Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

Югорский государственный университет

Институт цифровой экономики

КУРСОВАЯ РАБОТА

По дисциплине «Языки программирования высокого уровня»

«Разработка игры в жанре TowerDefense в Unity3D»

Студенты гр. 1191б\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/А.Н.Свита/

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Е.И.Сафонов/

(подпись)

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

г. Ханты-Мансийск

1. г.

**Оглавление**

[Ведение 3](#_Toc60074861)

[1 Игровой движок Unity3D 3](#_Toc60074862)

[1.1 Возможности 3](#_Toc60074863)

[1.2 Достоинства и недостатки 5](#_Toc60074864)

[2 Описание и создание игры 7](#_Toc60074865)

[2.1 Цель игры 7](#_Toc60074866)

[2.2 Создание и использования префабов (Prefabs) 7](#_Toc60074867)

[2.3 Работа со скриптами 8](#_Toc60074868)

[2.3.1 Spawn и передвижение врагов 8](#_Toc60074869)

[2.3.2 Кнопка выбора и расстановка башен 9](#_Toc60074870)

[2.3.3 Создание снарядов и стрельба по противникам 9](#_Toc60074871)

[2.3.4 Динамический интерфейс и подсчет очков 10](#_Toc60074872)

[2.3.5 Рандом появление врагов через каждую волну 10](#_Toc60074873)

[2.3.6 Menu, restart, и pause 11](#_Toc60074874)

[Заключение 12](#_Toc60074875)

[Список используемых источников 13](#_Toc60074876)

[Приложение А 14](#_Toc60074877)

[Приложение Б 15](#_Toc60074878)

# Ведение

Актуальность данной темы состоит в том, что жанр «TD» становится популярный для всех возрастов. Особенность жанр таких игр заключается в том, что у игрока стоит задача остановить врагов с помощью башен и других укреплений, не дав им дойти до края карты, где обычно враги наступают волнами, причем каждый их вид имеет свои слабые и сильные стороны, что нужно учитывать при выборе и улучшении башен.

Один из самых ранних представителей TD — аркадная игра Rampart, перенесённая впоследствии на множество платформ. Игра стала причиной создания ряда карт игры «StarCraft», получивших название Turret Defense, которые в свою очередь вдохновили на создание карт Sunken и Hero Defense (Warcraft III). Такие модификации стали популярными среди игроков Warcraft III и Age of Empires II, затем укрепившись в виде отдельного жанра. Первым представителем отдельной игры в жанре Tower Defense стала компьютерная игра «Master of Defense», вышедшая 7 ноября 2005 года. Она стала очень популярна, получила награду «Стратегия года» от GameTunnel в 2006-м.[1],[2]

Одной из самых популярных программ по созданию игр является Unity3D. Это объясняется понятным и гибким интерфейсом, который можно подстроить под себя и много чем еще, но об этом немного позже.

Все компоненты в Unity3D – это объекты, а, следовательно, мы в дальнейшем будем применять объектно-ориентированное программирование. ООП – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Цель работы: разработка игры TowerDefense на движке Unity3D

Задачи:

1. Разобраться что такое Unity3D
2. Использовать основные инструменты по создание игры Unity3D
3. Тестирование игры для определения ее работы

# Игровой движок Unity3D

## Возможности

Редактор Unity3D имеет простой Drag&Drop интерфейс, который легко настраивать, состоящий из различных окон, благодаря чему можно производить отладку игры прямо в редакторе. Движок использует для написания скриптов C#. Ранее поддерживались также Boo (диалект Python, поддержку убрали в 5-й версии) и модификация JavaScript, известная как UnityScript (поддержка прекращена в версии 2017.1). Расчёты физики производит физический движок PhysX от NVIDIA. Графический API – DirectX (на данный момент DX 11, поддерживается DX 12)

Проект в Unity3D делится на сцены (уровни) – отдельные файлы, содержащие свои игровые миры со своим набором объектов, сценариев, и настроек. Сцены могут содержать в себе как, собственно, объекты (модели), так и пустые игровые объекты – объекты, которые не имеют модели («пустышки»). Объекты, в свою очередь содержат наборы компонентов, с которыми и взаимодействуют скрипты. Также у объектов есть название (в Unity допускается наличие двух и более объектов с одинаковыми названиями), может быть тег (метка) и слой, на котором он должен отображаться. Так, у любого объекта на сцене обязательно присутствует компонент Transform – он хранит в себе координаты местоположения, поворота и размеров объекта по всем трём осям. У объектов с видимой геометрией также по умолчанию присутствует компонент Mesh Renderer, делающий модель объекта видимой.

