Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА
на тему:

Програмне забезпечення комп'ютерної 3D-гри в середовищі розробки Unity

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав студент 4 курсу, групи 1КІ-18б
спеціальності 123 — Комп'ютерна інженерія
Гулько О. С.

Керівник к.т.н., доц. каф. ОТ
Городецька О. С.

Рецензент д.т.н., проф. зав. каф. ЗІ
Лужецький В. А.

Допущено до захисту
д.т.н., проф. Азаров О.Д.

" 21 " червня 2022 р.

ВНТУ 2022

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітній рівень — бакалавр
Спеціальність — 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри
обчислювальної техніки
проф. Азаров О.Д.
«_9_» _лютого__2022 р.

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Гульку Олександру Сергійовичу

1 Тема роботи «Програмне забезпечення комп'ютерної 3D-гри в середовищі розробки Unity», керівник Городецька Оксана Степанівна к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від «24» березня 2022 року №66.

2 Строк подання студентом проекту 21 червня 2022.

3 Вихідні дані до роботи: ігровий рушій Unity 3D, середовище розробки JetBrains Rider.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, дослідження предметної області, аналіз і вибір програмних засобів, розробка та тестування 3D-гри в середовищі Unity 3D.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): структурна схема гри, схема ігрової карти.

6. Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти розділів таблиці

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1–3	к. т. н., доцент каф. ОТ Городецька О.С.		

7 Дата видачі завдання 9 лютого 2022 року.

8 Календарний план виконання БДР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів виконання комплексної бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	14.03.22	
2	Дослідження предметної області	15.03-17.03.22	
3	Визначення жанрів комп'ютерних ігор	18.03-22.03.22	
4	Аналіз і вибір програмних засобів для розробки	23.03-27.03.22	
5	Обгрунтування вибору ігрового рушія	30.03-03.04.22	
6	Обгрунтування вибору IDE	04.04-.07.04.22	
7	Обгрунтування вибору програм для 3D–моделювання	09.04-22.04.22	
8	Моделювання 3D–моделі танка	23.03-01.05.22	
	Розробка та тестування 3D-гри в середовищі Unity 3D	03.05-05.06.22	
9	Аналіз виконання роботи, висновки, додатки	07.06-12.06.22	
10	Перевірка якості виконання бакалаврського проекту та усунення недоліків	13.06-16.06.22	

Студент _____ Гулько О. С.

Керівник роботи _____ Городецька О. С.

АНОТАЦІЯ

Бакалаврська дипломна робота студента Гулька Олександра. Пояснювальна записка містить 86 сторінок, 51 рисунок, 1 таблицю та 14 посилань.

Під час виконання бакалаврської дипломної роботи було розроблено комп’ютерну 3D-гру з використанням ігрового рушія Unity 3D.

На основі дослідження предметної області та аналізу програмних засобів для розробки, обгрунтовано вибір найбільш оптимальних програмних інструментів, необхідних для розробки складових частин гри.

Згідно поставленого завдання була розроблена візуальна та програмна складова 3D-гри. Також була передбачена можливість імпорту гри на різні платформи.

Ключові слова: комп’ютерна 3D-гра, середовище розробки, ігровий рушій, архітектура, Unity.

ABSTRACT

Bachelor's thesis of student Gulko Olexander. The explanatory note contains 86 pages, 51 figures, 1 tables and 14 references.

During the bachelor's thesis, a computer 3D game was developed using the Unity 3D game engine.

Based on the study of the subject area and analysis of software for development, the choice of the most optimal software tools needed to develop the components of the game is justified.

According to the task, the visual and software component of the 3D game was developed. It was also possible to import the game to different platforms.

Keywords: Computer 3D game, development environment, game engine, architecture, Unity.

ЗМІСТ

ВСТУП

За останні два десятки років у світі виник і сформувався новий напрямок — комп'ютерні ігри.

Ігрова індустрія стрімко розвивається, її прибутки зростають й цього року поставлять новий рекорд. За результатами досліджень очікується, що у 2022 році галузь досягне $203,1 млрд прибутку завдяки витратам споживачів (і це на 5,4% більше, ніж минулого року) [1].

Бюджет сучасних ААА ігор складає від мільйона доларів і більше. Над ними працюють великі команди розробників (від 100 осіб). Час розробки може вимірюватись роками, а прибутки можуть перевищувати збори від прокату фільмів.

Бюджет інді-ігор дуже скромний в порівнянні з великими проектами, а то й взагалі відсутній. Але це зовсім не означає, що такі ігри будуть не популярні і в них ніхто не буде грати. Цікаві інді-ігри конкурують з ААА проектами. Головне щоб гра була цікава, та зуміла чимось зачепити гравців.

Великий потенціал ігрової індустрії та наявність попиту спонукає багатьох програмістів проявити себе в області розробки ігор.

Об'єктом дослідження є процес проектування та розробки комп'ютерної 3D-гри.

Предметом дослідження є програмне забезпечення комп'ютерної 3D-гри.

Метою даної бакалаврської дипломної роботи є розробка комп'ютерної гри в середовищі розробки Unity. Для досягнення поставленої мети необхідно вирішити такі задачі:

— дослідити предметну область;

— проаналізувати та вибрати програмні засоби розробки комп'ютерної гри;

— створити та оптимізувати 3D-моделі;
— налаштувати 3D-моделі в Unity 3D;
— розробити логіку управління танком в Unity 3D;
— створити звукові та візуальні ефекти в Unity 3D;
— розробити механіку стрільби;
— протестувати роботу комп’ютерної 3D-гри.

Апробація результатів роботи виконана у доповіді на LI науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії.

Матеріали роботи доповідались та опубліковувались [2]:

Гулько О. С. Вибір середовища розробки комп’ютерної 3D гри / О. С. Гулько, О. С. Городецька // Тези доповіді. LI науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії. Вінниця 2022 р. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15123/12755.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Жанри комп’ютерних ігор

Ком’ютерна гра — це комп’ютерна програма або частина комп’ютерної програми, яка використовується для організації ігрового процесу (геймплея), зв’язку з партнерами по грі, або сама гра виступає в якості партнера.

Часто комп’ютерні ігри створюються на основі книг, фільмів, хоча іноді буває навпаки — фільми чи книги основуються на комп’ютерних іграх [3].

Спеціально розроблені комп’ютерні ігри дозволяють використовувати гравців в науково-дослідницьких роботах. По деяким комп’ютерним іграм проводяться любительскі та професійні змагання. Змагання такого роду називаються кіберспортом.

Комп’ютерні ігри вплинули на суспільсво настільки, що в інформаційних технологіях появилась стійка тенденція до гейміфікації для неігрового прикладного програмного забезпечення. Прикладом цього є динозаврик в Google Chrome (рисунок 1.1), який з’являється щоразу, коли браузер не може отримати доступ до інтернету.

Рисунок 1.1 — Динозаврик в Google Chrome

Мета гри визначає її жанр. Гра може належати як до одного, так і до кількох жанрів, а в унікальних випадках — відкривати новий або бути поза всяких жанрів [4]. Розділяють вісім основних жанрів відеоігор, які у свою чергу діляться ще на декілька піджанрів:

Action — жанр комп'ютерних ігор, в яких успіх гравця у великій мірі залежить від його швидкості реакції і здатності швидко приймати тактичні рішення. Дія таких ігор розвивається дуже динамічно і вимагає напруження уваги і швидкої реакції на те, що відбувається в грі. При цьому в якості основного засобу прогресу в грі, як правило, використовують будь-яку зброю. Поділяється на шутери, аркади, файтинги та стелс-ігри. Цей жанр є найпопулярнішим та займає близько 70% ринку відеоігор.

Симулятори — це жанр відеоігор, характерною особливістю якого є якомога точніше відтворення фізичних законів реального світу, властивостей реальних предметів, процесів та подій.

Стратегії — ігри, що вимагають планування і вироблення певної стратегії для досягнення якоїсь конкретної мети, наприклад, перемоги у військовій операції. Гравець управляє не одним персонажем, а цілим підрозділом, підприємством або навіть всесвітом. Розрізняють покрокові стратегічні ігри (Turn-Based Strategy, TBS), де гравці по черзі роблять ходи , і кожному гравцеві відводиться необмежений або обмежений (залежно від типу і складності гри) час на свій хід, і стратегічні ігри в реальному часі (Real Time Strategy , RTS) , в яких всі гравці виконують свої дії одночасно, і хід часу не переривається.

Пригодницькі —жанр, в якому гравець керує ігровим персонажем, який рухається по сюжету та виконує зумовлені сценарієм завдання, покладаючись на свою уважність та логіку, здійснює пошуки підказок і вирішує загадки. Всередині жанру виділяються основні піджанри: інтерактивна література, інтерактивні фільми та візуальні романи. Часто за аналогією до пригодницьких фільмів пригодницькими називаються ті відеоігри, сюжет яких динамічно

розгортається, насичений яскравими подіями, швидкою зміною обстановки, а персонажі проявляють кмітливість та сміливість.

Музичні — в даному жанрі важлива музична складова, а від гравця потрібна наявність почуття ритму. Ігри даного жанру беруть за основу танці або виконання групою музичних композицій. Гравці повинні відповідно до того , що демонструється на екрані , натискати певні кнопки або виконувати танцювальні рухи. У разі успіху нараховуються очки. Багато музичних ігор також пропонують багатокористувацькі режими, де гравці або б'ються один з одним на рахунок , або грають в одній команді, уособлюючи собою групу. Деякі ігри жанру дозволяють використовувати стандартні геймпади або комп'ютерної миші, як , наприклад у грі Osu, Однак, більшість з них вимагають спеціальні контролери , виконані у вигляді музичних інструментів. Для багатьох танцювальних ігор випускають особливі килимки з реагуванням на натискання певних областей.

Комп'ютерна рольова гра (англ. Computer Role - Playing Game (CRPG або RPG) — це жанр комп'ютерних ігор, що заснований на елементах ігрового процесу настільних рольових ігор. У рольових іграх гравець управляє одним персонажем, або цілою групою, кожен з яких описаний набором чисельних характеристик, списком здібностей і вмінь, прикладами таких характеристик можуть бути показники здоро’я, показники сили, спритності, захисту, ухилення, рівень розвитку того чи іншого навику і т.п. Під час гри вони можуть змінюватися. Одним з характерних елементів ігрового процесу є підвищення можливостей персонажів за рахунок поліпшення їх параметрів та вивчення нових здібностей.

Головоломки — метою даного жанру є вирішення логічних завдань, що вимагають від гравця задіяння логіки, стратегії, інтуїції та іноді ерудиції й уважності. Головоломки можуть включатися до ігор інших жанрів як ключові елементи ігрового процесу або ж для його урізноманітнення як міні-ігри.

До головоломок не зараховуються ігри та їх елементи, де гравець покладється на удачу або швидкість реакції. Іноді термін використовується для загального позначення ігор з незвичайним ігровим процесом, як, наприклад, в іграх Every Extend Extra, Braid.

Interactive — в даному жанрі комп'ютерних ігор спілкування з гравцем здійснюється за допомогою текстової інформації. Розвиток цього жанру, у зв'язку з низькою вимогою до ресурсів, почався досить давно, і не припинився навіть з появою графічних ігор. Існують два види інтерфейсу — інтерфейс з введенням тексту з клавіатури або інтерфейс у вигляді меню, де гравець вибирає дію з декількох запропонованих.

1.2 Огляд популярних комп’ютерних ігор

За допомогою різних ігрових рушіїв було створено багато комп’ютерних ігор найрізноманітніших жанрів, які користуються популярністю і по цей день [5].

1.2.1 Kerbal Space Program

Kerbal Space Program — інді-відеогра в жанрі космічних симуляторів, створена незалежною мексиканською групою Squad для платформ Linux, OS X, Windows, Xbox One, PlayStation 4 [6].

Альфа-версія вийшла 24 червня 2011 і поступово вдосконалювалася, розповсюджуючись з офіційного вебсайту та сервісу Steam безкоштовно. Фінальна, платна, версія була випущена 27 квітня 2015 року. Скріншот з гри зображений на рисунку 1.2.

Порівняно із реальними умовами космосу та різних небесних тіл, гра спрощена. Так двигун гри не дозволяє відтворювати гравітацію декількох небесних тіл, замість цього в певній точці вона «перемикається» на тіло з більшим гравітаційним впливом. З цієї причини неможливо моделювати точки

Лагранжа та деякі інші ситуації — за стандартної версї гри. Але за допомогою моду Principia можна використовувати взаємодію N-тіл, враховувати точки Лагранжа і робити гравітаційні маневри.

Тяжіння небесних тіл є більшим, ніж було б в реальності при їх розмірах. Перша космічна швидкість для планети Кербін, як і друга, в 4 рази менша, ніж для Землі. Те саме стосується і решти об'єктів системи.

Проте деякі умовності наближуються до реальних за допомогою користувацьких модифікацій.

Рисунок 1.2 — Процес створення ракети

Гравець розпочинає власну космічну програму раси кербалів, жителів планети Кербін, схожої на Землю. Гравцеві слід створювати космічні апарати і запускати їх для проведення досліджень і підкорення небесних тіл зоряної системи Кербол. Кораблі та ракети складаються з окремих частин, таких як

кабіна пілотів, паливні баки, двигуни, обтічники, крила, антени і т. д. Проектування ракети і польоти включають враховування швидкості, опору повітря, витрату палива. Гра передбачає вирішення позаштатних ситуацій. Наприклад, запуск кораблів для порятунку застряглих на орбіті кербонавтів, або вихід кербонавтів у відкритий космос для ремонту корабля.

В Kerbal Space Program присутня велика кількість небесних тіл, на поверхні яких можна здійснити посадку та виконувати експерименти. Планетарна система зірки Кербол складається з великих планет, газового гіганта, карликових планет, а також супутників, що обертаються навколо них. У версії 0.23.5 включений пояс астероїдів, що може перетинати траєкторію планети Кербін, а в версії 1.10 додані комети.

В грі присутня своя зоряна система зі своїми планетами та спутниками, що віддалено нагадауєСонячну систему.

Мохо — аналог Меркурія. Кам'яниста планета без атмосфери та супутників. Майже червонного кольору. Є аномалія, через що на полюсі знаходиться дуже глибока западина, названа "Mohole".

Ева — аналог Венери. Планета земного типу з сильним тяжінням та щільною токсичною атмосферою. Має рідину на поверхні. Має астероїд-супутник Джілі.

Джіллі — малий за розмірами супутник Еви. Дуже гористий. Сила тяжіння дуже низька.

Кербін — подібна на Землю планета з океанами і материками, де розвивається цивілізація кербалів. Має два супутники: Мун та Мінмус.

Мун — схожий на Місяць супутник.

Мінмус — невеликий супутник. Присутні ідеально рівні низовини.

Дюна — аналог Марсу. Червона пустельна планета з кратерами і полярними шапками. Атмосфера щільніша за атмосферу Марса. Має один природний супутник Айк.

Айк — досить великий супутник. Дуже гористий.

Дрес — аналог Церери. Кам'яниста карликова планета з низьким тяжінням.

Джул — аналог Юпітера. Газовий гігант, найбільша і наймасивніша планета в системі. Має 5 супутників.

Лейт — великий супутник з щільною атмосферою. Майже повністю вкритий водою. В атмосфері присутній кисень.

Вал — досить великий супутник без атмосфери, аналог крижаних супутників Сатурна і Юпітера. Присутній кріовулканізм.

Тайло — дуже великий кам'янистий супутник. Немає атмосфери, й через це посадка на Тайло дуже складна.

Боп — невеликий кам'янистий супутник. Має досить великий кратер.

Пол — невеликий кам'янистий супутник з дуже високими й крутими скелями.

Іілу — аналог Плутона. Мала кам'яниста планета без атмосфери.

1.2.2 The Forest

The Fórest (англ. Forest — ліс) — відеогра з відкритим світом у жанрі survival horror, розроблена компанією «Endnight Games» [7]. Альфа-версія гри була випущена у Steam 30 травня 2014 року (скріншот з гри зображений на рисунку 1.3).

Головний герой перебуває на борту авіалайнера разом з дитиною на ім'я Тіммі. Раптово літак починає падати і він втрачає свідомість. Після авіакатастрофи головний герой крізь сон бачить, як місцевий абориген забирає хлопчика. Прокинувшись пізніше він помічає, що літак розламався навпіл і впав

у лісі, а всі пасажири загинули або були вбиті. Першочерговим завданням є вижити та знайти Тіммі.

Рисунок 1.3 — Скріншот з гри The Fórest

Гра отримала нагороди: "Golden Joystick Awards 2015" у категоріях: “Гра року”, “Найкраща графіка”, “Найкраща оригінальна гра”.

1.2.3 Subnautica

Subnautica — це пригодницька відеогра в жанрі виживання у відкритому ігровому світі, що була розроблена та опублікована студією Unknown Worlds Entertainment (рисунок 1.4).

Дії гри відбуваються у відкритому ігровому світі з точки зору від першої особи. Гравець втілюється в єдиного вцілілого члена екіпажу космічного корабля «Аврора», який був збитий невідомим енергетичним променем над вигаданою океанською планетою 4546B [8].

Рисунок 1.4 — Геймплей гри Subnautica

Основна мета гравця — дослідження ігрового світу та подолання викликів планети, одночасно з рухом ігровою історією. Subnautica дозволяє гравцеві збирати ресурси, конструювати інструменти, бази та підводні човни, а також взаємодіяти з дикою природою планети.

Особливістю гри є те, що вона не дає чіткого напрямку як треба рухатись сюжетом і що конкретно для цього потрібно. Всі рішення приймаються гравцем абсолютно вільно, проте логічні бар'єрі не дадуть потрапити в певне місце, без відповідного розвитку. Наприклад, щоб зануритись на глибину — потрібно змайструвати необхідне обладнання, а отже відшукати для нього креслення. Гра буде обережно підводити гравця до відкриття нових міст для дослідження — через сигнали порятунку від інших рятувальних капсул, які можуть надходити в той момент, коли гравцю може здатися, що він загубився.

1.2.4 The Long Dark

The Long Dark — відеогра в жанрі виживання з видом від першої особи з елементами відкритого світу, розроблена канадською компанією Hinterland Studio для Windows, macOS та Linux (рисунок 1.5). Альфа-версія відеогри була випущена 22 вересня 2014 року в рамках раннього доступу в Steam [9].

Дії гри розгортаються в глибоких нетрях півночі Канади після аварії літака в умовах глобальної катастрофи. За словами розробників, у відеогрі присутні елементи виживання, беруться до уваги показники температури тіла, кількість калорій, ступінь насичення/голоду, втома; також симулюється температура повітря і швидкість вітру, дика природа (у тому числі й життя диких тварин) і безліч інших факторів навколишнього середовища.

Рисунок 1.5 — Скріншот з гри The Long Dark

2 АНАЛІЗ І ВИБІР ПРОГРАМНИХ ЗАСОБІВ РОЗРОБКИ КОМП'ЮТЕРНОЇ ГРИ

2.1 Обгрунтування вибору ігрового рушія

Ігровий рушій — це готова архітектура, яку розробники використовують для створення гри.

Середньостатистичний ігровий рушій надає розробникам спосіб додавати такі речі, як:

— фізика;

— керування;

— виявлення колізій

— рендеринг;

— необхідні скрипти;

— штучний інтелект;

— візуальні ефекти;

— шейдери.

Ігровий рушій є багаторазовим компонентом, який розробники використовують для побудови гри. Це дає їм більше часу зосередитись на унікальних елементах, таких як моделі персонажів, текстури, взаємодія об'єктів тощо. Якби створення ігор відбувалось з нуля, то це зайняло б більше часу і зробити їх було б набагато складніше.

