КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота
на здобуття освітнього рівня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

РОЗРОБКА ІГРОВИХ ПРОГРАМ. РОЗРОБКА 3D ГРИ ЖАНРУ TOWER DEFENSE

Виконав студент 4-го курсу
Денис ШЕСТЕРНЯК _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Євгеній ІВАНОВ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто та допущено до захисту
на засіданні кафедри інтелектуальних програмних систем
«___» _____ 202_ р.,
протокол № __
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 42 сторінки, 15 ілюстрацій, 17 джерел посилання, 0 додатків.

TOWER DEFENSE, UNITY, ВЕЖА, ВОРОГ, ГРАВЕЦЬ, 3D, МОДЕЛЬ, ТЕСТУВАННЯ.

Об’єктом роботи є процес створення гри жанру Tower Defense. Предметом роботи є 3D гра жанру Tower Defense на двигуні Unity.

Метою роботи є створення прототипу однокористувацької трьохмірної гри жанру Tower Defense для комп’ютерів.

Методи розроблення: комп’ютерне моделювання, розробка програмного продукту на основі каскадної моделі. Інструменти розроблення: інтегровані середовища програмування та розробки Unity та JetBrains Rider 2019.3.4, Blender, Photoshop, мова програмування C#.

Результати роботи: виконано загальний огляд ігр та методів їх розробки, розроблено 3D гру жанру Tower Defense, яка надає можливість гравцеві можливість знищувати хвилі ворогів для розваги.

Гра не може застосовуватися для комерційного використання, оскільки є лише прототипом створенним для дослідження, а не повноцінною грою.

ЗМІСТ

ВСТУП

Індустрія комп'ютерних ігор з'явилася відносно недавно, близько 30 років тому, але вже змогла розвинутися в величезну галузь з колосальними доходами в кілька мільярдів доларів на рік. Зрозуміти подібне раптове зростання популярності віртуальних розваг дуже просто: все це завдяки широкому розповсюдженню комп'ютерних технологій, в тому числі появі мережі Інтернет. Завдяки цьому, на відміну від інших видів розваг, комп'ютерні ігри більш доступні для кінцевого користувача. Для того що б просто пограти гравцеві потрібно мати тільки комп'ютер або ігрову приставку і копію самої гри, а з широким розповсюдженням мережі Інтернет заради отримання копії гри не потрібно виходити з дому. Більш того, споживачеві не потрібно мати особливих знань для того щоб вибрати підходящу для нього гру, в той час як для більшості інших видів розваг необхідно розбиратися як мінімум в необхідної екіпіровки.

Так само варто взяти до уваги, що комп'ютерні ігри останнім часом перестали позиціонуватися як програми тільки для відпочинку і розваг. Наприклад, сьогодні, завдяки використанню ігрових технологій, створюються спеціальні комплекси по симуляції, які служать для навчання фахівців в різних областях: від лісорубів до пілотів реактивних літаків.

В Україні ж, на жаль, все трохи інакше. Ігрова індустрія у нас, так само як і в більшості інших країн пострадянського простору, розвинена дуже погано. Пов'язано це з тим, що культура комп'ютерних розваг прийшла до нас занадто пізно і практично не розвивалася. Через це, навіть при досить високому попиті, ми маємо дуже малу кількість компаній-розробників, які можуть скласти конкуренцію закордонним компаніям.

Тому розвиток технологій в даному напрямку можна вважати одним з найбільш перспективних, особливо в нашій країні

Об'єктом дослідження є розробка комп'ютерних ігор.

Предмет дослідження: технології розробки комп'ютерної гри жанру tower defense.

Мета випускної кваліфікаційної роботи - розробити прототип комп'ютерної гри жанру tower defense засобами середовища Unity.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1) вивчити особливості і стан комп'ютерної індустрії України;
2) вибрати жанр, вид і платформу для комп'ютерної гри;
3) розробити сценарій, концепцію основних елементів;
4) вибрати і вивчити засіб реалізації;
5) підготувати необхідні для гри анімації;
6) реалізація прототипу гри.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПОНЯТТЯ «КОМП'ЮТЕРНА ГРА» І ОГЛЯД ТЕХНОЛОГІЙ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР

1.1 Дослідження поняття і класифікації ігор

Терміном комп'ютерна гра (або іноді використовують неоднозначний термін відеогра) позначається комп'ютерна програма, яка служить для організації ігрового процесу (геймплея), зв'язку з партнером по грі, або сама виступає в якості партнера. Комп'ютерні ігри часто створюються на основі сторонніх джерел, таких як фільми або книги, але останнім часом стали з'являтися зворотні випадки, коли за відомою ігрової серії починають випускати додаткові матеріали, що розширюють всесвіт гри.

Більш того спеціально розроблені ігри можуть виступати в якості навчального матеріалу або дозволяють використовувати гравців в науково-дослідних цілях. Такі ігри рідко випускаються в широкі маси. За деякими ігор проводяться аматорські та професійні змагання, які називаються кіберспорту.

Комп'ютерні ігри надали настільки істотний вплив на суспільство, що останнім часом в інформаційних технологіях з'явилася стійка тенденція до Гейміфікації для неігрового прикладного програмного забезпечення. Таким чином, в деяких європейських школах почали використовувати відому гру Minecraft для навчання, а для потреб армій стали створювати спеціальні симулятори для тренувань солдатів. Комп'ютерні ігри так само з 2011 року офіційно визнані урядом США і американським Національним фондом окремим видом мистецтва, поряд з театром і кіно, а в Китаї кіберспорт офіційно був зарахований до видів спорту.

З усього цього випливає, що комп'ютерні ігри щільно влилися в нашу нинішню життя. Більш того, сфера їх використання за останні 10 років сильно зросла, тепер ігри використовують не тільки для розваги і відпочинку, але і для навчання людей і проведення наукових досліджень.

Комп'ютерні ігри класифікують за кількома основними ознаками:

- жанр;
- кількість гравців;
- візуальне уявлення;
- платформа

Жанр гри визначається метою і основною механікою гри. Основні жанри представлені в таблиці 1.

Таблиця 1 — Жанрова класифікація

Жанр	Особливості	Основні піджанри
Аркада (Arcade)	гри з навмисно примітивним ігровим процесом	• Платформер • Файтінг • Скролер
Пригодницька гра (Adventure)	головною частиною гри є історія	• Квест • Візуальна новела
Ролева гра (RPG)	головна особливість гри, гравець практично не обмежений у виборі ігрових предметів, напарників і діалогів.	• Тактична • Екшен RPG • Японська (Західна) • ZPG
Екшен (Action)	гра, яка характеризує частим і активним натисканням кнопок управління.	• Beat’em Up • Шутер • Слешер
Стратегічна гра (Strategy)	Основа гри - необхідність гравцеві робити нетривіальний вибір.	• Покрокова/У реальному часі • Глобальна • Варгейм • Економічна
Комп'ютерний симулятор (Simulator)	гравець робить безліч вправ і відточує свою техніку.	• Технічні • Симулятори життя
Головоломка (Puzzle)	вимагає аналітичного мислення.	
Навчальна гра (Educational)	гравець навчається під час виконання яких-небудь дій в грі	
Іграшки (Toys)	програми, взаємодіючи з якими, гравець отримує задоволення.	

Представлені тут жанри не можна назвати повними, так як останнім часом стали з'являтися ігри власних жанрів, які можна віднести як до одного з представлених тут жанрів, так і до самостійного окремо від них.

За кількістю гравців ігри поділяються на два види:

- однокористувацькі;
- розраховані на багато користувачів;

За візуального уявлення комп'ютерні ігри можна розділити на наступні види:

- текстові - мінімальне графічне представлення, спілкування з гравцем проходить за допомогою тексту;
- 2D - всі елементи намальовані у вигляді двовимірної графіки (спрайтів);
- 3D - всі елементи намальовані у вигляді тривимірної графіки (3D - моделі).

В даний список не включені такі категорії як 2,5D і псевдо-3D, так як вони є лише різновидами представлених категорій і технології їх створення практично не відрізняються від технологій створення основних представлених критеріїв.

За типом платформи:

- персональні комп'ютери;
- ігрові приставки / консолі;
- мобільні телефони;

На жаль, представлена тут класифікація не є повною і може бути доповнена. Так, наприклад, до жанрів можна додати такий специфічний вид ігор як музичні ігри, де основна увага приділяється музиці і / або звукам, а на багатокористувацькі ігри в свою чергу діляться на кілька підкатегорій. Але варто розуміти, що цієї класифікації досить для визначення більшості з існуючих ігор, так як на сьогоднішній день не існує однозначної і повної класифікації для комп'ютерних ігор. Це відбувається через те, що багато ігор не можна віднести до будь-яких критеріїв. Наприклад, гра може відповідати одразу кільком жанрам, вийти відразу на декількох платформах або мати як одно користувацький режим гри, так і розрахований на багато користувачів.

Все це обумовлено тим, що ігрова індустрія, на відміну від інших сфер розваг, з'явилася порівняно недавно. Але як вже було сказано, цієї класифікації досить, для того що б визначити більшість комп'ютерних ігор існуючих на даний момент.

Грунтуючись на даній класифікації комп'ютерних ігор було прийнято рішення розробити трьохвимірну однокористувацьку гру жанру Tower Defense для персональних комп'ютерів. Дане рішення було прийнято тому, що Tower Defense володіє простою і зрозумілою механікою, що означає легкість в розробці для початківців ігрових розробників. Так само трьохвимірні ігри цікавіші у виробництві, на відміну від двовимірних.

1.2 Загальний алгоритм реалізації проекту

Загальний алгоритм розробки комп'ютерної гри мало чим відрізняється від алгоритму розробки будь-якого іншого програмного продукту і включає в себе 3 великих етапи:

1) проектування;

2) розробка;

3) видання і підтримка.

На етапі проектування визначаються мета гри і засоби її розробки.

При визначенні мети виділяються ідея, жанр і сетинг гри. ідея - це те, що буде спонукати гравця грати в створювану гру, і вона дуже тісно пов'язана з жанром. Так, наприклад, основна ідея RPG - дозволити гравцеві прожити свою роль так, як йому захочеться, а основна ідея шутера - дозволити гравцеві взяти участь в реальних чи вигаданих бойових діях. Таким чином, визначивши основні ідеї гри, жанр буде підібраний практично відразу.

Визначившись з жанром і ідеєю гри, наступним кроком буде вибір сетинг. Сетинг - це середовище, в якому буде відбуватися основна дія гри. Він визначає місце, час і умови дії. Вибір сетинга може сильно полегшити розробку сценарію для гри, тому його краще вибирати заздалегідь і грунтуючись на смаки цільової аудиторії.

До засобів розробки в першу чергу відносять програмний код і ігровий двигун. Від їх грамотного вибору залежить як швидкість самої розробки, так і працездатність самого продукту надалі. Програмний код в першу чергу залежить від платформи, для якої буде створюватися комп'ютерна гра. Наприклад, якщо гра створюється для браузерів, то логічно буде використання мови Java або Flash, але, якщо гра створюється для персонального комп'ютера, оптимальним вибором буде, наприклад, мова програмування C #.

Ігровий двигун відповідає за низькорівневий опис фізики об'єктів, правил рендерингу графіки і ін. При виборі ігрового двигуна першим справою дивляться на його доступність і вже зроблений вибір мови програмування. Наприклад, ігровий движок Unity дозволяє розробляти ігри на мові C # і він є безкоштовним, якщо середній дохід кампанії не перевищує $ 100000 на рік.

Після вибору мети гри і засобів розробки, починається другий етап реалізації проекту - розробка.

Розробка найбільший і найдовший етап реалізації проекту, він включає в себе велику кількість кроків, без яких неможливо створити працездатний продукт.

Насамперед потрібно визначитися з сюжетом і ігровою механікою. Ігрова механіка ґрунтується на цілі гри, вона визначає всі об'єкти і правила, за якими гравець буде взаємодіяти з ними.

Зазвичай паралельно з розробкою ігрової механіки йде написання сюжету гри. Сюжет грає не останню роль, він визначає те, наскільки буде гравцеві цікаво грати в вашу гру. Сюжет представляють в двох варіантах: літературний і режисерський сценарій. Літературний сценарій описує основні

події і персонажів гри, які беруть участь в грі. Режисерський ж являє собою докладний опис рівнів гри, подій, які на цих рівнях відбуваються.

Так само на даному етапі починається раннє опрацювання графічної складової і дизайну гри. На основі сюжету і заздалегідь обумовленого дизайну, створюються ранні концепт-арти, на основі яких згодом буде опрацьовано основний вид гри і персонажів.

Після розробки сюжету і ігрової механіки починається найважливіша частина - розробка самої гри.

На основі сюжету і концепт-артів починається створення персонажів і об'єктів гри, паралельно з цим йде розробка ігрових рівнів. При розробці ігрових рівнів спочатку створюється його спрощений план, на якому схематично зображено сам рівень, а так же зображені предмети, з якими буде згодом взаємодіяти гравець.

