAtack));

}

}

/\*public override void TakeDamage(float Damage)

{

if (CurrentHealth <= 0)

{

Debug.Log("stop");

StopAllCoroutines();

}

base.TakeDamage(Damage);

}\*/

public bool GetEnemyIsBusy()

{

return EnemyIsBusy;

}

public override void Start()

{

base.Start();

numOfEnemies = M.GetComponent<WaveSpawner>().GetNumEnemies();

target = WayPoints.points[0];

}

public virtual void Update()

{

if(EnemyIsBusy && Input.GetMouseButtonDown(0))

{

Speed = StartSpeed;

EnemyIsBusy = false;

OneEnemyIsBusy = false;

transform.LookAt(target);

}

}

}

**Файл Epicentr.cs**

using UnityEngine;

public class Epicentr : MonoBehaviour

{

public bool Status = false;

public GameObject meteorPrefab;

private float p\_damage, p\_duration, p\_radius, p\_interval, p\_curDuration, p\_curInterval, p\_speed;

private int p\_countPerWave;

private Vector3 p\_spawnPoint;

public void Init(float \_damage, float \_duration, float \_radius, float \_interval, float \_speed ,int \_countPerWave, Vector3 \_spawnPoint)

{

p\_damage = \_damage;

p\_duration = \_duration;

p\_radius = \_radius;

p\_interval = \_interval;

p\_countPerWave = \_countPerWave;

p\_spawnPoint = \_spawnPoint;

p\_speed = \_speed;

Status = true;

}

private void Start()

{

transform.Rotate(new Vector3(90.0f, 0.0f, 0.0f));

}

private void Update()

{

if(Status)

{

p\_curInterval += Time.deltaTime;

if(p\_curInterval >= p\_interval)

{

for(int i = 0; i < p\_countPerWave; i++)

{

var \_spawnPosition = new Vector3(p\_spawnPoint.x, 10.0f, p\_spawnPoint.z);

var temp = Instantiate(meteorPrefab, \_spawnPosition, Quaternion.identity).GetComponent<Meteor>();

temp.Init(p\_damage, p\_speed);

}

p\_curInterval = 0.0f;

}

if(p\_curDuration >= p\_duration)

{

Destroy(gameObject);

}

p\_curDuration += Time.deltaTime;

}

}

}

**Файл Heroes.cs**

using UnityEngine;

using UnityEngine.AI;

using System.Collections;

public class Heroes : Mobs

{

private NavMeshAgent MyHero;

private bool notEnoughtHealth = false;

RaycastHit hit;

Ray MyRay;

public float RegenHealth;

public float RegenSpeed;

public Enemy E;

public bool HeroIsBusy = false;

protected IEnumerator Regen()

{

while(StartHealth > CurrentHealth)

{

if (CurrentHealth + RegenHealth > StartHealth)

{

CurrentHealth = StartHealth;

}

else {

CurrentHealth += RegenHealth;

}

HealthBar.fillAmount = CurrentHealth / StartHealth;

yield return new WaitForSeconds(RegenSpeed);

}

}

public override void TakeDamage(float Damage)

{

base.TakeDamage(Damage);

if (StartHealth > CurrentHealth && !notEnoughtHealth)

{

notEnoughtHealth = true;

StartCoroutine(Regen());

}

}

public override void Die()

{

base.Die();

}

public LayerMask ClickToMove;

public void OnTriggerEnter(Collider other)

{

}

public void Move(Vector3 PositionMove)

{

MyHero.SetDestination(PositionMove);

}

public void LookAtEachOther(Transform EnemyTransform)

{

transform.LookAt(EnemyTransform);

EnemyTransform.LookAt(transform);

StartCoroutine(DealDamage(damage, speedAtack));

}

public bool GetHeroIsBusy()

{

return HeroIsBusy;

}

public void SetDestinationToMousePosition()

{

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(ray, out hit))

{

Move(hit.point);

HeroIsBusy = true;

}

}

void Update()

{

if (Input.GetMouseButtonDown(0))

{

MyRay = Camera.main.ScreenPointToRay(Input.mousePosition);

Debug.DrawRay(MyRay.origin, MyRay.direction \* 10, Color.yellow);

if (Physics.Raycast(MyRay, out hit, 100))

{

if ( hit.collider.GetComponent(typeof(MeshFilter)) as MeshFilter )

{

MeshFilter filter = hit.collider.GetComponent(typeof(MeshFilter)) as MeshFilter;

if (filter.gameObject.tag == "Terrain")