Для цього потрібно використовувати інтерфейси додатків, таких як DirectX, OpenGL та XNA, а також комерційні та відкриті джерела бібліотек, що містять фізику, графічні сцени та бібліотеки графічного інтерфейсу. Створити власний рушій відеоігор — це непросте завдання, але іноді необхідне, якщо гра достатньо відрізняється від усіх інших і жодні існуючі рушії не будуть працювати.

Розглянемо найбільш популярні та практичні ігрові рушії на сьогоднішній день.

2.1.1 Unity 3D

Unity 3D — це багатоплатформний інструмент для розробки двовимірних та тривимірних застосунків та ігор, що працює на операційних системах Windows і OS X. Застосунки створені за допомогою Unity працюють на системамах Windows, OS X, Android, iOS, Linux, а також на гральних консолях Wii, PlayStation 4 та XBox 360 [10].

Присутня можливість створювати інтернет-застосунки за допомогою спеціального підключаємого модуля до браузера Unity, а також за допомогою експерментальної реалізації в межах модуля Adobe Flash Player. Застосунки, створені за допомогою Unity, підтримують OpenGL та DirectX.

Інтерфейс Unity є простим в розумінні та зручним у використанні (рисунок 2.1).

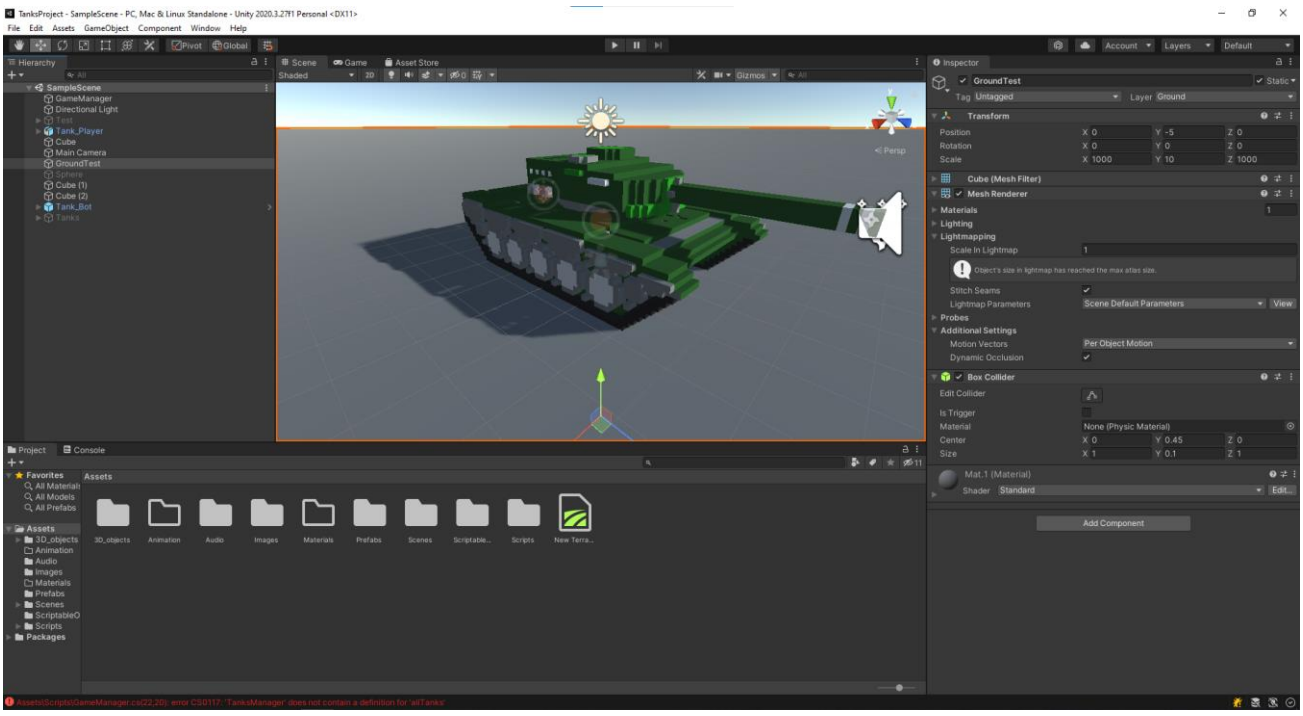

Рисунок 2.1 — Інтерфейс Unity

В залежності від типу проекту, головне вікно Unity можна кастомізувати по власному бажанні, змінюючи розмір та положення різних блоків, які, в свою чергу, можна додати або приховати (рисунок 2.2).

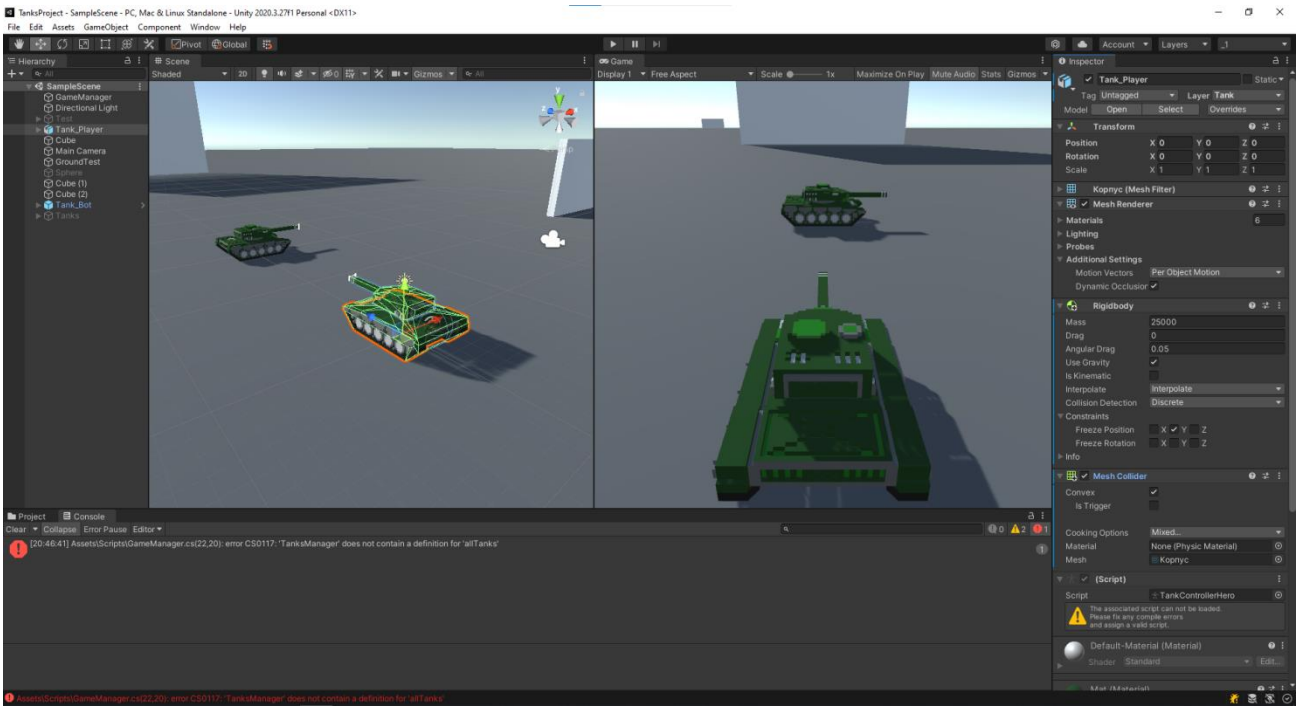

Рисунок 2.2 — Кастомізований інтерфейс Unity

Для рендеринга, ігровий рушій застосовує DirectX (Windows), OpenGL (Mac, Windows, Linux), OpenGL ES (Android, iOS), та спеціальне власницьке API для Wii. Присутня підтримка bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), динамічних тіней з використанням shadow maps, render-to-texture та повноекранних ефектів post-processing.

В Unity підтримуються файли з Cinema 4D, MagicaVoxel, 3ds Max, Maya, Softimage, Blender, Modo, ZBrush, Cheetah3D, Adobe Photoshop, Adobe Fireworks та Allegorithmic Substance. В ігровий проект Unity можна імпортувати об'єкти цих програм та робити налаштовування за допомогою графічного інтерфейсу.

Для написання шейдерів використовується ShaderLab, який підтримує шейдерні програми написані на Cg та GLSL. Шейдер може включати декілька варіантів реалізації, що дозволяє Unity визначати найкращий варіант для конкретної відеокарти. Unity також має вбудовану підтримку фізичного рушія Nvidia (колишнього Ageia) PhysX, (починаючи з Unity 3.0) підтримка в системі реального часу симуляції тканини на довільній та прив'язаній полігональній сітці, системи ray casts та шарів зіткнення.

Скриптова система ігрового рушія зроблена на Mono — вільний відкритий проект з реалізації .NET Framework. Програмісти можуть використовувати UnityScript (власна скриптова мова, подібна до JavaScript та ECMAScript), C# або Boo (мова програмування, подібна до Python). Починаючи з версії 3.0, до Unity входить перероблена версія MonoDevelop для налагодження скриптів.

В Unity присутня система контролю версій Unity Asset Server для ігрових скриптів та префабів. Дана система використовує PostgreSQL, роботу зі звуком, побудовану на основі бібліотеки FMOD (з можливістю програвати Ogg Vorbis аудіофайли), відеопрогравач із кодеком Theora, рушій для побудови ландшафтів рослинності, вбудовану систему карт освітлення (Beast), мережу для мультиплеєру (RakNet) та вбудовані навігаційні меші для пасфайндингу.

2.1.2 Unreal Engine

Unreal Engine (рисунок 2.3) — ігровий рушій, що пристосований у першу чергу для шутерів від першої особи, але також використовується і для інших жанрів [11]. Ігровий рушій написаний на мові програмування C++, він дозволяє створювати ігри для більшості платформ та операційних систем: Microsoft Windows, Linux, Mac OS і Mac OS X, консолей Xbox, Xbox 360, PlayStation 2, PlayStation Portable, PlayStation 3, PlayStation 4, Wii, Dreamcast і Nintendo GameCube.

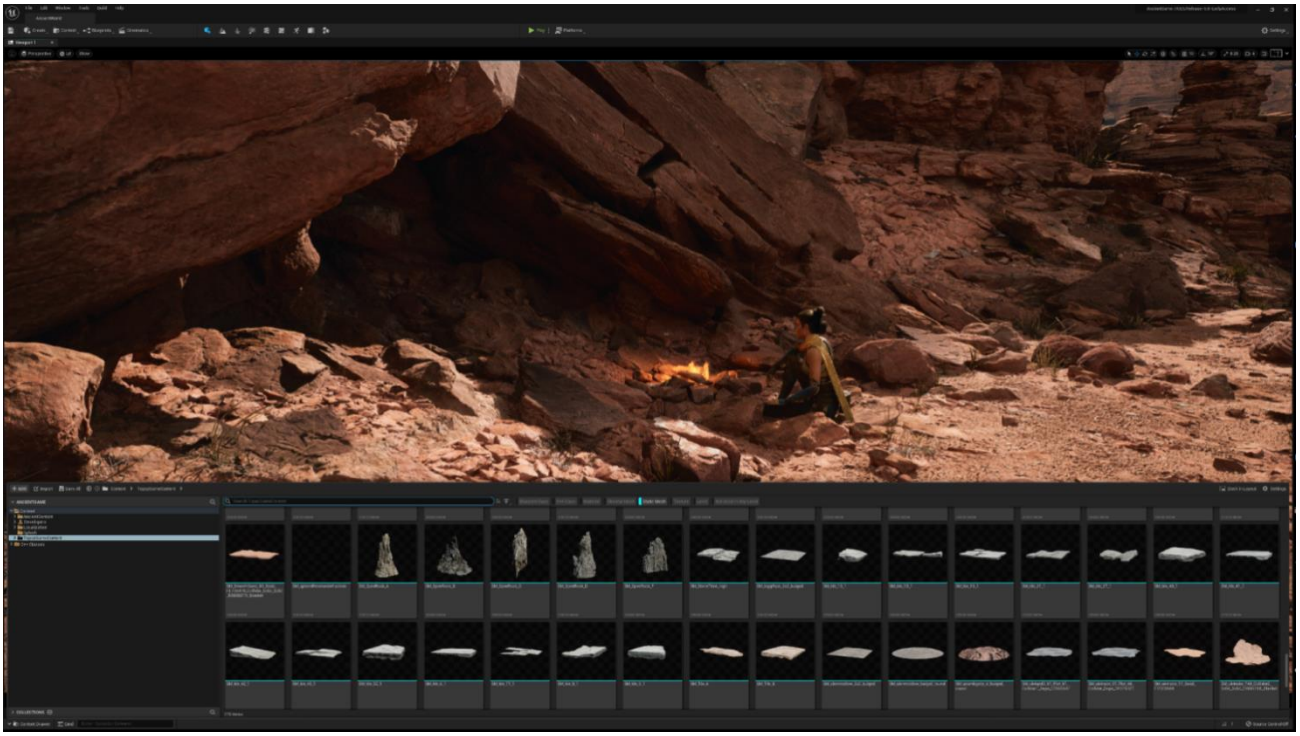

Рисунок 2.3 — Інтерфейс Unreal Engine 5

Для спрощенного портування, даний ігровий рушій використовує модульну систему залежних компонентів: підтримує різні системи рендерингу (Direct3D, OpenGL, Pixomatic; раніше підтримувалися Glide API, S3 Metal, PowerVR SGL), відтворення звуку (EAX, OpenAL, DirectSound3D; раніше підтримувалися A3D), засоби голосового відтворення тексту, розпізнавання мовлення (тільки для Xbox360, PlayStation 3, Nintendo Wii і Microsoft Windows, також планувалося для Linux і Mac), модулі для роботи з мережею й підтримку різних пристроїв вводу.

Технології Windows Live, Xbox Live, і GameSpy дають можливість грати по мережі та підключати до 64 гравців (клієнтів) одночасно. Попри те, що офіційно засоби розробки не містять у собі підтримки великої кількості клієнтів на одному сервері, рушій використовувався для створення MMORPG-ігор. Один з найвідоміших представників жанру, Lineage II, використовує рушій Unreal Engine.

Усі елементи ігрового рушія представлені у вигляді об'єктів, що мають набір характеристик і клас, який визначає доступні функції. У свою чергу, будь-який клас є «дочірнім» класом object. Серед основних класів і об'єктів можна виділити наступні:

— actor — базовий клас, що містить усі об'єкти, які мають відношення до ігрового процесу й мають просторові координати;

— pawn — фізична модель гравця або об'єкта, керованого штучним інтелектом;

— game level — об'єкт, що характеризує загальні властивості «простору», наприклад, силу тяжіння й туман.

Для роботи з простими та, як правило, нерухомими елементами ігрового простору (наприклад, стіни) використовується бінарна розбивка простору — увесь простір ділиться на «заповнений» та «порожній». В «порожній» частині простору розташовуються всі об'єкти, а також лише в ній може перебувати «точка спостереження» при рендерингу сцени. Можливість повного або часткового поміщення об'єктів в «заповнену» частину простору не виключається, однак може привести до неправильної обробки таких об'єктів (наприклад, розрахунки фізичної взаємодії) або неправильної рендеру у випадку поміщення туди «точки спостереження».

Поверхня (surface) є основним елементом бінарного дерева простору. Ці елементи створюються на грані перетину між «заповненою» і «порожньою» частинами простору. Група елементів бінарного дерева простору називається нодом. Цей термін, як правило, уживається в контексті node count — кількість нодів на екрані або в ігровому просторі взагалі. Кількість нодів, одночасно видимих на екрані, впливає на продуктивність при промальовуванні сцени. Якщо якийсь нод не потрапляє на екран або повністю перекривається іншими нодами, він не обраховується — це допомагає підвищити продуктивність, особливо в закритих просторах. Розбивка всього простору на групи нодів

називається зонуванням. Для цього іноді використовуються портали — невидимі поверхні, які служать для того щоб вручну розділити великий нод на два менші. Крім порталів використовуються антипортали, які обмежують області рендерингу.

Опис «заповнених» і «порожніх» частин простору виконується за допомогою набору замкнених тривимірних об'єктів, складених з непересічних поверхонь — брашів. Цей принцип побудови простору називається конструктивною суцільною геометрією. Геометрія може бути «адитивною» (увесь простір початково «порожній») і «попередньо прорахованою». Браші бувають декількох типів. Суцільні браші (solid) повноцінно беруть участь у бінарній розбивці простору. Напівсуцільні браші (semi-solid) не впливають напряму на бінарне дерево простору, однак впливають на його фізичну модель. Можуть тільки «заповнювати» простір. Слугують для створення «невидимих» перешкод, а також зниження числа полігонів і нодів. Порожні браші (non-solid) тільки створюють поверхні, не впливають на бінарне дерево простору. Використовуються переважно для створення об'ємів (volume) —частина простору, яка має властивості, відмінні від властивостей ігрового світу.

Існує чимало версій Unreal Engine, а також, їхніх модифікацій, розглянемо самі основні:

— Unreal Engine 1 — був випущений в 1998 році разом з грою Unreal;

— Unreal Engine 2 — дана версія появилась у 2002 році разом з грою Unreal Tournament 2003;

— Unreal Engine 3 — випущений у 2004 році з урахуванням персональних комп'ютерів, що використовують сучасні системи рендерингу (DirectX 9/10 і OpenGL 2/3, а з березня 2011 року було добавленно підтримку DirectX 11), і консолей наступного покоління (PlayStation 3 і Xbox 360);

— Unreal Engine 4 — з’явився у 2014 році, появилась система візуального скриптингу, яка дозволяє безперешкодно вибудовувати ігрову логіку навіть початківцям;

— Unreal Engine 5 — остання на даний момент версія ігрового рушія, була випущена на початку 2022 року, в ній появились такі технології як Lumen та Nanite.

2.1.3 CryEngine

CryEngine (рисунок 2.4) — ігровий рушіій, розроблений Crytek, та випущений у 2002 році. Вперше ігровий рушій був використаний при створенні гри FarCry 2. Запускати CryEngine можна лише на операційній системі Windows, проте ігри зроблені на даному рушію будуть запускатись на багатьох платформах: Microsoft Windows, Linux, PlayStation 3, PlayStation 4, Wii U, Xbox 360, Xbox One, iOS та Android.

Ігровий рушій характеризується прогресивними можливостями в розробці відеоігор та підтримкою сучасних передових технологій, таких як VR, Vulkan API, DirectX 12.

CryEngine дозволяє створювати ігри з сучасною реалістичною графікою. Проекти, розроблені на даному ігровому рушію, по якості графіки не поступаються передовим проектам, зроблених на Unreal Engine чи Unity (рисунок 2.5).

В CryEngine використовується власна технологія трасування променів, що працює на відеокартах Nvidia та AMD, та не потребує потужності відеокарт серії RTX.