К объектам можно применять коллизии (в Unity3D т. н. коллайдеры – collider), которых существует несколько типов.

Также Unity3D поддерживает физику твёрдых тел и ткани, а также физику типа Ragdoll (тряпичная кукла). В редакторе имеется система наследования объектов; дочерние объекты будут повторять все изменения позиции, поворота и масштаба родительского объекта. Скрипты в редакторе прикрепляются к объектам в виде отдельных компонентов.

При импорте текстуры в Unity3D можно сгенерировать alpha-канал, mip-уровни, normal-map, light-map, карту отражений, однако непосредственно на модель текстуру прикрепить нельзя – будет создан материал, которому будет назначен шейдер, и затем материал прикрепится к модели. Редактор Unity поддерживает написание и редактирование шейдеров. Редактор Unity имеет компонент для создания анимации, но также анимацию можно создать предварительно в 3D-редакторе и импортировать вместе с моделью, а затем разбить на файлы.

Unity3D поддерживает систему Level Of Detail (сокр. LOD), суть которой заключается в том, что на дальнем расстоянии от игрока высоко детализированные модели заменяются на менее детализированные, и наоборот, а также систему Occlusion culling, суть которой в том, что у объектов, не попадающих в поле зрения камеры не визуализируется геометрия и коллизия, что снижает нагрузку на центральный процессор и позволяет оптимизировать проект. При компиляции проекта создается исполняемый (.exe) файл игры (для Windows), а в отдельной папке – данные игры (включая все игровые уровни и динамически подключаемые библиотеки).

Движок поддерживает множество популярных форматов. Модели, звуки, текстуры, материалы, скрипты можно запаковывать в формат. unitypackage и передавать другим разработчикам, или выкладывать в свободный доступ. Этот же формат используется во внутреннем магазине Unity Asset Store, в котором разработчики могут бесплатно и за деньги выкладывать в общий доступ различные элементы, нужные при создании игр. Чтобы использовать Unity Asset Store, необходимо иметь аккаунт разработчика Unity. Unity имеет все нужные компоненты для создания мультиплеера (UNet депрекейтнули, начиная с версии 2018.4; на данный момент решение "из коробки" для мультиплеера отсутствует. Форк UNet под названием Mirror продолжает поддерживаться коммьюнити. В качестве стороннего решения существует платный Photon, облачный или On-Premises). Также можно использовать подходящий пользователю способ контроля версий. К примеру, Tortoise SVN, Git или Source Gear.

В Unity входит Unity Asset Server – инструментарий для совместной разработки на базе Unity, являющийся дополнением, добавляющим контроль версий и ряд других серверных решений. [3]

## Достоинства и недостатки

Как правило, игровой движок предоставляет множество функциональных возможностей, позволяющих их задействовать в различных играх, в которые входят моделирование физических сред, карты нормалей, динамические тени и многое другое. В отличие от многих игровых движков, у Unity3D имеется два основных преимущества: наличие визуальной среды разработки и межплатформенная поддержка. Первый 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. Под межплатформенной поддержкой предоставляется не только места развертывания (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).

Третьим преимуществом называется модульная система компонентов Unity3D, с помощью которой происходит конструирование игровых объектов, когда последние представляют собой комбинируемые пакеты функциональных элементов. В отличие от механизмов наследования, объекты в Unity3D создаются посредством объединения функциональных блоков, а не помещения в узлы дерева наследования. Такой подход облегчает создание прототипов, что актуально при разработке игр.

