ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики та інформатики

«До захисту допущено»
Завідувач кафедри ПМІ
_____ **О.А. Дмитрієва**
(підпис) (ініціали, прізвище)
«____»_____2021 р.

Випускна кваліфікаційна робота

магістра
(освітній ступінь)

на тему
«Дослідження етапів розробки комп'ютерних симуляторів із застосуванням Unity 3D»

Виконав (-ла): студент (-ка) 2 курсу, групи ІПЗм-20
(шифр групи)

напряму підготовки (спеціальності) спеціальності 121 Інженерія програмного забезпечення
(шифр і назва напряму підготовки)

Тетюшев Данило Артемович _____
(прізвище та ініціали) (підпис)

Керівник доц. кафедри ПМІ, к.т.н., доц. Маслова Н. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Студент _____
(підпис)

Покровськ – 2021

ДВНЗ «Донецький національний технічний університет»

ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики та інформатики

Освітній ступінь магістр

Напрям підготовки (спеціальність) 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ /О.А. Дмитрієва/
«____»_____2021 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Тетюшеву Данилу Артемовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження етапів розробки комп'ютерних симуляторів із застосуванням Unity 3D»

Спецчастина

Керівник роботи Маслова Н. О., к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом від « 22 вересня 2021 року № 570

2. Строк подання студентом роботи 6 грудня 2021 року

3. Вихідні дані до роботи результати переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно опрацювати) 1) Аналіз основних етапів розробки комп'ютерних симуляторів
2) Пошук серед розробки для реалізації проекту та їх порівняння;
3) Створення макету та формування ігрової логіки;
4) Проектування цілісного симулятору;
5) Тестування та виправлення можливих помилок.

5. Перелік графічного матеріалу
робота містить 45 рисунків, 1 блок-схему, 15 слайдів презентації та інший графічний матеріал, у кількості, достатній для демонстрації сутності роботи

-

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтролер	Доц. Назарова І.А.		

7. Дата видачі завдання 22 вересня 2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літератури з предметної області	23.09.2021 – 27.09.2021	
2	Збір інформації і методичних відомостей щодо застосування методів класифікації у криптографії та криптоаналізі	28.09.2021 – 05.10.2021	
3	Розробка структури програмного забезпечення, вибір даних для обробки та експериментів.	06.10.2021 – 20.10.2021	
4	Тестування системи	21.10.2021 – 10.11.2021	
5	Оформлення пояснювальної записки	11.11.2021 – 25.11.2021	
6	Нормоконтроль пояснювальної записки, її перевірка антиплагіатною системою	26.11.2021- 01.12.2021	
7	Оформлення супутніх матеріалів (розробка графічного матеріалу, написання доповіді, тощо) та рецензування роботи	02.12.2021- 13.12.2021	
8	Захист роботи	22.12.2021	

Студент _____ Тетюшев Д.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Маслова Н.О.
(підпис) (прізвище та ініціали)

А Н О Т А Ц І Я

Тетюшев Данило Артемович. Дослідження етапів розробки комп’ютерних симуляторів із застосуванням Unity 3D / Випускна кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 121 «Інженерія програмного забезпечення». – ДВНЗ ДонНТУ, Покровськ, 2021.

Мета роботи – дослідження методик та засобів створення комп’ютерних симуляторів. Розробка симулятору спортивної гри, який відповідає вимогам, поставленим до дипломної роботи, а також виконує всі поставлені перед ним функціональні завдання.

Тематикою роботи є «дослідження етапів розробки комп’ютерних симуляторів із застосуванням Unity 3D».

Завдання роботи – дослідити концепції створення комп’ютерних симуляторів та ігрових механік, а також засоби і методи їх створення.

Предметом дослідження роботи є алгоритми створення комп’ютерних симуляторів, а також ігрові механіки, які забезпечують коректну роботу симуляції та її взаємодії з гравцем.

Наукова новизна полягає в тому, що на даний момент часу у ігровій індустрії існує безліч симуляторів, починаючи з симулятора перегонів до симулятора фермера, але симулятори спортивної гри пейнтбол є рідкістю, яку не легко знайти.

Практичне значення полягає у можливості опублікувати симулятор на площадку Steam Greenlight, тим самим надати змогу гравцям оцінити гру та у майбутньому загрузити її на цифрові площадки і отримувати процент від продаж копій гри.

Ключові слова: симулятор, Unity 3D, мова C#, пейнтбол, спорт.

ABSTRACT

Tetyushev Danilo Artemovich. Research of stages of development of computer simulators with the use of Unity 3D / Graduation qualification work for the degree of "master" in the specialty 121 "Software Engineering". - SHEI DonNTU, Pokrovsk, 2021. The purpose of the work is to study the methods and means of creating computer simulators. Development of a sports game simulator that meets the requirements for the thesis, as well as performs all the functional tasks set before it. The topic of the work is "study of the stages of development of computer simulators using Unity 3D". The task of the work is to explore the concepts of creating computer simulators and game mechanics, as well as the means and methods of their creation. The subject of the study is the algorithms for creating computer simulators, as well as game mechanics that ensure the correct operation of the simulation and its interaction with the player. The scientific novelty is that at the moment there are many simulators in the gaming industry, from a racing simulator to a farmer's simulator, but paintball simulators are a rarity that is not easy to find. The practical value is the ability to publish the simulator on the site Steam Greenlight, thus allowing players to evaluate the game and in the future to upload it to digital sites and receive a percentage of sales of copies of the game.

Keywords: simulator, Unity 3D, C # language, paintball, sports.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

FPS – FIRST PERSON SHOOTER

COOL (C-STYLE OBJECT ORIENTED LANGUAGE)

ENTITY COMPONENT SYSTEM (ECS)

MVS – MICROSOFT VISUAL STUDIO

RPG - АНГЛ. ROLE PLAYING GAME

QoS – QUALITY OF SERVICE

F2P – FREE TO PLAY

ВСТУП

У сучасному світі комп’ютерні ігри стали одним з основних видів відпочинку як у підлітків, так і у дорослих. Ігрова індустрія створила безліч жанрів комп’ютерних ігор, одним з яких є симулятори.

Це такий тип ігор, де гравець може відчути себе на місці як пілота літака, так і фермера. У якості дипломного проекту було обрано дослідження засобів та методів створення комп’ютерного симулятора, на основі опанованих знань створити симулятор спортивної гри пейнтбол у вигляді FPS-шутеру.

Метою кваліфікаційної роботи є - дослідження методик та засобів створення комп’ютерних симуляторів. Аналіз етапів створення симулятору. Розробка симулятору спортивної гри, який відповідає вимогам, поставленим до дипломної роботи, а також виконує всі поставлені перед ним функціональні завдання. Тематикою роботи є «дослідження етапів розробки комп’ютерних симуляторів із застосуванням Unity 3D».

Завдання роботи – дослідити концепції створення комп’ютерних симуляторів та ігрових механік, а також засоби і методи їх створення.

Предметом дослідження роботи є алгоритми створення комп’ютерних симуляторів, а також ігрові механіки, які забезпечують коректну роботу симуляції та її взаємодії з гравцем.

Об’єктом дослідження є процесс створення комп’ютерної гри-симулятору засобами Unity 3D та Blender.

Структура та обсяг роботи. Випускна кваліфікаційна робота обсягом 76 машинописних сторінок, складається з вступу, чотирьох розділів, висновків, переліку використаних джерел, що складається з 50 найменувань. Робота містить 35 рисунків, 1 блок-схему, 15 слайдів презентації.

РОЗДІЛ 1
АНАЛІЗ НАЯВНИХ МЕТОДІВ ТА ТЕХНОЛОГІЙ, ДЛЯ ВИРІШЕННЯ ВІДПОВІДНОЇ ПОСТАВЛЕНОЇ ЗАДАЧІ

1.1 Аналіз засобів для реалізації проекту

На початку будь-якої роботи треба проаналізувати способи та методи її відтворення, що й було зроблено на першому етапі. Для створення ігрових світів використовують різноманітні середи та рушії, навички роботи з котрими є основою для створення якісного продукту.

Вибір технологій програмування та розробки - стартовий пункт у здійсненні будь-якого проекту. Від цього залежить функціональність, зручність майбутнього продукту. Розглянемо основні елементи, які будуть використані при створенні симулятору пейнтболу: рушій Unity 3D та його аналоги, програмний пакет для створення 3D-моделей Blender та його аналоги, мова програмування C# та її аналоги, середа розробки Microsoft Visual Studio.

Unity 3D – це міжплатформна середа для розробки комп’ютерних ігор. Середа дозволяє створювати проекти, які будуть працювати більш ніж на 25 платформах, які містять в собі ігрові консолі, персональні комп’ютери, мобільні пристрої. Unity 3D була випущена у 2005 році, та досі оновлюється.

Спершу планувалось, що середа буда працювати лише на системі MacOs, однак вже через декілька місяців після виходу, Unity отримала оновлення, яке містило у собі підтримку Windows. За декілька років ряд підтримуваних платформ лише поширювався. Вже за 4 роки, у 2009 році веб-сайт Gamasutra, присвячений розробці комп’ютерних ігор, назвав Unity одним з найважливіших учасників на ринку ігрових компаній.

Найперша версія середи створювалась лише трьома людьми: Девідом Хелгасоном, Джошимом Анте та Ніколасом Френсісом. Вони мали мету створити ігровий рушій, який би мав зручний графічний інтерфейс. Новітня версія Unity моє багато спільного з дебютною, яка з самого початку працювала

за принципом drag-and-drop, тобто перетягування елементів між підменю програми.

Однак назвати Unity повноцінним рушієм на момент випуску не вдавалось, середа підходила більш для створення якісних анімацій. Вже у версії 1.2 розробники додали у Unity різноманітні ефекти, тіні, інтегрований скрипт управління персонажем, систему ragdoll. Після цього нововведення середою почали цікавитись indie-розробники.

За два роки вийшла вже друга версія рушія. Були враховані та виправлені ранні помилки та недоліки. Середа Unity отримала поліпшений веб-плеєр, підтримку веб-стрімінгу та багато іншого. Однією з основних змін стала тотальна переробка графічного інтерфейсу користувача.

Версія Unity 2.6 стала безкоштовною у жовтні 2009 року, а також з’явилась версія для Wii. Кожне оновлення приносило із собою масу нових функцій. Так, у грудні 2011 року версія Unity 3.5. дозволила користувачам працювати з Adobe Flash та публікувати ігри у форматі .swf.

Розробники додали підтримку поширеного динамічного діапазону (HDR), можливість відображення віддалених об’єктів с гіршою деталізацією задля економії ресурсів (LOD – Levels Of Detail) та змогу використовувати можливості багатоядерних процесорів (мультипотоковий рендеринг). А інтерфейс зазнав кардинальних змін, якщо порівнювати першу версію (рис.1.1) з нею (рис.1.2).

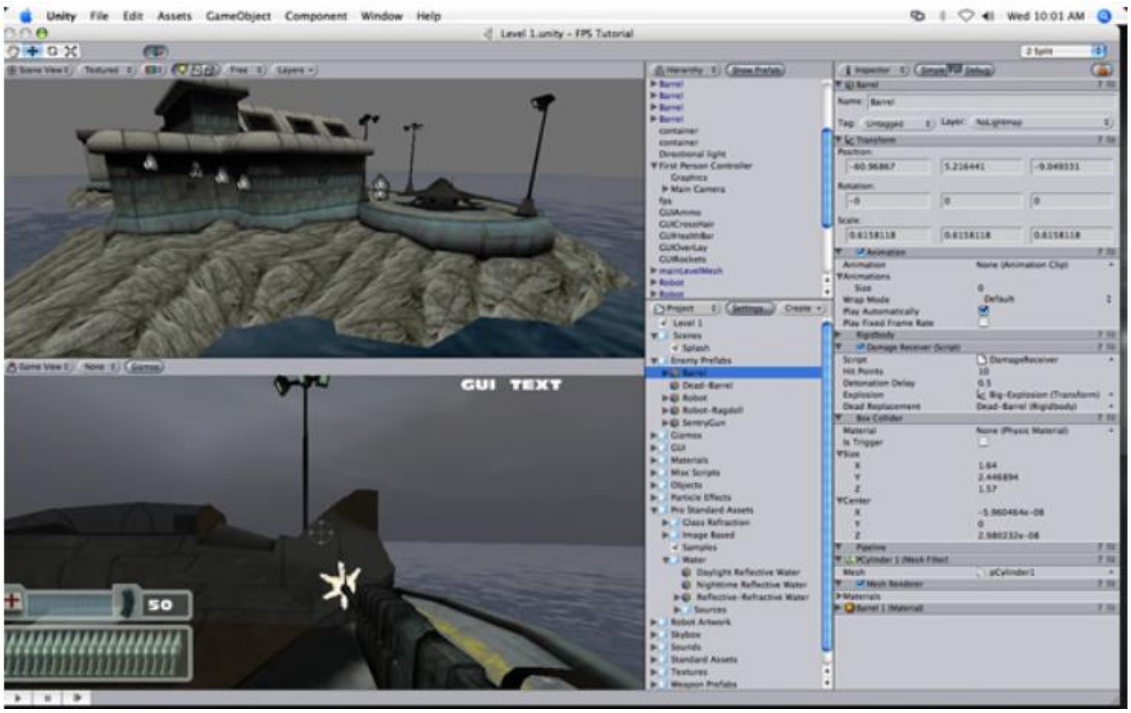


Рисунок 1.1 – Версія Unity 1

Рисунок 1.2 - Версія Unity 3.5

Новітня версія Unity вийшла у 2014 році та досі оновлюється. Версія Unity 5 отримала найбільший апдейт за усю історію розвитку рушія, а саме:

— нові інструменти графічного інтерфейсу окремо для 3D і 2D ігор;

— повноцінний звуковий редактор (можна в реальному часі об'єднувати різні звуки, додавати ефекти, пов'язувати їх з подіями в грі);

— підтримка WebGL - ігри працюють безпосередньо в браузері без установки веб-плеєра;

— глобальне освітлення в реальному часі для консолей нового покоління, ПК і мобільних платформ;

— віддзеркалення світла в реальному часі на основі Reflection Probes;

— фізично коректні матеріали;

— нові можливості 2D фізики: точкові сили притягання і відштовхування; тангенціальні сили (сили, спрямовані по дотичній до поверхні об'єкта); сили, спрямовані уздовж будь-яких осей; одностороннє зіткнення;

— відстеження завантаження процесора, відеокарти і пам'яті на часовій шкалі в режимі реального часу;

— додавання повноцінного 64х-бітного редактора. Інтеграція Terrain Speedtree;

— додавання нових API для 2D фізики і редактора анімації в Box2D, оновлення 3D фізики до nVidia PhysX3;

— перегляд сцен в HDR-режимі;

— налаштування для рендера сцени за допомогою заповнюючого (Ambient) світла;

— поліпшено роботу LOD (тепер немає падіння продуктивності для непропорційно скейлірованной геометрії);

— нові форми для перешкод Nav Mesh і стислі текстури для Cubemaps;

— підтримка джойстика для Windows Store;

— періодична анімація може пересувати персонажа.

Особливістю стало і те, що середа отримала підтримку роботи з шоломами віртуальної реальності, на той часи за першими Oculus Rift. За допомогою новітньої версії було створено багато популярних як ігор, так і симуляторів, наприклад симулятор виживання у лісі The Forest (рис 1.3), або пригодницька гра Firewatch (рис 1.4).

Рисунок 1.3 - Симулятор виживання у лісі The Forest)

Рисунок 1.4 - Пригодницька гра Firewatch

На сьогоднішній день Unity вважається одним із найперспективніших рушіїв, який поліпшується з кожним оновленням, завдяки старанням команди розробників.

Особливості рушія: Unity3D дає можливість розробляти гри, не вимагаючи для цього якихось особливих знань. Тут використовується компонентно-орієнтований підхід, в рамках якого розробник створює об'єкти (наприклад, головного героя) і до них додає різні компоненти (наприклад, візуальне відображення персонажа і способи управління ним). Завдяки зручному drag and drop інтерфейсу і функціональному графічному редактору, рушій дозволяє малювати мапи і розставляти об'єкти в реальному часі і відразу ж тестувати результат.

Друга перевага - наявність величезної бібліотеки ассетів і плагінів, за допомогою яких можна значно прискорити процес розробки гри. Їх можна імпортувати і експортувати, додавати в гру цілі заготовки - рівні, ворогів, патерни поведінки II і так далі.

Третя сильна сторона Unity 3D - підтримка величезної кількості платформ, технологій, API. Створені на рушії ігри можна легко перенести між ОС Windows, Linux, OS X, Android, iOS, на консолі сімейств PlayStation, Xbox, Nintendo, на VR- і AR-пристрої. Unity підтримує DirectX і OpenGL, працює з усіма сучасними ефектами рендеринга, включаючи новітню технологію трасування променів в реальному часі.

Unreal Engine - ігровий рушій, що розробляється і підтримується компанією Epic Games. Першою грою на цьому рушії був шутер від першої особи Unreal, випущений в 1998 році. Хоча рушій спочатку був призначений для розробки шутерів від першої особи, його наступні версії успішно застосовувалися в іграх самих різних жанрів, в тому числі стелс-іграх, файтингах і масових багатокористувацьких рольових онлайн-іграх. У минулому движок поширювався на умовах оплати щомісячна плата за користування.

Написаний на мові C ++, рушій 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, PlayStation 5, PSP, PS Vita, Wii, Dreamcast, GameCube і ін., а також на різних портативних пристроях, наприклад, пристроях Apple (iPad, iPhone), керованих системою iOS і інших. (Вперше робота з iOS була представлена в 2009 році, в 2010 році продемонстровано роботу рушія на пристрої з системою webOS).

У 1998 році Unreal Engine 1 був одним з перших ігрових рушіїв подібної універсальності; він поєднував в собі графічний рушій, фізичний рушій, штучний інтелект, управління файлової і мережевий системами і готову середу розробки для ігор - UnrealEd. З огляду на рівень продуктивності більшості комп'ютерів того часу, розробники дещо спростили деякі елементи рушія: систему виявлення зіткнень, мережевий код, код контролера для гравця.

Наразі останньою версією цього рушія є Unreal Engine 4, яка була презентована у березні 2014 року. 2 березня 2015 року Epic Games зробила заяву про зміну системи поширення: рушій став безкоштовним для всіх розробників, за умови, що прибуток від додатків, створених на основі рушія не перевищує $ 3000 за квартал.

Blender – це професійне програмне забезпечення, за допомогою якого створюється тривимірна комп'ютерна графіка. Середа містить у собі засоби для моделювання, скульптингу, анімації, рендерингу та інше. Наразі саме це програмне забезпечення користується найбільшою популярністю серед безкоштовних 3D- редакторів.

Одною з характерних рис Blender є його невеликий розмір у порівнянні з іншими подібними пакетами.