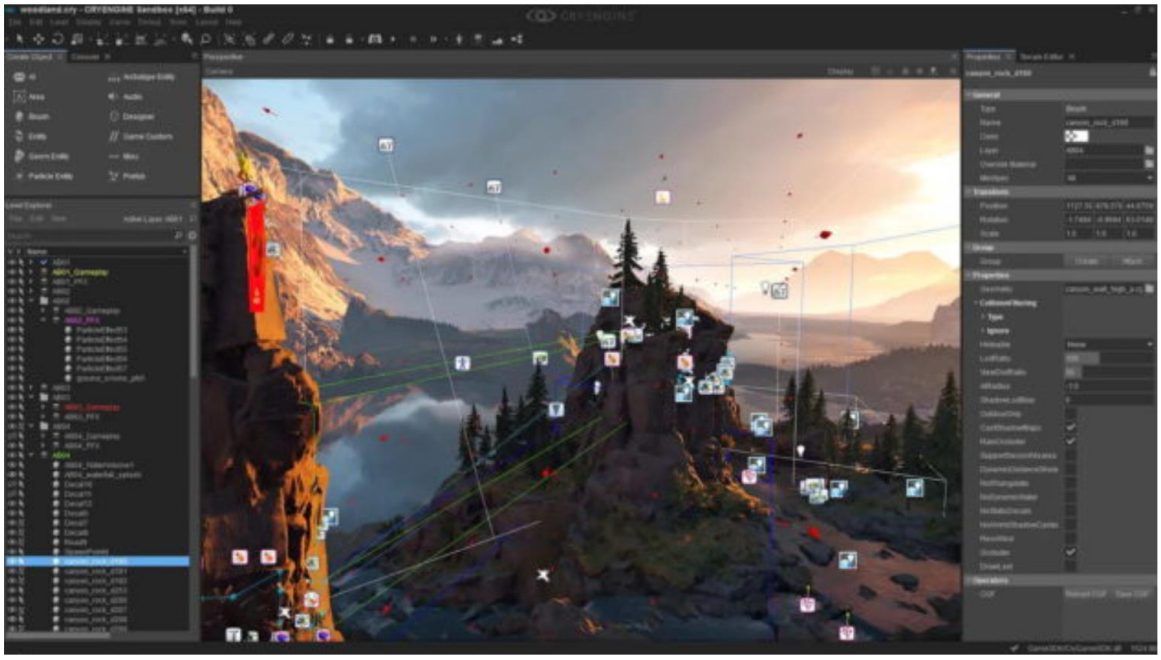

Рисунок 2.4 — Інтерфейс CryEngine 5.6

Незважаючи на свою потужність та використання передових технологій, CryEngine є досить складним ігровим рушієм для використання початківцями, необхідно мати глибокі знання в галузі розробки ігор, щоб вміло працювати в CryEngie.

Рисунок 2.5 — Сучасна графіка у грі Kingdom Come: Deliverance

2.1.4 Godot

Godot — ігровий рушій загального призначення, пристосований для роботи як з 2D так і з 3D проектами. Ігровий рушій постачається з повноцінним редактором ігор в який інтегровані інструменти для вирішення найпоширеніших потреб. Він включає в себе редактор коду, редактор анімації, редактор карт, редактор шейдерів, налагоджувач, профайлер та багато іншого (рисунок 2.6).

Написання ігрової логіки відбувається за допомогою мови програмування C++, або з використанням власної високорівневої динамічно типізованої скриптової мови програмування під назвою GDScript, що по синтаксису нагадує мову програмування Python. На відмінну від Python, GDScript містить строгу типізацію при оголошенні змінних. Також GDScript більш оптимізований під архітектуру Godot.

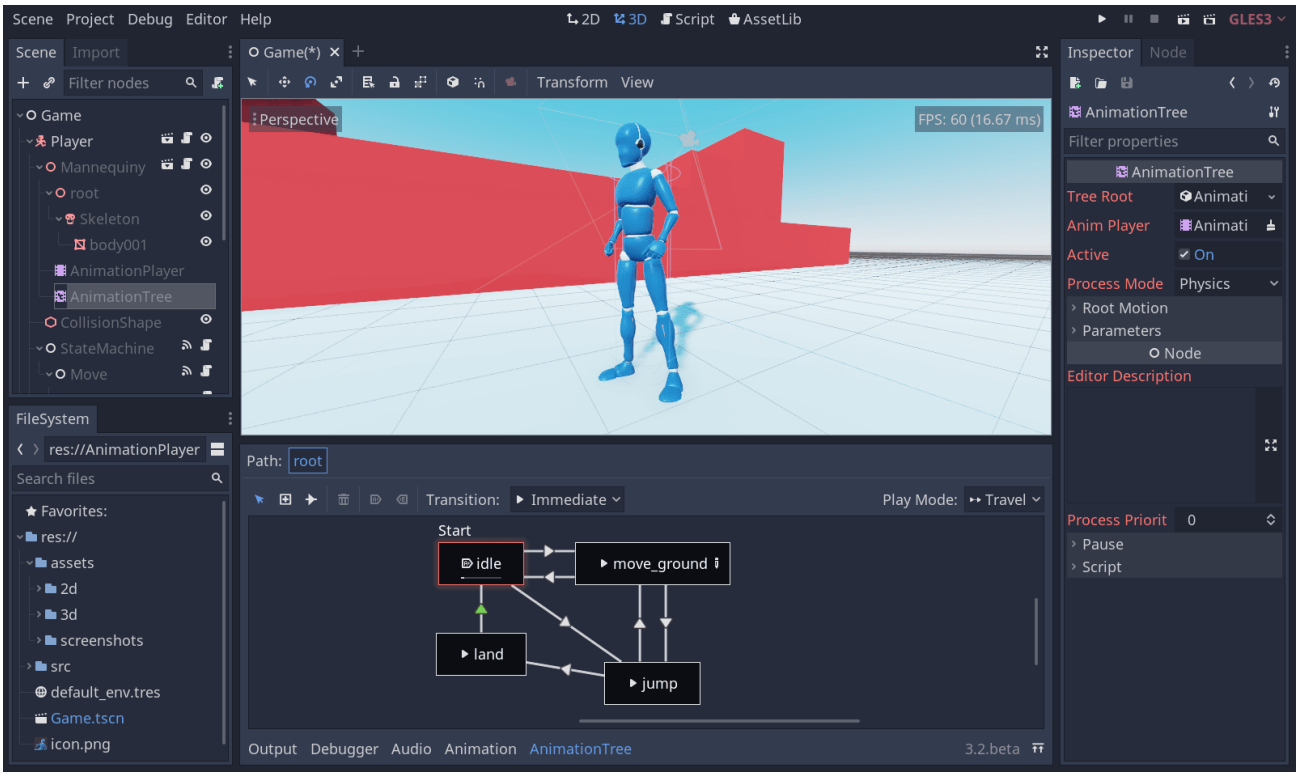

Рисунок 2.6 — Інтерфейс Godot

Godot містить свій інтегрований скриптовий редактор, що містить функції автодоповнення коду, авто-відступа, підсвітки синтаксичних помилок, швидкого доступа до повного API ігрового рушія та багатьма іншими можливостями. Крім цього в Godot присутній зручний профайлер, налагоджувач, статистику використання відеопам'яті та віддалений редактор ігрових сцен для контролю за обєктами в реальному часі.

2D та 3D фізична складова ігрового рушія побудована з нуля, це допомогло досягнути належного рівня оптимізації фізичної складової. Присутня можливість рейкастингу, виявлення зіткнень, руху твердих тіл і з'єднань між ними. Присутня власна реалізація контролера персонажа та 3D-контролер транспортних об'єктів із спрощеною системою підвіски.

Система для створення графіки для різних платформ побудована на OpenGL ES 2.0. Процес рендерингу містить технології, normal mapping order-independent transparency, specularity, повноекранні постеффекти типу bloom, DOF, FXAA, HDR, гама-корекції, distance fog, динамічні тіні, основані на shadow maps та інші.

Для написання шейдерів існує упрощена шейдерна мова, що є близькою підмножиною до мови GLSL. Шейдер використовується в матеріалах та в екранних ефектахь2D-візуалізації. Шейдери діляться на секції fragment та vertex. Також присутня можливість створювати шейдери у редакторі.

Godot містить окрему графічну підсистему для 2D, що використовується незалежно від 3D. Прикладом можливості 2D слугує графічний інтерфейс (робоче середовище Godot є основним прикладом можливості графічного інтерфейсу), тайлова графіка (квадратна, ізометрична та власний формат), parallax scrolling, спрайти, 2D світло та тіні (точкового типу), система частинок та інші можливості. Розробник може змішувати та комбінувати 2D та 3D в обох напрямках (за допомогою Viewport Node).

Ігрове середовище містить власну систему для анімування та керівні елементи для роботи з анімацією на основі скелету, шейп-анімацією та сценами-заставками. Завдяки основаному на нодах, дизайних рушіїв, через редактор анімацій можна анімувати будь-який параметр, що може бути присутній в грі. Пристуня можливість викликати будь-яких власні методи з ключами анімаційного треку, це дозволяє значно спростити процес анімування в складних сценах.

На даному ігровому рушієві можна створювати ігри під наступні платформи: Windows, Linux, OS X, BSD, Haiku, Android, iOS. Також, через компілювання рушія з SDK, можна проводити експорт на інші платформи.

Інші можливості Godot:

— багатопоточність (скрипти обраховуються в паралельних потоках та можуть самі їх створювати);

— occlusion culling і система порталів (ігнорування невидимих об’єктів);

— render targets (рендеринг зображення з камери в текстуру);

— рівні деталізації об’єктів (зменшують навантаження на відеокарту);

— light baking (змішування глобального статичного та динамічного прямого світла);

— система плагінів (плагіни пишуться на мовах програмування C++ або на GDScript);

— система вводу підтримує мишку, клавіатуру, геймпад та сенсорний екран (різні пристрої можуть бути назначені на будь-які дії, і вони будуть розглядатись незалежно від того, який був використаний метод вводу);

— конвеєр імпорту/експорту/компрессії текстур (гнучка система роботи з асетами та автоконвертування в потрібні формати);

— конвеєр імпорту цілісних 3D сцен (наприклад, з Blender можна експортувати сцену разом із камерами, мешами, освітленням, порталами та анімованими персонажами);

— граяічний редактор шейдерів (програма шейдера може бути створена у візуальному редакторі графів);

— текстурний атлас (може значно знизити навантаження на відеопам’ять на мобільних пристроях);

— підтримка відеокодека Theora (відео можна рендерити в текстуру);

— сітка навігації (для побудови шляхів на базі полігональних мешів);

— пошуковий алгоритм A* (швидкий алгоритм для побудови шляхів в простих матричних системах);

— аудіосистема підтримує кодеки WAV та Ogg Vorbis (Ogg Vorbis для потокового аудіо та WAV для аудіосемплів).

2.1.5 Source Engine

Source Engine (рисунок 2.7) — ігровий рушій, розроблений Valve Corporation. Особливостями даного ігрового рушія є гнучкість та модульна основа, технологія вираження емоцій, та продвинута фізика.

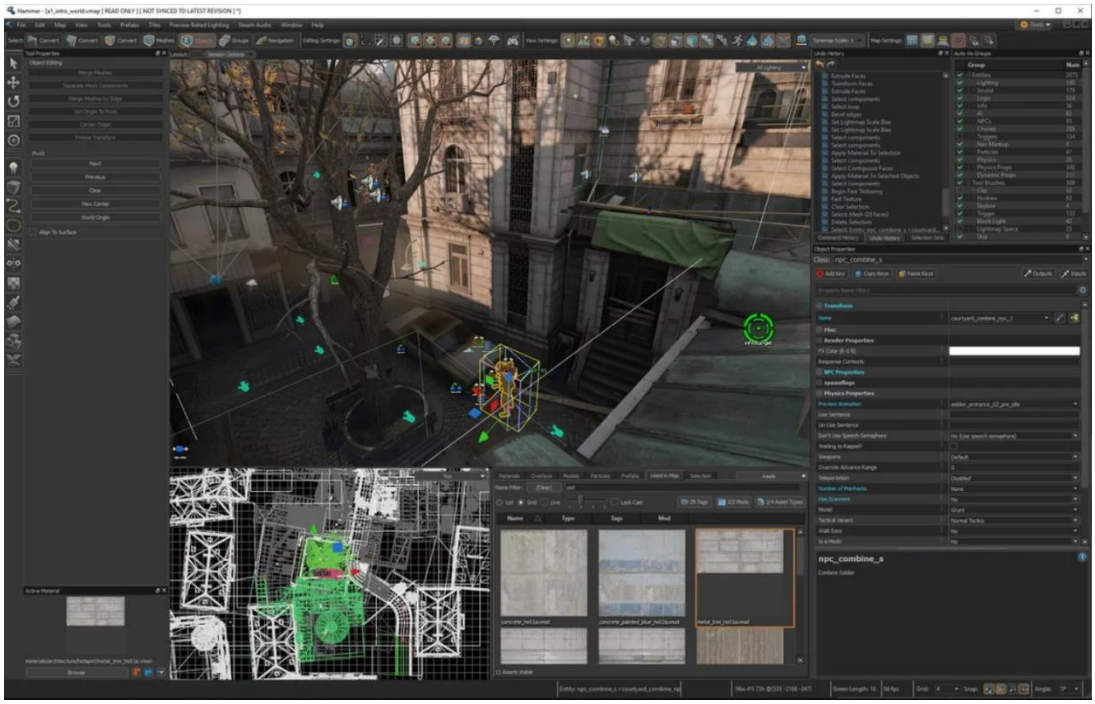

Рисунок 2.7 — Інтерфейс Source Engine

Ігри, розроблені за допомогою Source, підтримуються платформами Windows, Xbox, Xbox 360, PlayStation 3, PlayStation4, Mac OS X.

Ігрова фізика імітується завдяки переробленому фізичному рушію Havok. Він дозволяє рушію розраховувати різні фізичні об'єкти, такі як тверді тіла, пружні тіла, мотузки, поверхні тощо. В Source Engine використовується технологія для створення дуже реалістичних транспортних засобів, від машини до катера на повітряній подушці і вертольота. Для прорахунку поведінки транспортного засобу на дорозі або в повітрі використовується багато параметрів, наприклад, зчеплення коліс з дорогою, вага машини тощо. Для додання реалістичного руху тіла використовується ragdoll фізика. В Source Engine анімація може змішуватися з фізикою, що дає приріст реалістичності.

У Source використовується один із найбільш просунутих штучних інтелектів. Персонажі під управлінням штучного інтелекту можуть стрибати, бігати, літати, повзати, плавати, закопуватися, атакувати. Складна побудова карт переміщень допомагає неігровим персонажам обходити перешкоди, ховатися за ними.

Source використовує систему матеріалів для опису властивостей об'єкта. З чого зроблений об'єкт, чи він тоне, як впливає на інші об'єкти своєю поверхнею. Для надання об’єму використовуються карти нормалей, які визначають, як будуть висвітлені окремі точки на поверхні. В Source Engine підтримується змішування текстур, м'які переходи від однієї текстури до іншої.

Отже, в результаті огляду та аналізу ігрових рушіїв, оптимальним вибором для розробки комп’ютерної 3D-гри буде Unity 3D, адже він простий в освоєнні та ідеально підходить для невеликих проектів.

2.2 Обгрунтування вибору середовища розробки для написання коду

IDE — це інтегроване середовище розробки. IDE містить редактор коду, інструменти для автоматизації написання та відлагоджування програм.

Більшість сучасних середовищ розробки мають функцію автодоповнення коду, що дуже спрощує та пришвидшує процес програмування певної логіки.

2.2.1 Microsoft Visual Studio

Microsoft Visual Studio (рисунок 2.8) — середовище розробки від Microsoft, яке дозволяє створювати як консольні програми, так і програми з підтримкою графічного інтерфейса [12]. Дане середовище розробки підтримується операційними системами Windows та Mac OS. IDE підтримує такі мови програмування: C++, C#, CSS, Dart, F#, HTML, Java, JavaScript, JSON, Markdown, PHP, PowerShell, Python, SCSS, T-SQL, TypeScript та інші.

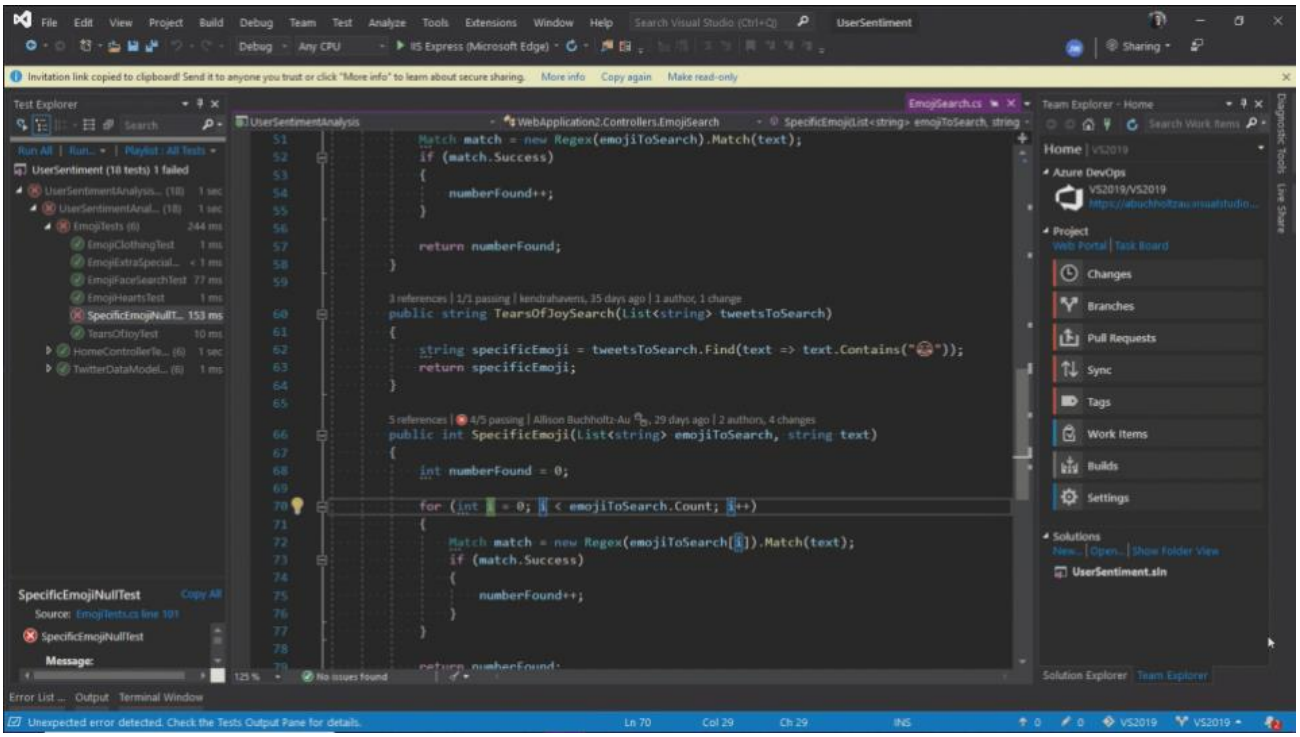

Рисунок 2.8 — Microsoft Visual Studio

До переваг Microsoft Visual Studio можна віднести вбудовану технологію розумного завершення коду IntelliSense, індивідуальне налаштування робочої панелі під власні потреби, підтримка функції розділеного екрану (split screen), велику бібліотеку доповнень, налаштовуванні параметри редактора коду.

Недоліками Microsoft Visual Studio є відсутність підтримки Linux, доволі високі системні вимоги, безкоштовна версія Community працює з обмеженнями, а вартість підписки на повноцінну версію починається від 50$.

2.2.2 Visual Studio Code

Visual Studio Code (рисунок 2.9) — простіший аналог Visual Studio. Дане середовище розробки пропонує не багато функцій, але дає можливість писати код на багатьох мовах програмування та містить інструменти відкладки. На відмінно від Visual Studio, окрім операційних систем Windows та Mac OS, Visual Studio Code підтримується операційною системою Linux.