В качестве недостатков приводятся ограничение визуального редактора при работе с многокомпонентными схемами, когда в сложных сценах визуальная работа затрудняется. Вторым недостатком называется отсутствие поддержки Unity3D ссылок на внешние библиотеки, работу с которыми программистам приходится настраивать самостоятельно, и это также затрудняет командную работу. Ещё один недостаток связан с использованием шаблонов экземпляров (англ. prefabs). С одной стороны, эта концепция Unity3D предлагает гибкий подход визуального редактирования объектов, но с другой стороны, редактирование таких шаблонов является сложным. Также, WebGL-версия движка, в силу специфики своей архитектуры (трансляция кода из C# С++ и далее в JavaScript), имеет ряд нерешённых проблем с производительностью, потреблением памяти и работоспособностью на мобильных устройствах. [3]

# Описание и создание игры

## Цель игры

В данной игре главное целью стоит не дать противнику дойти до конца и продержаться 10 волн и в этом помогут 3 типа башен, которые будут им мешать и которые помогут достичь победы (Рис. 1).



Рисунок 1 – Процесс игры.

## Создание и использования префабов (Prefabs)

Для начала работы нужно создать объекты и к счастью в Unity есть возможность создавать prefabs. Это особый тип ассетов, позволяющий хранить весь GameObject со всеми компонентами и значениями свойств. Prefabs выступает в роли шаблона для создания экземпляров хранимого объекта в сцене. Любые изменения в prefabs немедленно отражаются и на всех его экземплярах, при этом вы можете переопределять компоненты и настройки для каждого экземпляра в отдельности. Кстати важно заметить если перетаскивать файловый ассет в сцену, будет создан экземпляр того объекта и все такие экземпляры изменятся при изменении оригинального ассета. Однако, хоть его поведение и похоже, но ассет - это не prefabs, из-за этого не выйдет добавить к нему компоненты или использовать любые другие описанные ниже свойства prefabs. Что бы создать prefabs надо выбрать Asset, Create Prefab и перетащив объект со сцены в “пустой” prefabs, появившийся в проекте. После чего можно создавать экземпляры prefabs просто перетаскивая его из окна Project на сцену. Имена объектов-экземпляров prefabs, будут подсвечиваться синим в окне Hierarchy (имена обычных объектов имеют чёрный цвет). Как уже упоминалось выше, изменение в prefabs производится автоматически ко всем её экземпляров, однако можно отделять и изменять. [5]

## Работа со скриптами

### Spawn и передвижение врагов

Для того чтобы наши враги появлялись в начале мы создаем пустой GameObject и называем его «Spawn» и перенесем левую часть нашей сцене (рис. 2) и также мы создаем другой GameObject и называем «Fin» и ставим в конце дорожки (рис. 3) Также мы меняем теги у Spawn на «Spawn», а у Fin на «Finish» и кроме этого мы добавляем у Fin экземпляр Box Collider 2D и ставим галочку (is Trigger). После чего мы создаем опять GameObject и называем MovePoints и создаем в нем также Box Collider 2D и ставим галочку (is Trigger) и после мы копируем MovePoints расставляем по заданном пути (рис. 4). После всего этого создадим папку под название «Script» и создадим три, script-а под названием «Manager» [Листинг кода Б.1], «Loader» [Листинг кода Б.2] и «Eneme» [Листинг кода Б.3] где мы прописываем где они появляются и как они двигаются. В самом скрипте «Loader» мы используем Singleton для действия, spawn-а врагов где после этого в самом скрипте «Manager» мы удалил MonoBehaviour и вместо него напишем Loader<Manager>

Листинг кода 2 – Loader

public class Loader <T>: MonoBehaviour where T:MonoBehaviour

{

private static T instance;

public static T Instance

{

get

{

if (instance == null)

{

instance = FindObjectOfType<T>();

}

else if (instance != FindObjectOfType<T>())

{

Destroy(FindObjectOfType<T>());

}

DontDestroyOnLoad(FindObjectOfType<T>());

return instance;