Незважаючи на невеликі розміри, пакет виконує наступні функції:

— підтримка різноманітних геометричних примітивів, включаючи полігональні моделі, систему швидкого моделювання в режимі subdivision

surface (SubSurf), криві Безьє, поверхні NURBS, metaballs (метасфери), скульптурне моделювання та векторні шрифти;

— універсальні вбудовані механізми рендеринга і інтеграція з зовнішніми рендерерами YafRay, LuxRender і багатьма іншими;

— інструменти анімації, серед яких інверсна кінематика, скелетна анімація і сіткова деформація, ключові кадри, нелінійна анімація, редагування вагових коефіцієнтів вершин, обмежувачі;

— динаміка м'яких тел (включаючи визначення колізій об'єктів при взаємодії), динаміка твердих тіл на основі фізичного движка Bullet;

— система частинок включає в себе систему волосся на основі частинок;

— модифікатори для застосування неруйнівних ефектів;

— мова програмування Python використовується як засіб визначення інтерфейсу, створення інструментів і прототипів, системи логіки в іграх, як засіб імпорту / експорту файлів (наприклад, COLLADA), автоматизації завдань;

— базові функції нелінійного відео та аудіо монтажу;

— композітінг відео, робота з хромакею;

— трекінг камери і об'єктів;

— real-time контроль під час фізичної симуляції і рендеринга;

— процедурне і node-based текстурування, а також можливість малювати текстуру прямо на моделі;

— grease Pencil - інструмент для 2D-анімації в повному 3D-пайплайні;

— blender Game Engine - підпроект Blender, що надає інтерактивні функції, такі як визначення колізій, рушій динаміки і програмована логіка, також він дозволяє створювати окремі real-time-додатки починаючи від архітектурної візуалізації до відео ігор (вилучений у версії 2.8).

Програмний пакет був розроблений голландською анімаційною студією NeoGeo, його використовували в якості інструменту. Літом 1998 року автор Blender створив компанію Not a Number задля подальшого розвитку пакету. Він поширювався за схемою shareware (тобто умовно-безкоштовною).

На сьогоднішній день Blender є проектом з відкритим вихідним кодом та розвивається при підтримці Blender Foundation. Частіш за все програмний пакет використовується для створення комп’ютерних моделей для ігор, але Blender став відомим завдяки своєму функціоналу, котрий використовувався під час зйомок фільмів. Першим важливим кінофільмом, під час створення якого було задіяно програмний пакет, став кінокомікс «Людина-павук 2». Там його використовували для створення аніматики.

Популярним стало використання Blender й під час малювання мультфільмів, наприклад «Бунт пернатих», комерційний мультиплікаційний проект, був повністю створений у програмному пакеті.

Cinema 4D або скорочено C4D фірми Maxon є пакетом для створення тривимірної графіки і анімації. Cinema 4D є універсальною комплексною програмою для створення і редагування дво- і тривимірних ефектів і об'єктів.

Дозволяє рендерити об'єкти по методу Гуро. Підтримка моделювання, малювання, скульптінга, композітінга, трекінгу, анімації і високоякісного рендерингу. Відрізняється більш простим інтерфейсом, ніж у аналогів, і вбудованою підтримкою російської мови (включаючи повну російськомовну довідку), що робить її популярною серед російськомовної аудиторії.

Cinema 4D підтримує наступні мови програмування:

— Python - Код можна використовувати в менеджері для створення скриптів і плагінів, а також в об'єктах, тегах, вузлах Xpresso і в інших функціях програми;

— C ++ ;

— C.O.F.F.E.E. - скриптова мова програмування;

— Xpresso - нодова система програмування.

Крім вбудованих рендерів (standart CPU, physical CPU, prorender GPU) Cinema 4D може працювати і зі сторонніми рендерами, як вбудованимі безпосередньо в саму середу програми, так і за допомогою конекторів (експортерів). Частина з сторонніх рендерів до R19 безпосередньо підтримувався через вбудований в пакети Cinema 4D Visualize, Cinema 4D Studio і BodyPaint 3D коннектор CineMan.

C # - об'єктно-орієнтована мова програмування. Розроблена в 1998-2001 роках групою інженерів компанії Microsoft під керівництвом Андерса Хейлсберга і Скотта Вільтаумота як мову розробки додатків для платформи Microsoft .NET Framework. Згодом був стандартизований як ECMA-334 і ISO / IEC 23270.

C # відноситься до сім'ї мов з C-подібним синтаксисом, з них його синтаксис найбільш близький до C ++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, змінні, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі в форматі XML.

Перейнявши багато від своїх попередників - мов C ++, Delphi, Модула, Smalltalk і, особливо, Java - C #, спираючись на практику їх використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, C # на відміну від C ++ не підтримує множинне успадкування класів (між тим допускається множинна реалізація інтерфейсів).

Слід відзначити особливості мови C#: вона розроблялась як мова програмування прикладного рівня для CLR і, а тому, перш за все, вона залежить від можливостей самої CLR. Це стосується, перш за все, системи типів C #, яка відображає BCL. Присутність або відсутність тих чи інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльована у відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C #; подібної взаємодії слід

очікувати і в подальшому (проте, ця закономірність була порушена з виходом C # 3.0, що представляє собою розширення мови, що не спираються на розширення платформи .NET).

CLR надає C #, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування.

Проект C # був початий в грудні 1998 і отримав кодову назву COOL (C-style Object Oriented Language). Версія 1.0 була анонсована разом з платформою .NET в червні 2000 року, тоді ж з'явилася і перша загальнодоступна бета-версія; C # 1.0 остаточно вийшов разом з Microsoft Visual Studio .NET в лютому 2002 року.

На сьогодні останньою версією є 8.0.

C ++ - компільована, статично типізована мова програмування загального призначення.

C ++ широко використовується для розробки програмного забезпечення, будучи однією з найпопулярніших мов програмування. Область його застосування включає у себе створення операційних систем, різноманітних прикладних програм, драйверів пристроїв, додатків для вбудованих систем, високопродуктивних серверів, а також ігор.

Існує безліч реалізацій мови C ++, як безкоштовних, так і комерційних і для різних платформ. Наприклад, на платформі x86 це GCC, Visual C ++, Intel C ++ Compiler, Embarcadero (Borland) C ++ Builder і інші. C ++ зробив величезний вплив на інші мови програмування, в першу чергу на Java і C #.

Одним з найяскравіших прикладів використовування мови у створенні комп’ютерних ігор – шутер від третьої особи GTA V компанії RockStar Games. Ігра побудована виключно на мові C++.

Мова виникла на початку 1980-х років, коли співробітник фірми Bell Labs Бйорн Страуструп придумав ряд удосконалень до мови C під власні потреби.

Остання на поточний момент діюча версія стандарту - C ++ 17, опублікована в грудні 2017 року. Основною зміною стало введення в

стандартну бібліотеку паралельних версій стандартних алгоритмів і видалення застарілих і вкрай рідко використовуваних елементів.

Також слід проаналізувати різні направлення комп’ютерних ігор, щоб визначитись із вимогами до майбутнього продукту та його направленістю.

Комп'ютерні ігри можна класифікувати за різними критеріями:

— жанр: геймплей задає жанрову спрямованість, можливо кілька різновидів;

— режим гри: одиночний або розрахований на багато користувачів;

— візуальне уявлення: гра може, як використовувати графічні засоби оформлення, так і навпаки, бути текстової;

— платформозалежність: на яких платформах можливий запуск гри, так само гри бувають платформонезалежні (кросплатформні).

Класифікація за жанрами:

— пригодницька гра або квест (Adventure) - гра з продуманим і зазвичай лінійним сюжетом;

— бойовик (Action) - гра, яка вимагає від гравця постійних дій, насичена бойовими сценами, бійками і перестрілками;

— рольова гра (RPG - англ. Role Playing Game) - гра, відмінною рисою якої є наявність у персонажів певних навичок і характеристик, які можна знайти, а згодом розвивати, виконуючи будь-які дії;

— стратегічна гра (Strategy) - гра, що представляє собою управління глобальними процесами, як наприклад, розвиток економіки, створення армії, будівництво парків і т.д;

— комп'ютерний симулятор (Simulator) - гра, створена з метою симуляції реальних дій в житті, наприклад керування автомобілем або симулятор пілота.

Інтегроване середовище розробки (IDE) - це багатофункціональна програма, яка підтримує багато аспектів розробки програмного забезпечення. Інтегроване середовище розробки Visual Studio - це стартовий майданчик для написання, налагодження і складання коду, а також подальшої публікації

додатків. Крім стандартного редактора і відладчика, які є в більшості середовищ IDE, Visual Studio включає в себе компілятори, засоби автозавершения коду, графічні конструктори і багато інших функцій для поліпшення процесу розробки.

Серед основних переваг IDE-середовища Visual Studio можна виділити нижчеперелічені.

Вбудований Web-сервер. Для обслуговування Web-додатку ASP.NET необхідний Web-сервер, який буде очікувати Web-запити і обробляти відповідні сторінки. Наявність в Visual Studio інтегрованого Web-сервера дозволяє запускати Web-сайт прямо з середовища проектування, а також підвищує безпеку, виключаючи ймовірність отримання доступу до тестового Web-сайту з якого-небудь зовнішнього комп'ютера, оскільки тестовий сервер може приймати з'єднання лише з локального комп'ютера.

Підтримка безлічі мов при розробці. Visual Studio дозволяє писати код своєю рідною мовою чи будь-якою з інших бажаних мов, використовуючи весь час один і той же інтерфейс (IDE). Більш того, Visual Studio також ще дозволяє створювати Web-сторінки на різних мовах, але поміщати їх все в один і той же Web-додаток. Єдиним обмеженням є те, що в кожної Web-сторінці можна використовувати тільки якусь одну мову.

Інтуїтивний стиль кодування. За замовчуванням Visual Studio форматує код у міру його введення, автоматично вставляючи необхідні відступи і застосовуючи колірне кодування для виділення елементів типу коментарів. Такі незначні відмінності роблять код більш зручним для читання і менш схильним до помилок. Застосовувані Visual Studio автоматично параметри форматування можна навіть налаштовувати, що дуже зручно у випадках, коли розробник вважає за краще інший стиль розміщення дужок (наприклад, стиль K & R, при якому відкриває дужка розміщується на тому самому рядку, що і оголошення, якому вона передує).

Більш висока швидкість розробки. Багато з функціональних можливостей Visual Studio спрямовані на те, щоб допомагати розробнику робити свою роботу якомога швидше. Зручні функції, на зразок функції IntelliSense (яка вміє перехоплювати помилки і пропонувати правильні варіанти), функції пошуку і заміни (яка дозволяє відшукувати ключові слова як в одному файлі, так і в усьому проекті) і функції автоматичного додавання і видалення коментарів (яка може тимчасово приховувати блоки коду), дозволяють розробнику працювати швидко і ефективно.

Можливості налагодження. Пропоновані в Visual Studio інструменти налагодження є найкращим засобом для відстеження загадкових помилок і діагностування дивної поведінки. Розробник може виконувати свій код по рядку за раз, встановлювати інтелектуальні точки переривання, при бажанні зберігаючи їх для використання в майбутньому, і в будь-який час переглядати поточну інформацію з пам'яті.

1.2 Висновки за розділом

На першому етапі розробки дипломного проекту проаналізовані можливі варіанти виконання роботи, порівняні програмні середовища на предмет функціональності. Встановлено жанр майбутньої роботи та спосіб реалізації проекту за допомогою програмних середовищ.

РОЗДІЛ 2
АНАЛІЗ ПЕРЕВАГ ОБРАНИХ МЕТОДІВ

2.1 Виявлення переваг обраних програмних середовищ

Для виконання «Дослідження етапів розробки комп’ютерних симуляторів із застосуванням Unity 3D» було вирішено користуватись середою Unity 3D, програмним пакетом Blender та мовою програмування C#.

У Unity скрипти можна використовувати для розробки практично будь якого елементу гри або інтерактивного контенту з графікою реального часу. Unity підтримує скрипти на C #, створені відповідно до одного з двох основних підходів: традиційним і широко використовуваним об'єктно-орієнтованим підходом і інформаційно-орієнтованим підходом, який тепер теж підтримується в Unity в окремих випадках завдяки високопродуктивному багатопотоковому стеку інформаційно-орієнтованих технологій (DOTS).

Традиційна модель «ігровий об'єкт - компонент» добре працює і сьогодні, оскільки вона проста як для програмістів, так і інших користувачів, а також зручна для створення інтуїтивних інтерфейсів.

DOTS дозволяє вашій грі ефективно використовувати всі можливості новітніх багатоядерних процесорів. Стек складається з наступних компонентів:

— система завдань C # для ефективного виконання коду на багатопотокових системах;

— Entity Component System (ECS) для розробки високопродуктивного коду за замовчуванням;

— компілятор Burst для компіляції скриптів в оптимізований нативний код.

Багатопотокові системи DOTS допомагають створювати ігри для самих різних пристроїв і розробляти багаті ігрові світи з великим числом елементів і

складними симуляціями. Продуктивний код, в свою чергу, знижує тепловиділення і продовжує час автономної роботи мобільних пристроїв. Перехід від об'єктно-орієнтованого до інформаційно-орієнтованого підходу спрощує багаторазове використання коду, а іншим дозволяє легше зрозуміти і доповнити його при необхідності.

Налаштування та налагодження в Unity ефективні, тому що всі змінні ігрового процесу відображаються безпосередньо в процесі гри, що дозволяє міняти їх одразу без додаткового програмування. Гру можна призупинити в будь-який момент або переходити від одного оператора до іншого по черзі.

Blender - це безкоштовне програмне забезпечення для створення і редагування тривимірної графіки. З огляду на кросплатформність, відкритий вихідний код, доступність і функціональність пакет отримав популярність не тільки серед новачків, а й серед просунутих 3D-моделерів. У міру розвитку програми її вибирають в якості робочого інструменту для все більш серйозних проектів. Цей додаток практично не поступається за кількістю можливостей і функціоналу більш просунутим пакетам 3D графіки. І при цьому все безкоштовно.

На сьогоднішній день це повноцінний 3D редактор, в якому користувача зустрічає повністю програмований інтерфейс і унікальна внутрішня файлова система. Оболонка програми на перший погляд може здатися незручною і незрозумілою, але після налаштування гарячих клавіш працювати в Blender стає просто і зручно. В якості мови програмування додаток використовує Python, володіючи яким ви можете створювати власні інструменти, редагувати інтерфейс і сам принцип роботи програми. Приємним бонусом є доступність пакета на різних операційних системах обох розрядностей: освоїти програму зможуть власники комп'ютерів з ОС Windows, GNU / Linux і Mac OSX.

Функції програми:

— 3D-моделювання;

— анімація;

— текстурування та набори шейдерів;

— можливість малювання;

— візуалізація;

— базовий відеоредактор;

— ігровий рушій.

Для роботи над проектом обрано використовувати мову C#, тому слід проаналізувати його переваги над аналогами. Перш за все – на цій мові пишуться скрипти у середовищі Unity, що є вагомою перевагою, враховуючи вищезазначені ресурсі, які будуть використані.

Мова програмування C # претендує на справжню об'єктну орієнтованість (будь-яка мовна сутність претендує на те, щоб бути об'єктом).

Компонентно-орієнтований підхід до програмування, який сприяє меншій машинно-архітектурній залежності результуючого програмного коду, гнучкості і легкості повторного використання (фрагментів) програм. Мова більш орієнтована на безпеку коду, якщо порівнювати її з C++.

C# має уніфіковану систему типізації, та у неї розширена підтримка подієво-орієнтованого програмування.

З огляду на об'єктно-орієнтований дизайн, C# є гарним вибором для швидкого конструювання різних компонентів - від високорівневої бізнес логіки до системних додатків, що використовують низькорівневий код. Також слід зазначити, що C# є і Web орієнтованим - використовуючи прості вбудовані конструкції мови компоненти можуть бути перетворені на Web сервіси. Додатковими можливостями мови C# є використання Web технологій, таких як: XML (Extensible Markup Language) і SOAP (Simple Object Access Protocol).

2.2 Висновки за розділом

У другому розділі перелічені основні переваги обраних засобів для створення проекту. Описано функціональність кожного середовища та інструмента. Обрані засоби порівняні з аналогами.

РОЗДІЛ 3
ПРОЕКТУВАННЯ РОЗРОБЛЮВАНОГО СИМУЛЯТОРА

3.1 Створення ігрового світу та додання у нього головного героя

Для створення комп’ютерних ігор та симуляторів спочатку слід визначитись із жанром та основною концепцією майбутнього продукту. Від ідеї до релізу продукт зазвичай проходить певні етапи:

— придумування ідеї: що взагалі собою представлятиме проект, чого гравець повинен досягти для перемоги, який кінцевий результат;

— складання технічного завдання: воно полягає в перекладі спільного бачення задумки на технічну мову, її конкретизацію, формування більш чіткого уявлення про те, що потрібно робити фахівцям, які займаються проектом;

— дизайн: формування вигляду того, що гравець буде бачити на екрані свого ПК, смартфону, іншого гаджета;

— написання коду фронт частини: цей етап включає в себе програмування логіки гри, руху персонажів і т. д;

— написання серверного коду - це створення мережевої частини проекту, випуск його в інтернет;

— менеджмент - визначень його дуже багато, але в цілому все зводиться до формування професійної команди, грамотному розподілу обов'язків між її членами, організації ефективної взаємодії окремих учасників;

— тестування - складається тестування зазвичай з двох частин - закритої і відкритої: у першій беруть участь досвідчені тестувальники, які мають особливі навички, виявляються в основному критичні баги і помилки з якими показувати проект публіці неможливо, друга частина - це вже тестування звичайними користувачами і оптимізація проекту під великий мережевий трафік, відпрацювання монетизації.

Перш за все слід підкреслити, що фінальним продуктом буде симулятор пейнтболу, а це означає, що найбільш зручним и наближеним до реальності форматом буде створення FPS-шутеру.

FPS-шутер (шутер від першої особи) - жанр комп'ютерних ігор, в яких ігровий процес ґрунтується на боях з використанням вогнепальної або будь-якої іншої зброї з видом від першої особи.

Отже ігровий процес буде проходити від першої особи, основною зброєю гравця буде пейнтбольна рушниця (рис. 3.1)

Рисунок 3.1 – Ігровий процес від першої особи

Саме поняття комп’ютерного симулятора означає те, що він повинен максимально наближено відтворювати реальний процес. Тому слід ознайомитись та поглибитись у правила та основні нюанси гри у пейнтбол.

Пейнтбол - технічна спортивна гра з розряду екстремальних, що імітує швидкоплинні вогневі контакти на обмеженому просторі. Для такої імітації використовується спеціальний пневматичний пристрій (маркер) і желатинові кульки з фарбою. Ігри проводяться на лісових або відкритих полях, приміщеннях, з природними або штучними перешкодами і укриттями. Розрізняють пейнтбол як вид відпочинку - для всіх бажаючих, як спосіб

тренування - для охоронців, поліцейських або спецназу і як спорт - для турнірних команд. Поодинці або командою (від 3-х і більше осіб).

Саме традиційне поле - лісове, з додаванням укриттів з колод, покришок або бочок. Іноді поля настільки заросли чагарником, що додаткові укриття просто не потрібні. На відкритих майданчиках будуються штучні укриття з фанерних щитів, старих машин і т.д.

Спідбольні поля - невеликі, з однотипними симетрично розташованими укриттями будуються для швидкісної, високотехнічні гри.

Гіпербол - гра на відкритих полях з укриттями з пластикових труб різного діаметру.

Отже, виходячи з основ гри у пейнтбол, можна приступати до безпосередньої розробки продукту і почати слід із середовища, в умовах якого гравець буде діяти – лісу.

У середі Unity скористуємось інструментом Terrain Toolbox (рис. 3.2) та створимо пустий ландшафт і задамо йому параметри висоти, ширини та довжини.