Слідом після цього створюється перша версія рівня. Зазвичай, вона являє собою просто голу локацію, з мінімумом необхідних для проходження предметів. Ця версія рівня служить для того, що б протестувати рівень на прохідність. Після тесту, рівень починають поступово заповнювати іншими об'єктами.

Незабаром після створення перших рівнів триває складання першого прототипу гри, який називають альфа-версією гри. Вона необхідна для того, щоб розробник міг провести тестування (альфа-тестування) основний механіки гри, і перевірити наскільки вона відповідає заявленим вимогам. Часто в альфа-версії гри, у об'єктів навіть немає текстур або вони взагалі представлені у вигляді абстрактних об'єктів.

Якщо альфа-версія гри успішно проходить тестування, настає наступний етап розробки - опрацювання механіки і об'єктів гри.

На даному етапі йде доробка рівнів і механіки гри, і починають додавати перші сюжетні події в гру, такі як відеоролики, сюжетні діалоги і кат-сцени. Так само виправляються перші помилки і несправності в коді гри, які були виявлені при тестуванні альфа-версії гри.

Після цього настає етап створення другого прототипу гри, або, як прийнято говорити, бета-версії. Бета-версія служить для того, що протестувати гру на несправності, фактично бета-версія являє собою практично готову гру. У ній можуть бути відсутні які-небудь незначні елементи, які не впливають на геймплей гри. При тестуванні бета-версії гри перевіряється абсолютно все. Часто, особливо, якщо гра є мультіплеєрною, на тестування запрошуються звичайні гравці - це дозволяє сильно прискорити час проведення тестування, а так само зняти частину навантаження з команди розробки.

Якщо гра проходить бета-тестування, вона відправляється на остаточне доопрацювання і виправлення критичних помилок, після чого йде збірка фінальної версії гри і слідом настає реліз гри.

Після релізу гри подальша її підтримка. Підтримка полягає у випуску патчів (файлів виправлень помилок в готовому продукті). Також для того, що б продовжити життєвий цикл гри, для неї випускають додатковий контент у вигляді DLC (Downloadable content), які додають різні предмети і / або можливості для гравця.

Таким чином, стало зрозуміло, що етапи розробки комп'ютерних ігр мало відрізняються, від етапів розробки будь-якого іншого програмного продукту.

1.3 Засоби створення комп'ютерних ігор

1.3.1 Аналіз ігрових двигунів

Основними засобами розробки ігор є ігрові движки, які відповідають за низькорівневий опис фізики гри, правил рендерингу графіки та ін. І графічні редактори для відтворення графіки.

Для розробки комп'ютерної гри основним засобом є ігровий движок - програмний засіб, для розробки комп'ютерних ігор. Ігрові движки відразу включають в себе всі необхідні алгоритми для правильного функціонування гри і її розробки.

В даний час існує величезна кількість різних ігрових движків. Основні їх відмінності полягають в підтримуваних мовами програмування, функціональності і, що не менш важливо, в вартості ліцензії. При виборі середовища розробки основна увага була приділена саме на ці параметри. Для аналізу обрали наступні ігрові двигуни Unreal Engine, Unity, і Game Maker: Studio.

Game Maker: Studio [4] - ігровий движок розроблений і підтримуваний компанією YoYo Games, написаний на Delphi (наступна версія на C #). На даний момент є одним з найбільш популярних ігрових движків для розробки двовимірних ігор для всіх сучасних популярних платформ. Для розробки використовує власну спрощену мову програмування GML (Game Maker Language).

Даний ігровий движок має кілька версій поширення:

- Trial;
- Desktop;
- Web;
- UWP;
- Mobile;

Кожна з цих версій (крім Trial) відрізняється платформою, для якої буде створена гра. Trial - версія є безкоштовною і надається для тих, хто хоче спочатку випробувати ігровий движок, має ряд обмежень на кількість об'єктів використовуваних в грі, а також дозволять компілювати проект тільки для тесту на операційній системі Windows.

Unreal Engine [5] - ігровий движок розробляється і підтримується компанією Epic Games.

Перша гра, створена на цьому движку - Unreal - з'явилася в 1998 році. З тих пір різні версії движка були використані в більш ніж сотні ігор і інших проектів.

Написаний на мові C ++, движок дозволяє створювати ігри для більшості операційних систем і платформ: Microsoft Windows, Linux, Mac OS і Mac OS X; консолей Xbox, Xbox 360, Xbox One, PlayStation 2, PlayStation 3, PlayStation 4, PSP, PS Vita, Wii, Dreamcast, GameCube і ін., А також на пристроях керованих системою iOS і Android.

Для спрощення портування движок використовує модульну систему залежних компонентів; підтримує різні системи рендеринга (Direct3D, OpenGL, Pixomatic), відтворення звуку (EAX, OpenAL, DirectSound3D), засоби голосового відтворення тексту, розпізнавання мови, модулі для роботи з мережею та підтримуваних пристроїв введення. 2 березня 2015 року Unreal Engine 4 став безкоштовним. Однак, розробники ігор, як і раніше, повинні передавати 5% від прибутку гри компанії Epic Games, але за умови, що доходи гри складають більше $ 3000 за квартал.

Unity [6] - засіб для розробки двох-і тривимірних ігор, що є одною з найбільш популярних на сьогоднішній день систем розробки. Дозволяє створювати додатки, що працюють під більшістю сучасними операційними системами (Windows, OS X, Windows Phone, Android, Apple iOS, Linux), а також на ігрових приставках Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One і MotionParallax3D дисплеях (пристрої для відтворення віртуальних голограм), наприклад, Nettlebox. Є можливість створювати додатки для запуску в браузерах за допомогою спеціального модуля Unity (Unity Web Player), а також за допомогою реалізації технології WebGL. Останні версії Unity дозволяють створювати додатки для окулярів віртуальної реальності.

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Движок підтримує дві сценарних мови: C #, JavaScript (модифікація). Розрахунки фізики виробляє фізичний движок PhysX від NVIDIA.

Проект в Unity ділиться на сцени (рівні) - окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, власне, об'єкти (моделі), так і порожні ігрові об'єкти - об'єкти, які не мають моделі («пустишки »). Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти.

Unity поширюється безкоштовно, але крім безкоштовної, існують чотири збірки - стандартна Unity, Unity iOS Pro, Android Pro і командна ліцензія. Вони відрізняються вартістю і функціональністю.

Безкоштовна версія має деякі обмеження, проте можливість поширювати ігри є, за умови, що щорічний дохід з гри не перевищує 100 000 $. З виходом Unity 5 багато обмежень Free версії були прибрані, але щорічний дохід з гри все також не повинен перевищувати 100 000 $.

Все це робить Unity одним з найбільш пріоритетних движків для початківців ігрових розробників.

1.3.2 Аналіз 2D графічних редакторів

Для обробки двомірної графічної складової гри були відібрані три графічних редактора: Adobe Photoshop, CorelDRAW Graphics Suite і PaintTool SAI.

Adobe Photoshop [7] - потужний графічний редактор для фотографій і малювання, розроблений і підтримуваний Adobe Systems. Працює з растровими зображеннями, проте має кілька векторних інструментів. Лідер ринку в області комерційних засобів редагування растрових зображень і найбільш відомий продукт фірми Adobe.

Спочатку замислювався як потужний графічний редактор для растрових зображень в поліграфії, але швидко став популярний в області Web-дизайн. Дозволяє працювати в більшості популярних колірних моделях.

Все це робить Adobe Photoshop одним з кращих графічних редакторів.

CorelDRAW Graphics Suite [8] - векторний графічний редактор, розроблений і поширюваний компанією Corel. Є кращим векторних редактором. У пакет CorelDRAW Graphics Suite також входить редактор растрової графіки Corel PHOTO-PAINT і інші програми - наприклад, для захоплення зображень з екрану - Corel CAPTURE. Програма векторизації растрової графіки Corel TRACE до 12 версії входила в пакет як самостійна програма.

CorelDRAW дозволяє імпортувати проекти в більшість сучасних графічних форматах, а малювання в векторах дозволяє безболісно змінювати розміри зображення, що безсумнівно є корисною властивістю при розробці графічного оформлення для двовимірних ігор.

PaintTool SAI [9] - програма, призначена для цифрового малювання в середовищі Microsoft Windows, розроблена японською компанією SYSTEMAX.

Дана програма в першу чергу спрямована на тих, хто для малювання використовує графічні планшети. Володіє простим і зрозумілим інтерфейсом, значно «легше» щодо Adobe Photoshop, завдяки чому має високу швидкість роботи і запуску навіть на відносно слабких комп'ютерах. Підтримує всі популярні графічні формати.

Для розробки проекту був обраний ігровий движок Unity, так як він простіше в освоєнні і при цьому надає величезний функціонал в створенні різного роду ігор, а безкоштовна версія не володіє будь-якими серйозними обмеженнями.

Так само він набув значного поширення серед розробників ігор в Росії, що ще сильніше спрощує його освоєння.

Для створення графічної складової гри було вирішено використовувати Adobe Photoshop, так як розробник має досвід роботи з цим графічним редактором.

1.3.3 Аналіз 3D графічних редакторів

Для обробки трьохвимірної графічної складової гри були відібрані три графічних редактора: Blender, 3Ds MAX, MAYA.

Blender [10] - безкоштовний редактор тривимірної графіки з відкритим вихідним кодом на Python. У програми широкий функціонал, тому його часто вибирають не тільки новачки, але і професіонали. Позіціонується програма, як рішення для редагування 3D-моделей, візуалізації, анімації, розробки комп'ютерних ігор, скульптинга і ефектів.

3Ds Max [11] - один з найстаріших редакторів тривимірної графіки від Autodesk, який з'явився ще на початку нульових. Тому більшість моделерів починали (або працюють) з ним. 3Ds Max є стандартом в індустрії для інтер'єрів та архітектури і взагалі фотореалістичної подачі. Використовують його тому, що в редакторі зручні системи візуалізації, море доповнень для інтер'єрів та архітектури і нескінченна кількість фотореалістичних бібліотек моделей меблів і благоустрою для інтер'єрів та екстер'єрів.

А ще 3Ds Max - універсальне рішення для роботи з тривимірною графікою, яке підійде розробникам ігор і художникам. Редактор закриває весь функціонал по моделюванню: тут зручно робити будь-які матеріали, а візуалізація гнучко підключається через зовнішні рендери (V-RAY або Corona), які дозволяють зробити фотореалістичну картинку проекту.

Maya [12] - ще один професійний редактор тривимірної графіки від Autodesk, який на відміну від 3Ds Max орієнтований в бік анімації і спецефектів. Його часто використовують в роботі над фільмами і мультиплікацією - в програмі безліч спеціальних функцій, наприклад, є динаміка твердих і м'яких тіл при візуалізації. Плюс, є можливість ліпити органічних героїв як в ZBrush за допомогою кистей на «цифровий глині».

Для створення моделей гри було вирішено використовувати Blender, так як розробник має досвід роботи з цим графічним редактором.

1.4 Аналіз існуючих розробок

В якості проекту був обраний однокористувацький 3D – tower defense, тому основних конкурентів варто шукати в тій же жанровій категорії. Але варто пам'ятати, що даний жанр втратив свою популярність, через що сучасні ігри цього жанру часто не є чистими tower defense. Найяскравішими представниками останнім часом були:

- Orcs Must Die;
- Bloons TD;
- Dungeon of the Endless;
- Dungeon Defenders

Orcs Must Die [14] - це відеоігра жанру tower defense, розроблена та опублікована Robot Entertainment та Mastertronic. Це tower defense, який уникає традиційного подання зверху вниз подібних ігор, замість цього використовує орієнтовану на дії третьої особи точку зору. Після демонстрації на Penny Arcade Expo East 2011, гра була випущена через Xbox Live Arcade 5 жовтня 2011 року та для ПК з Windows 12 жовтня 2011 року.

Історія обертається навколо Ордену, елітної фракції чарівників та воїнів, які охороняють Розколи, магічні отвори між людським світом та «Мертвим світом», які забезпечують джерелом магічної сили в обох світах. Використовуючи цю силу, Орден здатний підтримувати ідеальний світ для людства, використовуючи магію для маніпулювання природою. Для того, щоб захистити людський світ, вони побудували фортеці, що працюють за допомогою магії, по всьому Мертвому світу, щоб охороняти Рифти, особливо від змагальної фракції, відомої як `` Натовп " - грубої орди істот, таких як Орки, Огри та Гноли, які, незважаючи на те, що вони не є розумними, інакше становлять велику загрозу для людського світу через їх величезну кількість.