}

}}

### Кнопка выбора и расстановка башен

Чтобы мы могли выбирать башни нужно создать UI под названием Button после создаем папку Towers и создаем скрипт TowerButton (Листинг кода Б.4) где мы прописываем GamrObject денег, спрайта, башен и методы, которые они будут возвращаться. После мы выделяем все три кнопки и перетаскиваем скрипт в button трех наших башен. После всего проделанного создаем в папке Towers другой скрипт TowerMen [Листинг кода Б.5] в котором прописываем проверку того нажимаем мы на кнопку или нет, можно ли ставить на определение место башню и после протаскиваем этот скрипт в Main Camera, а, чтобы было нашему коду понятно куда ставить мы создаем новый тег для точки их появления «Towerside».

### Создание снарядов и стрельба по противникам

Создаем prefabs трех разных типов снарядов стрела, камень и огненный шар после чего создаем скрипт Project [Листинг кода Б.6] где просто прописываем свойства наших снарядов урон и тип снаряда.

Листинг кода 6 – Project

public enum projectType

{

rock, arrow, fireball

}

public class Project: MonoBehaviour

{

[SerializeField]

int attackDam;

[SerializeField]

projectType pType;

public int AttackDamage

{

get

{

return attackDam;

}

}

public projectType Ptype

{

get

{

return pType;

}

}

}

После всего создадим в папке Towers новый скрипт «TowerControler» [Листинг кода Б.7] где мы прописываем на каком радиусе будет атаковать и чем будет атаковать наша башня также в скрипте Manager [Листинг кода Б.1] создадим public List<Eneme> EnemeList = new List<Eneme> (); где наш лист будет считывать сколько противников вышло или должны выйти, и он же будет считывать стрельбу и по кому будет он стрелять. Также в скрипте Eneme (Листинг кода Б.3) прописываем в нем здоровье и прописываем получение урона от заданного типа снаряда и также если его здоровье равно нулю, то он умирает и когда он умирает мы получаем определённое количество денег за которое мы тратим наши башни.

### Динамический интерфейс и подсчет очков

Все это прописано в скрипте Manager [Листинг кода Б.1] где происходит кроме, spawn-а противника также происходит подсчёт кого убили сколько денег было получено за противника и сколько было потрачено на башни и сколько врагов прошли до конца и все это выводится через UI интерфейс. Также было добавлена метод через кнопку, которая определяет следующую волну если противники были побеждены все и дает перейти на следующую волну, и также она определяет выиграл ли игрок или нет тогда в любом случае игра начинается заново.

### Рандом появление врагов через каждую волну

В скрипте Manager [Листинг кода Б.1] мы прописываем метод, который будет через каждую волну, которую мы прошли будут появляться разный типы противников и также разное их количество

Листинг кода 1 – Manager

IEnumerator Spawn()

{

if (enemiesPerSpawn > 0 && EnemeList.Count < totalEnemi)

{

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

{

if (EnemeList.Count < totalEnemi)

{

Eneme newEnemi = Instantiate(enemies[Random.Range(0,enemiesToSpawn)]) as Eneme;

newEnemi.transform.position = spawnPoi.transform.position;

}

}

yield return new WaitForSeconds(spawnDelai);

StartCoroutine(Spawn());

}

}

### Menu, restart, и pause

После всего сделанного создадим скрипты Menu [Листинг кода Б.8] который может начать игру или выйти, Restart [Листинг кода Б.9] который может начать игру заново, когда игрок выиграл или проиграл в самой игре, Pause [Листинг кода Б.10] который может останавливать игру по нажатие кнопки «Esc» где дается выбор продолжить или вернуться в главное меню.

# Заключение

После всего седланого можно понять, что создание игры очень сложное, но также очень интересное занятие. В ходе проекта было создана игра по жанру Tower Defense – TD в котором было реализованы постройка башен за золото, с которых игрок зарабатывает с убийств противника, также было сделано интерфейс где отображались золото, текущая волна и сколько прошло врагов до конца. Этот проект помог понять, что такое движок Unity и также помогло полностью принимать на практике ООП. После тестов игры потвердели, игра работает исправно учтены.