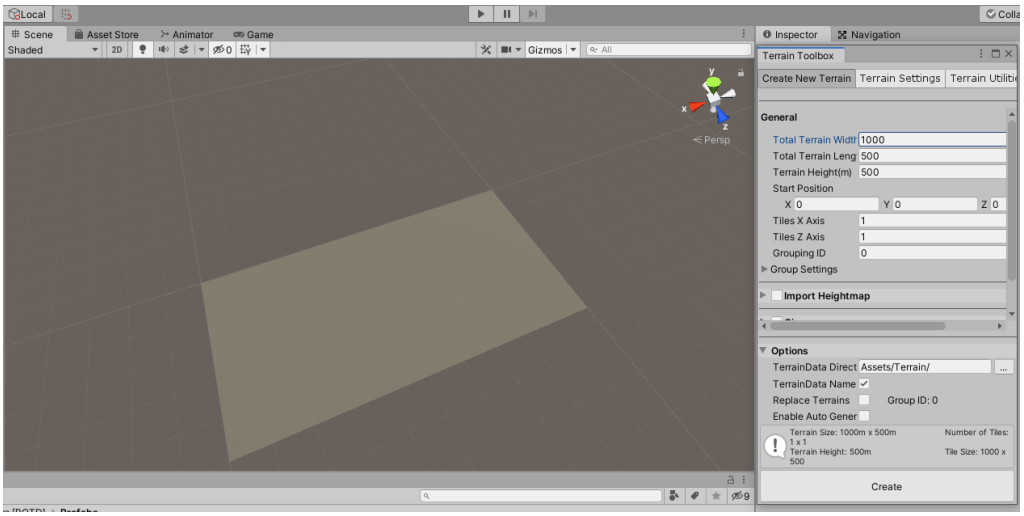


Рисунок 3.2 – Початок створення ландшафту

Використавши текстуру трави на створеному ландшафту було отримано засаджену зеленню місцевість (рис.3.3).

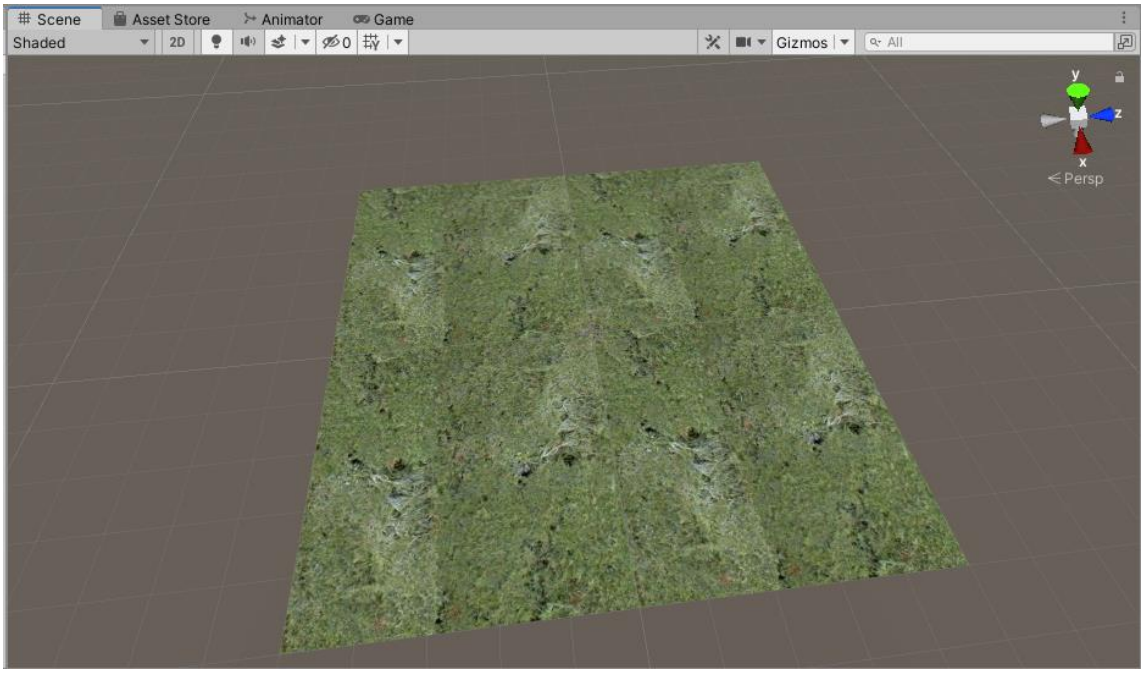


Рисунок 3.3 – Формування ландшафту

Тепер можна перейти до створення середи для проведення гри у пейнтбол. Гравцям потрібно мати укриття, з яких вони зможуть вести вогонь по своїм супротивникам та ховатися самі від них. Ігри у пейнтбол можуть бути проведені як у лісі, так і у старих приміщеннях, які вже не використовуються (будинки, ангари, старі офісні центри). Отже у середовищі Blender було створено було створено декілька моделей невеликих будинків, які можуть стати гарною вогневою позицією (рис 3.4 – 3.5).

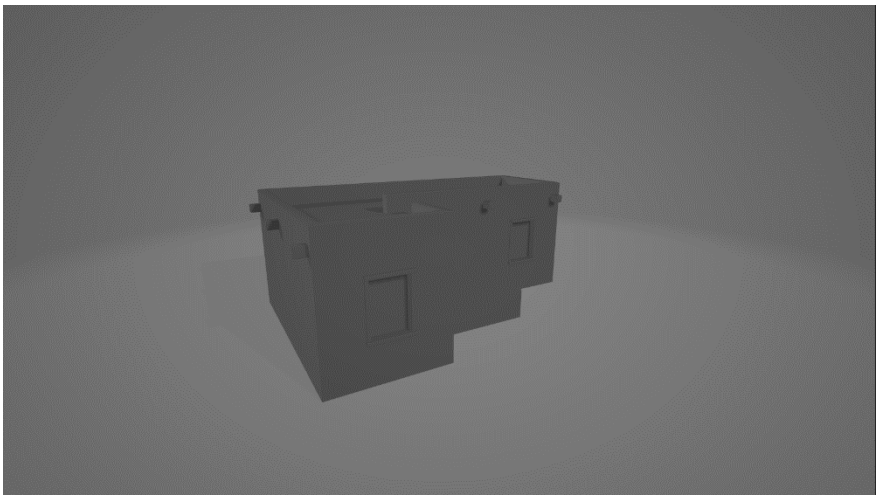


Рисунок 3.4 – Модель будинку 1

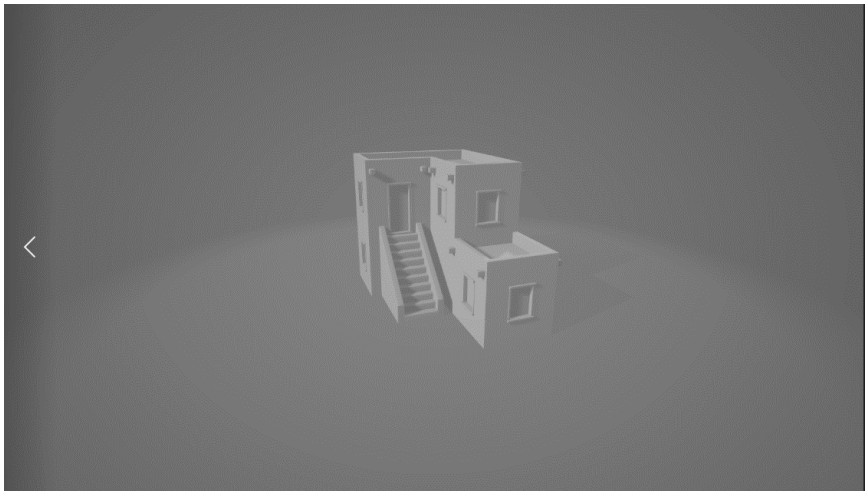


Рисунок 3.5 – Модель будинку 2

Після створення моделей на них було використані текстури, які зробили їх більш схожими на реальні будинки. На вікнах були використані скляні та дерев’яні текстурні матеріали, на стінах – цементні (рис. 3.6).

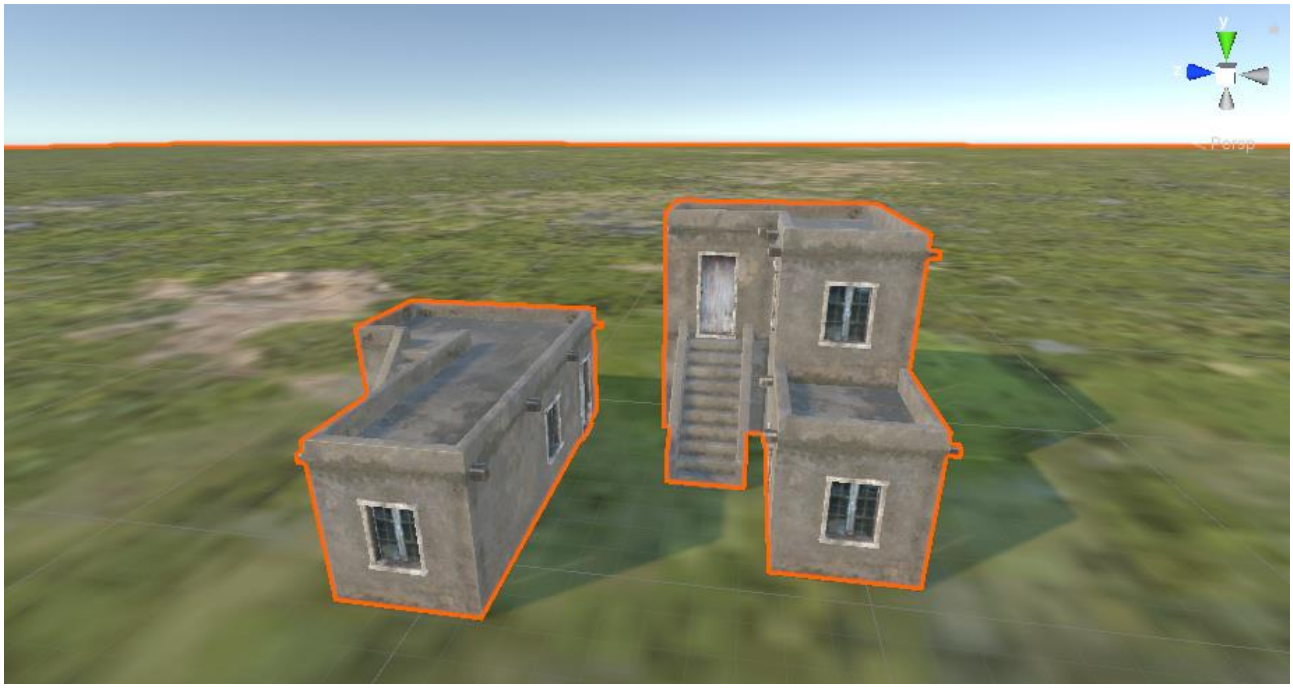


Рисунок 3.6 – Результат використання текстурних матеріалів

Окрім будинків було створено моделі бочок (рис. 3.7), дерев (рис. 3.8), колодязя (рис. 3.9), перешкод (рис. 3.10), каменів (рис. 3.11) та кущів (рис. 3.12), щоб майбутня карта не виглядала занадто просто.