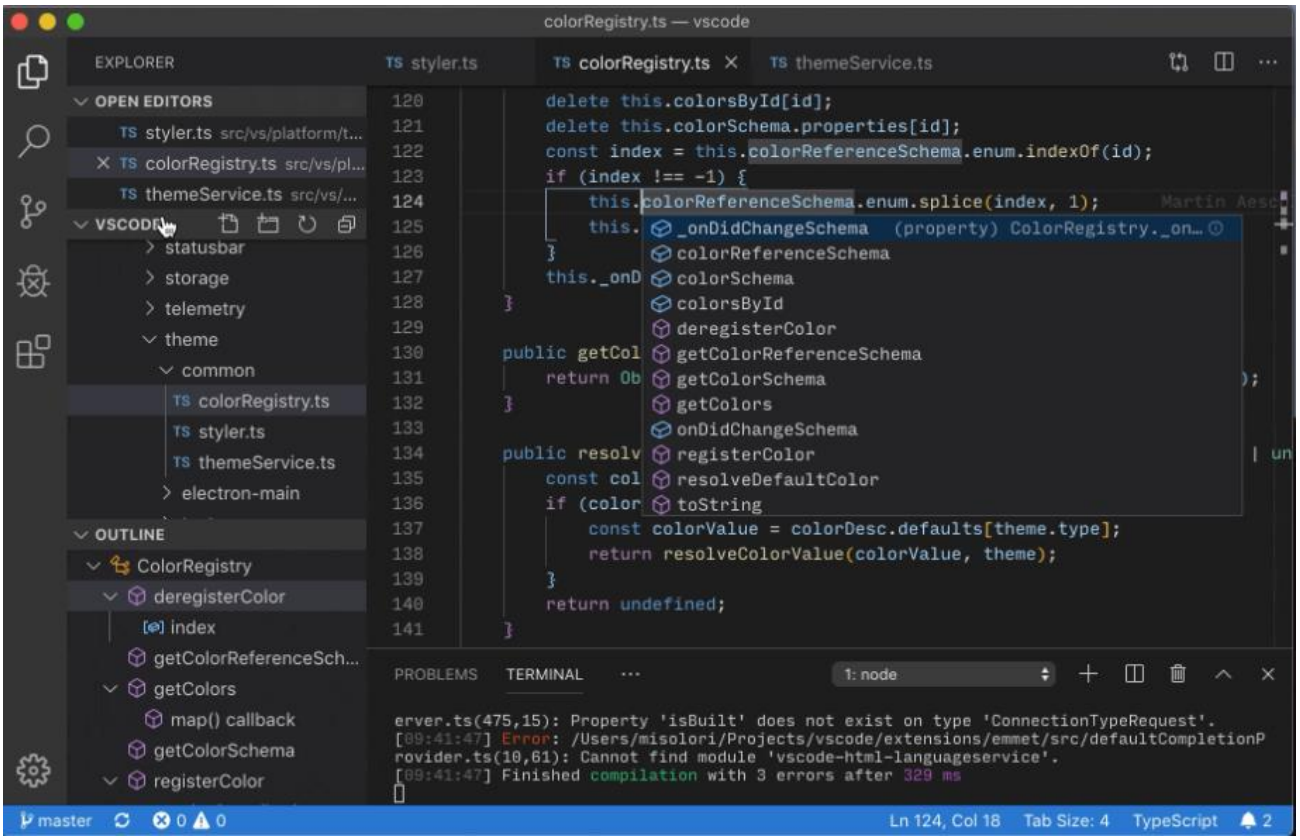

Рисунок 2.9 — Visual Studio Code

Підтримуванні мови програмування: C++, C#, CSS, Dart, F#, HTML, Java, JavaScript, JSON, Markdown, PHP, PowerShell, Python, SCSS, T-SQL, TypeScript та інші.

Перевагами Visual Studio Code є підтримка на операційній системі Linux, вбудована технологія автодоповнення коду IntelliSense, велика кількість розширень та те, що середовище розробки повністю безкоштовне. До недоліків можна віднести невелику кількість інструментів та відсутність підтримки split screen.

2.2.3 JetBrains Rider

JetBrains Rider (рисунок 2.10) — кросплатформне IDE, основане на платформах IntelliJ та ReSharper. Rider дозволяє створювати десктопні програми, .NET – сервіси і бібліотеки, ігри на рушіях Unity та Unreal Engine, мобільні додатки Xamarin, веб – додатки ASP.NET / ASP.NET Core та багато іншого.

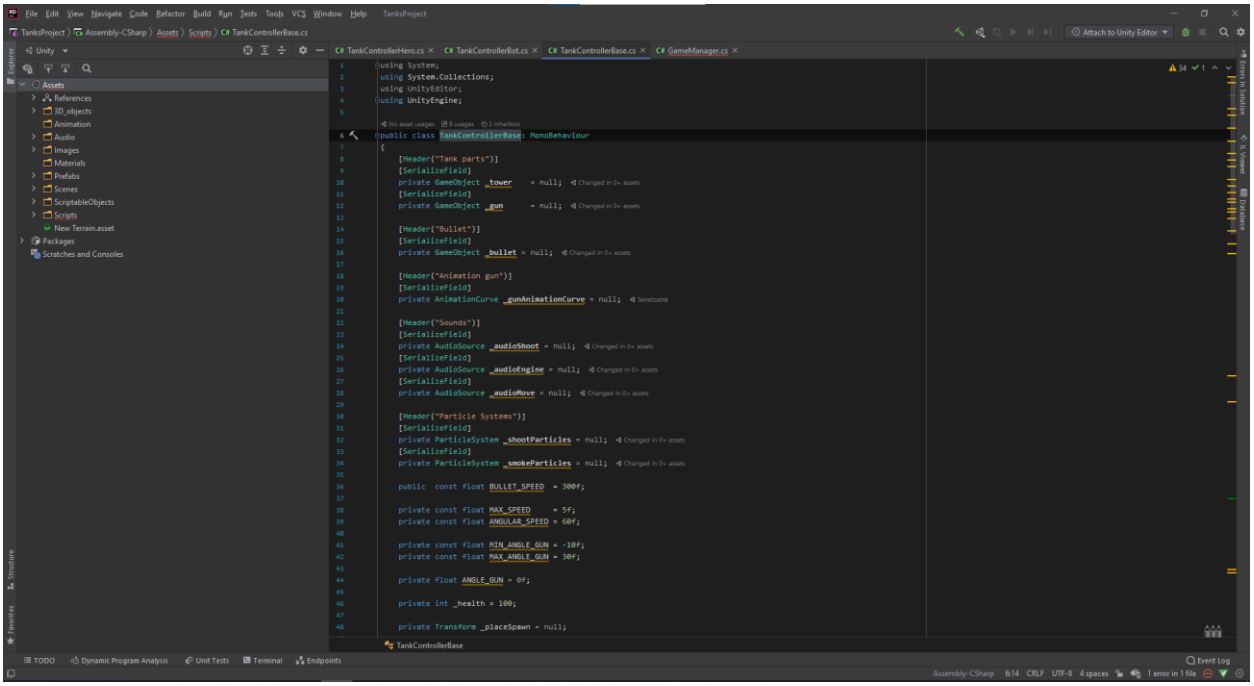

Рисунок 2.10 — JetBrains Rider

Перевагами JetBrains Rider є те, що Rider стабільніший та швидший ніж Visual Studio, має зручні та потужні інструменти рефакторингу, присутня функція автодоповнення коду (навіть при написанні шейдерів), підтримка підключення до баз даних та SQL. Недоліками JetBrains Rider є високі системні вимоги для роботи та те, що середовище розробки не безкоштовне.

2.2.4 Sublime Text

Sublime Text (рисунок 2.11) — це середовище розробки, що містить інструменти спрощення зміни кода: Goto Anything, співставлення скобок, множинне виділення та потужний Python API.

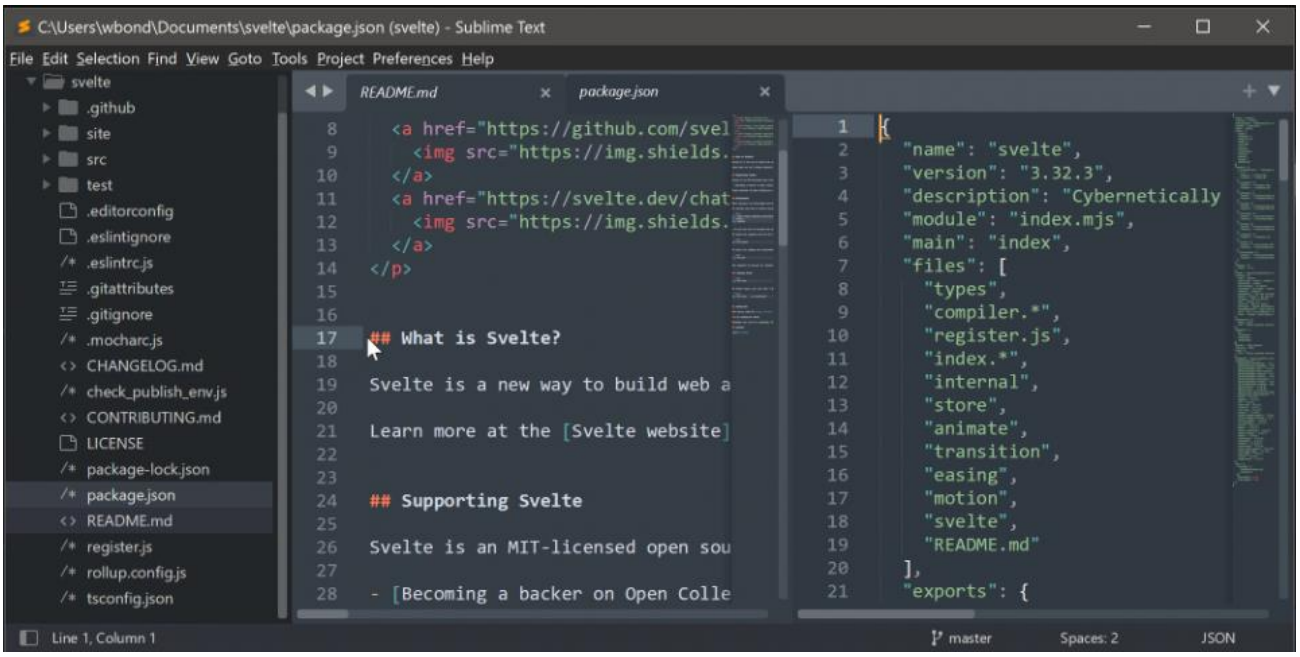

Рисунок 2.11 — Sublime Text

В даній IDE можна писати на таких мовах програмування: C++, C#, Python, CSS, JavaScript, HTML, PHP, SQL, Textile, XML, XSL та інших. Sublime Text підтримується на операційних системах Windows, Mac OS та Linux.

Перевагами Sublime Text є великий вибір розширених функцій синтаксису та редагування, навігація Goto Anything, яка дозволяє швидко отримувати

доступ до файлів, множинне виділення, яке дозволяє вносити зміни в декількох місцях одразу, можливість швидкого переходу між проектами з бистрим збереженням. До недоліків можна віднести відсутність функції автоматичної відкладки та необхідність платної ліцензії.

2.2.5 Eclipse

Eclipse — популярне середовище розробки, яке спочатку використовувалось лише для Java, але була добавлена підтримка інших мов програмування: C, C#, C++, Java, Perl, PHP, Python, Ruby. Дане IDE підтримується на операційних системах Windows, Mac OS та Linux.

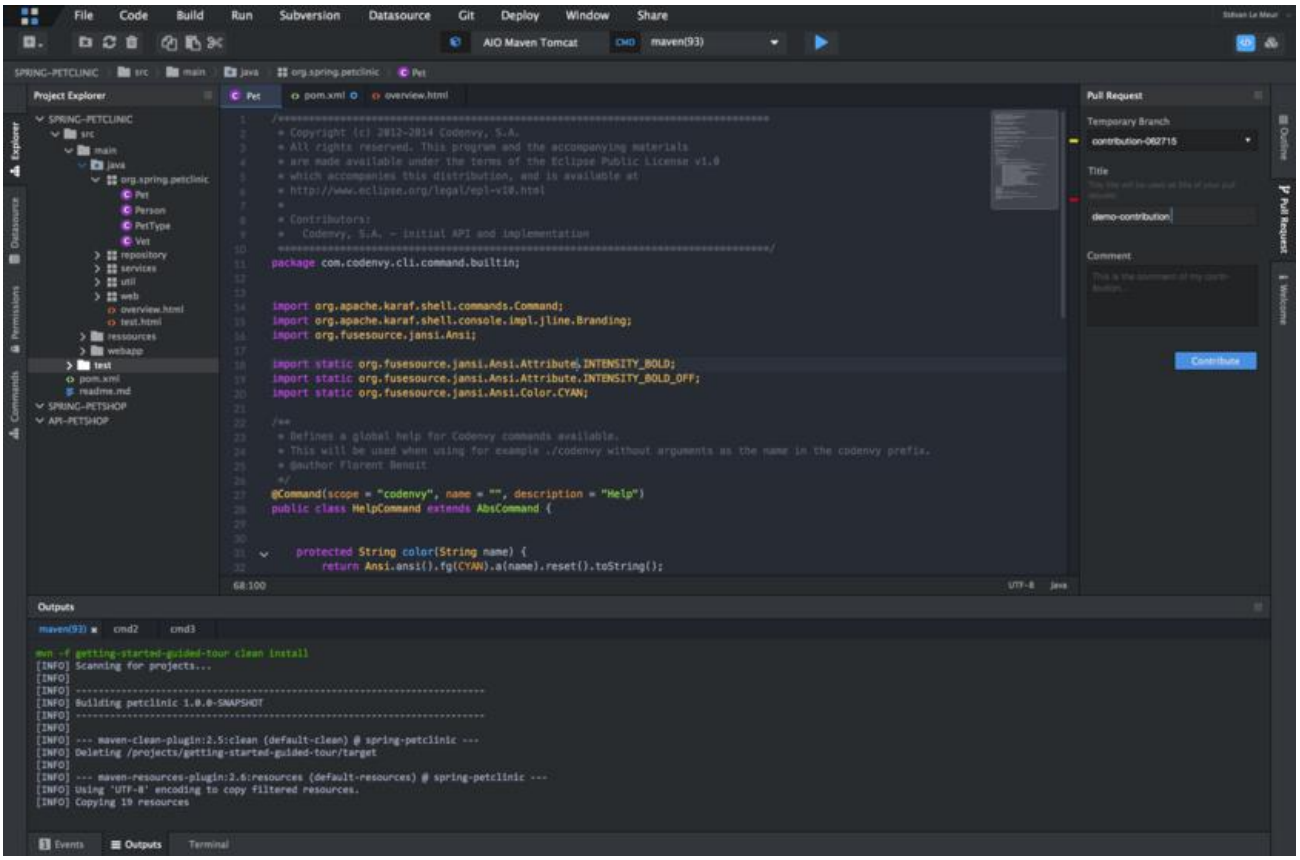

Рисунок 2.12 — Eclipse

Перевагами Eclipse є те, що середовище безкоштовне та з відкритим кодом, доступне налаштування та розширення додатковими функціями за

рахунок плагінів, графічний інтерфейс можна налаштувати, присутня можливість інтеграції JUnit тестування та проведення оптимізації тестів. Недоліками є те, що Eclipse буде важким для початківців та невеликий функціонал.

2.2.6 PyCharm

PyCharm (рисунок 2.13) — просте та зручне середовище розробки з відкритим кодом. Atom підтримує велику кількість мов програмування та чудово підходить для кросплатформної розробки.

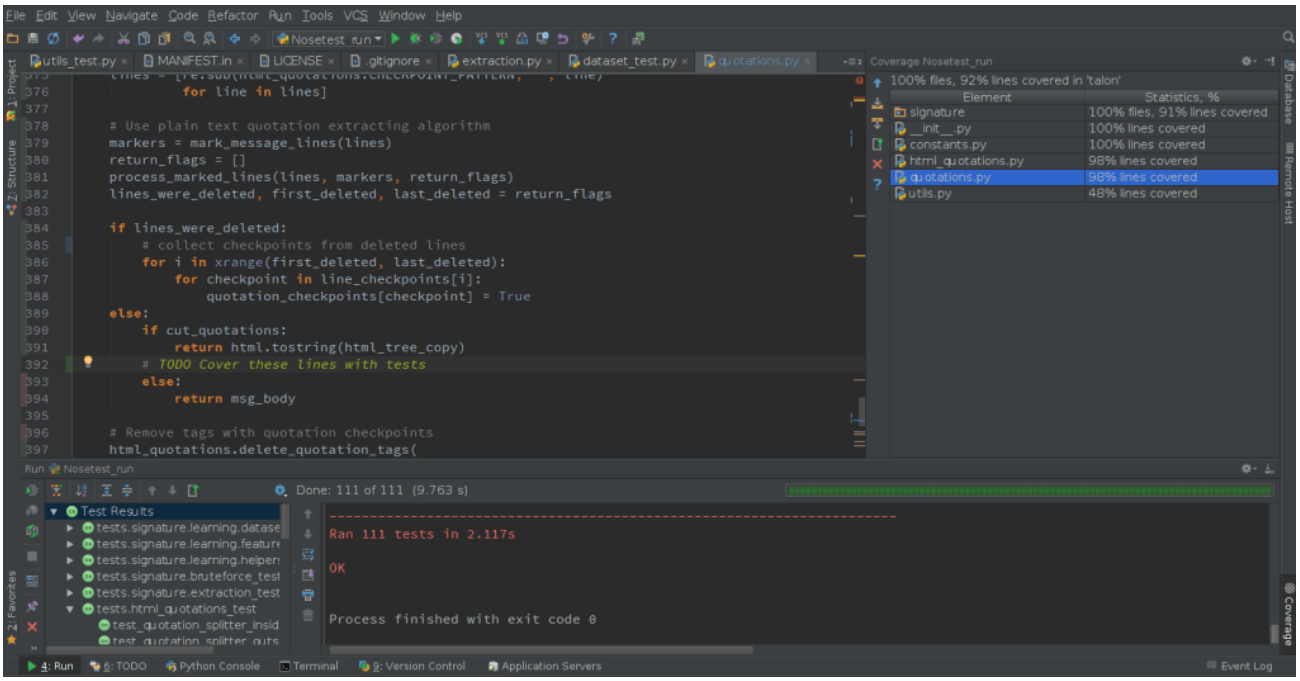

Рисунок 2.13 — PyCharm

До переваг PyCharm можна віднести автоматичне доповнення коду, автопошук та виправлення помилок, підтримку сучасних середовищ для веб – розробки, інтеграцію з системами контролю версій, широкий вибір налаштовуваних інструментів. Недоліками PyCharm є те, що в безкоштовній версії немає підтримки баз даних та висока вартість підписки — 800 $ в рік.

В результаті огляду та аналізу середовищ розробки, найбільш оптимальним середовищем розробки для написання коду є JetBrains Rider , адже, незважаючи на свою високу вартість, він швидкий та зручний, а також містить великий набір корисних інструментів та функцій.

2.3 Обгрунтування вибору програм для 3D–моделювання

Жодна 3D-гра не зможе обійтись без 3D–об’єктів, цими об’єктами можуть бути будинки, автомобілі, дерева та взагалі, будь – які просторові геометричні фігури [13]. Якість 3D–об’єктів, залежить від кількості полігонів, використаних на їх створення (рисунок 2.14).