Після несподіваної атаки натовпу, персонаж гравця, відомий лише як Герой, опиняється останнім живим членом Ордена, оскільки, здається, натовп раптово отримав сплеск як сили, так і інтелекту . Взявши на себе оборону людського світу, він захищає фортеці від Натовпу по черзі (на здивування свого вчителя, який розповідає сюжетну лінію). Зрештою Герой дізнається, що Натовп отримав силу від Чарівниці, колишня учениця Ордену, яка, незважаючи на величезний потенціал для захисту людства, натомість вирішила шукати владу для себе та використовувала магію, щоб захопити контроль натовпу.

У міру того, як атаки Натовпу стають дедалі агресивнішими, Герой врешті-решт вирішує забезпечити безпеку людського світу, пройшовши назад через Розлом і закривши їх назавжди простим заклинанням. Хоча це означає, що Натовп ніколи не може дійти до людського світу, це також означає, що людство більше не в змозі використовувати магію для підтримки себе, і весь світ, який став залежним від магії, починає погіршуватися. У Мертвому світі Чарівниця стає безсилою, оскільки Ріфти більше не можуть забезпечити її магією, і вона залишається на милість Натовпу, який був приведений до свого початкового стану.

Orcs Must Die є варіантом гри tower defense; гравець, як бойовий маг, повинен захищати один або кілька рифтів на кожному рівні від натиску орків та інших істот, що виходять з одного або декількох входів і проходять рівень. Як правило, кожен орк, який досягнув розколу, віднімає очко від початкової

оцінки розколу, тоді як спеціалізовані орки можуть зняти більше очок; якщо Rift Score опускається до нуля, рівень втрачається, і гравцеві доведеться починати знову.

Армії орків прибувають дедалі складнішими хвилями, кожен рівень має від чотирьох до дванадцяти хвиль. Як правило, хвилі починаються через кілька секунд після того, як гравець перемагає попереднього, але приблизно кожні три хвилі на всіх рівнях складності Nightmare, гравцеві дається перерва, де вони можуть встановити пастки та ініціювати наступну хвилю, коли вони будуть готові.

Як військовий маг, гравець має можливість розміщувати численні пастки та озброюватися зброєю та спорядженням, щоб перемогти орків. Гравець вибирає кількість пасток та спорядження (до десяти) зі своєї поточної Книги заклинань; як тільки гравець почне розміщувати пастки, він буде прив'язаний до цього вибору. Пастки можна розміщувати на будь-якій відповідній поверхні, але кожна пастка коштує розміщення в ігровій валюті, і пастки матимуть змінні витрати залежно від їх ефективності. Зазвичай пастки працюють один раз, а потім мають період скидання, перш ніж вони знову активуються. До таких пасток належать підлогові, настінні та стельові пастки, а також кілька Стражів, які діятимуть за власним бажанням. Гравець отримує суму грошей, яку потрібно витратити на розміщення пастки на самому початку, і заробляє більше, перемагаючи орків і після завершення хвиль; гроші також можуть скинути певні монстри, які повинен зібрати гравець. Пастки можна розміщувати в будь-який час, в тому числі в бою, а між хвилями розміщені пастки можна вилучати та повертати за їх вартість. Пізніше гравець отримує доступ до Ткачів, які за різні витрати можуть посилити певні ефекти на гравця або пастки, розміщені у формі дерева технологій; ці ефекти тривають лише для цього рівня.

Після того, як гравець ініціює хвилю, він може бігати близько рівня, щоб брати участь у бою або розміщувати додаткові пастки. Гравець повинен стежити за своїм здоров’ям і маною, використовуваною для зброї, але орки можуть викинути зілля для здоров’я та мани, які поповнять їх; або ж гравець може зцілити, перебуваючи в безпосередній близькості від Рифта. Якщо гравець втратить все своє здоров'я або впаде з рівня, він знову відродиться в Rift, але це буде коштувати їм кількох Rift Points.

Після перемоги над останньою хвилею, гравцеві присуджується кількість черепів, до 5 (або на рівні складності Учня, обмежене двома) на основі їхнього балу (відображає, скільки очок Rift залишилось) і часу, необхідного для перемоги над усіма хвилями . Черепи використовуються для придбання поліпшень пасток та зброї в книзі заклинань гравця, таких як збільшення шкоди, зменшення вартості або підвищення їх ефективності проти орків. Після першого проходження кожного рівня гравець також винагороджується новим типом пастки за свою книгу заклинань, яку він може використовувати на

наступних рівнях. Ці вдосконалення залишаються за гравцем, так що вони можуть повернутися на попередні рівні за допомогою цих вдосконалених пасток, щоб заробити більше черепів.

Bloons Tower Defense [15] (також відома як Bloons TD) - це серія ігор в жанрі tower defense, відгалуження серії Bloons. Розробник: Ninja Kiwi. Гра була спочатку створена як браузерна, випущена в 2007 році і була доступна за допомогою Adobe Flash. Пізніші гри цієї серії стали підтримувати Android, iOS, Windows Phone, PlayStation Portable і Nintendo DSi.

Гравці намагаються не допустити ситуації, при якій повітряні кульки (звані bloons замість правильного англ. Balloons) досягають кінця шляху пересування. Для цього потрібно розмістити вежі або предмети, які можуть їх лопати. Деякі вежі можуть затримати повітряні кульки і дати іншим башт більше часу, щоб їх лопнути. Гроші заробляються при Лопань куль і при проходженні хвиль. Вони можуть бути витрачені на нові вежі або поліпшення для існуючих веж. Також в грі є предмети: шипи (road spikes), лопається певне число куль і вибухають ананаси (exploding pineapples).

Гра відноситься до жанру «баштова захист», тому в ній гравець може вибирати різні типи веж, які будуть лопати кулі. Якщо кулька досягає кінця шляху, гравець втрачає життя (або здоров'я); якщо залишилося 0 життів, гра закінчується. Кулі ніколи не відступають від заданого їм курсу. Однак, вони можуть досягти виходу, можуть бути лопнути і можуть повернутися в початок траси (це - здатність однієї з веж).

У грі є два класи повітряних куль: звичайні (в грі не названі ніяк) і кулі класу MOAB. Звичайні кулі: червоні, блакитні, зелені, жовті, рожеві, чорні, білі, залізні, кулі кольору зебри, райдужні, фіолетові і керамічні. Кулі MOAB-класу - це дирижаблі. MOAB клас включає в себе:

MOAB (Massive Ornary Air Blimp)
BFB (Brutal Flying Behemoth)
ZOMG (Zeppelin Of Mighty Gargantuaness)
DDT (Dark Dirigible Titan)
BAD (Big Airship of Doom)

Деякі кулі отримують додаткові можливості. Так, це кулі camo (багато вежі не стріляють в них), regen (можуть відновлюватися), fortified (лопнути куля стає в два рази складніше). З кожним рівнем сила нападаючих куль збільшується.

Вежі - основний спосіб захисту в Bloons TD. Кожна вежа має свою силу і правила використання. Часто вежі можуть знищувати тільки певні кулі, вони не здатні лопнути інші типи куль. Кожна вежа може бути поліпшена, але для цього доведеться витратити ігрову валюту («гроші»). Гроші можна заробляти за допомогою лопания куль і за допомогою бонусу, що видається за закінчення раунду. У Bloons TD 4 і наступних іграх є бананові ферми, які можуть бути

використані для отримання додаткових коштів в середині раунду (в кінці раунду в Bloons TD 4).

У більш пізніх іграх серії є кілька рівнів складності; наприклад, в BTD5 - чотири рівні складності, в Bloons Monkey City їх 5. У BTD 5 складність можна вибрати, від неї залежить кількість життів і вартість поліпшень. У Bloons Monkey City кожна окрема карта отримує оцінку складності.

Dungeon of the Endless [16] - це неправдива гра-захист вежі, розроблена Amplitude Studios. Це третя гра з їх вільно пов’язаних серій Endless, яка включає Endless Space та Endless Legend. Він був випущений в жовтні 2014 року для систем Microsoft Windows і Mac OS X, серпня 2015 року для пристроїв iOS і для Xbox One в березні 2016 року. Порти PlayStation 4 і Nintendo Switch випущені в травні 2020 року. Перероблена версія для пристроїв iOS і Android, "Dungeon of the Endless: Apogee" планується розпочати 16 березня 2021 року.

У грі гравець виконує роль тих, хто вижив у в'язничному космічному кораблі, його рятувальний капсул розбився на поверхню дивної планети. Щоб врятуватися, вони повинні пронести енергетичний кристал через кілька поверхів, кожен з яких наповнений рядом небезпечних істот. Щоб допомогти, гравець може попросити своїх вижилих досліджувати випадково сформовані рівні для збору ресурсів, щоб забезпечити живлення різних кімнат і побудувати башточки, щоб відбивати ворогів, коли вони переносять кристал з початкової точки в ліфт на наступний поверх.

Dungeon of the Endless - це неправдива гра, що має процедурно сформовані рівні та поняття permadeath, так що кожне проходження гри різне. Гра заснована на направленні тих, хто вижив у в'язничному космічному кораблі (екіпаж, в'язні та цивільні особи), які потрапили в аварію на чужу чужу планету, через кілька рівнів, щоб досягти втечі з планети. На початку кожної гри гравець вибирає двох персонажів, кожен із персонажів має різну статистику, таку як стан здоров’я, атака, сила оборони та швидкість руху, а також ряд здібностей. Гравець може керувати кожним персонажем окремо або в тандемі, направляючи його переходити в різні кімнати поточного рівня або відкривати двері на цьому рівні, а також ініціювати будь-які спеціальні здібності або лікувати їх. В іншому випадку персонажі будуть діяти автономно, наприклад, відбиваючись від ворогів, які потрапляють у кімнату, як не можуть. На першому рівні гри персонажі починатимуться в аварійному втечу з енергетичним кристалом, а на наступних рівнях - на ліфті, який вони взяли з попереднього рівня.

Гра розгортається переважно покроково, кожен хід позначається, коли відкриваються нові двері на рівні. На початку кожного ходу гравець отримує фіксовані суми з трьох ресурсів: промисловість, наука та їжа. Промисловість використовується насамперед при будівництві веж та генераторів ресурсів;

Наука використовується для дослідження нових типів башточок і генераторів, а їжа використовується для оздоровлення персонажів або підвищення рівня їх досвіду, що покращує їх статистику та дає їм нові здібності. У деяких кімнатах можуть бути предмети, які можна вилучити за більшу частину цих ресурсів. Крім того, у кімнатах буде пил, який живить кристал. Оскільки кристал набирає більше енергії, гравець може «активувати» обмежену кількість безперервних наборів кімнат, дозволяючи їм будувати генератори ресурсів та башти в приміщеннях, які були активовані. Гравці також можуть у будь-який час деактивувати кімнату, щоб перенаправити електроенергію в іншу кімнату, відключивши башти та генератори в цій кімнаті. В інших кімнатах можуть бути скрині з предметами, крамарі та дослідницькі станції, або вони можуть містити вижилих, яких гравці можуть завербувати до своєї партії (загалом до 4 символів).

Відкриття нових дверей також може призвести до появи монстрів, які з'являться в довільних кімнатах на карті в будь-якій кімнаті, яка не активована або не має власного джерела живлення. Монстри атакують і партію, і кристал; вбивство істот має шанс створити більше пилу для запуску кристала. Персонажі втратять здоров’я атаками і помруть, якщо воно опуститься до нуля, хоча як тільки всі новонароджені монстри будуть переможені, їхнє здоров’я повністю відновиться. Якщо атакувати кристал, він назавжди втратить пил, а якщо він опуститься до нуля, кристал буде знищений і гра закінчена. Під час атак монстрів гравець може вільно переміщати персонажів між відкритими кімнатами, щоб отримати найкраще тактичне розташування, наприклад, битися з монстрами поряд з численними башточками. Після перемоги будь-яких породжених монстрів гравець може вільно пересувати персонажів і будувати захист, поки не вирішить відкрити наступні двері.

Як тільки буде знайдена точка виходу на рівні, гравець повинен мати одного персонажа, який збирає кристал і повільно несе його від початку до точки виходу, тоді як інші персонажі захищають носія від полчищ монстрів, які з'являються в цей момент. Якщо кришталевий носій загине, гра закінчена, але якщо всі гравці, що вижили, потрапляють до кімнати з точкою виходу, гравець переходить на наступний рівень. Мета гри - пройти 12 із цих рівнів, на кожному з яких з’являються більш складні монстри, принаймні один персонаж несе кристал до кінцевої точки виходу.

Мета-гра існує в тому, що гравець може розблокувати додаткових персонажів для початку гри, набираючи нових персонажів, яких вони знаходять на рівнях, або завершуючи гру або три рівні з цим персонажем живим. До того ж, досягнувши певних цілей, гравець може розблокувати різні режими гри, наприклад, такий, коли персонажі починатимуть набагато сильніше за ворогів, але не відновлять здоров’я після перемоги хвилі монстра, змушуючи їх покладатися на їжу, лікувальні вежі або предмети для зцілення.