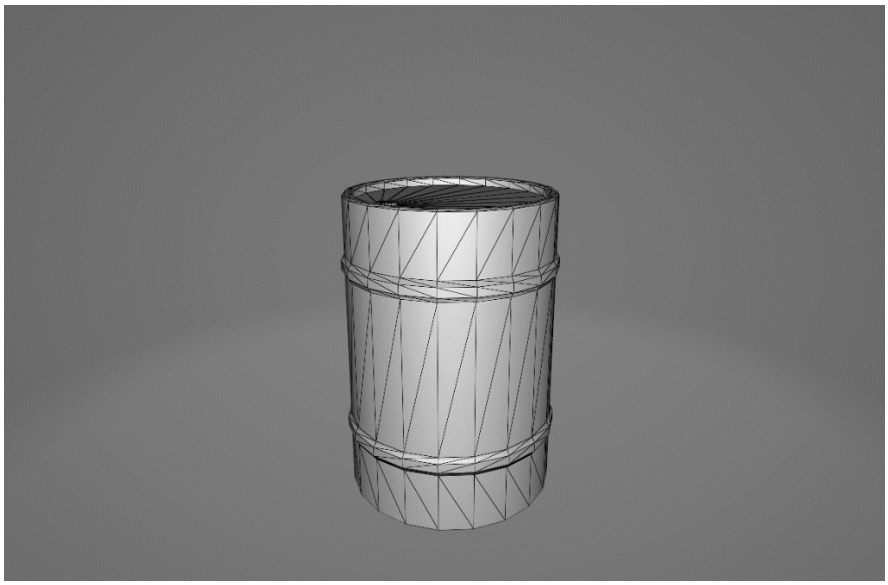


Рисунок 3.7 – Модель бочки

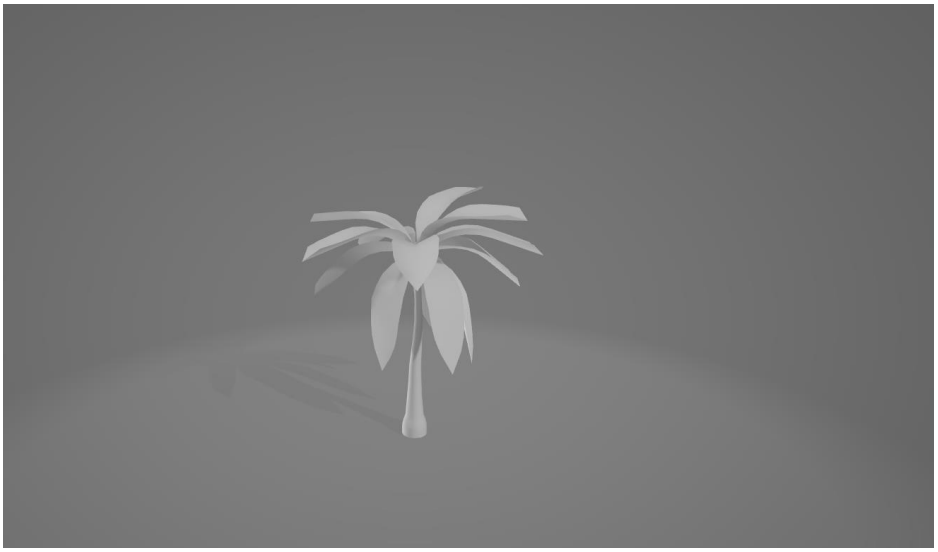


Рисунок 3.8 – Модель дерева

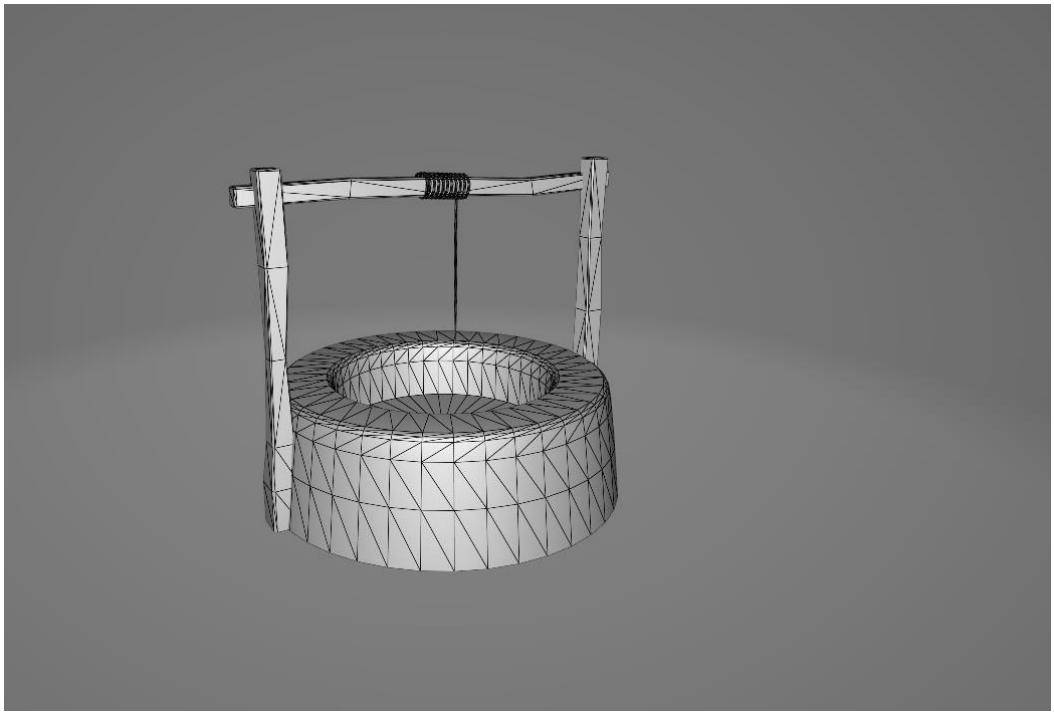


Рисунок 3.9 – Модель колодязя

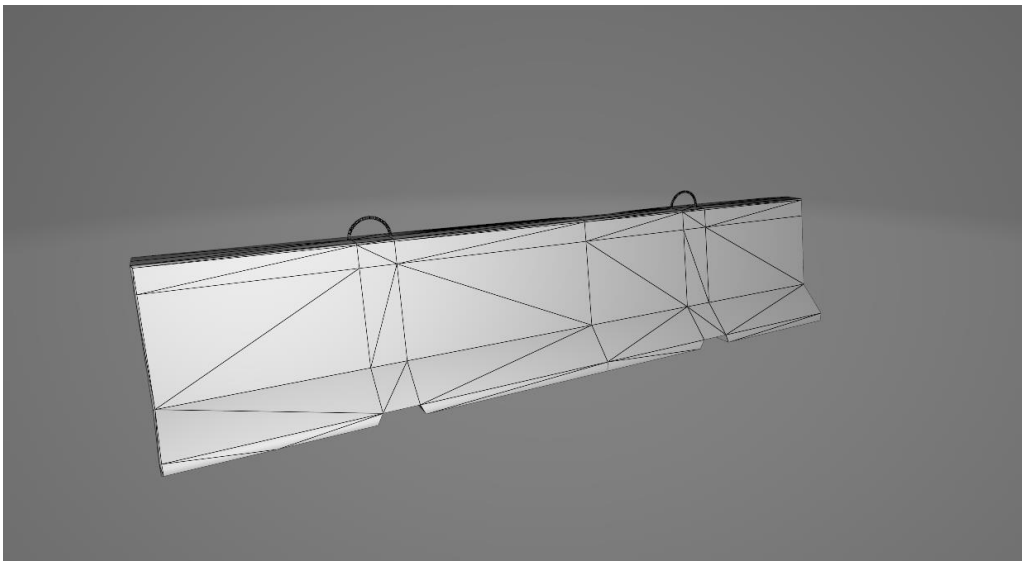


Рисунок 3.10 – Модель перешкоди

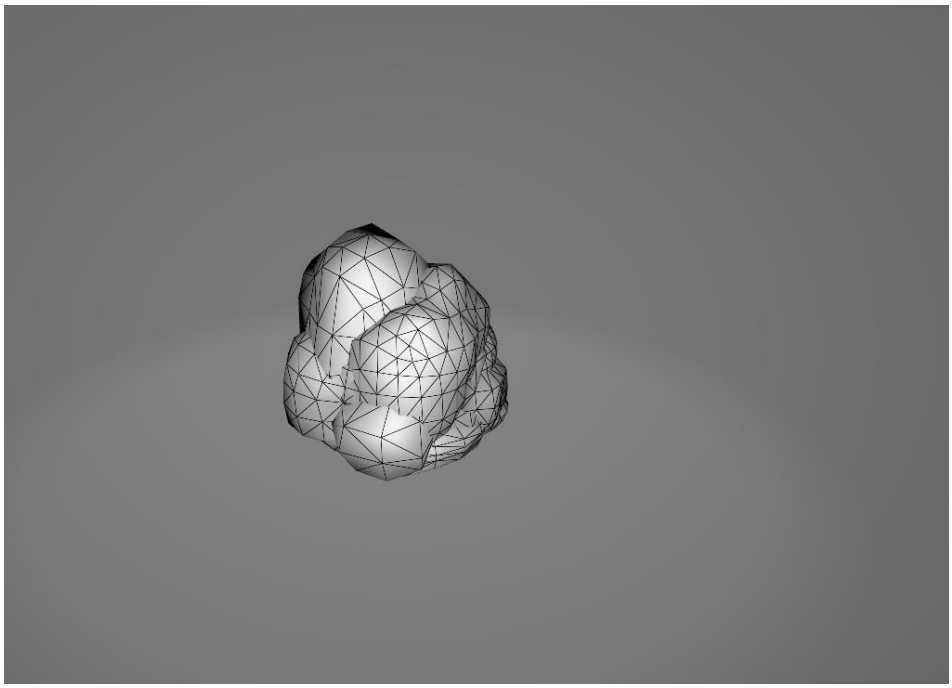


Рисунок 3.11 – Модель каменю

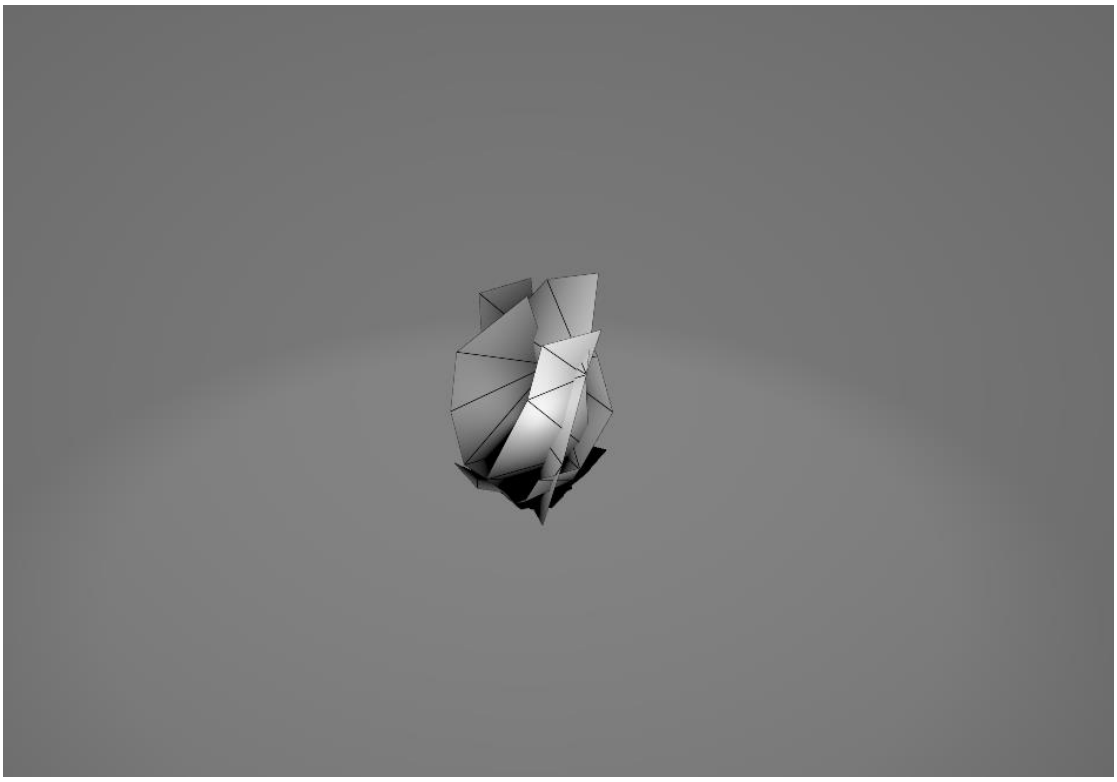


Рисунок 3.12 – Модель куща

3.2. Конструювання ігрового поля

Створивши необхідні елементи для заповнення вільного місця можна приступати до безпосереднього наповнення карти об’єктами. Першою картою буде спеціально відведене місце для гри, у центрі якого буду стояти міст – лінія розмежування території оппонентів (рис. 3.13 - 3.14).

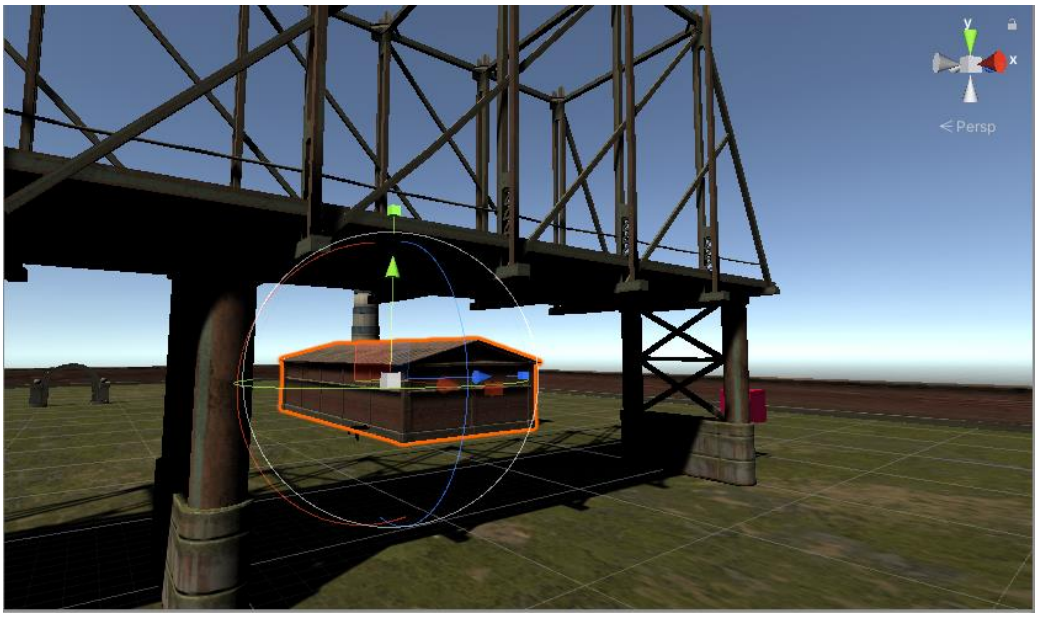


Рисунок 3.13 – Створення карти подій для гравців

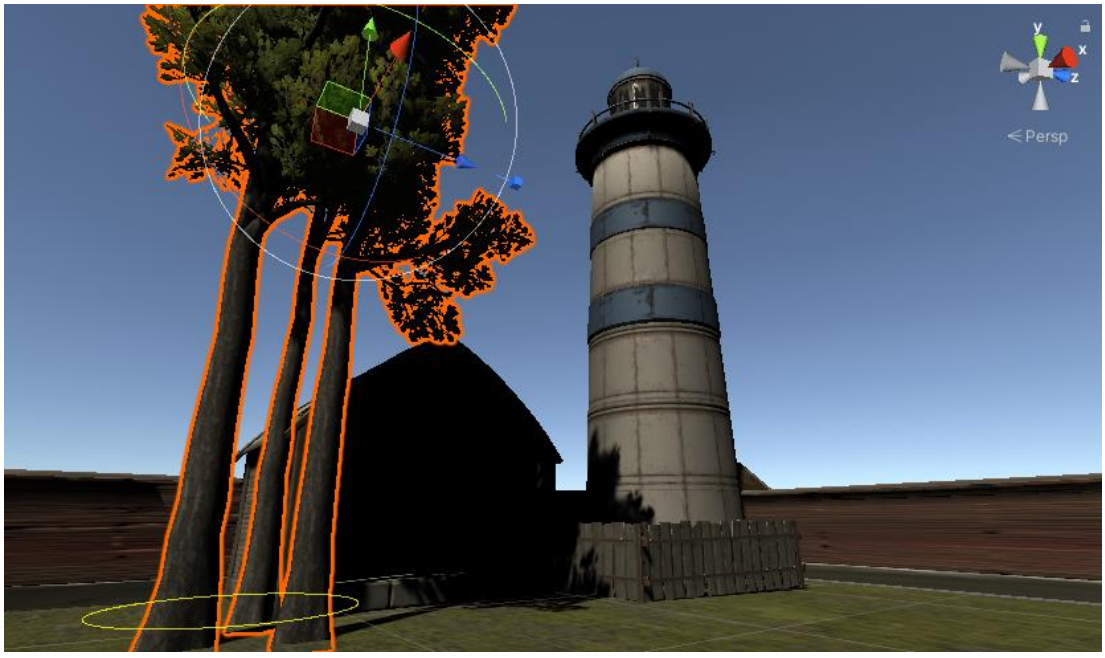


Рисунок 3.14 – Створення головної карти

3.3. Введення у гру персонажа користувача

Головним персонажем симулятора буде звичайна людина, яка прийшла випробувати себе у грі в пейнтбол, тому створімо макет гравця (рис. 3.15).

Рисунок 3.15 – Макет головного героя

Слід відмітити, що майбутній симулятор у своїй сутності буде шутером від першої особо, а тому і геймплейно буде відповідати цьому жанру.

Ігровий процес, або геймплей (англ. Gameplay), - компонент гри, який відповідає за взаємодію гри і гравця. Геймплей описує, як гравець взаємодіє з ігровим світом, як ігровий світ реагує на дії гравця і як визначається набір дій, який пропонує гравцеві гра. Термін частіше вживається в контексті комп'ютерних ігор. Геймплей безумовно не відноситься до таких компонентів гри, як графіка і звуковий супровід. Він являє собою патерн взаємодії гравця з грою на підставі її правил, визначає зв'язок між гравцем і грою, пропонований ігровий виклик і способи його подолання, сюжет як участь в ньому гравця. Терміни ігровий процес і геймплей можуть використовуватися як синоніми.

Отже додамо текстуру до моделі героя та помістимо його у ігровий світ (рис 3.16 – 3.17).

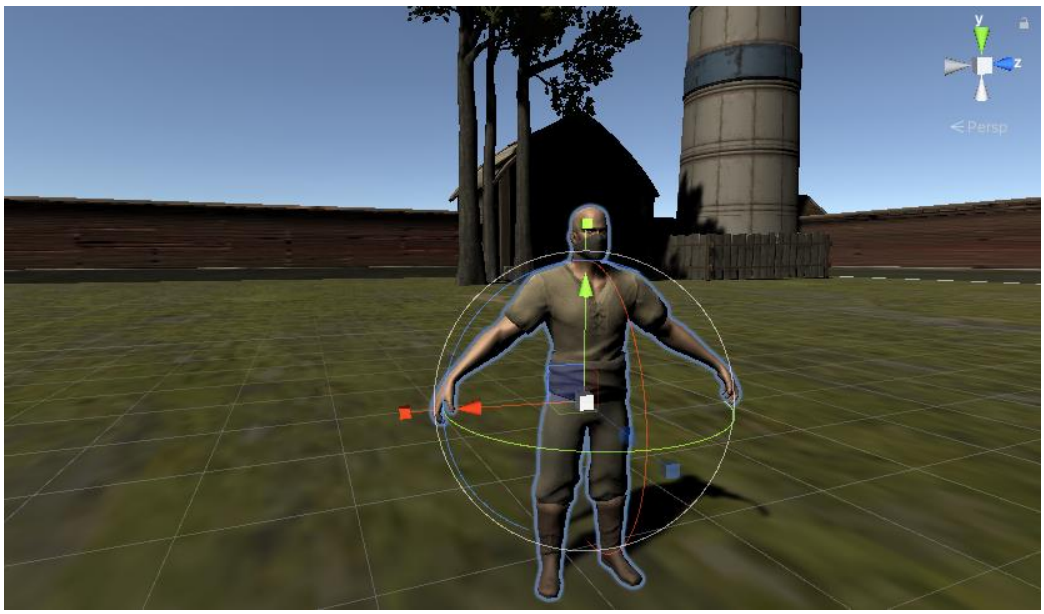


Рисунок 3.16 – Додання моделі персонажу у ігровий світ

Оскільки симулятор буде шутером від першої особи, то і бачити гравець повинен перед собою. Тому було створено камеру гравця, яка була поміщена на рівні очей моделі персонажу (рис. 3.18). Отриманий результат приведено на (рис. 3.19).

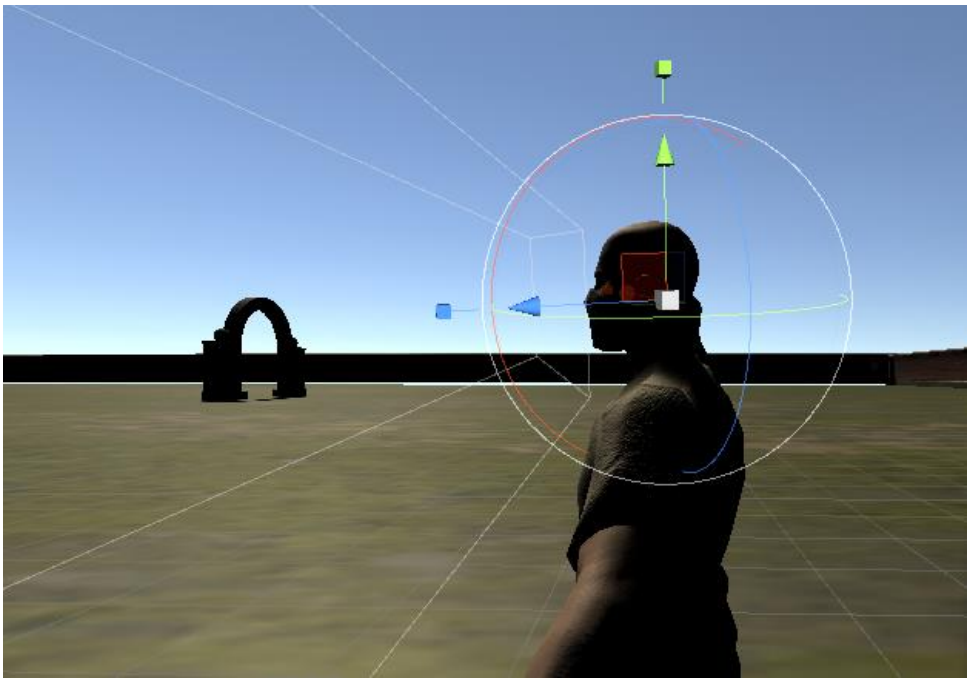


Рисунок 3.18 – Додання камери гравця

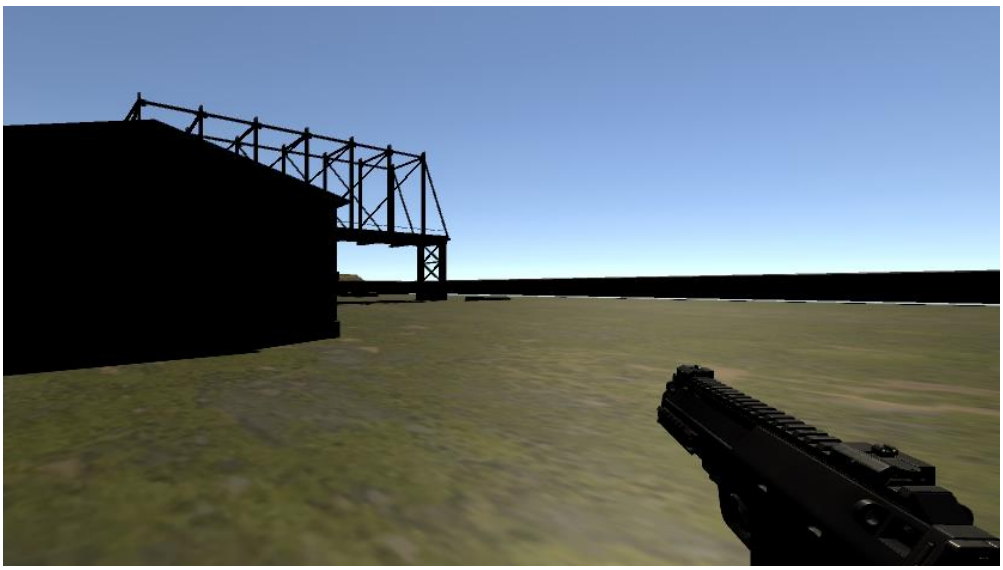


Рисунок 3.19 – Вигляд камери у грі

3.4. Створення скриптів для моделювання поведінки головного героя

Перед створенням моделі поведінки за допомогою скрипту, слід ознайомитись із тим, як вони працюють, та отримати розуміння його життєвого циклу (рис 3.20 – 3.21).