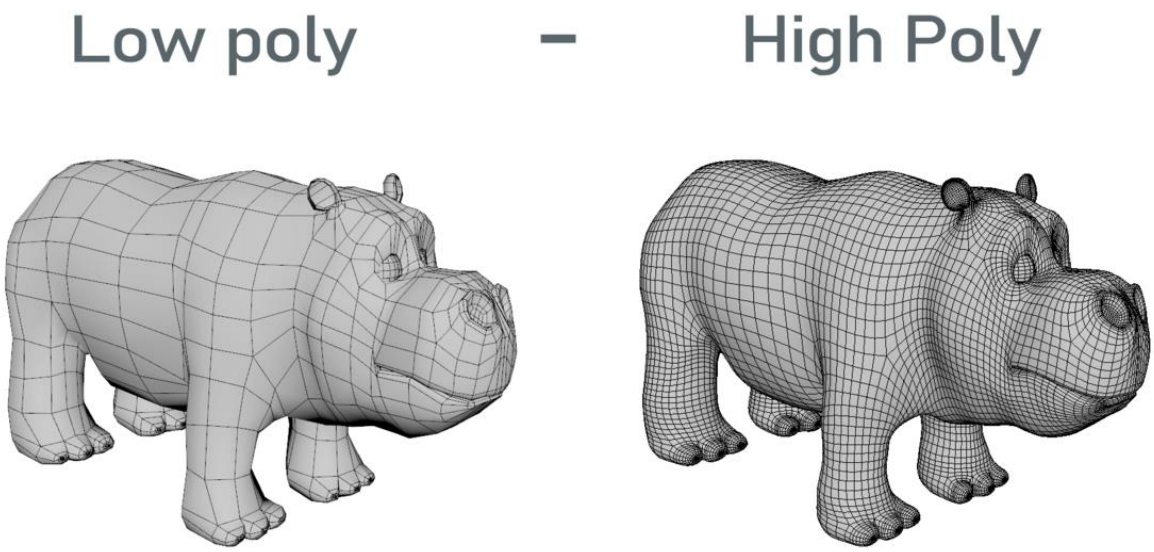

Рисунок 2.14 — Порівняння однакових об’єктів з різною кількістю полігонів

Процес 3D–моделювання полягає у створенні просторової віртуальної фігури, шляхом маніпулювання полігонами, вершинами та ребрами у віртуальному тривимірному просторі. 3D–моделювання використовується не лише в іграх, а й в інших галузях: в архітектурі, в мультфільмах, в інженерії, в медицині та інших.

Перед створенням 3D–об’єктів необхідно визначитись з тим, в якій програмі їх створювати, адже різні програми підходять для тих чи інших цілей, тому важливо наперед визначитись з тим що створювати і в якій програмі це робити.

2.3.1 Maxon Cinema 4D

Maxon Cinema 4D (рисунок 2.15) — програмне забезпечення для створення тривимірної графіки та анімації [14]. Інтерфейс програми зручний у використанні та інтуїтивно зрозумілий. Є можливість кастомізувати інтерфейс для своїх потреб, або ж просто вибрати потрібний з заготовлених пресетів. Пресети містять найбільш зручний інтерфейс для створення анімацій, для рендеру, для скульптингу, для налаштування матеріалів та ще для багато чого іншого.

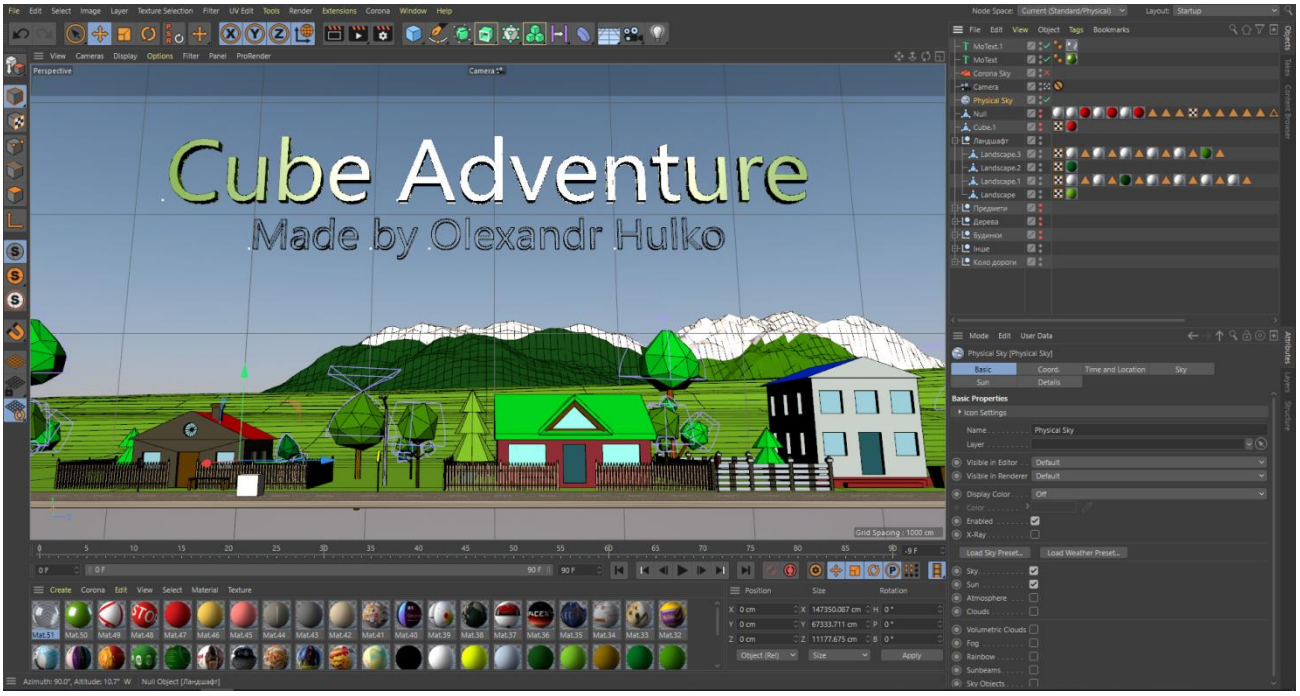

Рисунок 2.15 — Інтерфейс Maxon Cinema 4D

Програма містить інструменти для моделювання, текстурування, рендеру та анімації. Основою для створення об'єктів слугують примітиви на кшталт сфери чи площини, поділені на полігони. Об'єкти, як цілком, так за виділеними полігонами, можуть змінюватися базовими перетвореннями, такими як обертання, зміна розміру, та просунутими — скручування, тиснення, перетворення за формулою тощо. Програма надає ряд деформаторів і генераторів складних об'єктів, наприклад, ландшафтів. Особливістю Cinema 4D є інструмент «ніж» для ручного розділення більшого полігона на менші. Програма також дає змогу малювати полігональні стрічки, прив'язувати одні полігони до інших, перетворювати грані на дуги. Інструмент MoGraph дозволяє автоматично створити з базового об'єкта чи їх групи складний об'єкт, перетвореннями на кшталт клонування чи симетричного копіювання.

Cinema 4D надає гнучку систему створення матеріалів з параметричними шейдерами для швидкого визначення поверхні тривимірної моделі. Матеріали передбачають такі властивості, як текстура, відбиття світла, світіння, прозорість, рельєфне текстурування тощо (рисунок 2.16).

Рисунок 2.16 — Інтер’єр кімнати зроблений в Maxon Cinema 4D

Багатопрохідний рендеринг забезпечує відтворення кольору, тіней, відображень, розмиття. Підтримується експорт тривимірних моделей в програми, такі як Adobe Photoshop, Adobe After Effects, Final Cut Pro, Unity та інші. Крім основного рендера, Cinema 4D може працювати зі сторонніми рендерами, що вбудовуються до програми.

2.3.2 MagicaVoxel

MagicaVoxel (рисунок 2.17) — програма для моделювання 3D–об’єктів у воксельному стилі. Воксель — це аналог пікселя у тривимірному просторі. Вокселі мають форму куба і ними можна легко керувати – створювати, видаляти, переміщувати, красити в колір.

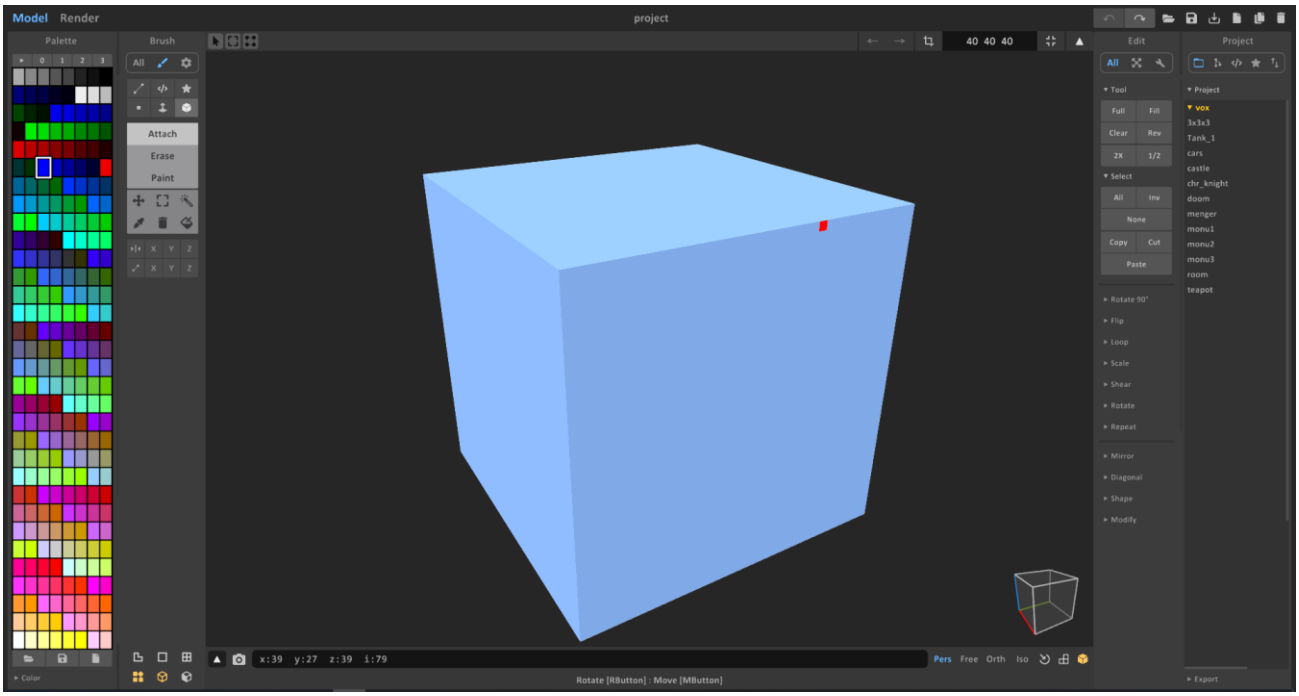

Рисунок 2.17 — Інтерфейс MagicaVoxel

Інтерфейс програми компактний та зрозумілий. Через свою простоту ознайомитись з усіма аспектами програми не важко. Багато хто робить свої перші кроки в 3D–моделюванні саме в MagicaVoxel.

Програма займає дуже мало місця в пам’яті та має невисокі системні вимоги.

Готову 3D–модель можна зберегти в форматі .vox для подальшого імпорту в інші програми. Окрім цього в програмі присутній механізм рендерингу, завдяки якому можна, виставивши потрібний ракурс та налаштувавши необхідні параметри, зберегти зображення об’єкта (рисунок 2.18).

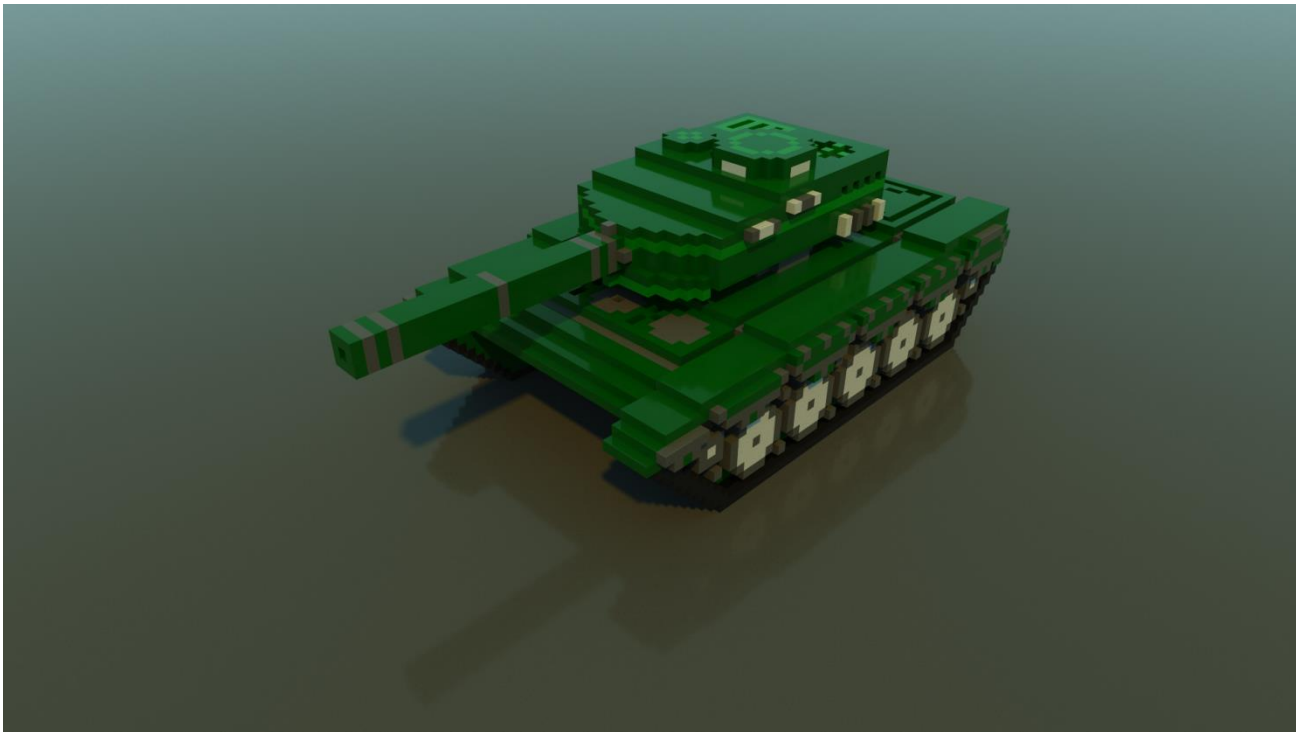

Рисунок 2.18 — Зображення створене в MagicaVoxel

2.3.3 Blender

Blender (рисунок 2.19) — програмне забезпечення, призначене для створення тривимірної графіки. Містить в собі інструменти моделювання, рендерингу, анімації та постобробки відео.

Особливостями програми є невеликий розмір, висока швидкість рендерингу, та підтримка багатьма операційними системами: FreeBSD,

GNU/Linux, Mac OS X, SGI Irix 6.5, Sun Solaris 2.8 (sparc), Microsoft Windows, SkyOS, MorphOS та Pocket PC.

Рисунок 2.19 — Інтерфейс Blender

Blender має такі функції, як симуляція динаміки твердих тіл, (Rigid Body), рідин (Liquid simulation) та м'яких тіл (Soft body), редагування матеріалів і геометрії за принципом вузлів (Nodes), велику кількість легко доступних розширень, написаних мовою Python.

У Blender будь-яка сутність, з якою взаємодіє користувач, називається об'єктом. Це може бути як полігональна модель чи крива, так і джерело світла, камера огляду, полігональна сітка моделі тощо, котрі видимі при редагуванні, але не відображаються в фінальній роботі. При цьому дані об'єкта (певна форма/функція) відділені від нього, тому декілька об'єктів здатні використовувати одні й ті ж дані.

Робота з тривимірними моделями відбувається у сцені, розкресленій координатною сіткою (рисунок 2.20).

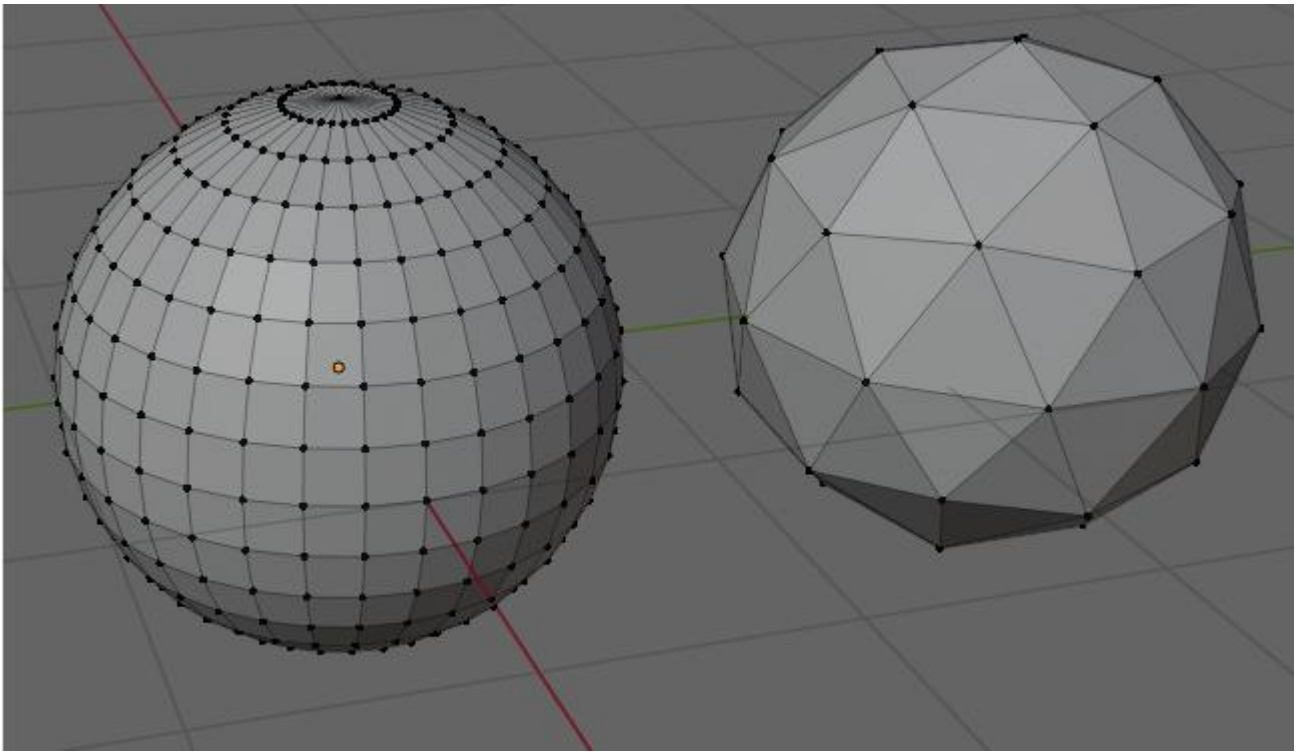

Рисунок 2.20 — Робота з моделями в Blender

Об'єкти сцени об'єднуються в так звані колекції. За промовчуванням кожна сцена має одну колекцію, але можна створювати нові колекції та переміщувати між ними об'єкти для згрупування роботи. Схема сцени відображається за замовчуванням праворуч вгорі. До сцени може додаватися фон або площина із зображенням – зразком для моделювання. Об'єкти можуть бути неактивними (з ними не відбувається взаємодії), активними (відбувається непряма взаємодія) та вибраними (користувач взаємодіє конкретно з цим об'єктом). Всі вони мають координати, що враховуються при переміщенні та деформації, та виходять з поворотної точки, що може перебувати за межами самого об'єкта. Blender містить інструменти для моделювання методом скульптингу, що симулює ліплення з глини.

На полігональні моделі можуть накладатися текстури та матеріали. Текстура визначає вигляд моделі (колір, прозорість, імітація шорсткості, блиску тощо), а матеріал додатково визначає взаємодію з іншими об'єктами

(заломлення променів світла, відображення, свічення). Створення текстур і матеріалів відбувається за допомогою наочних схем, які складаються з вузлів (нодів, nodes) і зв'язків між ними. Існують вузли, що задають масштаб, яскравість, змішування текстур і т. д. Текстури й матеріали можуть генеруватися процедурно.