{

SetDestinationToMousePosition();

}

}

}

}

if(MyHero.transform.position.x == hit.point.x && MyHero.transform.position.z == hit.point.z)

{

HeroIsBusy = false;

}

}

public override void Start()

{

base.Start();

MyHero = GetComponent<NavMeshAgent>();

}

}

**Файл Knight.cs**

using UnityEngine;

public class Knight : Enemy

{

void Update()

{

Move();

}

}

**Файл KnightHero.cs**

using UnityEngine;

public class KnightHero : Heroes

{}

**Файл LevelSelect.cs**

using UnityEngine;

using UnityEngine.SceneManagement;

public class LevelSelect : MonoBehaviour

{

public void ToLevel(string levelName)

{

SceneManager.LoadScene(levelName);

}

public void Quit()

{

Application.Quit();

}

}

**Файл Magic.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Magic : Tower

{

}

**Файл Map.cs**

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

public class Map : MonoBehaviour

{

private UIManager \_UI\_Manager;

public GameObject gameOverPanel;

public GameObject gameWinPanel;

public Heroes Hero;

[Header("Map Fileds")]

public int levelHealth;

public int gold;

[Header("Text Fileds")]

public Text healthText;

public Text goldText;

#region FireBall

[Header("FireBall")]

public GameObject selectCircle = null;

public float meteorTimeOut;

private bool Cast = false;

private Ray p\_ray;

private RaycastHit p\_hit;

private MeteorShower p\_meteorShower;

private float meteorTimer = 0f;

#endregion

void Start()

{

\_UI\_Manager = gameObject.GetComponent<UIManager>();

healthText.text = levelHealth.ToString();

goldText.text = gold.ToString();

p\_meteorShower = GetComponent<MeteorShower>();

selectCircle.SetActive(false);

}

private void Update()

{

if (Input.GetKey(KeyCode.Q) && meteorTimer <= 0)

{

Cast = true;

}

if (Cast)

{

p\_ray = Camera.main.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(p\_ray, out p\_hit))

{

selectCircle.transform.position = new Vector3(p\_hit.point.x, p\_hit.point.y + 0.5f, p\_hit.point.z);

selectCircle.SetActive(true);

}

if (Input.GetMouseButton(1))

{

Cast = false;

selectCircle.SetActive(false);

}

if (Input.GetMouseButton(0))

{

p\_meteorShower.Init(p\_hit.point);

Cast = false;

meteorTimer = meteorTimeOut;

}

} else {

selectCircle.SetActive(false);

}

if(meteorTimer >= 0) meteorTimer -= Time.deltaTime;

}

public void ApplyDamage(int Damage)

{

levelHealth -= Damage;

healthText.text = levelHealth.ToString();

if(levelHealth <= 0)

{

GameOver();

}

}

public void GameOver()

{

StopAllCoroutines();

gameOverPanel.SetActive(true);

\_UI\_Manager.col.SetActive(true);

\_UI\_Manager.SetGamePaused(false);

Time.timeScale = 0;

}

public void GameWin()

{

StopAllCoroutines();

gameWinPanel.SetActive(true);

\_UI\_Manager.col.SetActive(true);

\_UI\_Manager.SetGamePaused(false);

Time.timeScale = 0;

}

public void ToMap()

{

SceneManager.LoadScene("MainMap");

}

public void Again()

{

\_UI\_Manager.col.SetActive(false);

\_UI\_Manager.SetGamePaused(false);

Time.timeScale = 1;

SceneManager.LoadScene(SceneManager.GetActiveScene().name);

}

public void WasteMoney(int w\_Money)

{

gold -= w\_Money;

goldText.text = gold.ToString();

}

public bool HaveMoney(int h\_Money)

{

if (gold >= h\_Money) return true;

else return false;

}

public void AddMoney(int t\_Money)

{

gold += t\_Money;

goldText.text = gold.ToString();

}

public void CallMeteor()

{

if(meteorTimer <= 0) Cast = true;

}

public void CallReinforcement()

{

}

}

**Файл Meteor.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Meteor : MonoBehaviour

{

private float Damage;

private float Speed;

public void Init(float \_damage, float \_speed)

{

Damage = \_damage;

Speed = \_speed;

}

private void Start()

{

transform.Rotate(new Vector3(0.0f, 0.0f, -90.0f));

}

private void Update()

{

if(transform.position.y <= 0.0f)

{

Destroy(gameObject);

} else {

transform.position -= new Vector3(0.0f, Speed \* Time.deltaTime , 0.0f);