Гра підтримує онлайн-гру з до чотирма гравцями. Кожен гравець керує одним персонажем, і кожен отримує окремі ресурси при відкритті нових дверей. Нові персонажі можуть бути набрані одним гравцем, якщо в партії є місце, яке потім контролюється цим гравцем.

Dungeon Defenders [13] - це багатокористувацька відеоігра, розроблена Trendy Entertainment, яка поєднує в собі жанри Tower Defense та екшн-рольову гру. Він заснований на вітрині Unreal Engine 3 під назвою Dungeon Defense. Гра відбувається в фантастичній обстановці, де гравці керують молодими учнями чарівників та воїнів та захищаються від полчищ монстрів. Продовження Dungeon Defenders II вийшло в 2015 році.

Dungeon Defenders - це поєднання оборонних веж, рольових і екшен-пригод, де один-чотири гравці працюють разом, щоб захистити один або кілька кристалів Eternia від знищення хвилями ворогів, серед яких гобліни, орки, коболди, огри та виверни. . У грі є декілька рівнів, що складаються з приблизно п'яти загальних хвиль, у режимі кампанії, або інших рівнів, що є частиною проблем; деякі з цих рівнів можуть мати фінальну хвилю, яка включає битву боса проти унікального ворога. Персонажі гравців захищають кристали, магічно створюючи, підтримуючи та модернізуючи вежі або інші захисні елементи, які пошкоджують або відволікають ворожих монстрів, або використовуючи атаки ближнього та дальнього бою для безпосередньої перемоги над ворогами. На більш простих труднощах у гравців є невизначений час до хвилі, щоб вивчити карту рівня, щоб побачити, звідки походять монстри та тип або кількість монстрів, розмістити пастки та захист та керувати спорядженням персонажів. Пастки та захист обмежуються наявною маною, якою володіє персонаж - більше мани можна отримати, перемігши монстрів або відкривши скрині, що з’являються між хвилями, - а також загальною «цінністю захисту» рівня, обмежуючи кількість пасток, які можуть бути розміщеним. Персонажі можуть бути пошкоджені ворожими атаками і можуть бути вбиті в бою, але через кілька секунд відновляться, якщо не вибрано жорсткий режим. Якщо Кристал Етернії завдав занадто багато шкоди, він буде знищений, і гравці втратять цей рівень. Успішно захистивши хвилю, ви отримаєте очки досвіду персонажів на основі складності плюс додаткові бонуси, а саме запобігання пошкодженню кристалів, запобігання пошкодженню вашого персонажа та використання зброї лише для вбивства монстрів; Успішне проходження всіх хвиль рівня принесе гравцям великий приплив мани. Монстри також скинуть предмети, які можна продати за додаткову банківську ману.

Складність рівня встановлюється гравцем-хостингом після вибору рівня; сила, кількість та типи монстрів, з якими стикаються, залежить від цього вибору та кількості гравців у матчі, а також якості обладнання, яке може бути отримано як винагорода. Гравець має додаткові опції, такі як видалення

невизначеного часу між хвилями, замість цього примушуючи гравців готуватися до майбутньої хвилі за таймером зворотного відліку. Для рівнів кампанії гравці повинні пройти кожен рівень перед розблокуванням наступного, але можуть повернутися до попередніх рівнів, щоб спробувати покращити свою ефективність, кинути виклик рівню на вищій складності або просто перемолоти, щоб отримати кращі винагороди.

Персонажі стійкі для гравця. Гравець може керувати будь-якою кількістю символів, хоча символи зберігаються окремо між рейтинговими серверами Trendy та негранованою грою. Персонажі вибираються з доступних класів персонажів: чотири були поставлені разом з грою, тоді як додаткові класи були додані у вигляді завантажуваного вмісту у версії Microsoft Windows. [6] Кожен клас має унікальний набір пасток або захистів, певний набір зброї, яку вони можуть екіпірувати, і дві особливі здібності, які вони можуть використовувати в бою. По мірі того, як персонаж вирівнюється з накопиченим досвідом мани, гравець може розподіляти очки серед ряду характеристик, що впливають на персонажа, на пастки чи захист, які вони викладають. Додаткові бонуси до цих характеристик можуть виходити від спорядження, яким оснащений персонаж. Нове спорядження можна придбати, використовуючи банківську ману, або в магазині гри, або на аукціонах інших гравців, скинути вбивством монстрів або зібрати як нагороду після перемоги над хвилями ворогів. Банкова мана може бути витрачена на оновлення обладнання, що дозволяє гравцеві покращити бонуси, які обладнання надає персонажам. Гравець веде окремі запаси обладнання на серверах, що мають рейтинг та без рейтингу, але цей список буде загальним для всіх персонажів гравця в цьому режимі.

РОЗДІЛ 2 РОЗРОБКА ПРОЕКТУ

2.1 Опис цільової аудиторії

Для розробки комп'ютерної гри, як і для будь-якого іншого продукту, одну з найважливіших ролей відіграє визначення цільової аудиторії продукту. Коли визначена цільова аудиторія, розробнику набагато легше визначити багато аспектів продукту, що розробляється.

Для даного проекту, що розробляється була виділена досить велика цільова аудиторія, а саме молоді люди від 16 до 35 років. Вибір такої великої категорії людей обумовлюється відразу декількома причинами.

Перш за все, вибір такої категорії людей обумовлений середні віком активно грають в комп'ютерні ігри людей, який в Україні стабільно тримається в діапазоні від 30 до 33 років. Саме з цих міркувань обрана така верхня межа цільової аудиторії гри.

Нижня межа була обрана з тих міркувань, що хоча середній вік геймера і наближається до позначки 33 роки, реальний гравець розробляємої гри,

ймовірно, буде значно молодше. Це обумовлено тим, що люди починаючи з 30-35 річного віку, починають поступово переходити від жанрів з швидким геймплеєм до жанрів з більш повільним ігровим процесом.

Також не варто забувати про те, що дана категорія людей з більшою ймовірністю володіє достатніми коштами для купівлі розробляється гри, коли вона буде випущена. Обумовлено це тим, що проект відноситься до категорії малобюджетних комп'ютерних ігор, а значить більш доступний для широких мас потенційних покупців.

Таким чином, можна вважати, що виділена цільова аудиторія є основною для даного проекту, що розробляється.

2.2 Постановка завдання проекту

2.2.1 Актуальність, мета і призначення

На сьогоднішній день ігрова індустрія розвивається неймовірно швидко, кількість активних гравців зростає з кожним роком, як і прибуток, яка все за 6 виросла майже вдвічі і на кінець 2016 року становила більше 90 млрд доларів. Все це вказує на те, що розробка відеоігор є одним з найперспективніших напрямів в сфері розробки програмного софта.

Можна точно сказати, що розробляєма гра буде продаватися серед виділеної цільової аудиторії. Цьому сприяють вибрані жанр і стилізація проекту. Жанр tower defense досить популярний серед гравців, тому що являє собою простий і зрозумілий, а так само цікавий ігровий процес. Володіючи дуже низьким порогом входження, такі ігри доступні дуже широкому колу людей, в тому числі тим, хто іграми як засобом проведення власне дозвілля мало цікавиться.

Так само варто пам'ятати, що сучасне суспільство перенасичене великими проектами, що вимагають від гравця глибокого розуміючи механіки ігрового процесу. Для їх проходження найчастіше потрібна досить велика кількість часу. На їх фоні гра з простим і зрозумілим геймплеєм буде виглядати більш цікавою серед людей, яким складно приділяти досить багато часу.

На основі вище перерахованого легко виділяються мета і призначення проекту.

Мета проекту полягає в наданні кошти для приємного проведення часу.

Призначення проекту полягає в розвазі потенційного споживача.

2.2.2 Функціонал проекту

Проект являє собою комп'ютерну гру жанру трьохвимірний Tower Defense, основна мета якої полягає в розвазі і надання засобів для відпочинку та приємного проведення часу.

Перш за все, проект повинен відповідати наступним вимогам:

1. Гра повинна володіти простим і зрозумілим геймплеєм.
2. Основна механіка гри полягає в можливості гравця будувати вежі для знищення хвиль ворогів.

3. Не повинно бути занадто ускладнене проходження рівнів.

4. Вороги з'являються у певній стартовій точці та прямують у кінцеву точку обходячи перешкоди.

5. На рівні повинні бути присутніми декілька видів башт та ворогів.

6. При виграші або програші гравця, має бути можливість перейти на новий рівень або повторити цей.

Дані вимоги до функціональної частини є основними, повинні бути виконані в першу чергу.

2.2.3 Характеристика обладнання розробки

Так як для розробки був обраний ігровий движок Unity, а для написання і редагування програмного коду використовується JetBrains Rider 2019.3.4, то знадобиться обладнання, що володіє технічними характеристиками не нижче мінімально необхідних для коректно роботи даних засобів розробки. Тому розробка велася на комп'ютері з наступними параметрами:

• Процесор: Core i5-4590 3,3ГГц;
• Відеокарта: NVidia GeForce GTX 1050Ti;
• ОЗУ: 16 Гб;
• ОС: Windows 10.

2.3 Реалізація проекту

Tower Defense - це ігровий жанр, метою якого є знищення полчищ ворогів, перш ніж вони досягнуть місця призначення. Це робиться шляхом побудови веж, які атакують цих ворогів. Існує безліч варіацій жанру.

2.3.1 Етап розробки графічного оформлення

Варто взяти до уваги, що основні етапи реалізації проекту, а саме розробки графічного оформлення і розробка гри, проходили паралельно один одному. Цьому сприяв той факт, що для написання програмного коду необов'язково наявність готових анімацій та іншого, а для перевірки працездатності самого коду досить стандартних примітивів Unity. Так само такий спосіб реалізації проекту значно збільшує швидкість роботи.

Перш за все, варто виділити основні поняття, такі як тайл (Tile) і спрайт (Sprite).

Тайл (Tile) - невеликих розмірів повторюється фрагмент, який служить для побудови зображень великих розмірів (Тайловая графіка).

Часто використовується для створення рівнів для двовимірних ігор.

Спрайт (Sprite) - графічний об'єкт, який представляє собою растрове зображення. Використовується в комп'ютерній графіці як основна одиниця для анімацій двовимірних об'єктів.

В реалізації даного проекту було вирішено не створювати велику кількість графічних фрагментів і використовувати більш прості готові об`єкти в Unity та їх композицію.

Винятком є фрагменти показані на малюнках:

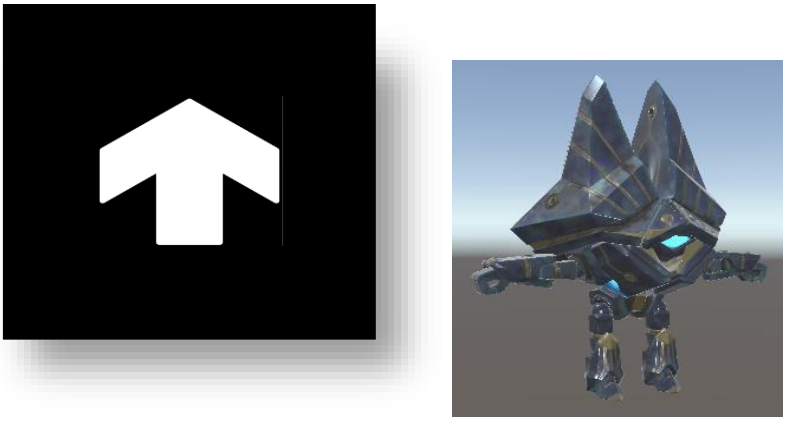

Рисунок 1 – графічні фрагменти гри

2.3.2 Початок розробки гри

Для реалізації комп'ютерної гри були обрані ігровий движок Unity і JetBrains Rider 2019.3.4 для опису програмного коду. При запуску Unity ми побачимо вікно проекту (рисунок 1), посередині знаходяться вікно Сцени, вікно Анімація, а так само вікно Ігри. перше служить для створення загальної композиції рівня і додавання нових об'єктів, друге представляє можливість для створення і редагування анімацій об'єктів, третє вікно представляє вид з камери показуючи, яким чином буде виглядати гра на даний момент.

Зліва знаходиться вікно ієрархії, тут показуються всі об'єкти, які беруть участь в цій сцені. З самого початку тут знаходиться тільки камера.
Справа вікно Інспектора, тут відображаються всі поточні властивості обраного об'єкта.

Знизу знаходяться три вікна: вікно Проекту, вікно Аніматора і Консоль. В вікні Проекту відображаються всі об'єкти, додані в поточну гру, в тому числі скрипти, анімації та інше. У вікні Аніматора створюються зв'язки і правила переходу між різними анімаціями. Консоль служить для -відображення помилок і виключень виникають під час роботи гри.