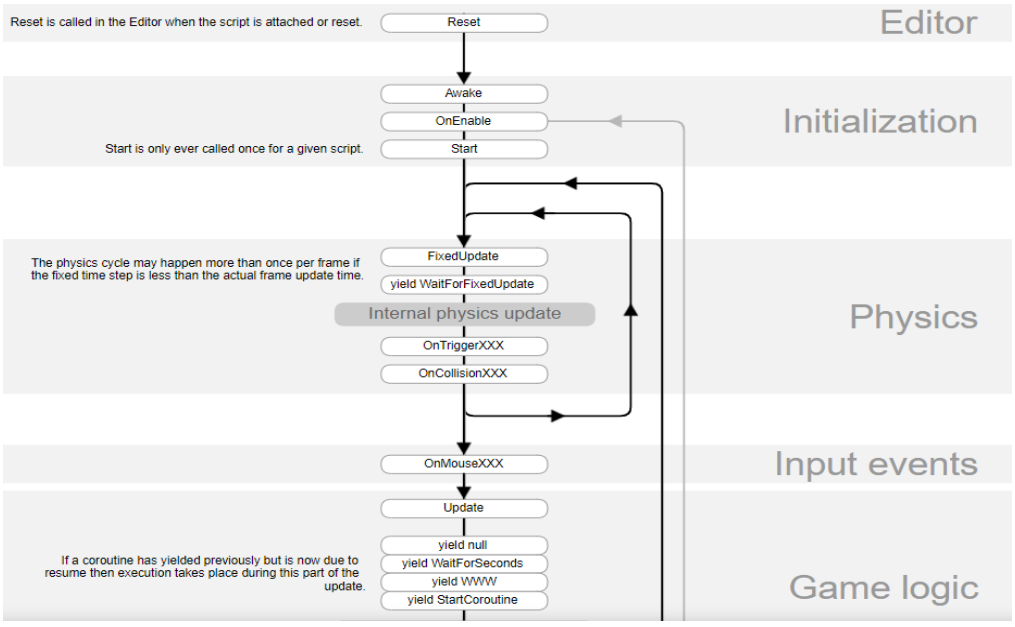


Рисунок 3.20 – Життєвий цикл скрипта 1

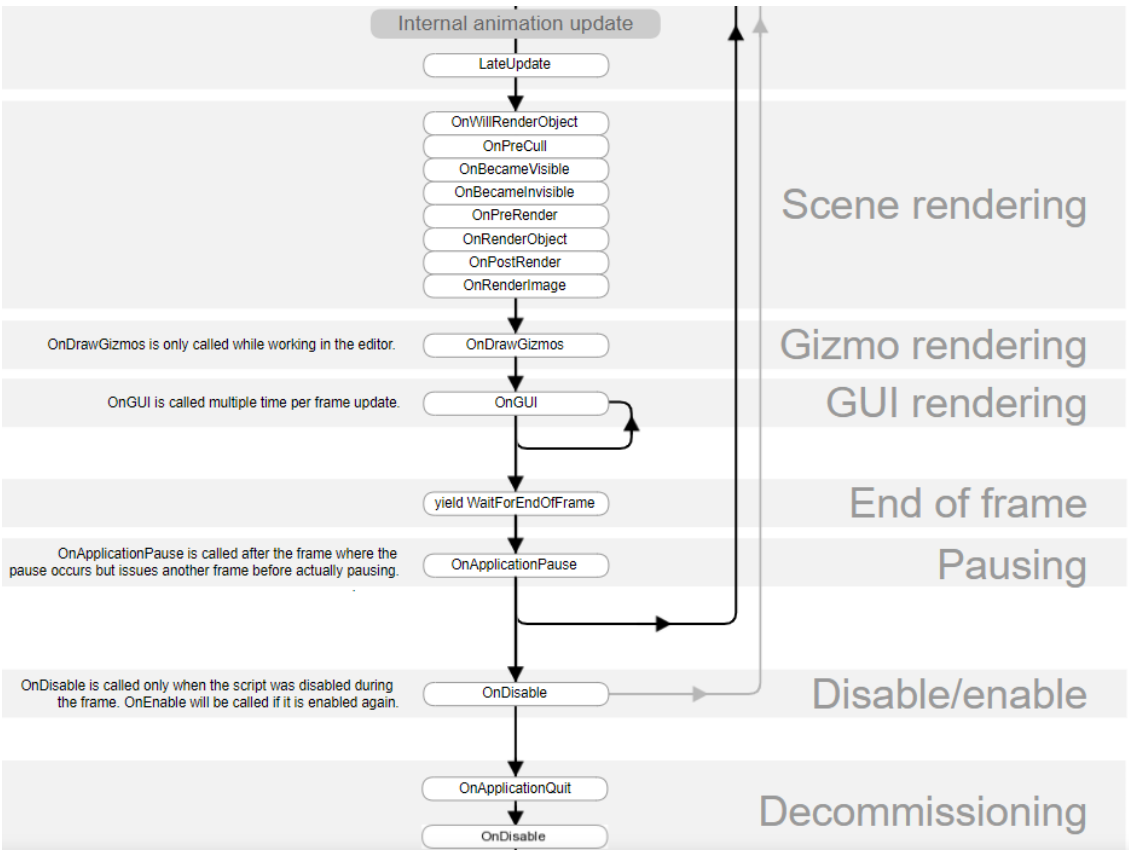


Рисунок 3.21 – Життєвий цикл скрипта

Усі перелічені на рисунках вище методи можуть використовуватись у скрипті, але не обов’язково усі – лише ті, що потрібні. Найголовніше – запам’ятати порядок виклику цих методів, щоб усе працювало вірно.

Найпершим викликається метод Reset, який потрібен для редагування даних.

Задля ініціалізації даних використовуються події Awake, OnEnable, Start.

Awake буде спрацьовувати тоді, коли сцена проекту ще не завантажилась, після чого спрацює вже Start.

Методи FixedUpdate, OnTrigger, OnCollision використовують для взаємодії з фізикою.

Задля додання різних подій є метод OnMouse.

Методи ігрової логіки мають назви Update, yield null, yield WaitForSeconds, yield WWW, yield StartCoroutine.

Для відображення сцени проекту використовують OnWillRenderObject, OnBecameVisible, OnPreRender, OnRenderObject та інші.

За відображення інтерфейсу відповідають методи OnDrawGizmos та OnGUI.

Для активації та деактивації різного роду об’єктів використовують метод OnDisable.

Методи OnApplicationQuit та OnDestroy дозволяють видаляти непотрібні об’єкти на сцені.

Тепер можна приступити безпосередньо до створення скриптів. Перш за все треба навчити головного героя пересуватись по ігровому світу.

У проекті Unity було створено скрипт FPSMovement (рис. 3.22), який буде відтворювати натискання на клавіші у команди, які рухають головного героя.

```
[RequireComponent(typeof (CharacterController))]
[RequireComponent(typeof (AudioSource))]
public class FirstPersonController : MonoBehaviour
{
    [SerializeField] private bool m_IsWalking;
    [SerializeField] private float m_WalkSpeed;
    [SerializeField] private float m_RunSpeed;
    [SerializeField] [Range(0f, 1f)] private float m_RunstepLenghten;
    [SerializeField] private float m_JumpSpeed;
    [SerializeField] private float m_StickToGroundForce;
    [SerializeField] private float m_GravityMultiplier;
    [SerializeField] private MouseLook m_MouseLook;
    [SerializeField] private bool m_UseFovKick;
    [SerializeField] private FOVKick m_FovKick = new FOVKick();
    [SerializeField] private bool m_UseHeadBob;
    [SerializeField] private CurveControlledBob m_HeadBob = new CurveControlledBob();
    [SerializeField] private LerpControlledBob m_JumpBob = new LerpControlledBob();
    [SerializeField] private float m_StepInterval;
    [SerializeField] private AudioClip[] m_FootstepSounds;    // an array of footstep sounds that will be randomly selected from.
    [SerializeField] private AudioClip m_JumpSound;           // the sound played when character leaves the ground.
    [SerializeField] private AudioClip m_LandSound;           // the sound played when character touches back on ground.

    private Camera m_Camera;
    private bool m_Jump;
    private float m_YRotation;
    private Vector2 m_Input;
    private Vector3 m_MoveDir = Vector3.zero;
    private CharacterController m_CharacterController;
    private CollisionFlags m_CollisionFlags;
    private bool m_PreviouslyGrounded;
    private Vector3 m_OriginalCameraPosition;
    private float m_StepCycle;
    private float m_NextStep;
    private bool m_Jumping;
    private AudioSource m_AudioSource;
```

Рисунок 3.22 – Фрагмент скрипту FPSMovement

У скрипті було реалізовано пересування персонажу, також під час бігу або ходьби буде відтворюватись звуки шагів, які були заготовлені раніше.

Головному герою була надана змога перестрибувати перешкоди, під час приземлення також буде відтворюватись відповідний звук. Слід не забувати, що камера повинна пересуватись разом із поглядом гравця, тому головна камера була прив’язана до миші. Тепер вона пересувається відповідно до рухів миші, які робить гравець.

Завдяки скрипту можна встановити швидкість пересування персонажу та висоту стрибків, щоб усе було максимально наближено до реальності.

Після додання моделі поведінки на головного героя пересуватись по мапі, але йому треба ще й захищатись від оппонентів, які будуть додані пізніше.

Отже, наступний етап – створення стрільби.

3.5. Стрільба персонажу та перевірка на попадання у ціль

На даному етапі слід враховувати, що пейнтбол – це спортивна гра, у якій замість куль використовуються шари із фарбою, тому під час пострілу відсутні будь-які ефекти, такі як дим, або спалахи.Насамперед було створено скрипт (рис. 3.23), який буде регулювати темп стрільби, відтворення відповідних звуків, та який відстежує попадання у ціль.

```
public class Weapom : MonoBehaviour
{
    public float damage = 100;
    public float fireRate = 1;
    public float force = 155;
    public float range = 15;
    public AudioClip shotSFX;
    public AudioSource _audioSource;
    public ParticleSystem muzzleFlash;
    public Transform bulletSpawn;
    public GameObject hitEffect;


    public Camera _cam;

    // Update is called once per frame
    void Update()
    {
        if (Input.GetButtonDown("Fire1"))
        {
            Shoot();
        }

    }

    void Shoot()
    {

        _audioSource.PlayOneShot(shotSFX);
```

Рисунок 3.23 – Скрипт відтворення пострілу

У даному скрипті була встановлена дальність польоту кулі, пауза між пострілами, бо пейнтбольна рушниця не має автоматичного режиму, було

створено об’єкт bulletSpawn, який надалі буде прикріплено до стволу рушниці. З нього будуть виходити кулі, також була встановлена границя ушкоджень, які будуть наноситись супротивнику.

Під час пострілу буде відтворюватись певний звук. Проводити постріл можна буде за допомогою лівої кнопки миші, яка у середовищі Unity позначається як Fire1. Перевірити, чи була натиснута відповідна клавіша, можна завдяки умови if та зверненню до управління, тобто методу Input.GetButtonDown.

Постріл буде проводитись за методом Raycast (рис. 3.24), тобто відтворення непомітних променів, які будуть супроводжуватись польотом кулі. Також у скрипті було додано відтворення інформації, куди саме потрапив промінь. Інформація про це буде виводитись у Log.

```
        RaycastHit hit;

        if (Physics.Raycast(_cam.transform.position, _cam.transform.forward, out hit, range))
        {
            Debug.Log("Popadaninye" + hit.collider);

            GameObject impact = Instantiate(hitEffect, hit.point, Quaternion.LookRotation(hit.normal));
            Destroy(impact, 2f);

            if(hit.rigidbody !=null)
            {
                hit.rigidbody.AddForce(-hit.normal * force);
            }
        }
    }
}
```

Рисунок 3.24 – Підключення Raycast

Після додання скрипту на рушницю, у середовищі Unity з’явився об’єкт bulletSpawner (рис. 3.25), який було надалі прикріплено до дула зброї.

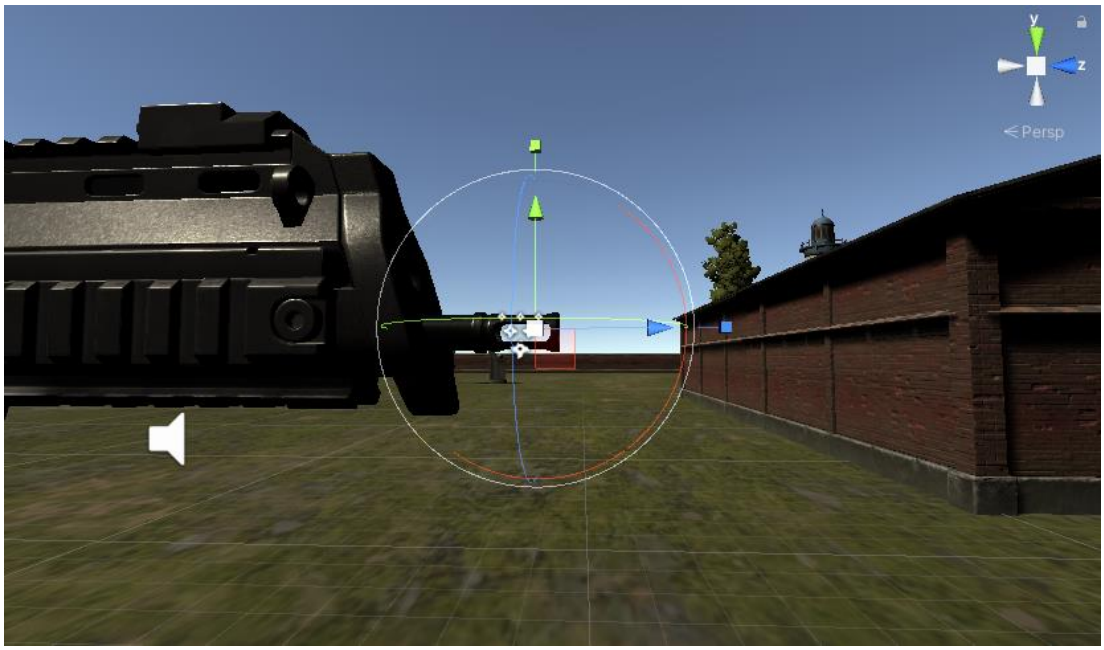


Рисунок 3.25 – Додання bulletSpawner на зброю

Тепер створена зброя має декілька параметрів (рис. 3.26), які надалі були заповнені.

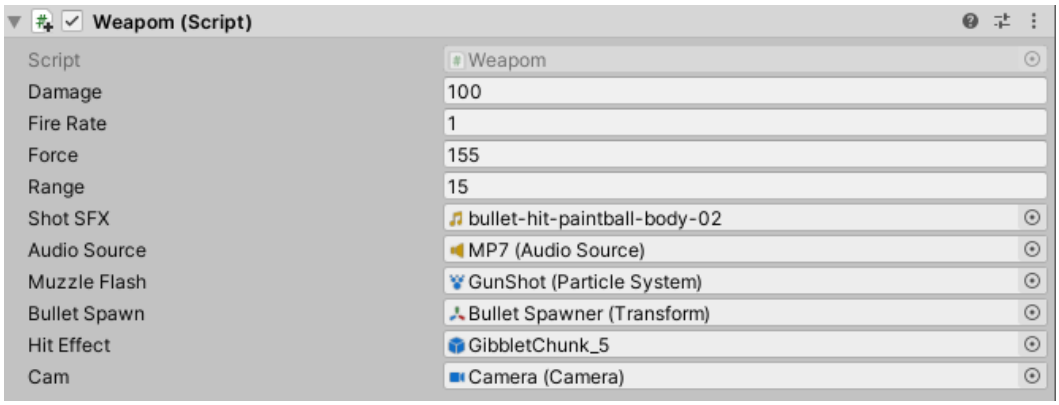


Рисунок 3.26 – Параметри зброї

Як можна бачити, до рушниці були додані ефекти пострілу, джерело, звідки буде братись звук та ефект, тобто бризки фарби від попадання.

У момент попадання фарба буде розбризкуватись по цілі, як це приведено на (рис 3.27).

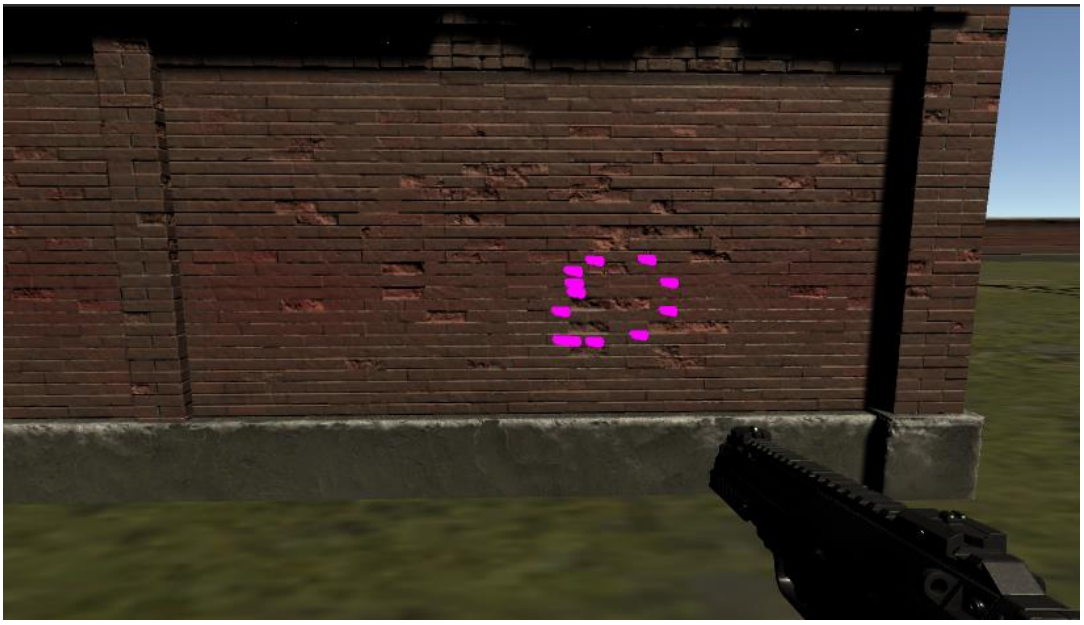


Рисунок 3.27 - Ефект влучення по цілі

3.6. Створення противників та додання їм штучного інтелекту

На цьому етапі до ігрового світу вводяться команда-супротивник. Їх моделі будуть відрізнятись від іншої команди, щоб не сплутати своїх із чужими.

Перш за все слід перемістити першого ворога до мапи (рис 3.28).

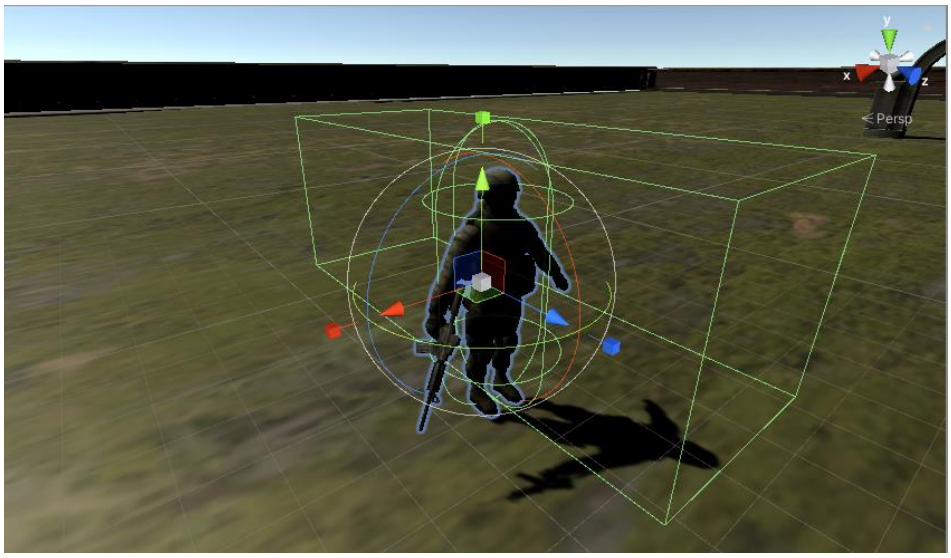


Рисунок 3.28 – Модель ворога

Тепер анімуємо його поведінку. Ворог буде мати анімації бігу, тримання рушниці, стрільби та смерті.

Порядок анімацій вибудовується завдяки функції Animation Controller (рис 3.29) .