}

}

private void OnTriggerEnter(Collider other)

{

if(other.tag == "Enemy")

{

other.GetComponent<Mobs>().TakeDamage(Damage);

}

}

}

**Файл MeteorShower.cs**

using UnityEngine;

public class MeteorShower : MonoBehaviour

{

[Header("Status")]

public bool Status = false;

[Header("Attribute")]

public float Damage;

public float Radius;

public float Duration;

public float Interval;

public float Speed;

public int countPerWave;

[Header("Ref")]

public GameObject epicenterPrefab;

private Vector3 p\_pointCast;

public void Init(Vector3 \_point)

{

p\_pointCast = new Vector3(\_point.x, 0.5f, \_point.z);

Status = true;

}

private void Update()

{

if(Status)

{

var temp = Instantiate(epicenterPrefab, p\_pointCast, Quaternion.identity).GetComponent<Epicentr>();

temp.Init(Damage, Duration, Radius, Interval, Speed, countPerWave, p\_pointCast);

Status = false;

}

}

}

**Файл Mobs.cs**

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

public abstract class Mobs : MonoBehaviour

{

public float StartHealth;

public float Speed;

public float range;

public float speedAtack;

public float damage;

protected Map M;

protected float StartSpeed;

protected GameObject MapObject;

[Header("Health fields")]

public Image HealthBar;

public float CurrentHealth;

public virtual void Die()

{

Destroy(gameObject);

}

public virtual void Start()

{

MapObject = GameObject.Find("Map");

StartSpeed = Speed;

CurrentHealth = StartHealth;

M = MapObject.GetComponent<Map>();

}

public void OnDrawGizmosSelected()

{

Vector3 GizmosPos = new Vector3(transform.position.x, transform.position.y - (float)0.4, transform.position.z);

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(GizmosPos, range);

}

public IEnumerator DealDamage(float \_Damage, float \_speedAtack)

{

while(true)

{

//StopAllCoroutines();

TakeDamage(\_Damage);

yield return new WaitForSeconds(\_speedAtack);

}

}

public virtual void TakeDamage(float Damage)

{

CurrentHealth -= Damage;

HealthBar.fillAmount = CurrentHealth / StartHealth;

if (CurrentHealth <= 0)

{

StopAllCoroutines();

Die();

}

}

}

**Файл Orc.cs**

using UnityEngine;

public class Orc : Enemy

{

public override void Update()

{

base.Update();

Move();

}

}

**Файл Tower.cs**

using UnityEngine;

public class Tower : MonoBehaviour

{

#region private Fileds

private Transform target;

private float fireCountdown = 0f;

private static WaveSpawner WS;

private static Map M;

private static GameObject SelectedTower;

private GameObject nextLevelPrefab;

private Vector3 btnPos;

private static bool btnIsActive = false;

#endregion

#region public Fields

[Header("Common Fields")]

public int towerPrice;

public float range = 5f;

public string EnemyTag = "Enemy";

public int Tower\_Level = 1;

public int sellPrice;

public GameObject[] allTowersPrefab;

[Header("Bullet Fields")]

public float fireRate = 1f;

public GameObject bulletPrefab;

public Transform firePoint;

public float speed = 70f;

public float towerDamage = 30f;

#endregion

public void Start()

{

btnPos = new Vector3(transform.position.x, transform.position.y - 2, transform.position.z);

GameObject map = GameObject.Find("Map");

M = map.GetComponent<Map>();

InvokeRepeating("UpdateTarget", 0f, 0.5f);

}

public void Update()

{

if (target == null) return;

if (fireCountdown <= 0f)

{

Shoot();

fireCountdown = 1f / fireRate;

}

fireCountdown -= Time.deltaTime;

}

void Shoot()

{

GameObject bulletGO = Instantiate(bulletPrefab, firePoint.position, firePoint.rotation);

Bullet bullet = bulletGO.GetComponent<Bullet>();

if (bullet != null)

{

bullet.Seek(target, towerDamage);

}

}

public void UpdateTarget()

{

GameObject[] enemies = GameObject.FindGameObjectsWithTag(EnemyTag);

float shortestDistance = Mathf.Infinity;

GameObject nearestEnemy = null;

foreach (GameObject enemy in enemies)

{

float DistanceToEnemy = Vector3.Distance(transform.position, enemy.transform.position);

if(DistanceToEnemy < shortestDistance)

{

shortestDistance = DistanceToEnemy;

nearestEnemy = enemy;