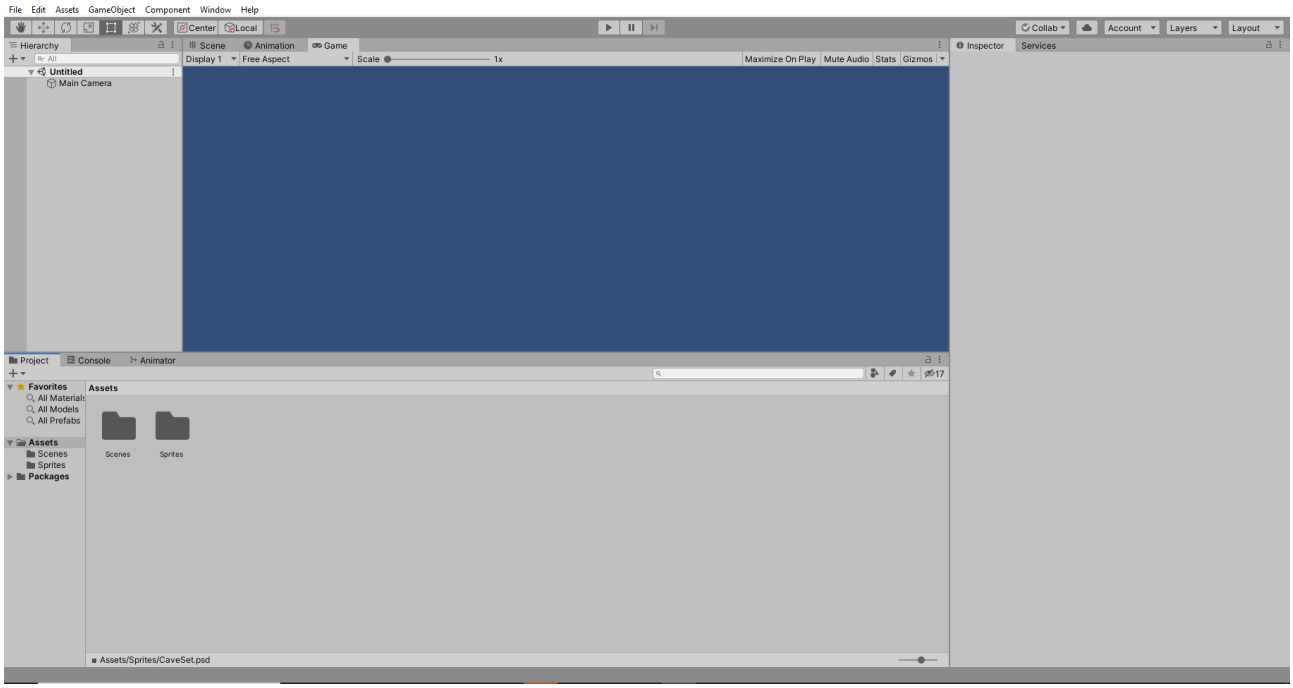

Рисунок 2 – Вікно проекту Unity

2.3.2 Створення ігрової дошки

Ігрова дошка є найважливішою частиною гри, тому ми створимо її першою. Це буде ігровий об’єкт із користувацьким компонентом GameBoard, який можна ініціалізувати 2D-розміром, для чого ми можемо використовувати значення Vector2Int. Дошка повинна працювати з будь-яким розміром, але ми вирішимо, який використовувати десь ще, тому для цього надамо загальнодоступний метод Initialize.

Окрім цього, ми візуалізуємо дошку за допомогою одного квадрата, що представляє землю. Ми не зробимо об'єкт дошки чотирикутником, натомість надамо йому дочірній об'єкт чотирикутник. При ініціалізації ми робимо масштаб XY землі рівним розміру дошки. Отже, кожна плитка - це одна квадратна одиниця.

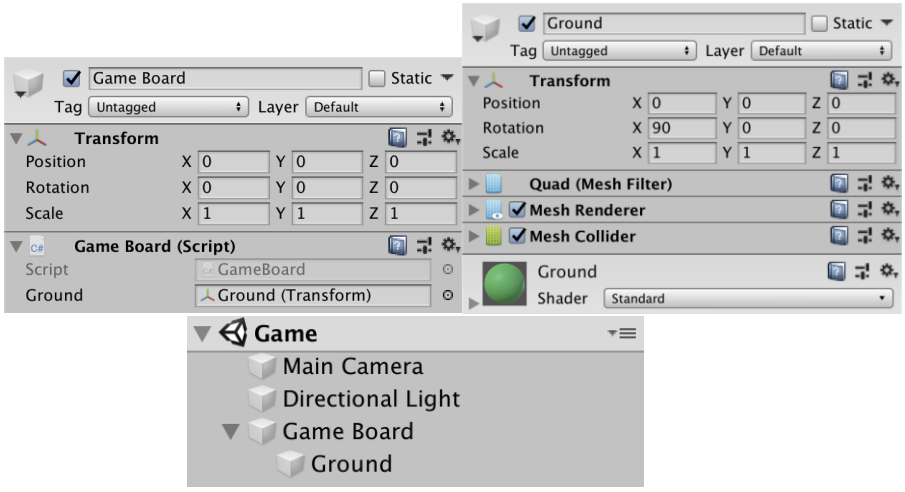

Рисунок 3 – Ігрова дошка

Дошка складається з квадратних плиток. Вороги зможуть ходити від плитки до плитки по краях, але не по діагоналі. Рух завжди буде спрямований до найближчого пункту призначення. Візуалізуємо напрямок руху на плитку зі стрілкою.

Ми будемо використовувати ігровий об’єкт для представлення кожної плитки у грі. Кожен з них матиме власний чотирикутник із матеріалом стрілки, як і дошка має свій чотирикутник. Ми також надамо плиткам власний компонент GameTile із посиланням на їх стрілку.

Наступним кроком є з’ясування правильного напрямку для кожної плитки. Ми робимо це, знаходячи шляхи, якими будуть йти вороги, щоб дістатися до місця призначення. Шляхи йдуть від плитки до плитки в північному, східному, південному або західному напрямку. Щоб спростити пошук, попросимо GameTile відстежувати посилання на своїх чотирьох сусідів. Відносини сусідів симетричні. Якщо плитка є східним сусідом другої плитки, то друга плитка є західним сусідом першої плитки. Додамо загальнодоступний статичний метод до GameTile, щоб встановити зв'язок між двома плитками.

Ми не будемо змушувати всіх ворогів постійно шукати шлях. Нам потрібно робити це лише один раз на плитку. Тоді вороги можуть запитати плитку, в якій вони плитці перебувають, куди йти далі. Ми зберігаємо цю інформацію в GameTile, додаючи посилання на наступну плитку на шляху. Крім того, ми також збережемо відстань до пункту призначення, виражене як кількість плиток, які ще потрібно ввести до досягнення пункту призначення. Це не корисна інформація для ворогів, але ми будемо використовувати її, коли знаходимо найкоротші шляхи.

GameBoard несе відповідальність за те, щоб усі його фрагменти містили правильні дані про шлях. Ми реалізуємо це, виконавши пошук у широту. Ми починаємо з плитки призначення, потім розвиваємо шлях до своїх сусідів, потім до сусідів цих плиток тощо. З кожним кроком відстань збільшується на одиницю, і доріжки ніколи не ростуть до плиток, які вже мають шлях. Це гарантує, що всі плитки в кінцевому підсумку вказують на найкоротший шлях до пункту призначення. Щоб здійснити пошук, ми повинні відстежувати плитки, які ми додали до контуру, але ще не виросли з них. Цю колекцію плиток часто називають межею пошуку. Важливо, щоб плитки оброблялись у тому самому порядку, що і додані до кордону, тож давайте використаємо Чергу. Ми закінчимо пошук не раз пізніше, тому визначте це як поле в GameBoard.

Сенс гри у захист вежі полягає в тому, щоб вороги не дійшли до місця призначення. Це робиться двома способами. По-перше, вбиваючи їх, а по-друге, уповільнюючи їх, щоб у вас було більше часу, щоб їх убити. На чотирикутниковій дошці основним способом приділити собі більше часу є збільшення відстані, яку вороги повинні пройти. Це робиться шляхом

розміщення на дошці перешкод, як правило, веж, які також вбивають ворогів, спочатку ми створимо перешкоди у вигляді стін.

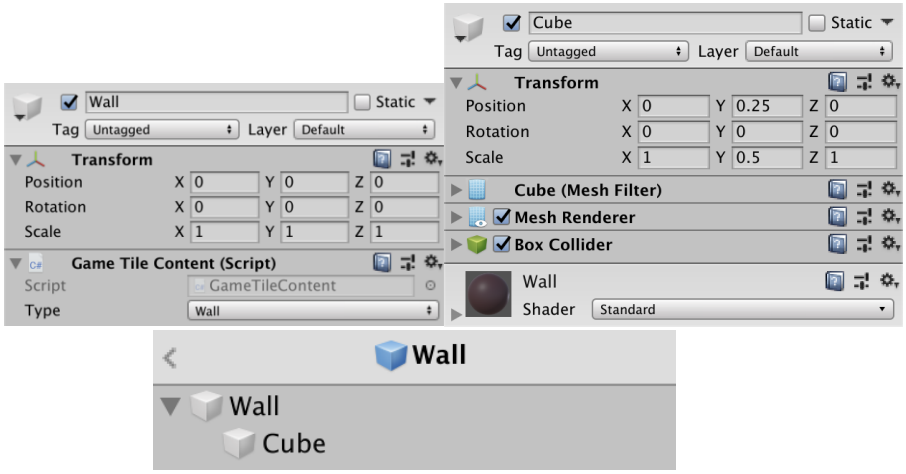

Рисунок 4 - Стіни

Ми лише підтримуватимемо перемикання між порожньою та настінною плиткою, не дозволяючи стінам безпосередньо замінювати пункти призначення. Тому робимо стіну лише тоді, коли плитка порожня. Крім того, ідея полягає в тому, що стіни перекриють пошук шляху. Але кожна плитка повинна мати шлях до місця призначення, інакше вороги можуть застрягти. Ще раз нам доведеться перевірити FindPaths на це та скасувати зміни, якщо ми створили недійсний стан дошки.

Візуалізація шляху дозволяє нам побачити, як працює пошук шляхів, і переконатися, що це справді правильно, але воно не призначене для показу гравцеві, принаймні не завжди. Тому зробимо можливим приховати стрілки.

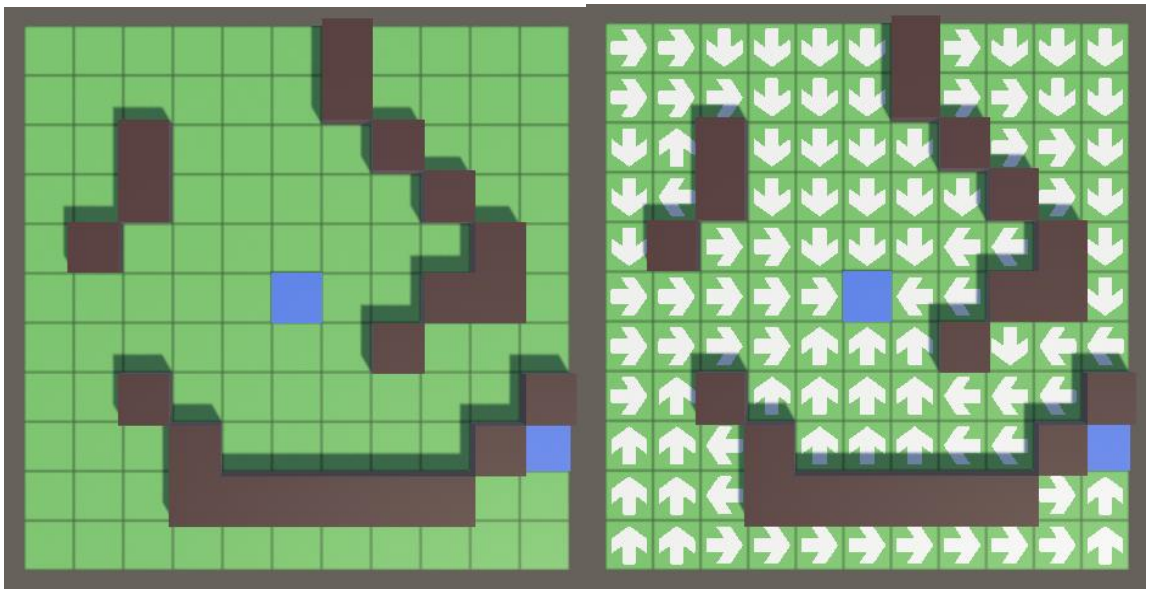

Рисунок 5 - Масштабована сітка, без і з візуалізацією шляху.

2.3.4 Створення ворогів

Гра має сенс лише за наявності ворогів, для чого потрібні точки їх появи. Отже, дійсна дошка повинна містити принаймні одну точку появи. Нам також

потрібно буде отримати доступ до точок появи пізніше при додаванні ворогів, тому використаємо список для відстеження всіх плиток з точками появи.