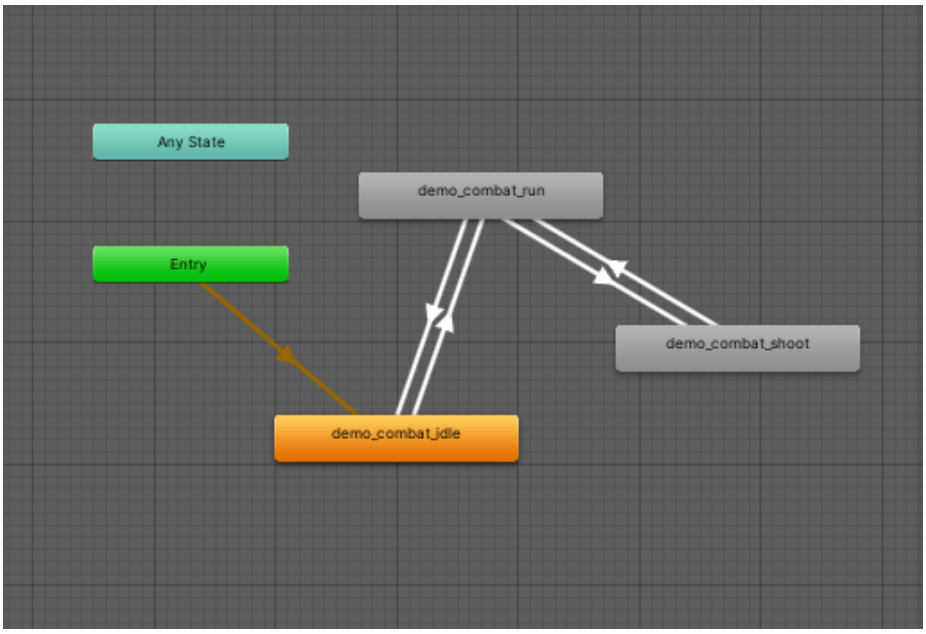


Рис 3.29 - Animation Controller

Контролер було переміщено на модель ворога, тепер анімації будуть пов’язані із ним. Також слід додати моделі ворога ряд властивостей, таких як коллайдер та фізики (рис. 3.30).

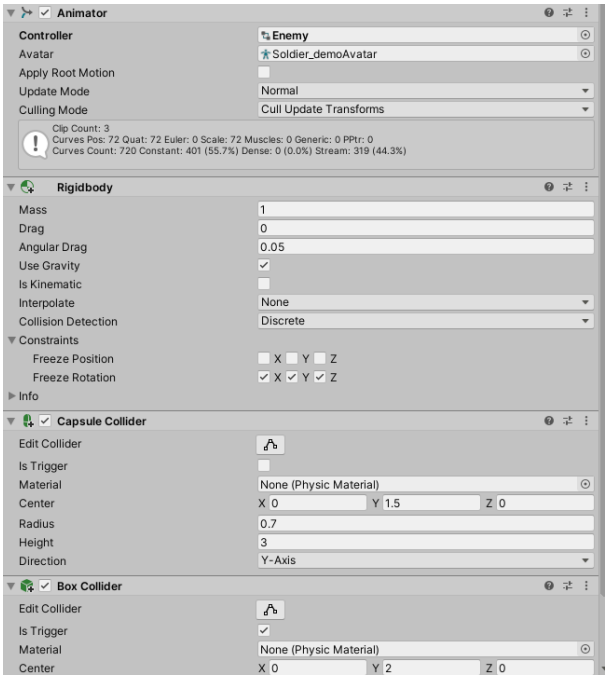


Рисунок 3.30 – Властивості моделі противника

У якості поля зору виступає компонент Box Collider, саме він буду тригером, заходячи у який противник відкриє вогонь.

Тепер потрібно перейти до написання штучного інтелекту, який буде керувати противниками.

Перш за все супротивникам було надано поле зору, у радіусі якого вони ідентифікували гравця як ворога. Якщо зайти до ворога із спини, то він не побачить, але якщо наблизитись у радіус поля зору, то він зреагує та відкриє вогонь.

Щоб супротивник пересувався до ворожого поля – додамо йому компонент NavMeshAgent (рис. 3.31), та напишемо скрипт для переслідування оппонентів.

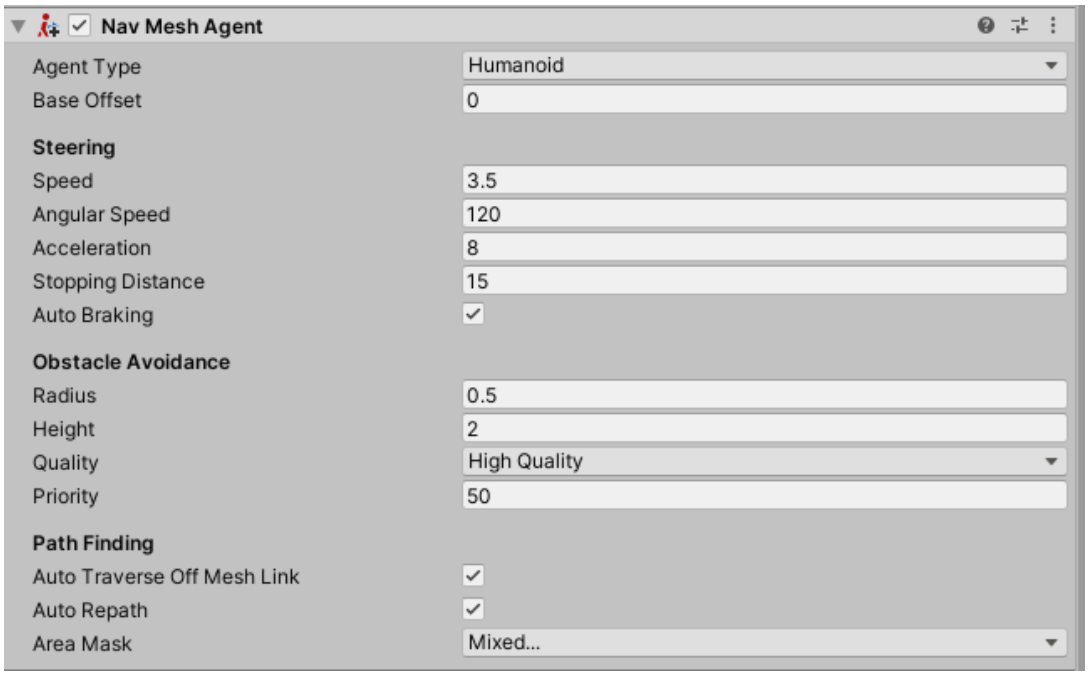
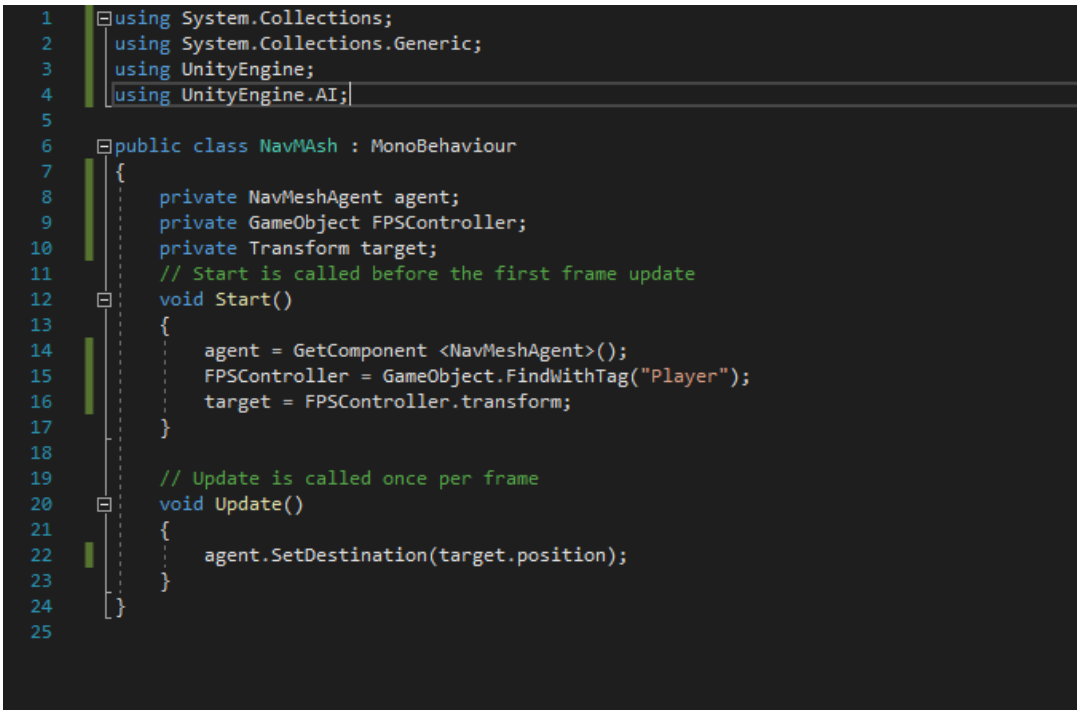


Рисунок 3.31 – компонент NavMeshAgent

Тепер перейдемо до написання скрипта. Потрібно, щоб ворог ідентифікував оппонентів та шукав їх на мапі. Тому додамо компонент NavMeshAgent також до іншої команди, там об’єднаємо іх одним тегом Player (рис. 3.32).



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class NavMAsh : MonoBehaviour
{
    private NavMeshAgent agent;
    private GameObject FPSController;
    private Transform target;
    // Start is called before the first frame update
    void Start()
    {
        agent = GetComponent <NavMeshAgent>();
        FPSController = GameObject.FindWithTag("Player");
        target = FPSController.transform;
    }