2.3.4 Autodesk 3Ds Max

Autodesk 3Ds Max (рисунок 2.21) — тривимірний графічний редактор з найсучаснішими засобами для художників, дизайнерів та архітекторів. Дана програма використовують для візуалізації моделей будівель, комп’ютерних ігор, тривимірних анімаційних мультфільмів, рекламних роликів тощо. За допомогою цього редактора зроблено безліч анімованих моделей для кінофільмів.

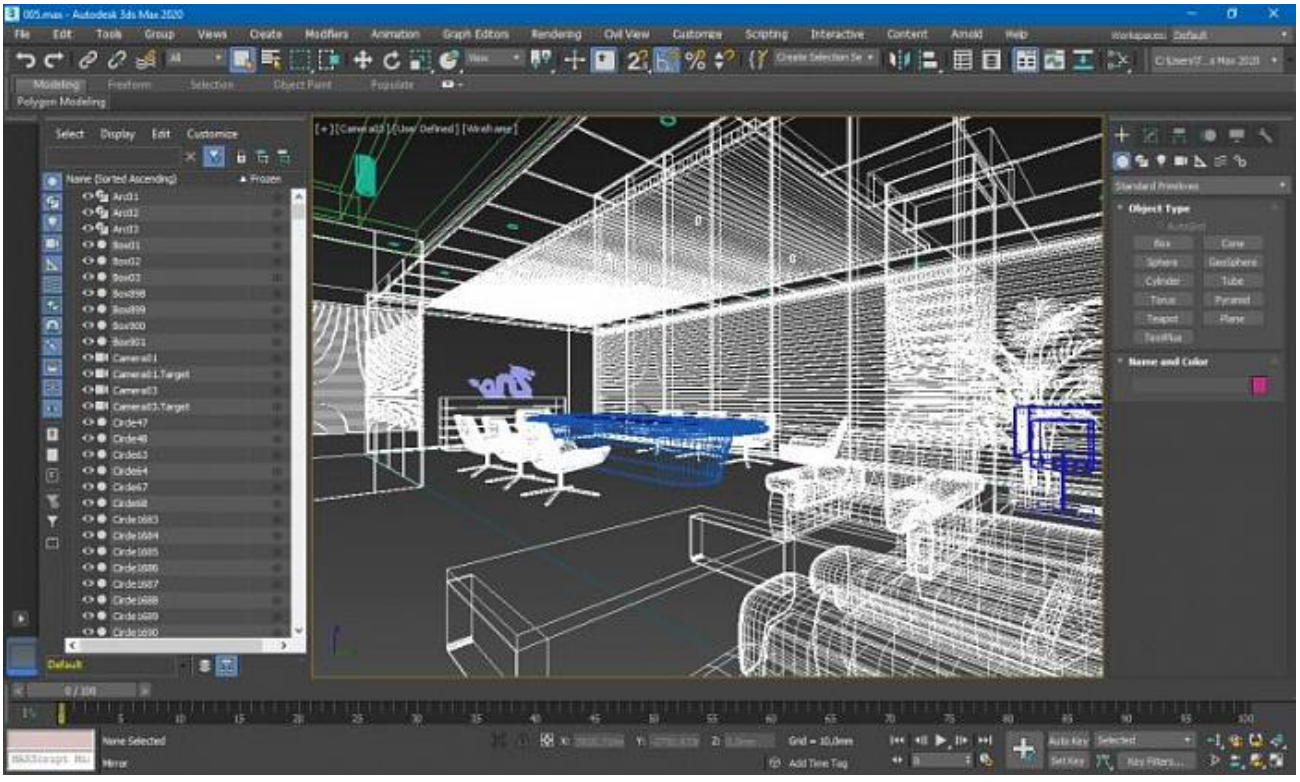

Рисунок 2.21 — Інтерфейс Autodesk 3Ds Max

В програмі не погано реалізована система частинок, а також інструменти для анімації. За рахунок добре продуманого механізму прорахунку фізики, моделювання поведінки твердих та м’яких тіл проводиться доволі реалістично. Присутня технологія HairandFur, що дозволяє буквально в два кліка налаштувати шерсть чи волосся.

Традиційно ця програма рахується професійним інструментом архітекторів та дизайнерів інтер’єру. Причиною цього слугує зручність моделювання об’єктів, великий вибір в створенні 3D–об’єктів, та якісні модулі для фотореалістичної візуалізації .Проте Autodesk 3Ds Max дозволяє виконувати велику кількість функцій, що далеко виходять за рамки архітектурних моделей.

В результаті огляду та аналізу програмних засобів для 3D–моделювання, найбільш оптимальним рішенням для створення 3D–об’єктів для гри, є використання програм MagicaVoxel та Cinema 4D, адже вони зручні у використанні та дозволяють створювати складні 3D–об’єкти за короткий час.

3 РОЗРОБКА ТА ТЕСТУВАННЯ 3D-ГРИ В СЕРЕДОВИЩІ UNITY 3D

Розробимо гру, в якій гравець управляє танчиком та змагається з іншими танчиками, які керуються комп’ютером. Всі танчики поділені на команди, гравець є частиною однієї з команд. Метою гравця є знищення танчиків ворожої команди.

Під час розробки гри потрібно пройти такі основні етапи: моделювання та налаштування моделі танка в програмах для 3D–моделювання, імпорт та налаштування моделей в Unity 3D, написання скриптів ігрової логіки, робота зі звуком, створення та налаштування візуальних ефектів.

3.1 Моделювання та оптимізація 3D–моделі танка

Перш за все необхідно зробити 3D–модель танка. Модель танка має бути у воксельному стилі, тому для моделювання найбільш оптимально використати програму MagicaVoxel (рисунок 3.1).

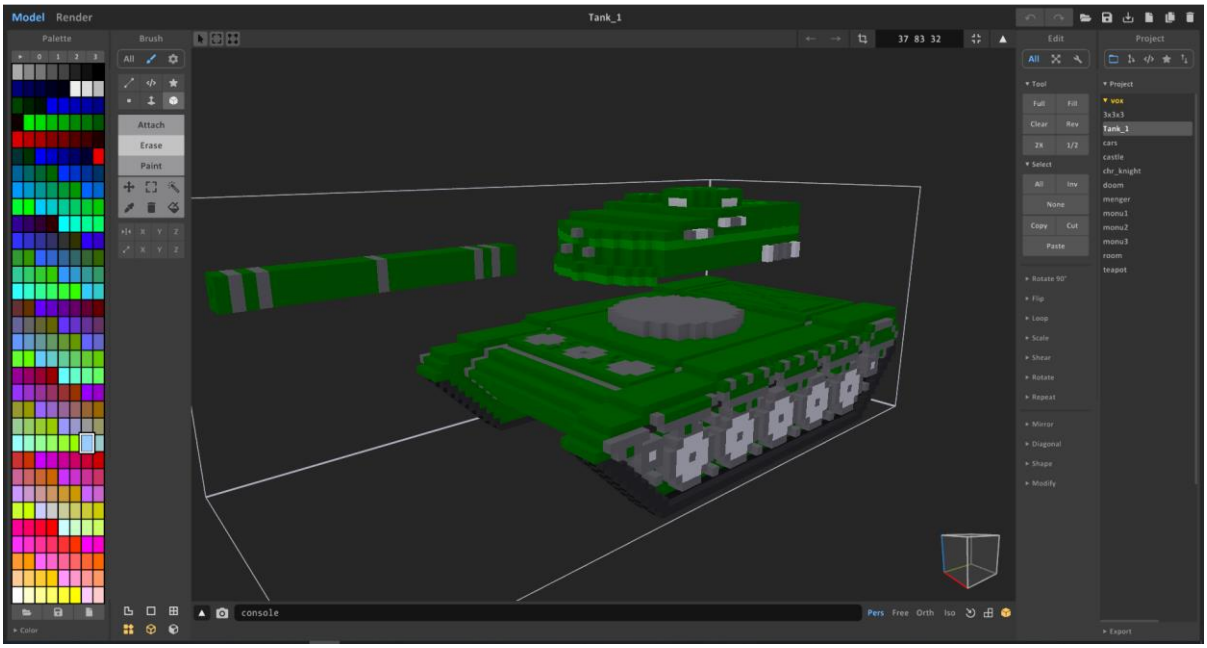

Рисунок 3.1 — Змодельована модель танка в MagicaVoxel

3D–модель танка розділена на окремі функціональні частини, адже їх потрібно буде окремо налаштовувати.

Після цього збережемо 3D–модель у форматі .vox для подальшого імпорту та налаштування у Cinema 4D (рисунок 3.2).

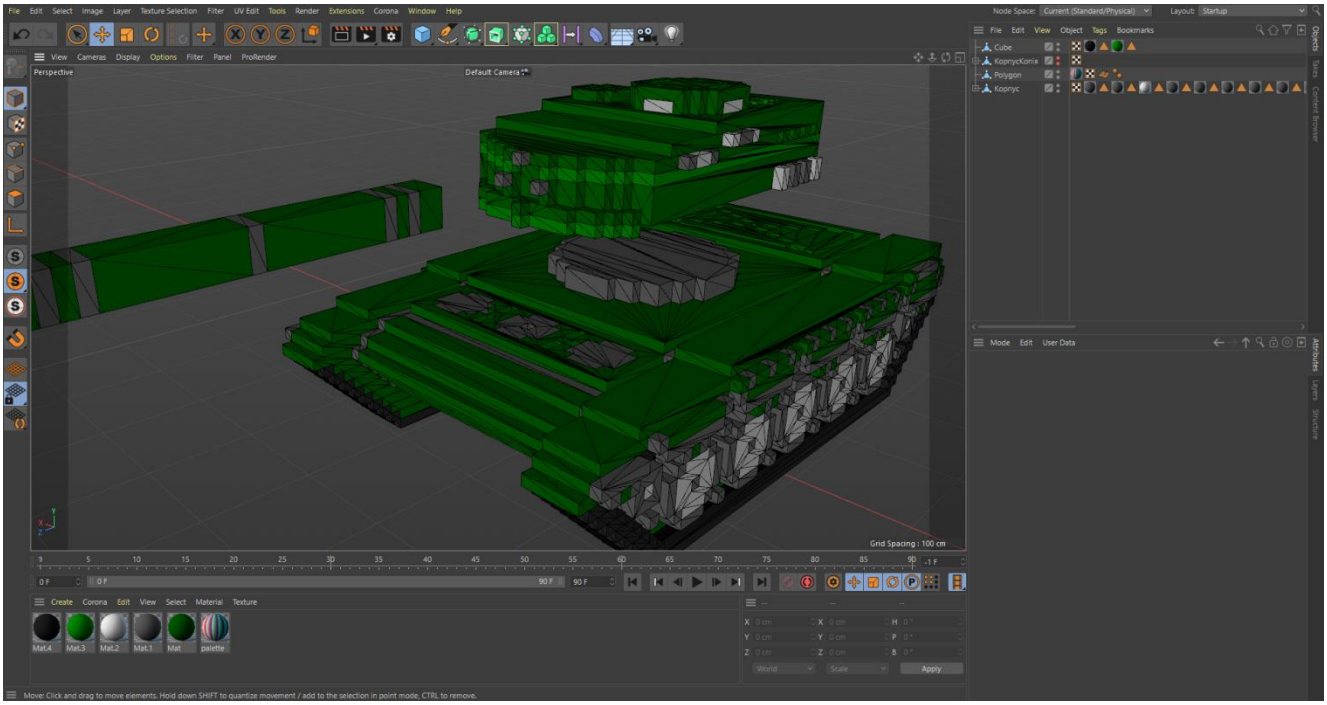

Рисунок 3.2 — Імпортована модель у Cinema 4D

Як видно рисунка 3.2 модель танка дуже не оптимізована та містить зайві полігони. Детальна інформація про модель зображена на рисунку 3.3

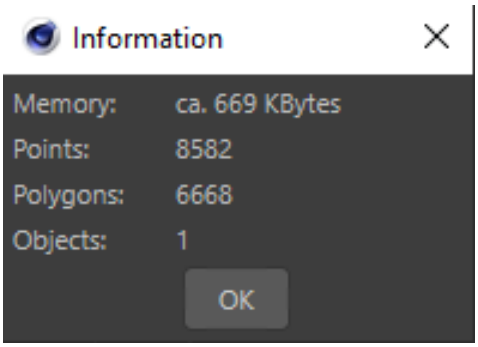

Рисунок 3.3 — Детальна інформація про 3D–модель танка

Проведемо оптимізацію 3D–моделі танка. Нажаль в Cinema 4D немає засобів для оптимізації полігональної сітки, тому це прийдеться робити вручну. Процес оптимізації полягає у виділенні всіх полігонів, які лежать на одній площині та заміні їх на один великий полігон. Таким чином зменшиться кількість вершин, ребер та полігонів, що призведе то зменшення розмірів займаної пам’яті об’єктом, та меншого навантаження на відеокарту. Полігональна сітка оптимізованої моделі танка зображена на рисунку 3.4.

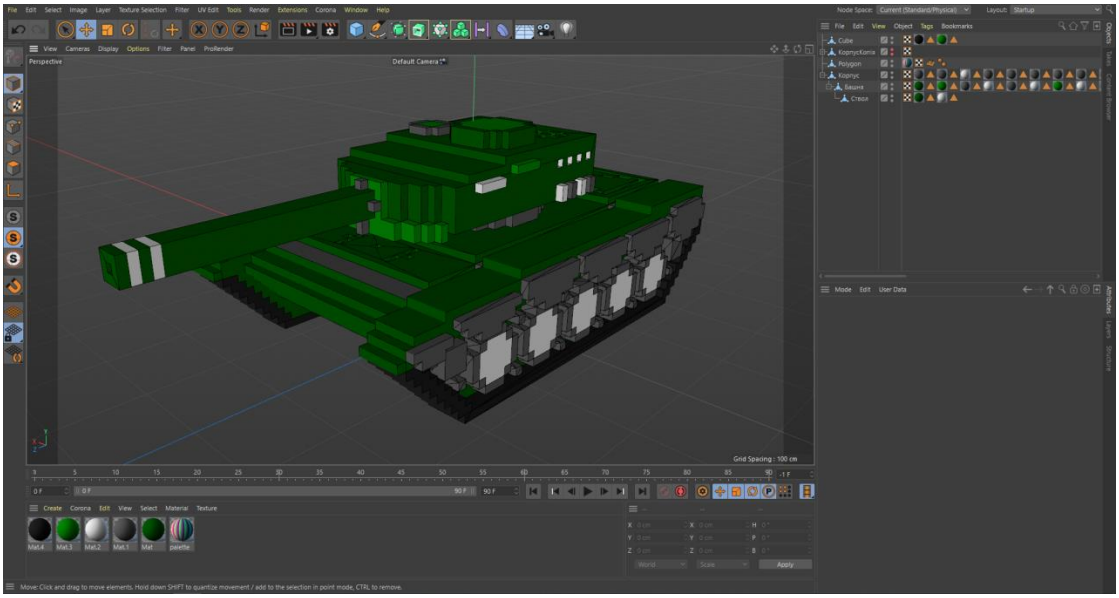

Рисунок 3.4 — Оптимізована 3D–модель танка

З рисунка 3.4 видно як зменшилась кількість полігонів. Детальна інформація про оптимізовану 3D – модель зображена на рисунку 3.5

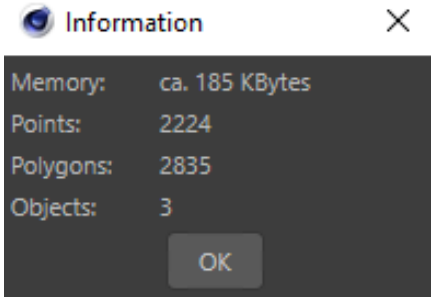

Рисунок 3.5 — Детальна інформація про оптимізовану 3D–модель танка

Порівнюючи детальну інформація до та після оптимізації, видно, що об’єм пам’яті, кількість вершин та полігонів зменшились приблизно в 3 рази. Завдяки оптимізації, у грі, при однаковому навантаженні на відеокарту, можна буде відобразити у 3 рази більше танків.

Окрім оптимізації моделі також необхідно змінити центри складових частин танка. Центр об’єкта — це точка, навколо якої відбувається поворот об’єкта та інші маніпуляції (рисунок 3.6).

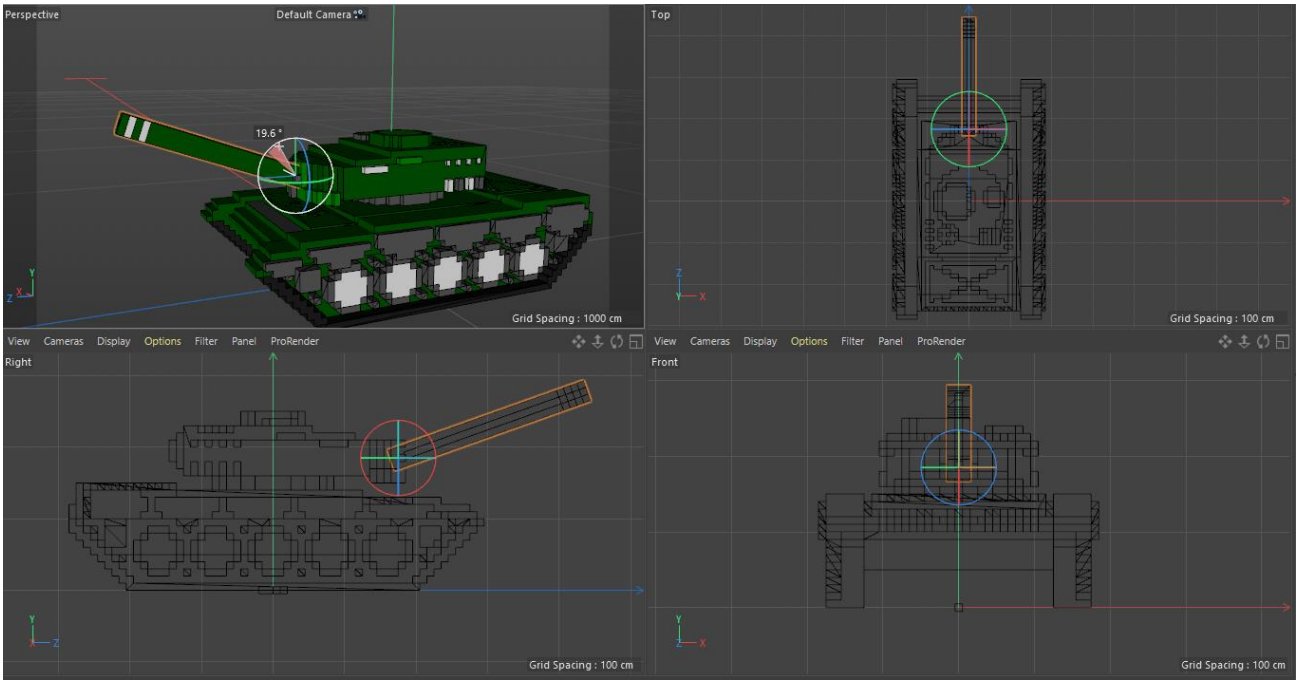

Рисунок 3.6 — Центр ствола

Модель танка складається з 3 частин: ствола, поворотної башні та корпусу. По замовчуванню в програмах для 3D–моделювання (Cinema 4D не є виключенням) центром 3D–об’єкта є його геометричний центр. В моєму випадку центр ствола має знаходитись в місці кріплення до поворотної башні, центр башні має знаходитись в місці кріплення до корпусу, а центр корпусу має знаходитись по центрі внизу. Внизу — тому, що при розташуванні об’єкта на ігровій сцені в Unity, він розташовується відносно свого центра. Якщо центр

знаходиться внизу об’єкта, то такий об’єкт розташується коректно, а якщо центром об’єкта є його геометричний центр, то він наполовину зайде в платформу, на якій розміщується.