Створення ворога дещо схоже на створення вмісту плитки. Ми створюємо префаб, який потім кладемо на дошку. Ми створимо фабрику для ворогів, яка розмістить все, що вона створює, на власній сцені. Ця функціональність ділиться із фабрикою, яку ми вже маємо, тому давайте помістимо код для цього в загальний базовий клас GameObjectFactory. Ми можемо вистачити одного методу CreateGameObjectInstance із загальним параметром prefab, який створює та повертає екземпляр та опікується всім управлінням сценами. Зробимо метод захищеним, що означає, що він доступний лише для самого класу та всіх типів, які його розширюють. Це все, що робить базовий клас, він не призначений для використання як повнофункціональна фабрика. Тож позначимо його як абстрактний, що унеможливлює створення його екземплярів об’єктів.

Ворогам потрібна візуалізація, яка може бути будь-якою. Робот, павук, привид або щось простіше, як куб, саме цим ми і будемо користуватися. Але загалом ворог має 3D-модель довільної складності. Щоб полегшити підтримку моделі, ми будемо використовувати кореневий об’єкт для нашої ієрархії prefab’ів ворогів, до якого приєднаний лише компонент Enemy.

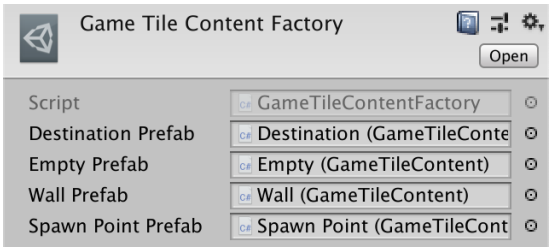

Рисунок 6 - Фабрика, що підтримує точки появи

Як тільки з’явиться ворог, він повинен почати рух по шляху до найближчого пункту призначення. Нам доведеться оживити їх, щоб це сталося.

Враховуючи плитку, з якої та плитку, до якої потрібно рухатись, вороги можуть визначити вихідну і кінцеву точку для подорожі однією плиткою. Ворог може взаємодіяти між ними, відстежуючи його прогрес. Після завершення процесу процес повторюється для наступної плитки. Але шляхи могли змінитися в будь-який час. Замість того, щоб з’ясовувати, куди рухатись, ми продовжуємо рухатись по запланованому маршруту та переоцінюватимемо, як тільки з’явиться наступна плитка.

```
public bool GameUpdate () {
    progress += Time.deltaTime * progressFactor;
    while (progress >= 1f) {
        if (tileTo == null) {
            OriginFactory.Reclaim(this);
            return false;
        }
        progress = (progress - 1f) / progressFactor;
        PrepareNextState();
        progress *= progressFactor;
    }
    if (directionChange == DirectionChange.None) {
        transform.localPosition =
            Vector3.LerpUnclamped(positionFrom, positionTo, progress);
    }
    else {
        float angle = Mathf.LerpUnclamped(
            directionAngleFrom, directionAngleTo, progress
        );
        transform.localRotation = Quaternion.Euler(0f, angle, 0f);
    }
    return true;
}
```

Лістинг 1 – Метод для руху ворогів

Переміщення між центрами плитки та раптова зміна напрямку виглядає чудово для абстрактної гри, де вороги ковзають кубиками, але в цілому плавніший рух виглядає краще. Першим кроком до цього є переміщення між краями плитки замість центрів.

Хоча вороги рухаються по стежках, в даний час вони ніколи не змінюють напрямку погляду. Щоб змусити їх подивитися, куди вони йдуть, вони повинні знати напрямок шляху, яким вони йдуть. Ще раз визначимо це, коли знайдені шляхи, щоб ворогам не потрібно було їх обчислювати. Замість того, щоб негайно переходити до нового напрямку погляду, краще, якщо ми інтерполюємо між обертаннями, як ми інтерполюємо між позиціями.

Ми можемо покращити рух, якщо вороги рухаються по кривій під час повороту. Замість того, щоб прямувати від краю до краю, ми змусимо їх рухатись по чверті кола. Центр цього кола лежить на куті, розділеному плитками Від і До, на тому самому краю, що ворог увійшов у плитку Від.

До цього моменту швидкість наших ворогів завжди становить одну плитку в секунду, незалежно від того, як вони рухаються всередині плитки. Але відстань, яку вони долають, залежить від їх стану, тому їх швидкість, виражена в одиницях за секунду, змінюється. Щоб підтримувати цю швидкість постійною, ми повинні регулювати швидкість прогресу залежно від стану.

У нас є потік ворогів, які є однаковими, що рухаються з однаковою швидкістю. Результат може виглядати більше як довга змія, ніж окремі вороги.

Зробимо їх трохи виразнішими шляхом рандомізації їх розміру, зміщення та швидкості.

2.3.5 Створення веж

Стіни лише уповільнюють ворогів, збільшуючи довжину шляху, яким вони повинні пройти. Але мета гри - усунути ворогів до того, як вони досягнуть місця призначення. Це робиться шляхом розміщення на дошці веж, які їх стріляють.

Башта буде обертатися, і оскільки у неї є колайдер, фізичному двигуну доведеться це відстежувати. Але нам насправді не потрібно бути настільки точним, тому що все, для чого ми використовуємо баштові коллайдери, це вибір клітин. Ми можемо обійтися наближенням. Відрегулюємо колайдер башти куба так, щоб він охоплював обидва куби.

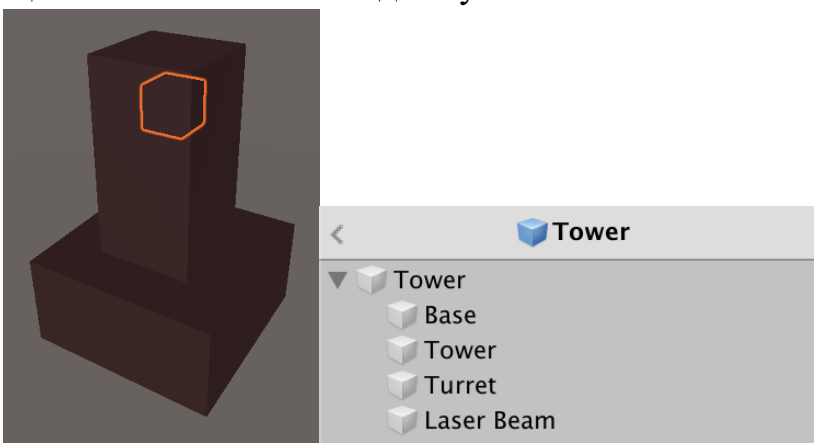

Рисунок 7 - Вежа

Наша вежа буде стріляти лазерним променем. Існує багато способів візуалізувати це, але ми просто використаємо напівпрозорий куб, який розтягнемо, щоб сформувати промінь. Кожній вежі знадобиться один свій, тому додамо його до prefab'a вежі. Помістімо його всередину башти, щоб він був прихований за замовчуванням, і надамо йому менший масштаб, наприклад 0,2. Зробимо це дочірньою структурою збірного кореня, а не куба башти.

Ймовірно, що гравець в кінцевому підсумку замінить стіни вежами. Спершу прибирати стіну незручно, і ворогам буде можливо пробратися крізь тимчасову щілину. Ми можемо зробити пряму заміну можливою, встановивши GameBoard.ToggleTower також перевіряє, чи є плитка на даний момент стіною в ній. Якщо так, то безпосередньо замініть його вежею. У цьому випадку нам не потрібно знаходити нові шляхи, оскільки плитка все ще блокує їх.

```
public void ToggleTower (GameTile tile) {
    if (tile.Content.Type == GameTileContentType.Tower) {
        updatingContent.Remove(tile.Content);
        tile.Content = contentFactory.Get(GameTileContentType.Empty);
        FindPaths();
    }
    else if (tile.Content.Type == GameTileContentType.Empty) {
        tile.Content = contentFactory.Get(GameTileContentType.Tower);
        if (FindPaths()) {
            updatingContent.Add(tile.Content);
        }
        else {
            tile.Content = contentFactory.Get(GameTileContentType.Empty);
            FindPaths();
        }
    }
    else if (tile.Content.Type == GameTileContentType.Wall) {
        tile.Content = contentFactory.Get(GameTileContentType.Tower);
        updatingContent.Add(tile.Content);
    }
}
```

Лістинг 2 – Розташування башт

Башта може робити свою справу лише в тому випадку, якщо вона може знайти ворога. Після того, як ворог знайдений, він також повинен вирішити, на яку частину ворога цілитися.

Ми будемо використовувати фізичний механізм для виявлення цілей. Подібно до баштового колайдера, нам не потрібен колайдер ворога, щоб точно відповідати його формі. Ми можемо обійтися найпростішим колайдером, який є сферою. Після виявлення ми використовуватимемо положення ігрового об’єкта з прикріпленим до нього коллайдером як точку для прицілювання.

```
bool AcquireTarget () {
    Vector3 a = transform.localPosition;
    Vector3 b = a;
    b.y += 3f;
    int hits = Physics.OverlapCapsuleNonAlloc(
        a, b, targetingRange, targetsBuffer, enemyLayerMask
    );
    if (hits > 0) {
        target =
            targetsBuffer[Random.Range(0, hits)].GetComponent<TargetPoint>();
        Debug.Assert(target != null, "Targeted non-enemy!", targetsBuffer[0]);
        return true;
    }
    target = null;
    return false;
}

bool TrackTarget () {
    if (target == null) {
        return false;
    }
    Vector3 a = transform.localPosition;
    Vector3 b = target.Position;
    float x = a.x - b.x;
    float z = a.z - b.z;
    float r = targetingRange + 0.125f * target.Enemy.Scale;
    if (x * x + z * z > r * r) {
        target = null;
        return false;
    }
    return true;
}
```

Лістинг 3 – Виявлення та відслідковування ворога

Лазер - не єдиний вид зброї, який ми могли б встановити на вежі. Ми додамо другий тип башти, який перекриває снаряди, які вибухають при ударі та пошкоджують усіх сусідніх ворогів. Щоб це стало можливим, ми повинні підтримувати більше одного типу веж.

Міномет працює, стріляючи снарядом під кутом, тому він потрапляє через перешкоди і вражає ціль зверху. Як правило, використовуються снаряди, які детонують при ударі або поки вони все ще перевищують ціль. Щоб бути простим, ми завжди будемо націлюватись на землю, тому снаряди будуть спрацьовувати, як тільки їх висота буде зменшена до нуля.

Для наведення міномета потрібно як направити його до цілі горизонтально, а потім відрегулювати вертикальну орієнтацію, щоб снаряд потрапив на правильну відстань. Ми починаємо з першого кроку, спочатку використовуючи фіксовані відносні точки замість рухомих цілей, щоб полегшити перевірку правильності наших розрахунків.

Наступним кроком є з’ясування кута, під яким повинен бути запущений снаряд. Ми повинні це вивести з фізики траєкторії обстрілу. Ми не будемо розглядати опір, вітер чи будь-який інший вид перешкод, лише швидкість запуску v та гравітацію g = 9.81.

Зміщення d оболонки співпадає з цільовим трикутником і може бути описане двома компонентами. Горизонтальне зміщення є простим $d_x = v_x t$ де t це час після запуску. Вертикальна складова подібна, але також піддається негативному прискоренню через гравітацію, отже $d_y = v_y t - \frac{gt^2}{2}$.

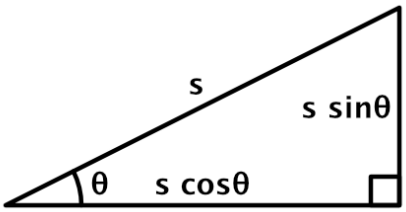

Рисунок 8 - Виведення швидкості запуску

Ми запускаємо снаряди з фіксованою швидкістю s що не залежить від кута запуску θ. Тому $v_x = s\cos\theta$ та $v_y = s\sin\theta$. Підставляючи, ми приходимо до $d_x = st\cos\theta$ та $d_y = st\sin\theta - \frac{gt^2}{2}$. Ми запускаємо оболонку таким чином, щоб її час польоту t достатньо довгий, щоб досягти мети. Оскільки горизонтальне переміщення є найпростішим, ми можемо виразити час, використовуючи $t = \frac{d_x}{v_x}$. У пункті призначення $d_x = x$, тому $t = \frac{x}{s\cos\theta}$. Це означає що $y = x\tan\theta - \frac{gx^2}{2s^2cos^2\theta}$. Використовуючи це, ми знаходимо $\tan\theta = \frac{s^2 \pm \sqrt{s^4 - g(gx^2 + 2ys^2)}}{gx}$. Є два можливі кути запуску, тому що можна цілити як низько, так і високо. Низька траєкторія швидша, оскільки вона наближається до прямої лінії до цілі. Але висока траєкторія візуально цікавіша, тому ми використаємо її.