    // Update is called once per frame
    void Update()
    {
        agent.SetDestination(target.position);
    }
}

```

Рисунок 3.32 – Написання скрипту переслідування

Тепер вороги будуть шукати опонентів, щоб поцілити у них з рушниці. У команді з 5 гравців 2 будуть охороняти свою базу, патрулюючи її по заданому маршруту, а інші відправляться на ворожу територію.

Аналогічно вищевказаних етапів було створено 4 гравця у команді головного героя та 4 додаткових опонента, щоб гра відбувалась 5 на 5.

3.7 Висновки за розділом

У основній частині дипломного проекту було проведено роботи з створення моделей персонажів та об'єктів ландшафту. Сформовано мапу. Проведено роботу з головною камерою та освітленням. Для формування поведінки ігрових персонажів у симуляторі були створені скрипти, в тому числі підключено штучний інтелект для ботів, якими буде керувати система. Сформовано ігрову логіку та стиль гри був наближений до відповідного

жанру. Проведена робота із звуком, щоб він відтворювався коректно під час заданих дій. Опановано технологію Raycast, що дозволила реалізувати основну задачу симулятора – стрільбу.

РОЗДІЛ 4
ОБГРУНТУВАННЯ ЖАНРОВОГО І ПРОГРАМНОГО ВИБОРУ

4.1 Аналіз використаних методів та їх ефективності

Unity може працювати під управлінням більш ніж двадцяти операційних систем він комфортний у використанні, що дозволяє геймммейкерам створювати додатки для широкого кола ОС і ігрових платформ і тим самим розширюючи кола гравців.

У цьому середовищі використаний компонентно-орієнтований підхід, з його допомогою розробник може створити об'єкт, наприклад, основного героя. Крім цього, він може додавати різні елементи. Це може бути зображення головного героя і методи контролю над ним.

Одна з незаперечних переваг Unity 3D полягає в тому, що він оснащений великою бібліотекою ассетів і плагінів, що дозволяють істотно прискорити хід розробки ігрової програми. Програміст може їх імпортувати і експортувати, вбудовувати готові заготовки - рівні, героїв, ворогів.

Unity 3D підтримується великою кількістю платформ, API та ін. Створені в цьому середовищі гри можна легко перенести з OS Windows на Linux і ігрові консолі типу PlayStation, Xbox, Nintendo. Unity 3D може працювати з DirectX і OpenGL. Unity 3D дозволяє формувати складні анімації. Зіткнення між ігровими об'єктами.

Unity 3D - це оптимальне рішення для розробників-початківців. З одного боку їх не влаштовує функціонал простіших пакетів, наприклад, RPG Maker. З іншого боку, вони не хочуть нести витрати на придбання більш просунутих і відповідно дорогих рушіїв.

C # має виразний синтаксис, і його легко вивчати. Фігурні дужки дізнаються всі, хто працював в C, C ++ або Java. Синтаксис мови усуває складності C ++ і надає такі потужні можливості, як анулювання значень типів, перерахування, делегати, лямбда-вирази і прямий доступ до пам'яті. C # підтримує універсальні методи і типи, які підвищують безпеку типів і

продуктивність. Ітератори дозволяють класам творців колекцій визначати незвичайні поведінки ітерацій, які легко використовувати в клієнтському коді. Вирази інтегрованої мови запитів (LINQ) роблять строго типізований запит першокласною конструкцією мови.

Популярність мови - ще одна значуща перевага. Велика кількість шанувальників C # сприяють його розвитку. Також це сприятливо впливає на ріст числа вакансій, пов'язаних з розробкою на мові Microsoft. Програмісти, добре знайомі з C #, затребувані в індустрії, незважаючи на їх велику і постійне збільшення кількості.

Мова C # практично універсальна. Можна використовувати її для створення будь-якого ПЗ: просунутих бізнес-додатків, відеоігор, функціональних веб-додатків, додатків для Windows, macOS, мобільних програм для iOS і Android.

C # без перебільшення вкрай популярна серед творців відеоігор. Мова використовується для розробки ігор під Windows, macOS, Android і iOS. Вся справа в Unity - платформі для роботи з 3D-графікою. C # краще за інші мови адаптован під роботу з цим рушієм. Тому програмісти зазвичай не вибирають, а відразу використовують зв'язку Unity + C #.

Blender починався як комерційний проект, але пізніше був закритий і відроджений вже з відкритими початковими кодами. У порівнянні з комерційними розробками розмір цього редактора зовсім мізерний - лише кілька десятків мегабайт. Одною з найголовніших переваг програми є кросплатформність. Blender однаково добре і стабільно працює в Linux і Windows. Крім того, програма може функціонувати навіть на ПК з дуже слабкими конфігураціями, навіть до нетбуків. Мінімальні вимоги до системи більш ніж скромні: процесор з одним ядром, що працює на частоті 1 ГГц, оперативна пам'ять 512 Мбайт і відеокарта з підтримкою Open GL і обсягом пам'яті не нижче 64 Мбайт. Програма включає в себе великий арсенал засобів для створення тривимірної графіки.

Так, в Blender можна оперувати системами частинок, контролювати ваги окремих частинок при текстуруванні, застосовувати напрямні при анімації та використовувати зовнішні сили, наприклад вітер. Крім того, в програмі є симулятор флюїдів, який відкриває перед користувачем величезні можливості по створенню ефектів текучих тіл, таких як дим або рідини. У режимі реального часу користувач може прораховувати фізичні завдання, наприклад моделювати поведінку м'яких тел. Програма дає можливість редагувати NURBS-поверхні, використовувати метаболи і налаштовувати оснащення персонажів.

Microsoft Visual Studio - лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення і ряд інших інструментальних засобів. Дані продукти дозволяють розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, підтримуваних Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone, Android, IOS, .NET Compact Framework і Silverlight. Підтримує наступні мови: Visual Basic, C ++, C #, F #.

Visual Stiduo має ряд функцій, які роблять її більш зручною:

— IntelliSense - технологія автодоповнення Microsoft.

— Code Anilizer - функціонал, який допомагає знайти помилки в коді.

— Perfomance Analizer -інструмент, що відображає витрати ресурсів при роботі додатка / сервісу у вигляді статистики і графіків.

— Test Manager - вбудований менеджер тестів.

— Extension / Updates Manager - менеджер плагінів, адаптерів, провайдерів.

— Nuget - система управління пакетами для платформ розробки Microsoft, в першу чергу бібліотек .NET Framework.

— Git Manager - вбудований менеджер контролю версій.

— Archivator - архіватор проектів.

— File Manager - для додавання нового файлу в проект існує вбудований менеджер файлів.

— Views - велика кількість різних вкладок для відображення різної корисної інформації, на зразок списку "GOTO", або відображення даних об'єкта в Debug режимі.

— Customization - можливість змінити зовнішній вигляд Visual Studio під себе.

— Setting - налаштування всього вище перерахованого функціоналу.

4.2 Подальше просування проекту та комерційна частина

Після створення певного продукту треба подумати над його просуванням та пошуком покупців. Перш за все слід пам’ятати, що створений симулятор – це перш за все цифровий продукт, який буде розповсюджуватись скоріш за все через всесвітню павутину, тобто інтернет. Раніше для цього робили копії ігор на фізичних носіях, але на сьогоднішній день ця ера пройшла. Звісно, й на сьогодні випускаються коллекційні видання на компакт-дисках, але зазвичай продаж копій ігор відбувається на цифрових площадках, таких як Steam, Epic Games Store, Origin, тощо.

Steam – це один із найкрупніших на сьогодняшний день онлайн-магазинів, спочатку на ньому розповсюджувались в основному програмні продукти компаніх Valve, але у теперішній час ця площадка працює разом из сотнями компаніями з виробництва цифрових товарів, починаючи з програмного забезпечення до ігор з доповненою реальністю.

Як це працює: кожен зареєстрований користувач отримує акаунт із своєю «бібліотекою», у якій міститься усе придбане програмне забезпечення.

Але найважливішою функцією для подальшого просування розробленого продукту є функція Steam Greenlight (рис. 4.1).

Steam Greenlight – окрема ніша торгової площадки Steam. Завдяки цій функції розробники-початківці можуть заробити гроші на своєму продукті, але всьому є свої проблеми.

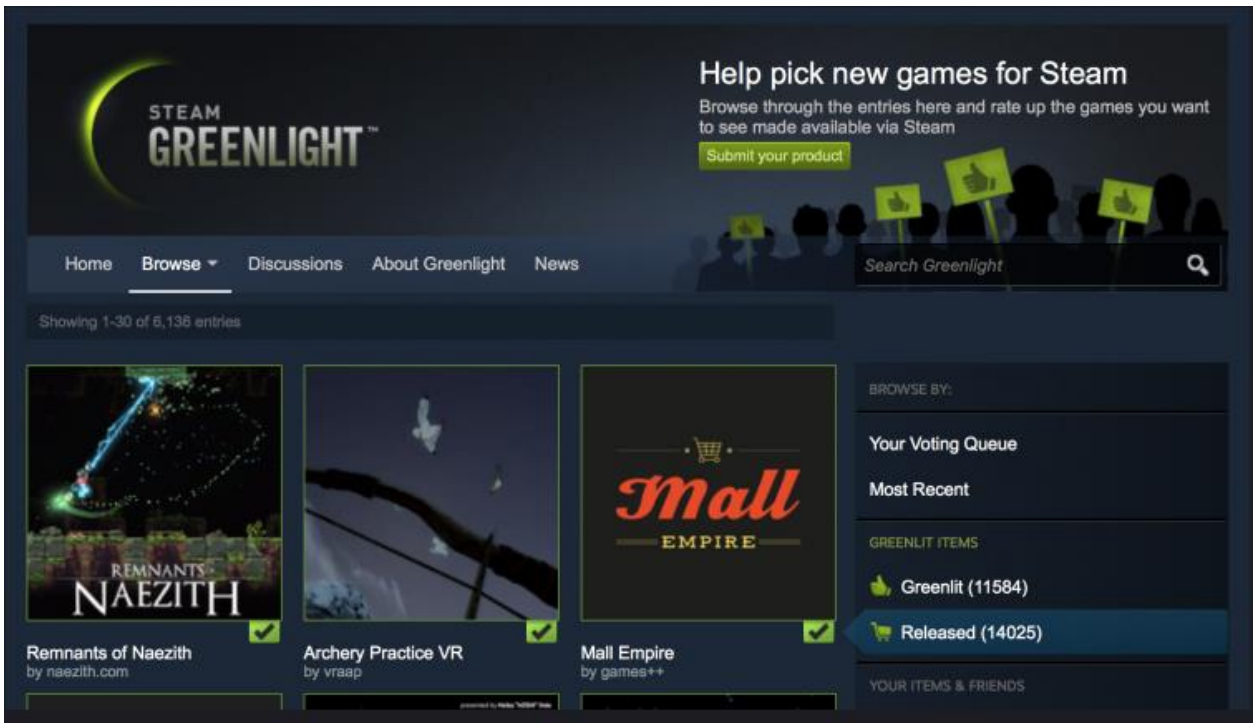


Рисунок 4.1 – Steam Greenlight

Перш за все, якщо розробник бажає почати продавати свій програмний продукт, він вносить певну сумму за роміщення його гри або ПЗ на площадці Steam Greenlight. Певний час вона буде знаходитись там. Власник продукту може загрузити опис, характеристики, прикріпити скріншоти або трейлери ігрового процесу, щоб зацікавити майбутніх користувачів. За той час, що продукт знаходиться на площадці, він збирає відгуки гравців, які оцінюють, чи зможе їм спободатись той чи інший проект. Якщо розміщений продукт по результатам місяця увійде до десятки проектів, що найбільш зацікавили користувачів, то він гарантовано буде розміщений на торговій площадці Steam. Там вже власник ПЗ може встановити ціну за свою роботу та отримувати виплати від продаж, отдаючи відсоток площадці за розміщення. Як правило, цей відсоток не перевищує 10% від встановленої ціни.

Вся ця процедура була можлива до весни 2017 року. Тепер же на заміну «зеленому світлу» прийшла функція Steam Direct (рис. 4.2).

Процедура продажу програмного забезпечення ускладнилась. Тепер власники продуктів мають платити площадці за кожне розміщення своїх позицій, а не один раз, як це було із Steam Greenlight.

Рисунок 4.2 – Steam Direct

Більш того, тепер перед розміщенням розробнику доведеться оформити певні електронні документи, пройти процедуру ідентифікації, а також надати налогову інформацію.

4.3 Оновлення, маркетинг та отримання прибутку

Для успіху будь-якого проекту треба завжди прислуховуватись до бажань користувачів та приймати своєчасні міри. Так як створений програмний продукт я в першу чергу комп’ютерною грою, то вона вимагає постійного оновлення, наповнення контентом, який буде цікавим гравцям, якісного обслуговування та стабільної роботи.

У еру інтернету локальні ігри стали користатись меншим попитом, тому у майбутньому у симулятор потрібно додати можливість мультиплеєру, щоб гравці могли змагатись між собою у командних поєдинках. Для цього слід скористатись хостингом ігрового серверу. Слід відмітити, що завдяки Unity цей пункт розвитку не буде важким, бо Unity представляє своїм користувачам послугу Multiplay. Облачна платформа Multiplay забезпечує стабільність роботи серверу та надає подальшу підтримку без серйозних проблем. Тим паче, алгоритм Quality of Service () дозволяє підібрати оптимальний регіон для підключення до гри, щоб усі гравці мали більш-менш однакову затримку відгуку від серверу. Міжнародна інфраструктура надає можливість стабільного підключення незалежно від місцезнахождення гравця.

Також слід відмітити, що для просування свіжої гри невідомого розробника потрібно надати відомості про нього самого. Наприклад, на сторінці гри надати інформацію про адресу соціальних мереж розробника, або ж на окрему групу, у якій будуть публікуватись свіжі новини про оновлення, івенти та поліпшення гри. Звісно, гравці перш ніж випробувати гру на собі оцінюють її візуальний стиль. Не зайвим буде зробити трейлер, який би демонстрував продукт з ліпшої сторони, але не перекривав її недоліки, щоб не розчарувати майбутніх користувачів.

Нести інформацію в маси – гру невідомого розробника можуть просто не побачити на фоні геймдев-гігантів. У якості рекламної кампанії можна використати блоги або портали, які займаються оглядом новинок у індустрії. Достатньо будет надати журналістам основну інформацію про новий програмний продукт та зробити невеликий огляд. Також слід не забувати, що на ринку ігрових симуляторів симулятор пейнтболу не користується великою популярністю, бо частіш за все програмні продукти цього жанру подаються як аркади, а тому якісний симулятор цієї спортивної гри може стати одним з основних повноцінних продуктів.

Під час оновлення та привнесення чогось нового у ігровий процес слід звертати увагу на професіоналів, на їх підхід, та орієнтуватись на гігантів геймдеву, які вже не перший рік займаються цією роботою.

Просування нової гри у маси не може обійтись без маркетингу та піар-кампанії. Якісний трейлер може зачепити користувачів. Також у останні десять років стало популярним використання закритого бета-тесту (ЗБТ), під час якого декільком гравцям надається доступ до демо-версії гри на певний час. Після цього вони публікують свої оцінки та висловлюють думки щодо створеного продукту, що дозволяє розробникам виявити слабкі сторони свого продукту та поліпшити його до виходу повноцінної версії. Таким підходом користувались розробники FPS-головоломки Superhot (рис. 4.3) та стратегії This War of Mine (рис. 4.4).

Рисунок 4.3 – FPS-головоломка SuperHot

Рисунок 4.4 – стратегія This War of Mine

Ще однією корисною функцією у Steam є можливість запустити банер «Скоро вийде», який буде з’являтись на головному вікні торговою площадки, сповіщаючи користувачів про те, що незабаром вийде той чи інший програмний продукт, тим самим спонукаючи гравців ознайомитись із новою грою.

Слід врахувати, що випуск нової гри, яка не може скласти конкурецнію більш відомим ігровим франшизам, слід планувати на той період, коли не

планується вихід біль значущих проектів, які можуть перекрити симулятора пейнтболу.

Однією з переваг обраного формату симулятора є те, що у своєй сутності пейнтбол – це спортивна гра, тобто сама гра не отримає обмеження за віком гравців та буде доступна усім користувачам. У той же час потрібно враховувати, що у симулятор можуть зайти діти, то гра повинна адаптуватись під рівень навичок гравців. Саме тому була обрана функція Multiplay, яка буде збалансовувати команди у майбутньому мультиплеєрі, відсортовуючи гравців за рівнем їх навичок.

Новий керуємий сервіс має гнучку систему налаштувань логіки підбору супротивників та легко інтегрується з системою масштабування ігрового серверу, дозволяя кожному гравцю знайти ідеальний матч.

Отримання прибутку – це кінцева мета проекту. Підійти до цього питання можна з декількох боків, які на сьогодняшний день активно використовуються у ігровій індустрії.

Перший – випуск free-to-play (F2P) гри, яка буде доступна гравцям безкоштовно. Отримувати прибуток можна буде завдяки внутрінньому магазину. Там можна придбати будь-які бонуси, або ж предмети для кастомизації персонажу за реальні гроші. Отримані бонуси не будуть впливати на ігровий процес, ставлячи інший гравців у скрутне положення, а лише рмінювати зовнішній вид деяких об’єктів, як це було реалізовано у королівських битвах Fortnite (рис. 4.5) та Apex Legends (рис. 4.6).

Рисунок 4.5 – F2P гра Fortnite

Рисунок 4.6 – F2P гра Apex Legends

Інший варіант релізу симулятору пейнтболу – встановлення фіксованої ціни, яка може урізатись під час розпродажу на торговій площадці задля більшого її поширення. Зазвичай у таких випадках гравці можуть заробляти гроші просто заходячи у гру, що робить продукт більш цікавим, бо може окупити сам себе. Як правило, це відбувається завдяки продажу ігрових предметів, які с деякою долею ймовірності можуть дістатись гравцям під час ігрової сесії. Отримані гроші користувачі можут витратити як на предмети у

грі, що їх більш цікавлять, або ж на виручені кошти можуть придбати нове програмне забезпечення на самій площадці Steam. Найяскравішими представниками подібних ігрових продуктів є королівска битва Player Unknown Battlegrounds (PUBG) (рис. 4.7) та FPS-шутер CounterStrike: Global Offensive (рис. 4.8).

Рисунок 4.7 – комп’ютерна гра PUBG

Рисунок 4.8 – комп’ютерна гра CounterStrike: Global Offensive

Також слід відмітити, що навіть у комп’ютерних ігор з фіксованою вартістю є мікротранзакції, які виконуються завдяки механіки так званих «кейсів». Концепція полягає в тому, що під час гри користувачу може

дістатись той самий «кейс», який можна відкрити тільки ключем, що можна лише придбати. З кейсу гравець може дістати косметичне оформлення для моделей персонажів або ж зброї, які також не впливають на ігровий процес, але мають грошову цінність (рис. 4.9). З маленьким шансом гравцю може дістатись унікальна модель, вартість якої у декілька разів перевищує вартість самої гри.

Тому користувачі з усього світу випробовують свою удачу, відкриваючи кейси, а отже – це ще один варіант отримання прибутку.

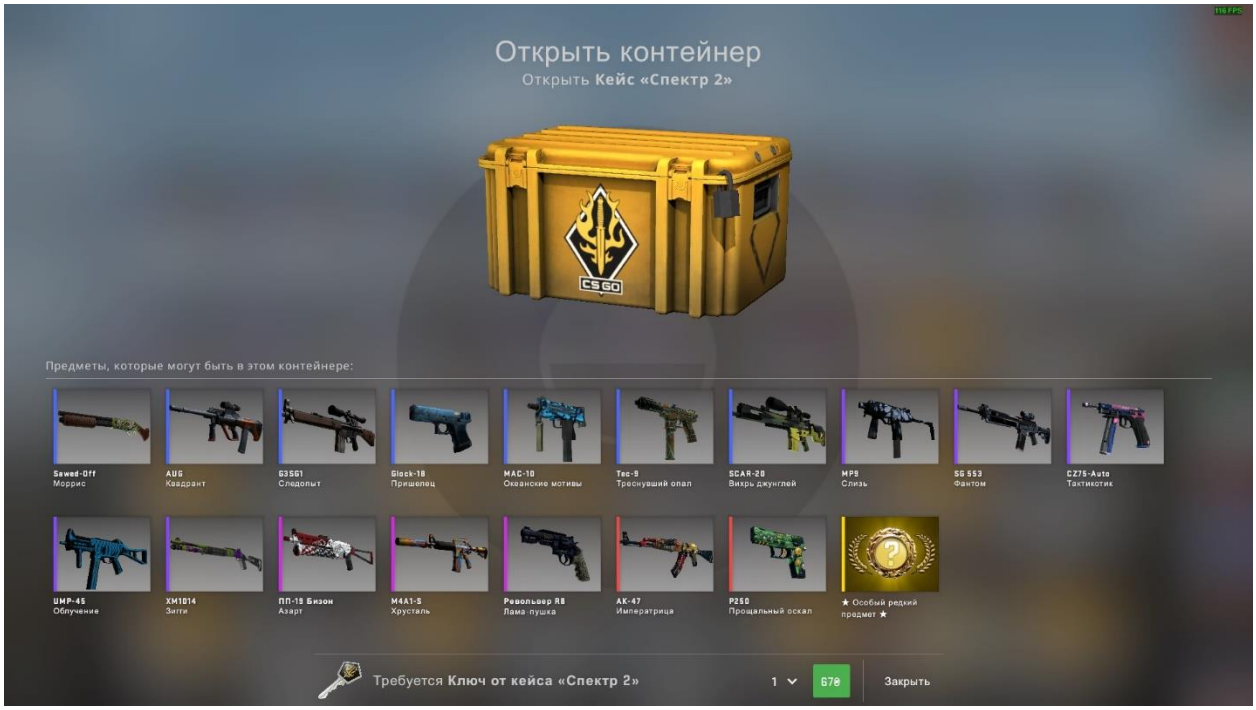


Рисунок 4.9 – механіка отримання прибутку за допомогою кейсів

4.4 Висновки за розділом

У розділі проведена робота з аналізу та обгрунтування обраної тематики та методів її реалізації. Приведені явні переваги програмних середовищ над їх аналогами, починаючи від підтримки операційних систем до швидкодії та навантаження комп’ютеру. Перелічені головні необхідні для створення проекту функції на інструменти. Проаналізовано головні етапи просування програмного продукту та методи отримання прибутку. Сформульовано план з

розміщення симулятора на цифровій торговій площадці та подальшої підтримки програмного продукту.

ВИСНОВКИ

Під час виконання магістерської роботи розглянуто та досліджено теоретичну базу за обраною темою проекту.

Проаналізовано значення теми роботи і доведена її актуальність. В межах дослідження предметної області розглянуто і проаналізовано схожі продукти та сформульована мета проекту.

Проаналізовано популярні засоби для створення ігрових додатків та пакетів для роботи з 3D моделями.

Досліджено алгоритми створення комп'ютерних симуляторів, а також ігрові механіки, які забезпечують коректну роботу симуляції та її взаємодії з гравцем. Розглянено методи створення комп'ютерних ігрових симуляторів.

Проаналізовано ринок комп'ютерних спортивних симуляторів та виявлено, що наразі на ринку майже не має симуляторів, які б максимально наближено відтворювали процес гри у пейнтбол.

Досліджено етапи просування створеного проекту на цифрові торгові площадки. Оцінена конкуренція в обраному направленні та досліджено шляхи отримання прибутку за рахунок продажу створеного продукту у мережі інтернет.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «Ігрова термінологія» X [електронний ресурс].-режим доступу:http://www.intuit.ru;
2. «Історія розвитку комп'ютерних ігор» [електронний ресурс].-режим доступу: www.igrover.ru/node/503;
3. Alan Thorn Learn Unity for 2D Game Development / Apress, 2013;
4. Alan Thorn Mastering Unity Scripting / Packt Publishing, 2015;
5. Alex Okita Learning C# Programming with Unity 3D / Routledge, 2014;
6. Andrew Troelsen and Philip Japikse Pro C# 7: With .NET and .NET Core / Apress, 2017;
7. Anne Boehm Murach's C# / Mike Murach & Associates, 2016;
8. Ashley Godbold Mastering Unity UI Development / Packt Publishing, 2018;
9. Aung Sithu Kyaw, Clifford Peters, Thet Naing Swe Unity 4.x Game AI Programming / Packt Publishing, 2013;
10. Benjamin Smith C#: Simple and Effective Tips and Tricks to Learn C# Programming Effectively / Independently published, 2020;
11. Chris Dickinson Unity 2017 Game Optimization / Packt Publishing, 2017;
12. Chris Totten Game Character Creation with Blender and Unity / Sybex, 2012;
13. Colleen Macklin and John Sharp Games, Design, and Play / Addison-Wesley Professional, 2016;
14. Dave Calabrese Unity 2D Game Development / Packt Publishing, 2014;
15. David Baron Hands-On Game Development Patterns with Unity 2019 / Packt Publishing, 2019;
16. David Horachek Creating E-Learning Games with Unity / Packt Publishing, 2014;
17. Georgios N. Yannakakis and Julian Togelius Artificial Intelligence and Games / Springer, 2019;
18. Guy Somberg Game Audio Programming: Principles and Practices / CRC Press, 2016;

19. Harrison Ferrone Learning C# by Developing Games with Unity 2020 / Packt Publishing, 2020;
20. Ian Griffiths Programming C# 8.0: Build Windows, Web, and Desktop Applications / O'Reilly Media, 2020;
21. Jamie Chan C#: Learn C# in One Day and Learn It Well, LCF Publishing / Learn Coding Fast, 2015;
22. Jason Gregory Game Engine Architecture / A K Peters/CRC Press, 2018;
23. Jeff W. Murray C# Game Programming Cookbook for Unity 3D / A K Peters/CRC Press, 2014;
24. Jeremy Gibson Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C# / Addison-Wesley Professional, 2017;
25. Joe Hocking Unity in Action: Multiplatform Game Development in C# with Unity 5 / Manning, 2018;
26. John M. Quick Learn to implement games with code / A K Peters/CRC Press, 2016;
27. Jon Skeet C# in Depth: Fourth Edition / Manning, 2019;
28. Mark J. Price C# 8.0 and .NET Core 3.0 / Packt Publishing, 2019;
29. Matt Smith and Chico Queiroz Unity 4.x Cookbook / Packt Publishing, 2013;
30. Matt Smith and Chico Queiroz Unity 5.x Cookbook / Packt Publishing, 2015;
31. Michael L Croswell Game Development: Using Unity and C# / Morgan Kaufmann, 2013;
32. Mike Geig Unity 2018 Game Development in 24 Hours, Sams Teach Yourself / Sams Publishing, 2018;
33. Nicolas Alejandro Borromeo Hands-On Unity 2020 Game Development / Packt Publishing, 2020;
34. Patrick Felicia Unity from Zero to Proficiency (Beginner) / Independently published, 2019;
35. Paul Deitel and Harvey Deitel C# 6 for Programmers (Deitel Developer) / Pearson, 2016;

36. Philip Walker Unity Certified Programmer: Exam Guide / Packt Publishing, 2020;
37. RB Whitaker The C# Player's Guide (3rd Edition) / Starbound Software, 2016;
38. Robert Nystrom Game Programming Patterns / Genever Benning, 2014;
39. Robert Wells Book on Unity 2020 By Example / Packt Publishing, 2020;
40. Ryan Henson Creighton Unity 3D Game Development by Example Beginner’s Guide / Packt Publishing, 2010;
41. Simon Jackson Unity 3D UI Essentials / Packt Publishing, 2015);
42. Stefan Boeykens Unity for Architectural Visualization / Packt Publishing, 2013;
43. Stephen Cleary Concurrency in C# Cookbook / O'Reilly Media, 2014;
44. Sue Blackman Beginning 3D Game Development with Unity / Apress, 2011;
45. Sue Blackman and Jenny Wang Unity for Absolute Beginners / Apress, 2014;
46. Sue Blackman Beginning 3D Game Development with Unity 4: All-in-one, multi-platform game development (Technology in Action) / Apress; 2nd ed. Edition, 2013;
47. Sue Blackman Unity for Absolute Beginners / Apress, 2014;
48. Terry Norton Learning C# by Developing Games with Unity 3D Beginner's GuideSep / Packt Publishing, 2013;
49. Vahe Karamian Introduction to Game Programming: Using C# and Unity 3D / Noorcon Inc., 2016;
50. Will Goldstone Unity Game Development Essentials / Packt Publishing, 2019.

Додаток А

Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження

Додаток Б

Фрагмент лістингу програми

Скрипт FPSController

```
using System;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;
using UnityStandardAssets.Utility;
using Random = UnityEngine.Random;

#pragma warning disable 618, 649
namespace UnityStandardAssets.Characters.FirstPerson
{
    [RequireComponent(typeof (CharacterController))]
    [RequireComponent(typeof (AudioSource))]
    public class FirstPersonController : MonoBehaviour
    {
        [SerializeField] private bool m_IsWalking;
        [SerializeField] private float m_WalkSpeed;
        [SerializeField] private float m_RunSpeed;
        [SerializeField] [Range(0f, 1f)] private float m_RunstepLenghten;
        [SerializeField] private float m_JumpSpeed;
        [SerializeField] private float m_StickToGroundForce;
        [SerializeField] private float m_GravityMultiplier;
        [SerializeField] private MouseLook m_MouseLook;
        [SerializeField] private bool m_UseFovKick;
        [SerializeField] private FOVKick m_FovKick = new FOVKick();
        [SerializeField] private bool m_UseHeadBob;
        [SerializeField] private CurveControlledBob m_HeadBob = new
CurveControlledBob();
```

```
        [SerializeField] private LerpControlledBob m_JumpBob = new LerpControlledBob();
        [SerializeField] private float m_StepInterval;
        [SerializeField] private AudioClip[] m_FootstepSounds;    // an array of footstep sounds that will be randomly selected from.
        [SerializeField] private AudioClip m_JumpSound;           // the sound played when character leaves the ground.
        [SerializeField] private AudioClip m_LandSound;           // the sound played when character touches back on ground.