}

}

if(nearestEnemy != null && shortestDistance <= range)

{

target = nearestEnemy.transform;

} else

{

target = null;

}

}

public void OnDrawGizmosSelected()

{

Vector3 GizmosPos = new Vector3(transform.position.x, transform.position.y, transform.position.z);

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(GizmosPos, range);

}

public void OnMouseUp()

{

SelectedTower = gameObject;

btnIsActive = !btnIsActive;

if (btnIsActive == false && M.GetComponent<UIManager>().switchAllBtn.activeSelf == false) btnIsActive = !btnIsActive;

M.GetComponent<UIManager>().SwitchButtons(btnPos, 1, btnIsActive);

}

public void SellTower(GameObject Map)

{

M = Map.GetComponent<Map>();

WS = Map.GetComponent<WaveSpawner>();

int MySellPrice = 0; // переменная чтобы не изменялось значение в префабе

// если ещё не вызвали первую волну

if (WS.GetWaveNumber() <= 0) {

for (int i = 0; i < SelectedTower.GetComponent<Tower>().Tower\_Level; i++)

{

MySellPrice += SelectedTower.GetComponent<Tower>().allTowersPrefab[i].GetComponent<Tower>().towerPrice;

}

} else {

for (int i = 0; i < SelectedTower.GetComponent<Tower>().Tower\_Level; i++)

{

MySellPrice += SelectedTower.GetComponent<Tower>().allTowersPrefab[i].GetComponent<Tower>().sellPrice;

}

}

M.AddMoney(MySellPrice);

btnIsActive = !btnIsActive;

M.GetComponent<UIManager>().SwitchButtons(btnPos, 1, btnIsActive);

Destroy(SelectedTower);

}

public void LevelUp(GameObject Map)

{

if( SelectedTower.GetComponent<Tower>().Tower\_Level < SelectedTower.GetComponent<Tower>().allTowersPrefab.Length)

{

nextLevelPrefab = SelectedTower.GetComponent<Tower>().allTowersPrefab[SelectedTower.GetComponent<Tower>().Tower\_Level];

M = Map.GetComponent<Map>();

if (M.HaveMoney(nextLevelPrefab.GetComponent<Tower>().towerPrice))

{

Instantiate(nextLevelPrefab, SelectedTower.GetComponent<Transform>().position, SelectedTower.GetComponent<Transform>().rotation);

M.WasteMoney(nextLevelPrefab.GetComponent<Tower>().towerPrice);

btnIsActive = !btnIsActive;

M.GetComponent<UIManager>().SwitchButtons(btnPos, 1, btnIsActive);

Destroy(SelectedTower);

}

}

}

}

**Файл TowerManager.cs**

using UnityEngine;

public class TowerManager : MonoBehaviour

{

private Tower T;

private Map M;

private static Vector3 PosSelectedTower;

private Color startColor;

private static bool btnIsActive = false;

private Vector3 btnPos;

public GameObject[] Tower\_Prefabs;

// =========== МЕТОДЫ ===========

public void Start()

{

GameObject map = GameObject.Find("Map");

M = map.GetComponent<Map>();

startColor = GetComponent<Renderer>().material.color;

}

// =========== СТРОИМ БАШНЮ ===========

public void Build\_Tower(string towerName)

{

for (int i = 0; i < Tower\_Prefabs.Length; i++)

{

T = Tower\_Prefabs[i].GetComponent<Tower>();

if (M.HaveMoney(T.towerPrice) && Tower\_Prefabs[i].name == towerName)

{

PosSelectedTower = new Vector3(PosSelectedTower.x, PosSelectedTower.y + (float)0.4, PosSelectedTower.z);

Instantiate(Tower\_Prefabs[i], PosSelectedTower, Quaternion.Euler(0, 0, 0));

M.WasteMoney(T.towerPrice);

btnIsActive = !btnIsActive;

M.GetComponent<UIManager>().SwitchButtons(btnPos, 0, btnIsActive);

}

}

}

// =========== ЕСЛИ НАЖАЛИ НА КУБ ===========

public void OnMouseUp()

{

btnPos = new Vector3(transform.position.x + 3, transform.position.y, transform.position.z);

PosSelectedTower = transform.position;

btnIsActive = !btnIsActive;

if ( btnIsActive == false && M.GetComponent<UIManager>().switchAllBtn.activeSelf == false ) btnIsActive = !btnIsActive;

M.GetComponent<UIManager>().SwitchButtons(btnPos, 0, btnIsActive);