Кроме того, мы познакомились со средой разработки Unity3D и доказали, что человек даже с отсутствием какого-либо опыта создания игр, при должном усердии и желании в состоянии разработать собственную игру.

В дальнейшем возможное развитие игры – это добавление новых башен и врагов и добавление способностей, как и для игрока так и для самих башен.

# Список используемых источников

1. Tower Defense и классика жанра [Электронный ресурс] Режим доступа: http://android.mobile-review.com/articles/44015/
2. Что такое Tower Defense [Электронный ресурс] Режим доступа: https://igrasan.ru/tower-defense/
3. Unity (игровой движок) – Википедия [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/Unity\_(игровой\_движок)
4. Unity – Руководство: Префабы (Prefams) [Электронный ресурс] Режим доступа: https://docs.unity3d.com/ru/530/Manual/Prefabs.html#:~:text=Использование%20префабов,из%20окна%20Project%20на%20сцену.

# Приложение А



Рисунок А.1. – Процесс игры

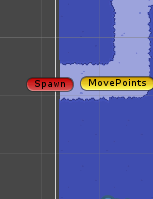


Рисунок А.2. – Spawn

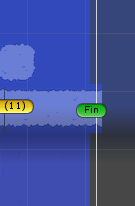


Рисунок А.3. – Fin



Рисунок А.4. – MovePoints

# Приложение Б

Листинг кода Б.1 – Manager

public enum gameStatus

{

next, play, gameover,win

}

public class Manager : Loader<Manager>

{

[SerializeField]

int totalWaves;

[SerializeField]

Text totalMoneyLabel;

[SerializeField]

Text currentWave;

[SerializeField]

Text playBtnLabel;

[SerializeField]

Text totalEscapedLabel;

[SerializeField]

Button playBtn;

[SerializeField]

GameObject spawnPoi;

[SerializeField]

Eneme[] enemies;

[SerializeField]

int totalEnemi=1;

[SerializeField]

int enemiesPerSpawn;

int waveNumber = 0;

int totalMoney = 10;

int totalEscaped = 0;

int roundEscaped=0;

int totalKilled = 0;

int whichEnemiesToSpawn = 0;

int enemiesToSpawn = 0;

gameStatus currentState = gameStatus.play;

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

public List<Eneme> EnemeList = new List<Eneme>();

const float spawnDelai = 0.5f;

public int TotalEscaped

{

get

{

return totalEscaped;

}

set

{

totalEscaped = value;

}

}

public int RoundEscaped

{

get

{

return roundEscaped;

}

set

{

roundEscaped = value;

}

}

public int TotalKilled

{

get

{

return totalKilled;

}

set

{

totalKilled = value;

}

}

public int TotatlMoney

{

get

{

return totalMoney;

}

set

{

totalMoney = value;

totalMoneyLabel.text = TotatlMoney.ToString();

}

}

void Start()

{

playBtn.gameObject.SetActive(false);

ShowMenu();

}

private void Update()

{

HamdEscape();

}

IEnumerator Spawn()

{

if (enemiesPerSpawn > 0 && EnemeList.Count < totalEnemi)

{

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

{

if (EnemeList.Count < totalEnemi)

{

Eneme newEnemi = Instantiate(enemies[Random.Range(0,enemiesToSpawn)]) as Eneme;

newEnemi.transform.position = spawnPoi.transform.position;

}

}

yield return new WaitForSeconds(spawnDelai);

StartCoroutine(Spawn());

}

}

public void RegitEnemy(Eneme enemy)

{

EnemeList.Add(enemy);

}

public void UnregEnemy(Eneme enemy)

{

EnemeList.Remove(enemy);

Destroy(enemy.gameObject);

}

public void DestroiEnemy()

{

foreach(Eneme eneme in EnemeList)

{

Destroy(eneme.gameObject);

}

EnemeList.Clear();

}

public void addMoney(int amout)

{

TotatlMoney += amout;

}

public void sutractMoney(int amout)

{

TotatlMoney -= amout;

}

public void IsWaveOver()

{

totalEscapedLabel.text = "Прошли " + TotalEscaped + "/10";

if ((RoundEscaped+TotalKilled)==totalEnemi)