        private Camera m_Camera;
        private bool m_Jump;
        private float m_YRotation;
        private Vector2 m_Input;
        private Vector3 m_MoveDir = Vector3.zero;
        private CharacterController m_CharacterController;
        private CollisionFlags m_CollisionFlags;
        private bool m_PreviouslyGrounded;
        private Vector3 m_OriginalCameraPosition;
        private float m_StepCycle;
        private float m_NextStep;
        private bool m_Jumping;
        private AudioSource m_AudioSource;

        // Use this for initialization
        private void Start()
        {
            m_CharacterController = GetComponent<CharacterController>();
            m_Camera = Camera.main;
            m_OriginalCameraPosition = m_Camera.transform.localPosition;
```

```
        m_FovKick.Setup(m_Camera);
        m_HeadBob.Setup(m_Camera, m_StepInterval);
        m_StepCycle = 0f;
        m_NextStep = m_StepCycle/2f;
        m_Jumping = false;
        m_AudioSource = GetComponent<AudioSource>();
                m_MouseLook.Init(transform , m_Camera.transform);
    }


    // Update is called once per frame
    private void Update()
    {
        RotateView();
        // the jump state needs to read here to make sure it is not missed
        if (!m_Jump)
        {
            m_Jump = CrossPlatformInputManager.GetButtonDown("Jump");
        }

        if (!m_PreviouslyGrounded && m_CharacterController.isGrounded)
        {
            StartCoroutine(m_JumpBob.DoBobCycle());
            PlayLandingSound();
            m_MoveDir.y = 0f;
            m_Jumping = false;
        }
        if (!m_CharacterController.isGrounded && !m_Jumping &&
m_PreviouslyGrounded)
        {
```

```
            m_MoveDir.y = 0f;
        }

        m_PreviouslyGrounded = m_CharacterController.isGrounded;
    }


    private void PlayLandingSound()
    {
        m_AudioSource.clip = m_LandSound;
        m_AudioSource.Play();
        m_NextStep = m_StepCycle + .5f;
    }


    private void FixedUpdate()
    {
        float speed;
        GetInput(out speed);
        // always move along the camera forward as it is the direction that it being aimed at
        Vector3 desiredMove = transform.forward*m_Input.y + transform.right*m_Input.x;

        // get a normal for the surface that is being touched to move along it
        RaycastHit hitInfo;
        Physics.SphereCast(transform.position, m_CharacterController.radius, Vector3.down, out hitInfo,
                           m_CharacterController.height/2f, Physics.AllLayers, QueryTriggerInteraction.Ignore);
```

```
        desiredMove = Vector3.ProjectOnPlane(desiredMove,
hitInfo.normal).normalized;

        m_MoveDir.x = desiredMove.x*speed;
        m_MoveDir.z = desiredMove.z*speed;


        if (m_CharacterController.isGrounded)
        {
            m_MoveDir.y = -m_StickToGroundForce;

            if (m_Jump)
            {
                m_MoveDir.y = m_JumpSpeed;
                PlayJumpSound();
                m_Jump = false;
                m_Jumping = true;
            }
        }
        else
        {
            m_MoveDir +=
Physics.gravity*m_GravityMultiplier*Time.fixedDeltaTime;
        }
        m_CollisionFlags =
m_CharacterController.Move(m_MoveDir*Time.fixedDeltaTime);

        ProgressStepCycle(speed);
        UpdateCameraPosition(speed);
```

```
        m_MouseLook.UpdateCursorLock();
    }


    private void PlayJumpSound()
    {
        m_AudioSource.clip = m_JumpSound;
        m_AudioSource.Play();
    }


    private void ProgressStepCycle(float speed)
    {
        if (m_CharacterController.velocity.sqrMagnitude > 0 && (m_Input.x
!= 0 || m_Input.y != 0))
        {
            m_StepCycle += (m_CharacterController.velocity.magnitude +
(speed*(m_IsWalking ? 1f : m_RunstepLenghten)))*
                         Time.fixedDeltaTime;
        }

        if (!(m_StepCycle > m_NextStep))
        {
            return;
        }

        m_NextStep = m_StepCycle + m_StepInterval;

        PlayFootStepAudio();
    }
```

```
    private void PlayFootStepAudio()
    {
        if (!m_CharacterController.isGrounded)
        {
            return;
        }
        // pick & play a random footstep sound from the array,
        // excluding sound at index 0
        int n = Random.Range(1, m_FootstepSounds.Length);
        m_AudioSource.clip = m_FootstepSounds[n];
        m_AudioSource.PlayOneShot(m_AudioSource.clip);
        // move picked sound to index 0 so it's not picked next time
        m_FootstepSounds[n] = m_FootstepSounds[0];
        m_FootstepSounds[0] = m_AudioSource.clip;
    }


    private void UpdateCameraPosition(float speed)
    {
        Vector3 newCameraPosition;
        if (!m_UseHeadBob)
        {
            return;
        }
        if (m_CharacterController.velocity.magnitude > 0 &&
m_CharacterController.isGrounded)
        {
            m_Camera.transform.localPosition =
```

```
m_HeadBob.DoHeadBob(m_CharacterController.velocity.magnitude +
                                (speed*(m_IsWalking ? 1f : m_RunstepLenghten)));
            newCameraPosition = m_Camera.transform.localPosition;
            newCameraPosition.y = m_Camera.transform.localPosition.y -
m_JumpBob.Offset();
        }
        else
        {
            newCameraPosition = m_Camera.transform.localPosition;
            newCameraPosition.y = m_OriginalCameraPosition.y -
m_JumpBob.Offset();
        }
        m_Camera.transform.localPosition = newCameraPosition;
    }


    private void GetInput(out float speed)
    {
        // Read input
        float horizontal =
CrossPlatformInputManager.GetAxis("Horizontal");
        float vertical = CrossPlatformInputManager.GetAxis("Vertical");

        bool waswalking = m_IsWalking;

#if !MOBILE_INPUT
        // On standalone builds, walk/run speed is modified by a key press.
        // keep track of whether or not the character is walking or running
        m_IsWalking = !Input.GetKey(KeyCode.LeftShift);
```

```
#endif
        // set the desired speed to be walking or running
        speed = m_IsWalking ? m_WalkSpeed : m_RunSpeed;
        m_Input = new Vector2(horizontal, vertical);

        // normalize input if it exceeds 1 in combined length:
        if (m_Input.sqrMagnitude > 1)
        {
            m_Input.Normalize();
        }

        // handle speed change to give an fov kick
        // only if the player is going to a run, is running and the fovkick is to be used
        if (m_IsWalking != waswalking && m_UseFovKick && m_CharacterController.velocity.sqrMagnitude > 0)
        {
            StopAllCoroutines();
            StartCoroutine(!m_IsWalking ? m_FovKick.FOVKickUp() : m_FovKick.FOVKickDown());
        }
    }


    private void RotateView()
    {
        m_MouseLook.LookRotation (transform, m_Camera.transform);
    }
```

```
            private void OnControllerColliderHit(ControllerColliderHit hit)
            {
                Rigidbody body = hit.collider.attachedRigidbody;
                //dont move the rigidbody if the character is on top of it
                if (m_CollisionFlags == CollisionFlags.Below)
                {
                    return;
                }

                if (body == null || body.isKinematic)
                {
                    return;
                }
                body.AddForceAtPosition(m_CharacterController.velocity*0.1f,
        hit.point, ForceMode.Impulse);
            }
        }
    }
```

Скрипт MouseLook

```
using System;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;

namespace UnityStandardAssets.Characters.FirstPerson
{
    [Serializable]
    public class MouseLook
    {
        public float XSensitivity = 2f;
        public float YSensitivity = 2f;
        public bool clampVerticalRotation = true;
        public float MinimumX = -90F;
```

```
    public float MaximumX = 90F;
    public bool smooth;
    public float smoothTime = 5f;
    public bool lockCursor = true;


    private Quaternion m_CharacterTargetRot;
    private Quaternion m_CameraTargetRot;
    private bool m_cursorIsLocked = true;

    public void Init(Transform character, Transform camera)
    {
        m_CharacterTargetRot = character.localRotation;
        m_CameraTargetRot = camera.localRotation;
    }


    public void LookRotation(Transform character, Transform camera)
    {
        float yRot = CrossPlatformInputManager.GetAxis("Mouse X") *
XSensitivity;
        float xRot = CrossPlatformInputManager.GetAxis("Mouse Y") *
YSensitivity;

        m_CharacterTargetRot *= Quaternion.Euler (0f, yRot, 0f);
        m_CameraTargetRot *= Quaternion.Euler (-xRot, 0f, 0f);

        if(clampVerticalRotation)
            m_CameraTargetRot = ClampRotationAroundXAxis
(m_CameraTargetRot);

        if(smooth)
        {
            character.localRotation = Quaternion.Slerp (character.localRotation,
m_CharacterTargetRot,
                smoothTime * Time.deltaTime);
            camera.localRotation = Quaternion.Slerp (camera.localRotation,
m_CameraTargetRot,
                smoothTime * Time.deltaTime);
        }
        else
        {
            character.localRotation = m_CharacterTargetRot;
            camera.localRotation = m_CameraTargetRot;
        }
```

```
    UpdateCursorLock();
}

public void SetCursorLock(bool value)
{
    lockCursor = value;
    if(!lockCursor)
    {//we force unlock the cursor if the user disable the cursor locking helper
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}

public void UpdateCursorLock()
{
    //if the user set "lockCursor" we check & properly lock the cursos
    if (lockCursor)
        InternalLockUpdate();
}

private void InternalLockUpdate()
{
    if(Input.GetKeyUp(KeyCode.Escape))
    {
        m_cursorIsLocked = false;
    }
    else if(Input.GetMouseButtonUp(0))
    {
        m_cursorIsLocked = true;
    }

    if (m_cursorIsLocked)
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }
    else if (!m_cursorIsLocked)
    {
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}

Quaternion ClampRotationAroundXAxis(Quaternion q)
```

```
    {
        q.x /= q.w;
        q.y /= q.w;
        q.z /= q.w;
        q.w = 1.0f;

        float angleX = 2.0f * Mathf.Rad2Deg * Mathf.Atan (q.x);

        angleX = Mathf.Clamp (angleX, MinimumX, MaximumX);

        q.x = Mathf.Tan (0.5f * Mathf.Deg2Rad * angleX);

        return q;
    }

}
    }
```

Додаток В

Матеріали презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»
ФАКУЛЬТЕТ КОМП'ЮТЕРНО-ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА АВТОМАТИЗАЦІЇ
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ

Випускна кваліфікаційна робота магістра на тему:

«Дослідження етапів розробки комп'ютерних симуляторів із застосуванням Unity 3D

Виконав студент групи ІПЗ-м20 Тетюшев Данило Артемович

Керівник роботи Маслова Н.О., к.т.н., доцент

1. Актуальність тематики роботи

- Індустрія комп'ютерних ігор з кожним роком охоплює все більшу аудиторію, що стало здебільше помітно у час пандемії. Кількість гравців різного віку за цей час збільшилась у декілька разів

2. Загальні аспекти роботи

- Об’єкт дослідження - комп’ютерні симулятори, а також засоби та методи їх створення
- Предмет дослідження - алгоритми роботи скриптів, а також можливості двигуна для забезпечення коректної роботи симулятору
- Мета роботи -дослідження можливостей двигуну Unity 3D, засобів та методів створення комп’ютерних симуляторів

3. Постановка завдання роботи

- дослідити теоретичну базу за обраною темою проєкту, проаналізувати актуальність обраної тематики;

- виявити й проаналізувати аналогічні продукти та сформулювати мету проекту;

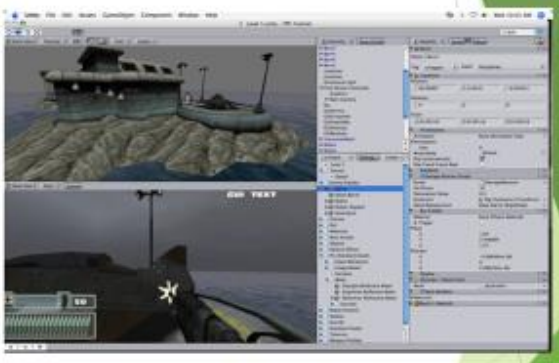
- дослідити популярні засоби створення ігрових додатків та пакетів для роботи з 3D моделями.

- **дослідити алгоритми створення комп’ютерних симуляторів, а також ігрові механіки, які забезпечують коректну роботу симуляції та її взаємодії з гравцем;**
- **розглянути методи створення комп’ютерних ігрових симуляторів;**

- проаналізувати ринок комп’ютерних спортивних симуляторів й дослідити етапи просування створеного проекту на цифрові торгові площадки

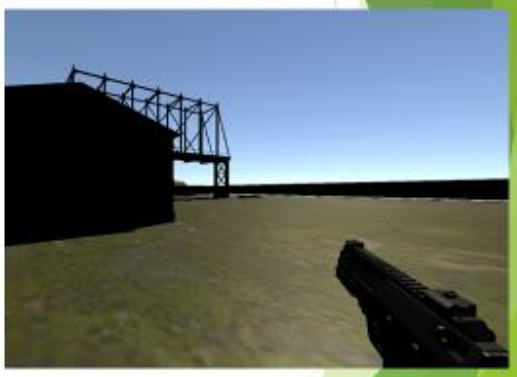
4. Середа розробки

- У якості середи розробки було обрано Unity 3D. Двигун майже нічим не поступається своїм конкурентам та дозволяє користуватись всіма необхідними функціями на безкоштовній основі



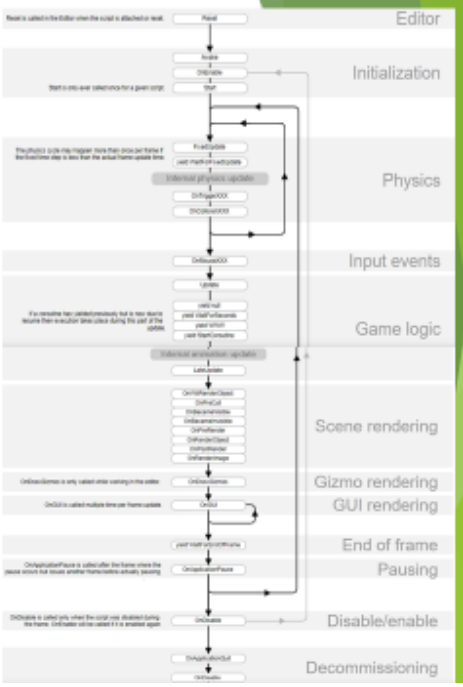
5. Загальні відомості про роботу

- Незважаючи на те, що робота є симулятором спортивної гри пейнтбол, її ще можна класифікувати як FPS-шутер (шутер від першої особи).
- Усі події будуть відбуватись від першої особи, та перед собою гравець буде бачити лише власну зброю



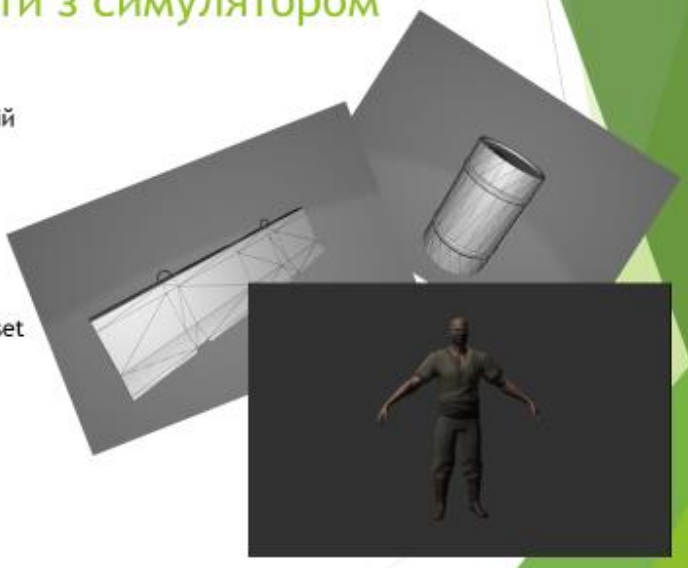
6. Використані методи

Методи FixedUpdate, OnTrigger, OnCollision використовують для взаємодії з фізикою.
Задля додання різних подій є метод OnMouse.
Методи ігрової логіки мають назви Update, yield null, yield WaitForSeconds, yield WWW, yield StartCoroutine.
Для відображення сцени проекту використовують OnWillRenderObject, OnBecameVisible, OnPreRender, OnRenderObject та інші.
За відображення інтерфейсу відповідають методи OnDrawGizmos та OnGUI.
Для активації та деактивації різного роду об'єктів використовують метод OnDisable.
Методи OnApplicationQuit та OnDestroy дозволяють видаляти непотрібні об'єкти на сцені.



7. Початок роботи з симулятором

- Перш за все у роботі було вирішено створити мапу подій та наповнити її об'єктами ландшафту.
- Моделі були створені за допомогою функціоналу Blender. Складні моделі персонажів були взяті з відкритого джерела Unity Asset Store



8. Технологія Raycasting та стрільба

- Стрільба у симуляторі реалізована за допомогою технології Raycasting.
- Ray casting, рейкастинг або Кидання променів — використання методу перетину променів з поверхнею для вирішення різних завдань в комп'ютерній графіці та обчислювальній геометрії

9. Вороги та поле зору

- Під час виконання роботи були створені вороги - боти, яким був даний штучний інтелект, який відповідає за пошук опонентів, реакцію на постріл та додає їм штучне поле зору. За переміщення по мапі відповідає компонент NavMesh

10. Практична цінність та отримання прибутку

- Steam Greenlight - окрема ніша торгової площадки Steam. Завдяки цій функції розробники-початківці можуть заробити гроші на своєму продукті, але всьому є свої проблеми

11. Можливі методи доступу до симулятора

- Free-to-play - доступ на безкоштовній основі, прибуток надходить від внутрішньоігрових транзакцій, лутбоксів.
- Платна основа - доступ до матеріалів на платній основі

12. Висновки

- Метою дипломної роботи було дослідження можливостей комп'ютерних симуляторів й методів їх створення
- Під час виконання магістерської роботи розглянуто та досліджено теоретичну базу за обраною темою проекту.
- Проаналізовано значення теми роботи і доведена її актуальність. В межах дослідження предметної області розглянуто і проаналізовано схожі продукти та сформульована мета проекту.
- Проаналізовано популярні засоби для створення ігрових додатків та пакетів для роботи з 3D моделями.
- Досліджено алгоритми створення комп'ютерних симуляторів, а також ігрові механіки, які забезпечують коректну роботу симуляції та її взаємодії з гравцем. Розглянено методи створення комп'ютерних ігрових симуляторів.
- Проаналізовано ринок комп'ютерних спортивних симуляторів та виявлено, що наразі на ринку майже не має симуляторів, які б максимально наближено відтворювали процес гри.
- Досліджено етапи просування створеного проекту на цифрові торгові площадки. Оцінена конкуренція в обраному направленні та досліджено шляхи отримання прибутку за рахунок продажу створеного продукту у мережі інтернет.
- Результатом роботи став завершений програмний продукт, симулятор спортивної гри пейнтбол, який має перспективи на поширення на цифрових площадках та подальшого розвитку

Дякую за увагу!