Сенс обчислення траєкторій полягає в тому, що ми тепер знаємо, як запускати снаряди. Наступним кроком є їх створення та запуск. Ми запускаємо снаряди, бо вони заповнені вибухівкою. Коли снаряд досягає цілі, він повинен підірвати та пошкодити всіх ворогів у зоні вибуху. Радіус вибуху та величина збитку залежать від того, які снаряди вистрілює мінометом, тому додайте параметри конфігурації для них у MortarTower.

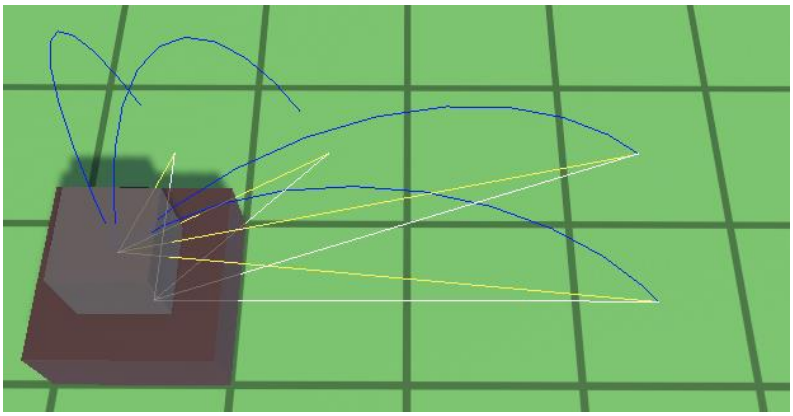

Рисунок 9 - Параболічні траєкторії польоту до однієї секунди.

Ми можемо вдосконалити свою гру за допомогою візуалізації вибуху, коли снаряд детонує. Окрім того, що він виглядав цікавішим, він також надавав корисні візуальні відгуки гравцеві. Ми зробимо це, створивши prefab вибуху, подібний до лазерного променя, за винятком того, що це сфера, вона прозоріша і має яскравіший колір. Надамо йому новий компонент вибуху із настроюваною тривалістю, за замовчуванням півсекунди, яка є короткою, але достатньо довгою для чіткої реєстрації. Дамо йому метод Initialize, щоб встановити його положення та радіус вибуху. Нам потрібно подвоїти радіус при встановленні масштабу, оскільки радіус сітки сфери дорівнює 0,5. Це також гарне місце для нанесення шкоди всім ворогам, що перебувають в зоні дії, тому воно також повинно мати параметр шкоди. Крім того, йому потрібен метод GameUpdate, який просто перевіряє, чи не минув його час.

Оскільки снаряди невеликі і мають відносно високу швидкість, їх важко побачити. І при перегляді знімка екрана одного кадру траєкторії зовсім не зрозумілі. Ми могли б зробити це більш очевидним, додавши до снарядів ефект сліду. Це нереально для звичайних оболонок, але ми можемо заявити, що це трасувальні оболонки. Такі снаряди спеціально зроблені, щоб залишити яскравий слід з метою зробити їхні траєкторії видимими.

2.3.6 Створення ігрових сценаріїв

Завжди створювати одного і того ж синього кубикового ворога не дуже цікаво. Першим кроком до створення більш цікавих сценаріїв ігрового процесу є підтримка декількох видів ворогів.

Як ви розробляєте типи ворогів, вирішувати вам, але для цього проекту використаємо якомога простіші. Продублюємо оригінальний prefab ворога і використаємо його для всіх трьох розмірів, змінюючи лише їх матеріал: жовтий для маленького, синій для середнього та червоний для великого. Не будемо змінювати масштаб prefab'a куба, натомість використаємо масштабну конфігурацію для зміни їх розміру. Також дамо їм збільшення здоров'я та зменшення швидкості відповідно.

Зараз фабрика ворогів визначає набір із трьох ворогів. Наша фабрика виробляє кубики трьох розмірів, але ніщо не заважає нам створити ще одну фабрику, яка виробляє щось інше, наприклад сфери трьох розмірів. Ми можемо змінити, які вороги породжуються, присвоївши грі іншу фабрику, тим самим переключившись на іншу тему.

Другий крок створення сценаріїв ігрового процесу - це більше не створювати ворогів на фіксованій частоті. Натомість вороги повинні породжуватися послідовними хвилями, поки сценарій не буде завершений або гра не програна.

Одна ворожа хвиля складається з групи ворогів, які породжуються один за одним, поки хвиля не завершиться. Хвиля може містити поєднання ворогів, і затримка між послідовними нерестами може бути різною. Щоб зробити це простим у реалізації, ми починаємо з базової послідовності спауну ворогів, яка виробляє той самий тип ворога на фіксованій частоті. Тоді хвиля - це просто список таких послідовностей появи.

У деяких сценаріях вам може знадобитися пройти всі хвилі більше одного разу. Ми можемо підтримати це, зробивши можливим повторення сценаріїв, проїжджаючи по всіх хвилях кілька разів. Ви можете додатково уточнити це, наприклад, повторити лише останню хвилю, але в цьому підручнику ми просто повторимо весь сценарій.

Якщо гравцеві вдалося обіграти цикл один раз, він повинен мати можливість бити його знову без проблем. Нам доведеться збільшувати складність, щоб сценарій залишався складним. Найпростіший спосіб це зробити, зменшуючи всі періоди перезарядки послідовно. Це призводить до того, що вороги з’являються швидше і неминуче пригнічують гравця у сценарії виживання.

2.3.7 Додавання анімації

До цього моменту наші вороги просто ковзають по дошці. Це може бути чудово для абстрактної гри, яка використовує куби та сфери для ворогів, але навіть таких ворогів можна зробити цікавішими, змусивши їх рухатися більш органічно. Ми могли б змусити їх відскакувати, компенсуючи їх вертикальне положення чимось на зразок абсолютної синусоїди, заснованої на часі, але загальний підхід полягає у використанні анімаційного кліпу. Ми використовуватимемо анімацію, оскільки це дозволяє набагато складніший рух, а також дає змогу імпортувати існуючу анімацію.

Ми можемо створювати анімаційні кліпи в редакторі Unity, записуючи коригування ієрархії об’єктів. Перетягнемо середній збірний екземпляр куба ворога на сцену або окрему сцену, призначену для запису анімації. Потім виберемо дочірній куб Моделі ворога та відкриємо вікно анімації через Вікно / Анімація / Анімація.

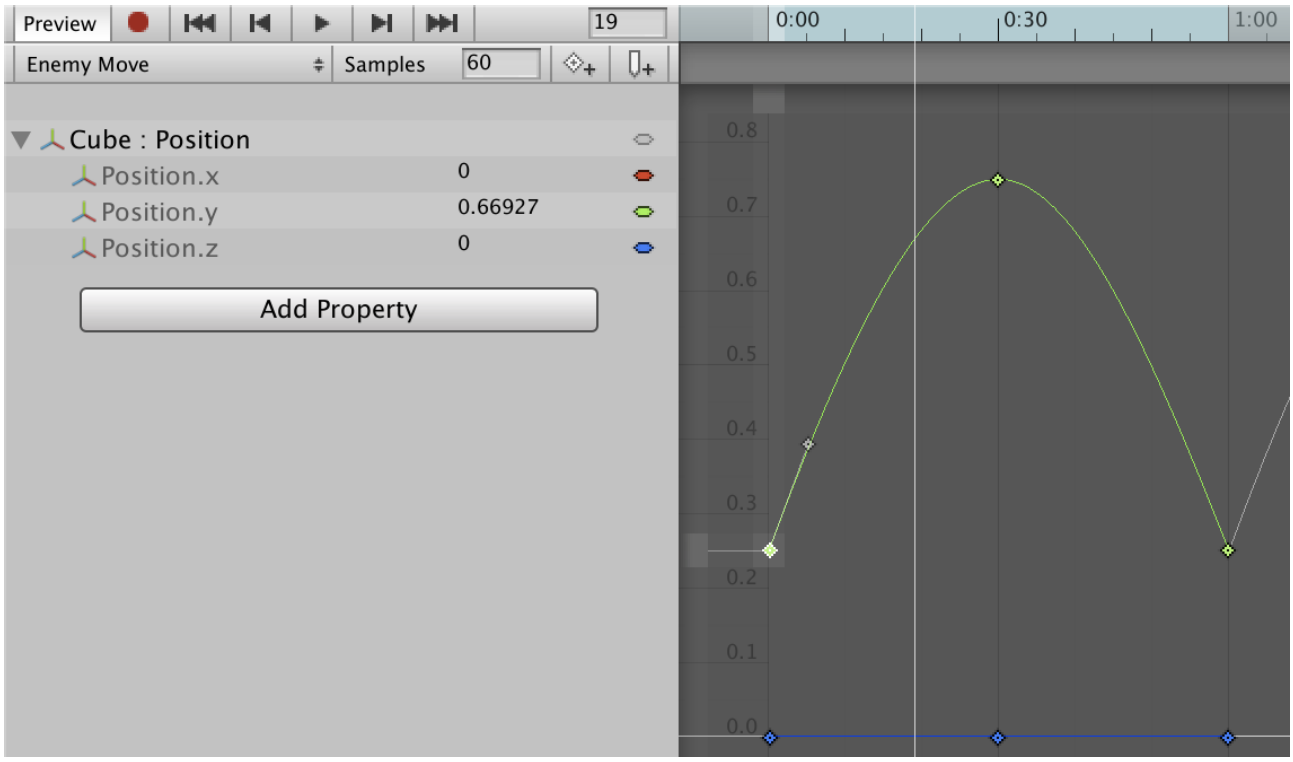

Рисунок 10 – Анімація ворога

Контролер анімації можна використовувати для анімації ворогів, але це досить жорсткий підхід для нашої простої поведінки ворога. Крім того, одночасно може бути багато ворогів, і всі вони потребуватимуть власного контролера, тому логіка управління анімацією повинна бути якомога простішою. Нарешті, ми хочемо використовувати різні анімації для кожного ворога, хоча всі вони мають однакову логіку. Тому замість того, щоб покладатися на контролер анімації Unity, ми створимо власний. Контролер анімації Unity потрібен лише для запису анімації.

```
#if UNITY_EDITOR
    double clipTime;
#endif

    bool hasAppearClip, hasDisappearClip;

    public Clip CurrentClip { get; private set; }

    public bool IsDone => GetPlayable(CurrentClip).IsDone();

#if UNITY_EDITOR
    public bool IsValid => graph.IsValid();
#endif

    public void Configure (Animator animator, EnemyAnimationConfig config) {
        hasAppearClip = config.Appear;
        hasDisappearClip = config.Disappear;

        graph = PlayableGraph.Create();
        graph.SetTimeUpdateMode(DirectorUpdateMode.GameTime);
        mixer = AnimationMixerPlayable.Create(
            graph, hasAppearClip || hasDisappearClip ? 6 : 4
        );

        var clip = AnimationClipPlayable.Create(graph, config.Move);
        clip.Pause();
        mixer.ConnectInput((int)Clip.Move, clip, 0);

        clip = AnimationClipPlayable.Create(graph, config.Intro);
        clip.SetDuration(config.Intro.length);
        clip.Pause();
        mixer.ConnectInput((int)Clip.Intro, clip, 0);
```

Лістинг 3 – Фрагмент коду контролера анімацій

Перевірити працездатність проекту можна прямо у вікні Unity, для цього необхідно натиснути кнопку запуску проекту в верхній частині екрану. Поряд з нею знаходяться кнопки паузи і закінчення тесту гри. Вони мають такий же вигляд, як кнопки управління в будь-якому іншому аудіо- або відеоплеєрі.