{

if (waveNumber<=enemies.Length)

{

enemiesToSpawn = waveNumber;

}

SetCurretGameState();

ShowMenu();

}

}

public void SetCurretGameState()

{

if (totalEscaped >= 10)

{

currentState = gameStatus.gameover;

}

else if(waveNumber==0&& (RoundEscaped + TotalKilled) == 0)

{

currentState = gameStatus.play;

}

else if (waveNumber >= totalWaves)

{

currentState = gameStatus.win;

}

else

{

currentState = gameStatus.next;

}

}

public void PlayButtonPressed()

{

switch (currentState)

{

case gameStatus.next:

waveNumber += 1;

totalEnemi += waveNumber;

break;

default:

TotalEscaped = 0;

TotatlMoney = 10;

enemiesToSpawn = 0;

TowerMen.Instance.DestroyAllTower();

TowerMen.Instance.RenameTagBuild();

totalMoneyLabel.text = TotatlMoney.ToString();

totalEscapedLabel.text = "Прошли" + TotalEscaped + "/ 10";

break;

case gameStatus.gameover:

SceneManager.LoadScene("SampleScene");

break;

case gameStatus.win:

SceneManager.LoadScene("SampleScene");

break;

}

DestroiEnemy();

TotalKilled = 0;

RoundEscaped = 0;

currentWave.text = "Волна " + (waveNumber + 1);

StartCoroutine(Spawn());

playBtn.gameObject.SetActive(false);

}

public void ShowMenu()

{

switch (currentState)

{

case gameStatus.gameover:

playBtnLabel.text = "Конец игры";

break;

case gameStatus.next:

playBtnLabel.text = "Некст волна";

break;

case gameStatus.play:

playBtnLabel.text = "Начать играть";

break;

case gameStatus.win:

playBtnLabel.text = "Победа";

break;

}

playBtn.gameObject.SetActive(true);

}

private void HamdEscape()

{

if (Input.GetKeyDown(KeyCode.Escape))

{

TowerMen.Instance.DisabelDrag();

TowerMen.Instance.towerNtnPress=null;

}

}

}

Листинг кода Б.2 – Loader

public class Loader <T>: MonoBehaviour where T:MonoBehaviour

{

private static T instance;

public static T Instance

{

get

{

if (instance == null)

{

instance = FindObjectOfType<T>();

}

else if (instance != FindObjectOfType<T>())

{

Destroy(FindObjectOfType<T>());

}

DontDestroyOnLoad(FindObjectOfType<T>());

return instance;

}

}

}

Листинг кода Б.3 – Eneme

public class Eneme : MonoBehaviour

{

[SerializeField]

Transform exit;

[SerializeField]

Transform[] wayPoint;

[SerializeField]

float navigate;

[SerializeField]

int health;

[SerializeField]

int rewardAmount;

int target = 0;

Transform enemy;

Collider2D enemyCollider;

Animator anim;

float navigateTime = 0;

bool isDead = false;

public bool IsDead

{

get

{

return isDead;

}

}

void Start()

{

enemy = GetComponent<Transform>();

enemyCollider = GetComponent<Collider2D>();

anim = GetComponent<Animator>();

Manager.Instance.RegitEnemy(this);

}

void Update()

{

if (wayPoint != null&& isDead==false)

{

navigateTime += Time.deltaTime;

if (navigateTime > navigate)

{

if (target < wayPoint.Length)

{

enemy.position = Vector2.MoveTowards(enemy.position, wayPoint[target].position, navigateTime);

}

else

{

enemy.position = Vector2.MoveTowards(enemy.position, exit.position, navigateTime);

}

navigateTime = 0;

}

}

}

void OnTriggerEnter2D(Collider2D collision)

{

if (collision.tag == "MovePoi")

{

target += 1;

}

else if (collision.tag == "Finish")

{

Manager.Instance.RoundEscaped += 1;

Manager.Instance.TotalEscaped += 1;

Manager.Instance.UnregEnemy(this);

Manager.Instance.IsWaveOver();