3.2 Налаштування моделі танка в Unity 3D

Unity підтримує безліч форматів 3D–об’єктів, тому імпорт моделей не складає великих труднощів. Після імпорту потрібно налаштувати матеріали (рисунок 3.7).

Рисунок 3.7 — Налаштування матеріалів

У вікні налаштування матеріалу є багато налаштування, самі основні з яких: вибір кольору, прозорості, металічності та шороховатості.

Після налаштування матеріалу, на кожну частину танка потрібно добавити колайдер. Колайдери необхідні для того, щоб одні об’єкти не проходили крізь інші. Колайдери не можуть по своїй формі повністю відтворити форму об’єкта, адже виявлення зіткнень для нього буде дуже

завантажувати процесор. Зазвичай колайдери мають спрощену форму у порівнянні з об’єктом на якому вони є (рисунок 3.8).

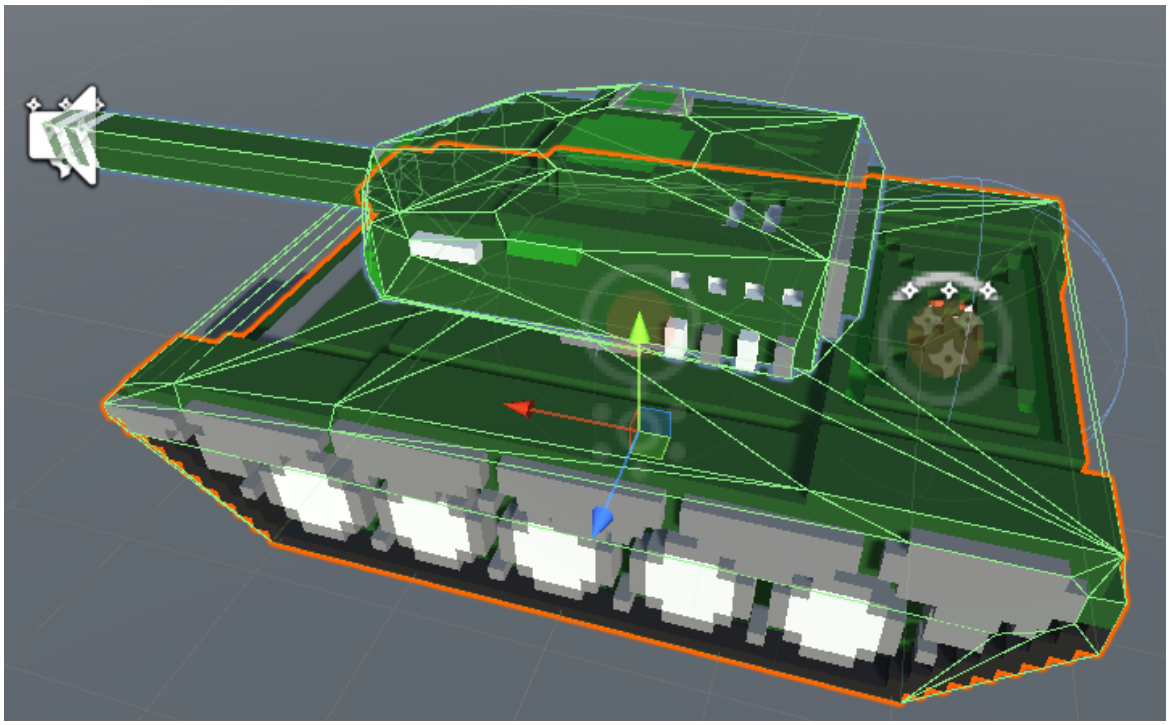

Рисунок 3.8 — Колайдери на частинах танка

Окрім колайдерів необхідно добавити компонент Rigidbody. На об’єкти з даним компонентом обраховується фізика, в залежності від налаштованих параметрів. Налаштування компоненту Rigidbody зображено на рисунку 3.9.

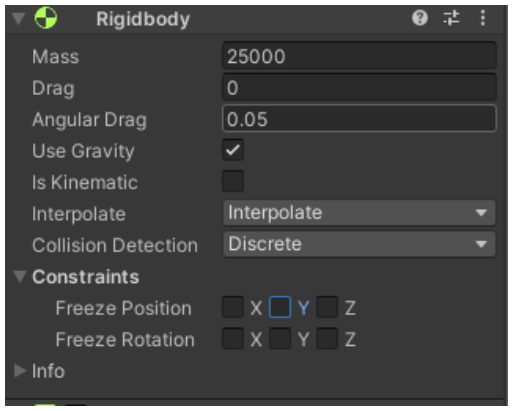

Рисунок 3.9 — Налаштування Rigidbody

Розглянемо параметри Rigidbody:

— Mass —маса об’єкта;

— Drag — опір повітря;

— Angular Drag — кутовий опір;

— Use Gravity — дія гравітації на об’єкт;

— Is Kinematic — розрахунок фізики для об’єкта;

— Interpolate — синхронізація фізики з рендером;

— Freeze Position/Rotation — вимикає фізичний розрахунок руху/повороту по вибраним осям.

3.3 Розробка логіки управління танком в Unity 3D

Для зручності код управління танком винесений в окремий клас TankControllerBase (додаток Б), від цього класу наслідуються класи TankControllerHero та TankControllerBot. Кожен з двох класів наслідників використовує однакові методи батьківського класу, але в TankControllerHero ці методи визиваються в залежності від нажаття клавіш гравцем, а в TankControllerBot визиваються алгоритмом поведінки.

В Unity є два способи руху об’єкта — фізичний та не фізичний. Фізичний спосіб імітує прикладення фізичної сили до об’єкта, через що об’єкт рухається плавно та реалістично і взаємодіє з іншими об’єктами. Не фізичний спосіб руху переміщає об’єкт шляхом зміни його позиції в ігровому просторі, тобто фактично це не рух а телепортація. Через те що телепортація відбувається на невеликі відстані – гравцем це сприймається як плавний рух.

В грі реалізований фізичний спосіб руху. Рух відбувається через задання швидкості та напрямку руху компоненту Rigidbody. Реалізації наведена в лістингу 3.1.

Лістинг 3.1 — Метод фізичного руху танка

```
protected void MoveTank( Vector3 direction, float MAX_SPEED =
MAX_SPEED)
{
_tankBody.velocity = direction * MAX_SPEED;
}
```

Поворот об’єкта відбувається через прикладення обертальної сили до об’єкта (лістинг 3.2).

Лістинг 3.2 — Метод фізичного поворотоу танка

```
protected void RotateTank( Vector3 direction )
{
_tankBody.AddTorque(direction * Time.deltaTime * 100,
ForceMode.VelocityChange);
}
```

Для поворота встановлене обмеження в 60 градусів за секунду, адже безперервна дія обертальної сили на об’єкт призвела б до того, що об’єкт буде постійно пришвидшувати обертатись:
_tankBody.maxAngularVelocity = Mathf.Deg2Rad * ANGULAR_SPEED;
Константа ANGULAR_SPEED = 60. Оскільки максимальна кутова швидкість задається в радіанах, то потрібно перевести значення з градусів в радіани за допомогою Mathf.Deg2Rad.

Поворот таких частин танка, як ствол та башня, реалізований не фізичним способом, адже в даному випадку він недоцільний. Кут повороту ствола танка обмежений діапазоном від -10 до 30 градусів.

3.4 Створення звукових ефектів у середовищі Unity 3D

Для програвання звуку потрібно додати компонент Audio Source (рисунок 3.10).

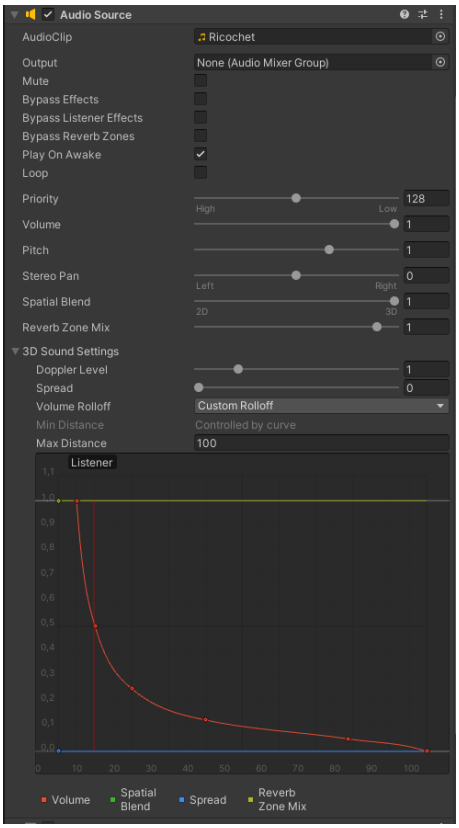

Рисунок 3.10 — Компонент Audio Source

Audio Source має багато налаштовуваних параметри, основні з яких: звуковий файл для відтворення, гучність звуку, можливість зациклення, налаштування кривої гучності в залежності від відстані та інші.

Коли танк стоїть на місці — відтворюється зациклений тихий звук двигуна, коли танк повертається або рухається то відторюється звук руху. Гучність даного звуку залежить від швидкості руху та обертання (лістинг 3.3).

Лістинг 3.3 — Метод контролю гучності звуку

```
protected void ControlMoveVolume()
```

```
{
float move_audio  = Mathf.Clamp(_tankBody.velocity.magnitude, 0, 0.5f);
float rotate_audio  = Mathf.Clamp(_tankBody.angularVelocity.magnitude, 0, 0.5f);
_audioMove.volume = move_audio + rotate_audio;
}
```

При пострілі лунає звук пострілу, при потраплянні снаряда в цілі, на місці потрапляння лунає звук вибуху, при рикошеті, на місці рикошета лунає звук рикошета. Також при прольоті снаряда недалеко від гравця, лунає характерний свист. Це дасть змогу гравцю зрозуміти звідки та наскільки близько летів снаряд. Гучність свисту залежить від швидкості снаряда.

Кожен компонент Audio Source налаштовувався відповідно до характеру звуку. Наприклад звук руху танка буде чутний недалеко, а от звук пострілу чи вибуху гравець почує здалеку.

3.5 Розробка візуальних ефектів у середовищі Unity 3D

За допомогою системи частинок можна симулювати дим, вогонь, вибухи, дощ та інші ефекти. Для їх створення в Unity необхідно вибрати та додати компонент Particle System (рисунок 3.11).

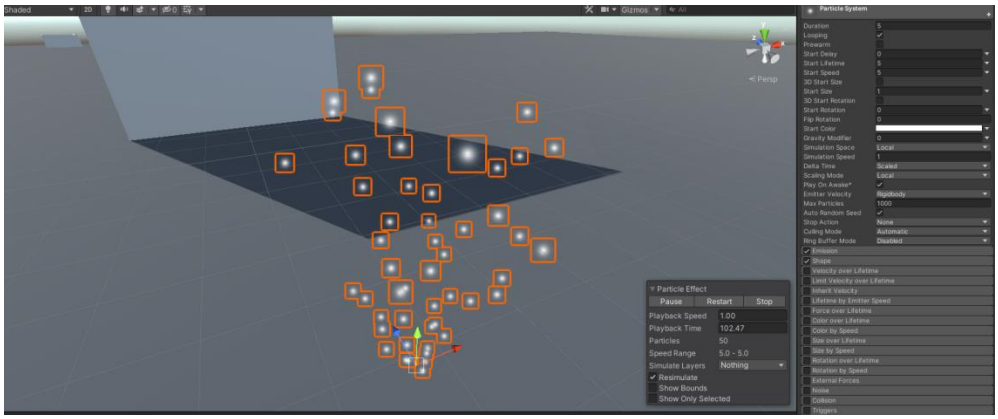

Рисунок 3.11 — Система частинок по замовчуванню

В налаштуваннях Particle System є безліч різних параметрів, основні з яких: колір частинок, їх швидкість, час життя, форма, розмір та форма розповсюдження.

При пострілі з танка, на кінці ствола відтворюється відповідна система частинок (рисунок 3.12). Це надає грі більшій реалістичності.

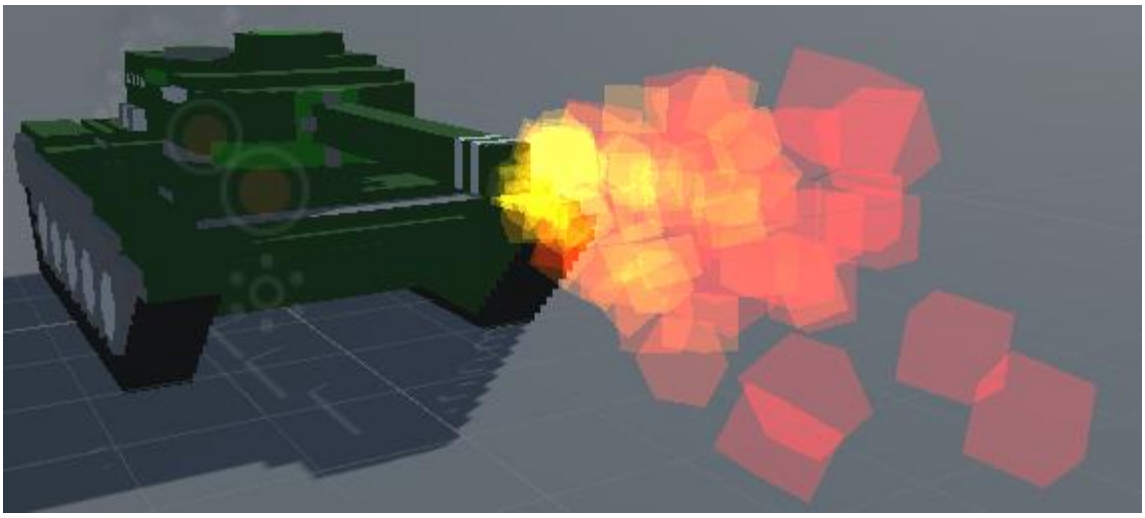

Рисунок 3.12 — Постріл з танка

При вибуху снаряда на місці попадання відтворюється система частинок, що імітує вибух (рисунок 3.13).

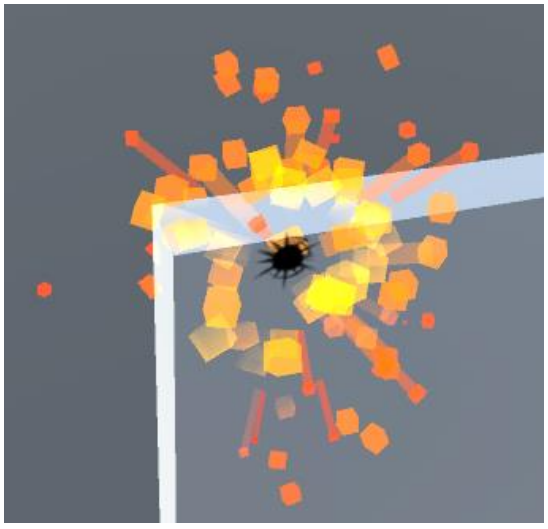

Рисунок 3.13 — Вибух снаряда

На танкові відтворюється система частинок, що імітує вихлопні гази від роботи двигуна (рисунок 3.14).

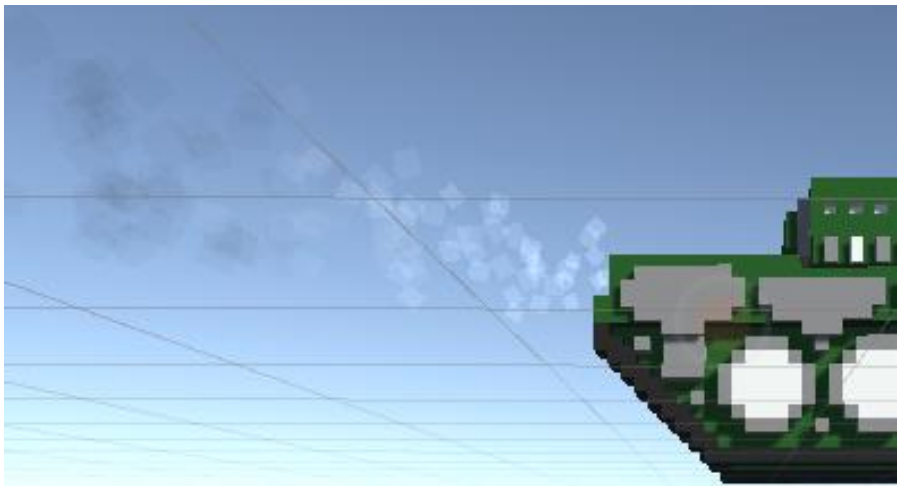

Рисунок 3.14 — Вихлопні гази

Зі зменшенням кількості здоров’я вихлопні гази темніють (рисунок 3.15). Це допомагає зрозуміти наскільки пошкоджений танк.

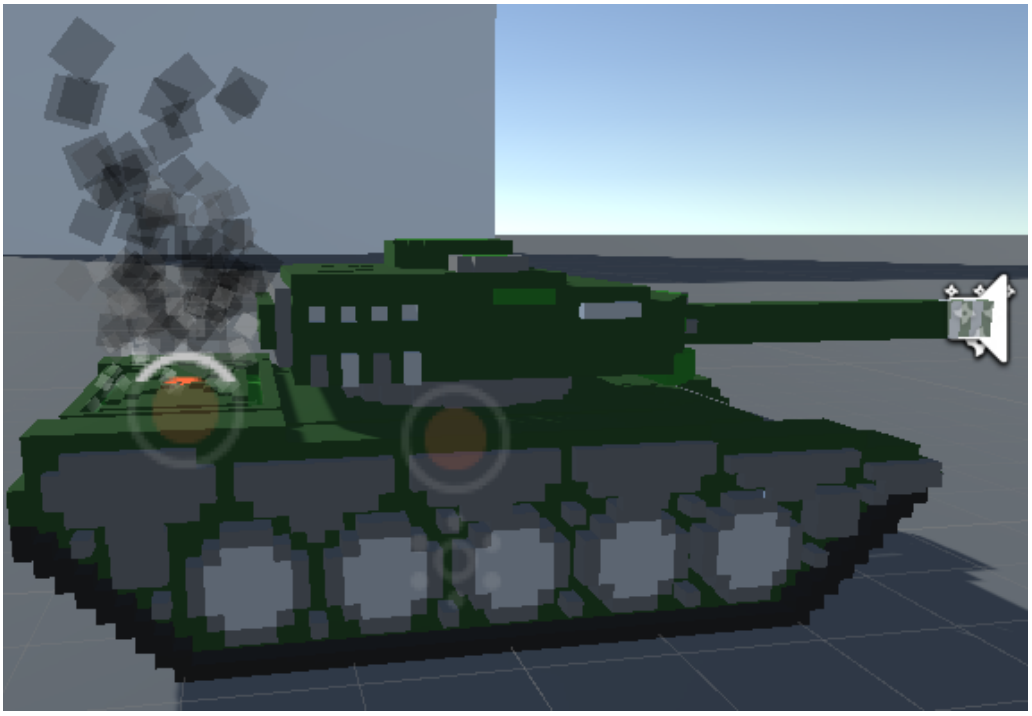

Рисунок 3.15 — Вихлопні гази підбитого танку

Метод контролю кольору системи частинок вихлопних газів підписаний на подію, яка визивається при зміні кількості здоров’я наведений в лістингу 3.4.