}

// =========== ЕСЛИ НАВЕЛИ МЫШКУ НА КУБ ===========

public void OnMouseEnter()

{

GetComponent<Renderer>().material.color = Color.green;

}

// =========== ЕСЛИ УБРАЛИ МЫШКУ С КУБА ===========

public void OnMouseExit()

{

GetComponent<Renderer>().material.color = startColor;

}

}

**Файл UIManager.cs**

using UnityEngine;

public class UIManager : MonoBehaviour

{

private Vector3 WorldPos;

private static bool gamePaused = false;

[Header("Buttons Fields")]

public GameObject [] allBtns;

public GameObject switchAllBtn;

public GameObject col;

[Header("Pause Game Fields")]

public GameObject pauseGame;

public void SwitchButtons(Vector3 towerPos, int num\_arr, bool \_btnIsActive)

{

WorldPos = Camera.main.WorldToScreenPoint(towerPos); // мировая позиция текущего объекта

col.SetActive(\_btnIsActive);

allBtns[num\_arr].GetComponent<Transform>().position = new Vector3(WorldPos.x, WorldPos.y, WorldPos.z);

allBtns[num\_arr].SetActive(\_btnIsActive);

switchAllBtn.SetActive(\_btnIsActive);

}

public void HideAllButtons()

{

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

{

allBtns[i].SetActive(false);

}

switchAllBtn.SetActive(false);

col.SetActive(false);

}

public void SwitchGamePause()

{

gamePaused = !gamePaused;

pauseGame.SetActive(gamePaused);

col.SetActive(gamePaused);

Time.timeScale = gamePaused == true ? 0 : 1;

}

public void SetGamePaused(bool \_gamePaused)

{

gamePaused = \_gamePaused;

}

}

**Файл WaveSpawner.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

[System.Serializable]

public struct Enemy\_Settings

{

public GameObject Enemy\_Prefab;

public int num\_Enemies;

public float countdown;

}

[System.Serializable]

public struct Wave\_Settings

{

public List<Enemy\_Settings> enemy\_settings;

}

public class WaveSpawner : MonoBehaviour

{

#region public fields

public List<Wave\_Settings> Waves;

[Header("Common Fields")]

public Transform spawnPoint;

public GameObject Btn\_Wave\_Call;

public float timeBeetweenWaves;

[Header("Text Fields")]

public Text WavesText;

#endregion

#region private fields

private int numEnemies;

private int NumLastWave;

private int waveNumber = 0;

#endregion

public int GetWaveNumber()

{

return waveNumber;

}

public int GetNumLastWave()

{

return NumLastWave;

}

public void Start()

{

foreach (Wave\_Settings W in Waves)

{

foreach (Enemy\_Settings ES in W.enemy\_settings)

{

for (int i = 0; i < ES.num\_Enemies; i++)

{

numEnemies++;

}

}

}

NumLastWave = Waves.Count; // запоминаем кол-во волн

WavesText.text = "Волна " + waveNumber.ToString() + "/" + NumLastWave.ToString();

}

public int GetNumEnemies()

{

return numEnemies;

}

public void SpawnWave()

{

Btn\_Wave\_Call.SetActive(false); // деактивируем кнопку

StopAllCoroutines();

StartCoroutine(SpawnEnenmy());

}

public bool IsLastWave()

{

if (waveNumber >= NumLastWave) return true;

else return false;

}

IEnumerator SpawnEnenmy()

{

foreach (Wave\_Settings W in Waves)

{

waveNumber++;

WavesText.text = "Волна " + waveNumber.ToString() + "/" + NumLastWave.ToString();

foreach (Enemy\_Settings ES in W.enemy\_settings)

{

for (int i = 0; i < ES.num\_Enemies; i++)

{

yield return new WaitForSeconds(ES.countdown);

Instantiate(ES.Enemy\_Prefab, spawnPoint.position, spawnPoint.rotation); // создаём врага

}

}

if (Waves.Count >= 2)

{

Waves.RemoveAt(0);

Btn\_Wave\_Call.SetActive(true); // активируем кнопку

yield return new WaitForSeconds(timeBeetweenWaves); // время ожидания между волнами

}

break;

}

if (IsLastWave())

{

Btn\_Wave\_Call.SetActive(false);

} else SpawnWave();

}

}

**Файл WayPoints.cs**

using UnityEngine;

public class WayPoints : MonoBehaviour

{

public static Transform[] points;

void Awake()

{

points = new Transform[transform.childCount];

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

{

points[i] = transform.GetChild(i);

}

}

}