}

else if (collision.tag == "Project")

{

Project newP = collision.gameObject.GetComponent<Project>();

EnemiHit(newP.AttackDamage);

Destroy(collision.gameObject);

}

}

public void EnemiHit(int hitPoints)

{

if (health-hitPoints>0)

{

health -= hitPoints;

//damage

anim.Play("Hit");

}

else

{

anim.SetTrigger("didDie");

Die();

//die

}

}

public void Die()

{

isDead = true;

enemyCollider.enabled = false;

Manager.Instance.TotalKilled += 1;

Manager.Instance.addMoney(rewardAmount);

Manager.Instance.IsWaveOver();

}

}

Листинг кода Б.4 – TowerButton

public class TowerButton : MonoBehaviour

{

[SerializeField]

TowerControler towerObject;

[SerializeField]

Sprite dragSprite;

[SerializeField]

int towerPrice;

public TowerControler TowerObject

{

get

{

return towerObject;

}

}

public Sprite DragSprite

{

get

{

return dragSprite;

}

}

public int TowerPrice

{

get

{

return towerPrice;

}

}

}

Листинг кода Б.5 – TowerMen

public class TowerMen : Loader<TowerMen>

{

public TowerButton towerNtnPress{ get; set; }

SpriteRenderer spriterand;

private List<TowerControler> TowerLst = new List<TowerControler>();

private List<Collider2D> Buildlist = new List<Collider2D>();

private Collider2D buildTile;

void Start()

{

spriterand = GetComponent<SpriteRenderer>();

buildTile = GetComponent<Collider2D>();

spriterand.enabled = false;

}

void Update()

{

if (Input.GetMouseButtonDown(0))

{

Vector2 mousePoin=Camera.main.ScreenToWorldPoint(Input.mousePosition);

RaycastHit2D hit = Physics2D.Raycast(mousePoin, Vector2.zero);

if (hit.collider.tag=="Towerside")

{

buildTile = hit.collider;

buildTile.tag = "TowersideFull";

RegisterBuildSite(buildTile);

PlaceTower(hit);

}

}

if (spriterand.enabled)

{

FollowMouse();

}

}

public void RegisterBuildSite(Collider2D buildTag)

{

Buildlist.Add(buildTag);

}

public void RegisterTower(TowerControler tower)

{

TowerLst.Add(tower);

}

public void RenameTagBuild()

{

foreach(Collider2D buildTag in Buildlist)

{

buildTag.tag = "Towerside";

}

Buildlist.Clear();

}

public void DestroyAllTower()

{

foreach(TowerControler tower in TowerLst)

{

Destroy(tower.gameObject);

}

TowerLst.Clear();

}

public void PlaceTower(RaycastHit2D hit)

{

if (!EventSystem.current.IsPointerOverGameObject() && towerNtnPress!=null)

{

TowerControler netTower = Instantiate(towerNtnPress.TowerObject);

netTower.transform.position = hit.transform.position;

BuyTower(towerNtnPress.TowerPrice);

RegisterTower(netTower);

DisabelDrag();

}

}

public void BuyTower(int price)

{

Manager.Instance.sutractMoney(price);

}

public void SelectTower(TowerButton towerSelect)

{

if (towerSelect.TowerPrice<=Manager.Instance.TotatlMoney)

{

towerNtnPress = towerSelect;

EnabelDrag(towerNtnPress.DragSprite);

}

}

public void FollowMouse()

{

transform.position = Camera.main.ScreenToWorldPoint(Input.mousePosition);

transform.position = new Vector2(transform.position.x, transform.position.y);

}

public void EnabelDrag(Sprite sprite)

{

spriterand.enabled =true;

spriterand.sprite = sprite;

}

public void DisabelDrag()

{

spriterand.enabled = false;

}

}

Листинг кода 6 – Project

public enum projectType

{

rock, arrow, fireball

}

public class Project : MonoBehaviour

{

[SerializeField]

int attackDam;

[SerializeField]

projectType pType;

public int AttackDamage

{

get

{

return attackDam;

}

}

public projectType Ptype

{

get

{

return pType;