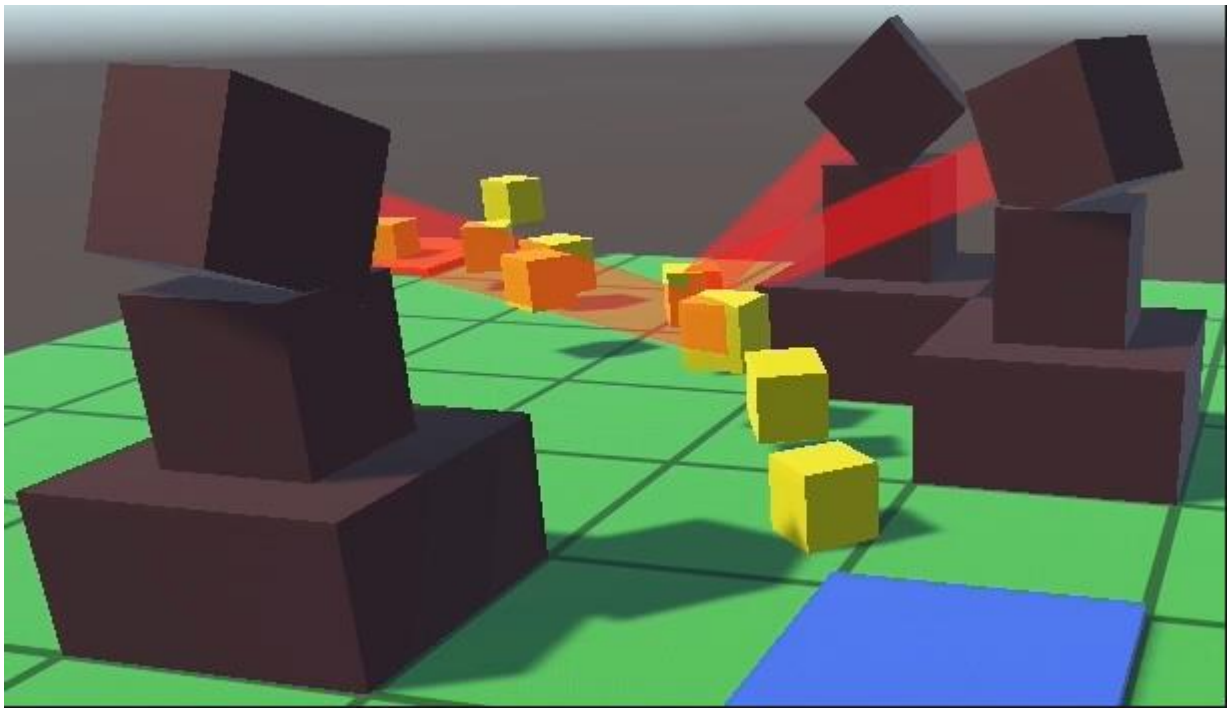

Рисунок 11 – Вигляд гри

2.3.8 Додавання моделей

Хоча в редакторі Unity можна створювати прості анімації, вони зазвичай імпортуються разом із 3D-моделями. Ви або створили їх самостійно в окремій програмі, або отримали звідкись, наприклад, Unity Asset Store.

Після створення та імпорту нашої моделі у Unity, ми додамо її замість як префаб для ворога. Надамо гренадеру власну конфігурацію анімації. Ми можемо використовувати анімацію GrenadierWalk для руху, GrenadierCloseRangeAttack як для вступу, так і для виходу, і GrenadierDeath для смерті. Усі вони знаходяться в папці AnimationClips всередині об’єктів із знаком @ перед їх іменем.

Швидкість ходьби гренадера не збігається зі швидкістю в грі, що спричиняє ковзання ніг навіть при русі прямо вперед. Це відбувається тому, що анімаційний кліп не охоплює жодного юніту в секунду. Ми компенсуємо це, додавши опцію конфігурації швидкості анімації переміщення до EnemyAnimationConfig, встановлену за замовчуванням на 1.

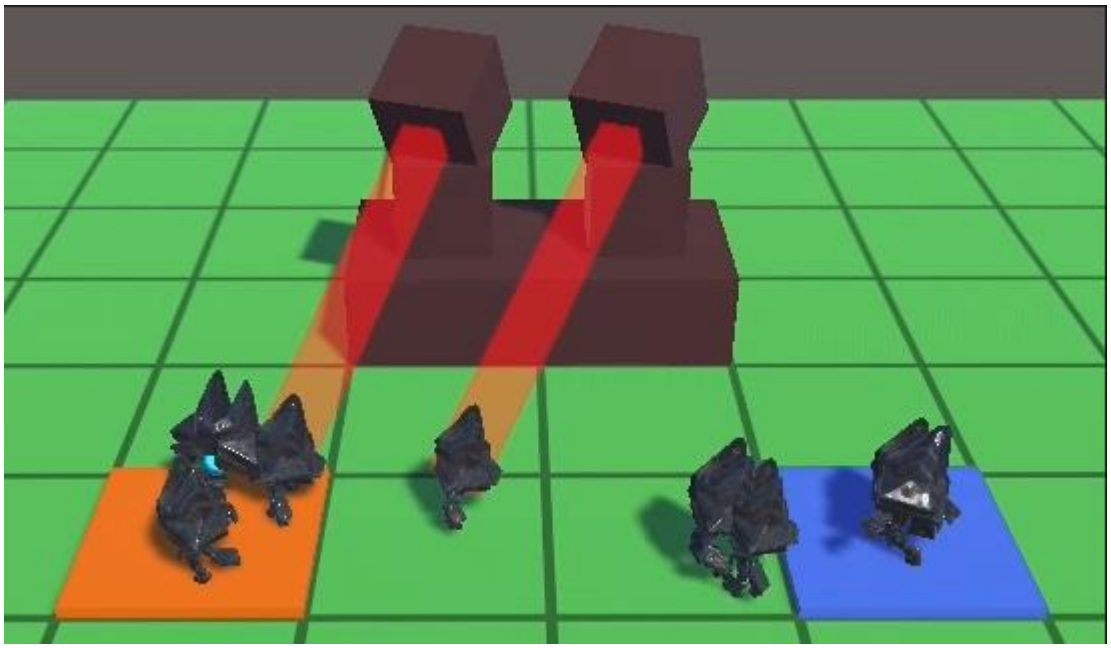

Рисунок 12 – Вигляд моделей на ігровому полі

2.3.9 Тестування

Слід відзначити, що на даному етапі реалізації проекту проводити як-небудь крупне тестування смисла ні, так як в грі реалізована лише мала частина можливостей, що передбачається реалізувати. Саме за цієї причини, дане тестування, є лише невеликою перевіркою на відповідність основним функціональним вимогам.

Таблиця 2 - Тестування

№	Призначення	Очікуваний результат	Отриманий результат	Підсумок
1	Перевірка на коректне встановлення веж	Вежі будуть встановлюватися на один блок та робити цей блок недійсним для пересування ворогів	Вежі становлювались так як і очікувалось	Пройдено
2	Перевірка на коректне знищення ворогів	При зменшенні здоров’я ворога до 0 він буде знищуватися	Як тільки здоров’я ворога падає до 0, вмикається анімація смерті та ворог зникає	Пройдено
3	Перевірка на коректне виявлення	При попаданні ворога у зону атаки вежі, вона повинна виявити	При попаданні ворога у зону атаки вежі, вона виявляє його та	Пройдено

	ворогів вежею	його та атакувати, якщо не атакує іншого ворога	атакує, якщо не атакує інших	
4	Перевірка на коректну поведінку ворогів	Вороги при появленні повинні йти по маршруту до кінцевої точки та обходити перешкоди	При появі ворогів у початковій точці, вони починають рух до кінцевої точки й правильно минують перешкоди	Пройдено
5	Перевірка на зміну рівнів	При проходженні хвилі ворогів, хвиля змінюється на іншу відмінну від попередньої	Коли башти гравця знищують всіх ворогів, рівень вважається пройденим та змінюється на інший	Пройдено
6	Перевірка на програш гравця	Коли певна кількість ворогів доходить до кінцевої точки, гравець програє та повинен почати спочатку	При доходженні заданої кількості ворогів до кінцевої точки гравець починає спочатку	Пройдено

2.3.10 Технічні характеристики

Перш за все, варто розуміти, що даний проект не є повністю готовою до релізу комп'ютерною грою, і за своєю суттю є прототипом стоять приблизно на рівні ранньої альфа-версії. Викликано це було тим, що створення якісного продукту займає колосально кількість часу. Так, наприклад, розробка невеликої гри для мобільних телефонів з процедурно генеруються рівнем може займати більше року, при умови, що над проектом працює не більше трьох осіб. Збільшення кількості осіб може прискорити розробку, але не сильно.

На даний момент дуже складно визначити якими системними вимогами буде володіти проект в майбутньому, так як велика кількість
елементів просто не реалізовані. На даний момент ми маємо лише елементи, які відповідають за основну ігрову механіку, а значить, в майбутньому

технічні характеристики для розроблюваної комп'ютерної гри можуть дуже сильно змінитися.

Зараз можна з упевненістю стверджувати лише те, що для запуску даного прототипу буде потрібно комп'ютер, що володіє системними характеристиками не нижче мінімально допустимих для коректної роботи ігрового движка Unity, для якого найбільш важливим аспектом є тільки відеокарта. Вона повинна підтримувати DirectX 9 з шейдерами не нижче версії 3.0. Це означає, що в даний момент створений прототип з великою ймовірністю піде на більшості комп'ютерах.

ВИСНОВКИ

Під час аналізу доступних джерел було проведено дослідження поняття комп'ютерна гра, під час якого була проведена класифікація ігор за 4 критеріями, але через порівняльну молодість ігрової індустрії, а також те, що класифікація комп'ютерних ігор не була систематизована, скласти детальну класифікацію не вдалося.

Додатково був складений алгоритм розробки відеоігор. Були проаналізовані популярні засоби розробки. В ході аналізу, було проведено їх порівняння і обрані найбільш актуальні засоби розробки для початківців розробників. Вибір пріоритетних засобів розробки проходив за двома критеріями: доступність і функціональність.

При аналізі існуючих розробок, було проведено їх порівняння і виділені їхні переваги і недоліки. В ході аналізу стало зрозуміло, що при розробці комп'ютерної гри з простою ігровою механікою, варто звернути увагу на додаткові елементи гри, такі як сюжет і графічне оформлення. Це потрібно для того, що б утримати потенційного гравця і продовжити життєвий цикл розробки.

Грунтуючись на всю отриману в ході дослідження інформацію, було вирішено розробити прототип трьохвимірної Tower Defense для одного гравця на ігровому движку Unity. Таке рішення було прийнято з кількох причин:

1) тривимірна графіка, на відміну від двовимірної цікавіша в створенні;
2) по ігровій механіці, гра жанру Tower Defense простіше реалізується;
3) ігровий движок Unity поширюється безкоштовно і дозволяє розробляти програми на мові програмування C #.

Після вибору засобів розробки було розпочато вивчення Unity, а так само розробка самого проекту. В ході розробки був вивчений ігровий движок Unity і були придбані необхідні знання та вміння, а саме:

• створення сцен;
• створення анімацій;
• створення і написання скріптів;
• настройка об'єктів;
• створення UI;
• компіляція проекту.

Освоєння середовища розробки Unity несе не маловажний характер, так як в сучасному світі індустрія розробки ігор все сильніше поширюється в нашому суспільстві. Ігри перестали бути лише предметом для розваг, і тепер використовуються і в інших областях, наприклад, в науці або в навчанні користувачів. Тому розвиток в даному напрямку можна вважати одним з найважливіших в сучасному суспільстві.

В ході реалізації проекту були виконані наступні завдання:

1) вивчені особливості і стан комп'ютерної індустрії України;
2) обрані жанр, вид і платформа для комп'ютерної гри;

3) розроблено сценарій і концепція основних елементів;
4) вибрано і вивчено засіб реалізації;
5) підготовлені необхідні для гри анімації;
6) реалізований прототип гри.

Також освоєння таких графічних редакторів як Adobe Photoshop та Blender несе за собою велику цінність, оскільки це є безпосередньою частиною розробки ігор, а також має великий обсяг інших сфер в яких знання цих редакторів є дуже важливим.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Компьютерные игры как искусство ? [Електронний ресурс] – Режим доступу до ресурсу: http://gamesisart.ru
2. Торн А. Основы анимации в Unity / Торн А., 2016. - 176 с.
3. Хокинг Дж. Unity в действии. Мультиплатформенная разработка на C# / Хокинг Дж., 2019. – 352 с.
4. Game Maker: Studio [Електронний ресурс] – Режим доступу до ресурсу: https://www.yoyogames.com/gamemaker
5. Unreal Engine [Електронний ресурс] – Режим доступу до ресурсу: https://www.unrealengine.com
6. Unity Game Engine [Електронний ресурс] – Режим доступу до ресурсу: https://unity3d.com/ru
7. Adobe Photoshop [Електронний ресурс] – Режим доступу до ресурсу: http://www.adobe.com/ru/products/photoshop
8. CorelDRAW [Електронний ресурс] – Режим доступу до ресурсу: http://www.coreldraw.com/ru
9. PaintTool SAI [Електронний ресурс] – Режим доступу до ресурсу: http://mypainttoolsai.ru/
10. Blender [Електронний ресурс] – Режим доступу до ресурсу: https://www.blender.org/
11. 3Ds Max [Електронний ресурс] – Режим доступу до ресурсу: https://www.autodesk.ru/products/3ds-max
12. Maya [Електронний ресурс] – Режим доступу до ресурсу: https://www.autodesk.ru/products/maya
13. Dungeon Defenders [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/65800/Dungeon_Defenders/
14. Orcs Must Die [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/102600/Orcs_Must_Die/
15. BLOONS TD [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/960090/Bloons_TD_6/
16. Dungeon of the Endless [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/249050/Dungeon_of_the_Endless/
17. Unity Manual [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/index.html