}

}

}

Листинг кода Б.7 – TowerControler

public class TowerControler : MonoBehaviour

{

[SerializeField]

float timeBetweenAttack;

[SerializeField]

float attackRadius;

[SerializeField]

Project project;

Eneme targetEneme = null;

float attackCouter;

bool isAttack = false;

void Start()

{

}

void Update()

{

attackCouter -= Time.deltaTime;

if (targetEneme == null||targetEneme.IsDead)

{

Eneme nearestEnemy = GetNearesEnemy();

if (nearestEnemy != null&& Vector2.Distance(transform.position, nearestEnemy.transform.position) <=attackRadius)

{

targetEneme = nearestEnemy;

}

}

else

{

if (attackCouter <= 0)

{

isAttack = true;

attackCouter = timeBetweenAttack;

}

else

{

isAttack = false;

}

if (Vector2.Distance(transform.position, targetEneme.transform.position) > attackRadius)

{

targetEneme = null;

}

}

}

public void FixedUpdate()

{

if (isAttack == true)

{

Attack();

}

}

public void Attack()

{

isAttack = false;

Project newProject = Instantiate(project) as Project;

newProject.transform.localPosition = transform.localPosition;

if (targetEneme == null)

{

Destroy(newProject);

}

else

{

//move project to enemy

StartCoroutine(MoveProject(newProject));

}

}

IEnumerator MoveProject(Project project)

{

while(GetTargetDistane(targetEneme)>0.20f && project != null && targetEneme != null)

{

var dir = targetEneme.transform.localPosition - transform.localPosition;

var angleDirection = Mathf.Atan2(dir.y, dir.x) \* Mathf.Rad2Deg;

project.transform.rotation = Quaternion.AngleAxis(angleDirection, Vector3.forward);

project.transform.localPosition = Vector2.MoveTowards(project.transform.localPosition, targetEneme.transform.localPosition, 5f \* Time.deltaTime);

yield return null;

}

if(project!=null|| targetEneme == null)

{

Destroy(project);

}

}

private float GetTargetDistane(Eneme thisEneme)

{

if (thisEneme == null)

{

thisEneme = GetNearesEnemy();

if (thisEneme == null)

{

return 0f;

}

}

return Mathf.Abs(Vector2.Distance(transform.localPosition, thisEneme.transform.localPosition));

}

private List<Eneme> GetEnemes()

{

List<Eneme> enemiInRange = new List<Eneme>();

foreach(Eneme eneme in Manager.Instance.EnemeList)

{

if(Vector2.Distance(transform.localPosition, eneme.transform.localPosition) <= attackRadius)

{

enemiInRange.Add(eneme);

}

}

return enemiInRange;

}

private Eneme GetNearesEnemy()

{

Eneme nerestEneme = null;

float smallesDistance = float.PositiveInfinity;

foreach(Eneme eneme in GetEnemes())

{

if (Vector2.Distance(transform.localPosition, eneme.transform.localPosition) < smallesDistance)

{

smallesDistance = Vector2.Distance(transform.localPosition, eneme.transform.localPosition);

nerestEneme = eneme;

}

}

return nerestEneme;

}

}

Листинг кода Б.8 – Menu

public class Menu : MonoBehaviour

{

// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void Update()

{

}

public void Play()

{

SceneManager.LoadScene(1);

}

public void Quite()

{

Application.Quit();

}

}

Листинг кода Б.9 – Restart

public class Restart : MonoBehaviour

{

public void RestartLevel()

{

SceneManager.LoadScene("SampleScene");

}

}

Листинг кода Б.10 – Pause

public class Pause : MonoBehaviour

{

[SerializeField]

GameObject pause;

void Start()

{

pause.SetActive(false);

}

// Update is called once per frame

void Update()

{

if (Input.GetKeyDown(KeyCode.Escape))

{

pause.SetActive(true);

Time.timeScale = 0;

}

}

public void PauseOff()

{

pause.SetActive(false);

Time.timeScale = 1;

}

public void Menu()

{

SceneManager.LoadScene(0);

Time.timeScale = 1;

}

}