Лістинг 3.4 — Зміна кольору системи частинок

```
onDamage += ControlSmokeParticles;
private void ControlSmokeParticles(int health)
{
ParticleSystem.MainModule smokeParticlesMain = _smokeParticles.main;
Color color = smokeParticlesMain.startColor.color;
int alpha = Mathf.Clamp(30 + (100 - health), 30, 130);
color.a = alpha / 255f;
smokeParticlesMain.startColor = color;
}
```

3.6 Розробка механіки стрільби

При стрільбі відбуваються чотири речі: спавниться снаряд, відтворюється звук стрільби, відтворюється система частинок, програється анімація віддачі ствола.

Спавн снаряда (скрипт снаряда наведений в додатку В) відбувається через метод Instantiate, в який потрібно передати три параметра: об’єкт, що спавниться, вектор позиції та поворот. Після цього снаряду необхідно задати швидкість по вектору, який направлений вперед відносно снаряда:
GameObject currentBullet = Instantiate(_bullet, _spawnBullet, _placeSpawn.rotation); currentBullet.GetComponent<Rigidbody>().velocity = currentBullet.transform.forward * BULLET_SPEED;

Константа BULLET_SPEED = 300, тобто снаряд полетить зі швидкістю 300 м / с. Для того, аби можна було відслідковувати рух снаряда на такій

великій швидкості, на снаряд був добавлений та налаштований компонент Trail Renderer (рисунок 3.16).

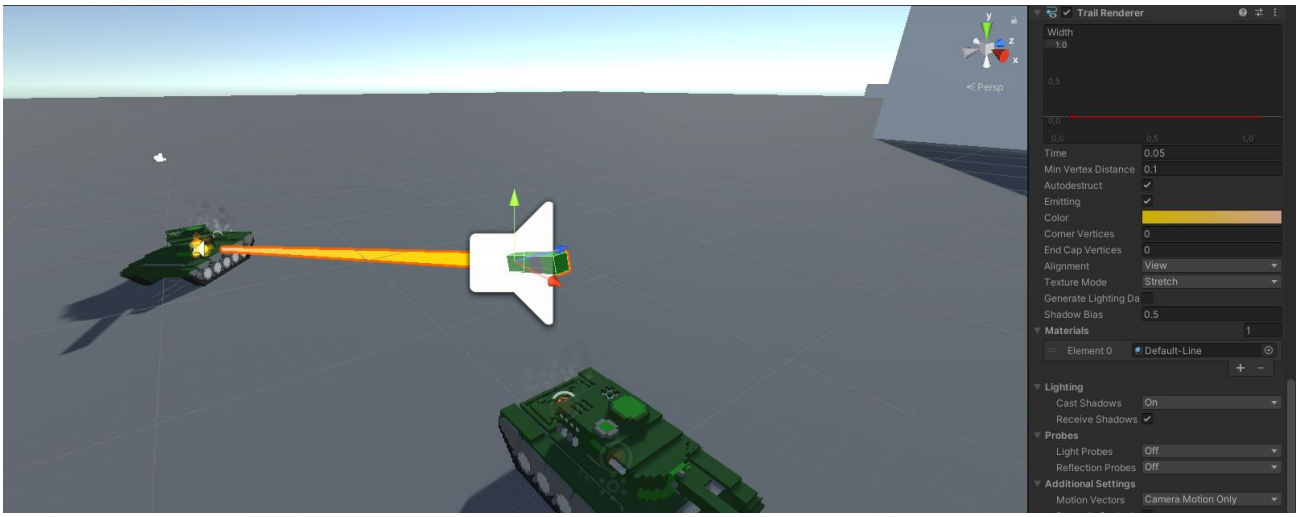

Рисунок 3.16 — Trail Renderer

Trail Renderer рисує лінію руху об’єкта. Даний компонент можна гнучко налаштувати під власні потреби.

Для більшої реалістичності при стрільбі, ствол танка рухається назад, а потім в начальну позицію, таким чином створюється ефект віддачі. Рух ствола контролюється анімаційною кривою AnimationCurve (рисунок 3.17);

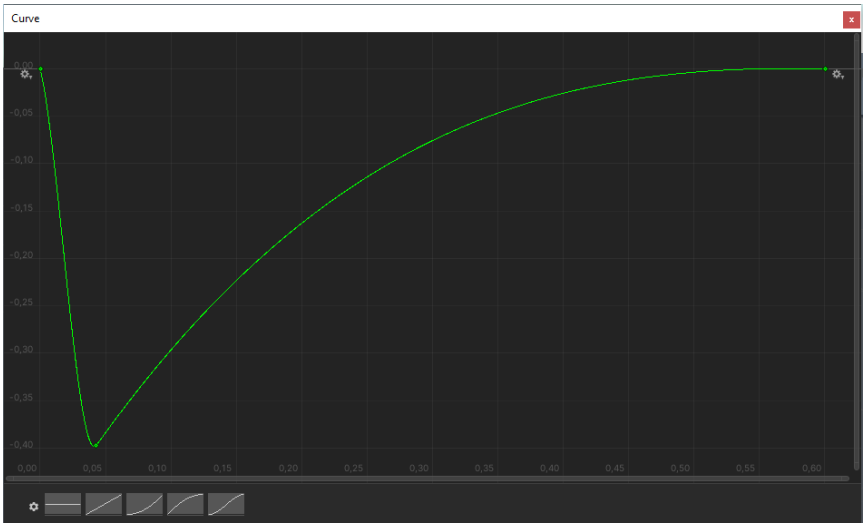

Рисунок 3.17 — Контроль анімації віддачі

З рисунка 3.17 видно, що значення анімаційної кривої різко спадає, а потім плавно переходить в початкове значення. Значення з AnimationCurve зчитуються в корутіні ShootAnimation, в якій відтворюється анімація (лістинг 3.5)

Лістинг 3.5 — Код анімації віддачі ствола танка

```
StartCoroutine(ShootAnimation(_gun, _gunAnimationCurve, _tower));
IEnumerator ShootAnimation(GameObject gun, AnimationCurve animationCurve, GameObject parentObject)
{
Vector3 relativeVector;
float time = 0;
while (time <= animationCurve.keys[animationCurve.length - 1].time)
{
time += Time.deltaTime;
relativeVector = parentObject.transform.position + parentObject.transform.forward * 1.75f + parentObject.transform.up * 0.4f;
gun.transform.position = relativeVector + gun.transform.forward * animationCurve.Evaluate(time);
yield return null;
}
}
```

Корутіна представляє собою ітератор, що повертає значення IEnumerator та використовує ключове слово yield. Корутіна виконується паралельно основному потоку. Корутіни дуже зручно використовувати у випадках, коли потрібно запустити метод, що повинен працювати деякий час паралельно основному потокові.

Снаряд при попаданні може не вибухнути а відрикошетити. Є декілька варіантів обрахунку того, чи снаряд відрикошетить. Перший і самий простий варіант — при попаданні снаряда він може відрикошетити з деякої ймовірністю (наприклад 10 %). Цей варіант в деяких випадках може виглядіти нереалістично. Другий варіант — визначати кут між вектором нормалі поверхні в яку попав снаряд, та вектором швидкості снаряда. Цей варіант підходить в тих випадках, коли снаряд сферичний. В грі снаряд має форму прямокутного паралелепіпеда, тому такий розрахунок при певних умовах дасть некоректні дані. Третій варіант (реалізований у грі) — обрахувати швидкість до та в момент зіткнення, якщо швидкість змінилась менш ніж на 50 м / с, то снаряд рикошетить, інакше — вибухає (лістинг 3.6)

Лістинг 3.6 — визначення рикошету або вибуху снаряда

```
_speed_after_collision = _rigidbody.velocity.magnitude;
_speed_collision = _bullet_speed - _speed_after_collision;
_explosion = _speed_collision >= _explosive_speed;
_point = other.GetContact(0);
if (_explosion)
Explosion(_point);
else
Ricochet(_point);
_bullet_speed = _speed_after_collision;
```

На місці вибуху, окрім звуку вибуху та системи частинок спавниться спрайт з слідом від снаряда (рисунок 3.18).

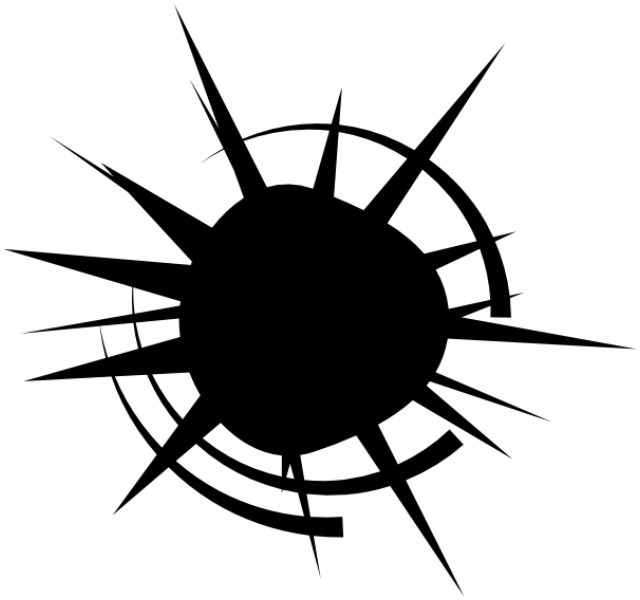

Рисунок 3.18 — Слід від вибуху снаряда

Для оптимізації, роздільна здатність спрайта зменшена до 32 х 32. Метод спавна спрайта наведений в лістингу 3.7.

Лістинг 3.7 — спавн спрайта на місці зіткнення снаряда

```
private void SpawnHoleSprite(ContactPoint point)
{
if (!point.otherCollider.gameObject.isStatic)
return;
Vector3 position = point.point + point.normal.normalized * 0.001f;
Quaternion rotation = Quaternion.LookRotation(point.normal) *
Quaternion.Euler(0, 0, Random.Range(0, 360));
PrefabSpawner.SpawnPrefab(_container.holeSprite, position, rotation);
}
```

Метод SpawnHoleSprite підписаний на подію, яка визивається при вибуху снаряда, він приймає об’єкт класу ContactPoint, в якому є інформація про вектор точки зіткнення, вектор нормалі поверхні з якою було зіткнення, інформацію

про колайдер що зіткнувся і колайдер з яким зіткнулись. Якщо об’єкт не позначений статичним, то спрайт не спавниться (статичними позначаються лише ті об’єкти, які не будуть рухатись). Вектор місця спавна = вектору точки зіткнення + нормалізований вектор нормалі поверхні, помножений на 0.001, це потрібно для того, щоб уникнути графічних артефактів, коли дві поверхні розташовуються на однакових координатах (рисунок 3.19).

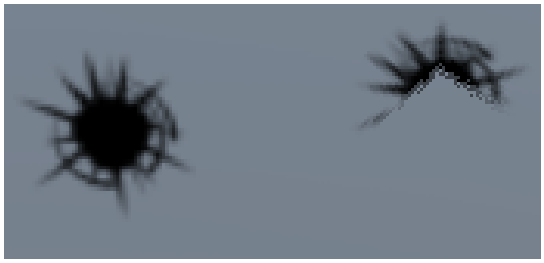

Рисунок 3.19 — Правильно(зліва) та неправильно (справа) налаштована позиція

Окрім налаштування позиції також потрібно налаштувати поворот спрайту, за допомогою Quaternion.LookRotation(point.normal) спрайт повертається перпендикулярно вектору нормалі, та адекватно розміщується на різних поверхнях. Окрім цього кожен спрайт повертається по локальній осі Z на випадкову кількість градусів, в межах від 0 до 360. Таким чином кожен спрайт виглядає по різному (рисунок 3.20).

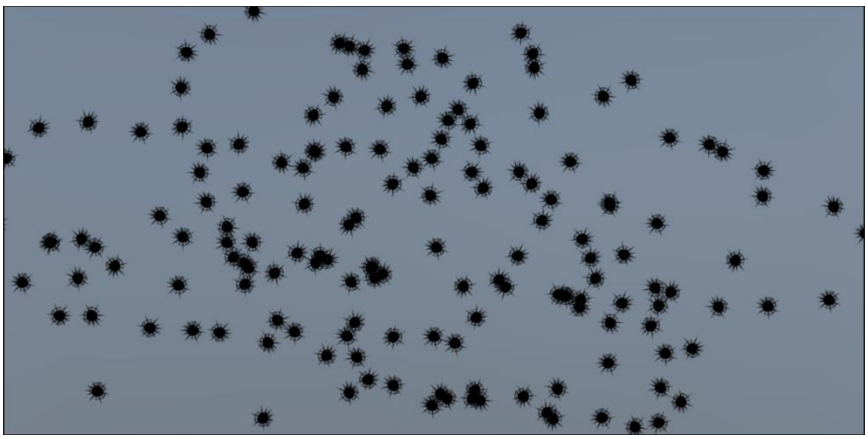

Рисунок 3.20 — Спавн спрайтів з різним поворотом

Якщо кількість спрайтів на ігровій сцені досягає декількох тисяч, це погано впливає на продуктивність, тому кожний спрайт відображається лише одну хвилину, після чого, за секунду плавно розчиняється та пропадає (лістинг 3.8), весь скрипт наведений в додатку Г.

Лістинг 3.8 — плавне зникнення спрайту

```
private IEnumerator DestroyAsSprite()
{
SpriteRenderer spriteRenderer = GetComponent<SpriteRenderer>();
Color color = spriteRenderer.color;
yield return new WaitForSeconds(time);
while (color.a > 0)
{
color.a -= Time.deltaTime;
spriteRenderer.color = color;
yield return null;
}
Destroy(gameObject);
}
```

На рисунку 3.21 показано зникнення спрайту сліда від снаряду.

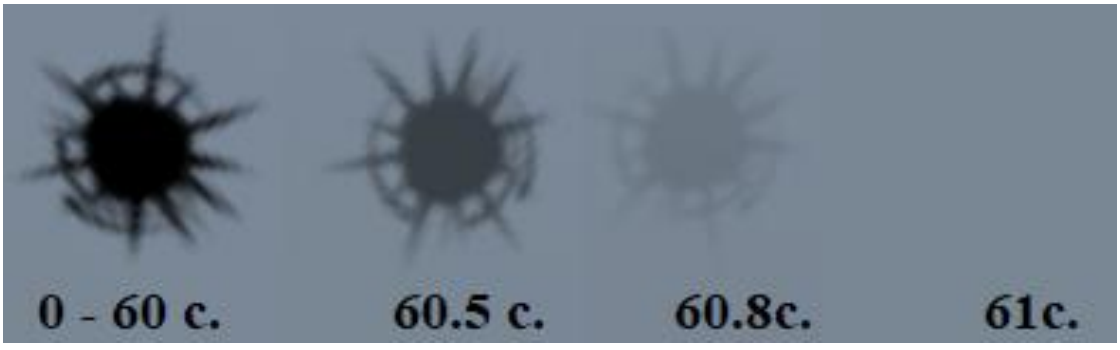

Рисунок 3.21 — Плавне зникнення сліда від снаряда

3.7 Тестування роботи комп’ютерної 3D-гри

Запустимо гру, внизу в центрі екрана розташований танчик гравця (рисунок 3.22).

Рисунок 3.22 — Танчик гравця на ігровій сцені

Тепер перевіримо управління танком, поворот та рух танка керується клавішами WASD або стрілками на клавіатурі, поворот башні та ствола реалізовується через рух комп’ютерної миші, ПКМ відповідає за стрільбу, ЛКМ — за прицілювання (рисунок 3.23).

Рисунок 3.23 — Управління танчиком

Після попадання по статичних об’єктах, в місцях попадання появляється слід від вибуху снаряда (рисунок 3.24).

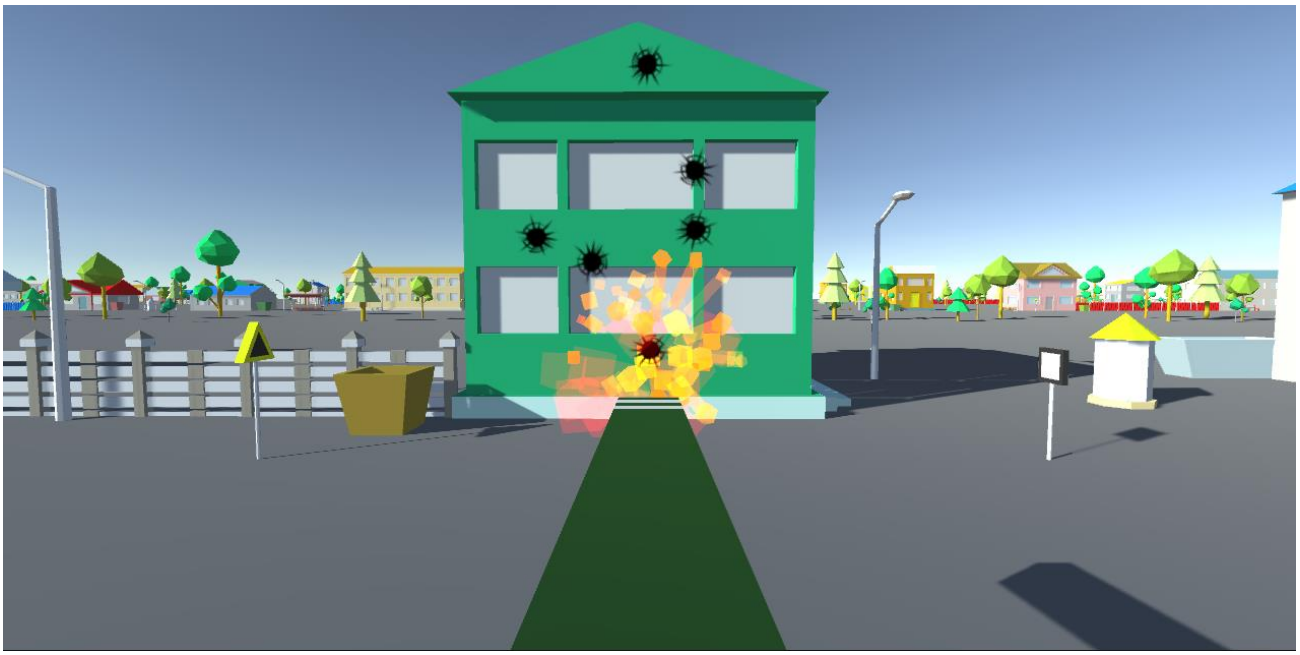

Рисунок 3.24 — Слід від попадання

З рисунків видно, що 3D об’єкти налаштовані належним чином та коректно відображаються і взаємодіють.

При попаданні в танк, він начинає диміти (рисунок 3.25).

Рисунок 3.25 — Попадання по танку гравця

З рисунку 3.25 видно, що фізичні моделі танків коректно реагують на виявлення зіткнення зі снарядом, також ці зіткнення правильно обробляються ігровими скріптами.

Під час тестування були протестовані різні аспекти гри. За результатами тестування доведено працездатність та коректність роботи гри.

ВИСНОВКИ

Під час виконання бакалаврської дипломної роботи було досліджено різні жанри комп’ютерних ігор. Був проведений огляд популярних комп’ютерних ігор, розроблених за допомогою Unity 3D.

Був проведений аналіз програмних засобів для розробки, а саме — ігрових рушіїв, IDE для написання коду, програм для 3D–моделювання. Згідно аналізу було підібрано найбільш оптимальне програмне забезпечення, яке підходило б під потреби гри.

Було проведено моделювання та оптимізацію 3D–моделей, налаштування їх в Unity 3D. Розроблена логіка управління танком, механіка стрільби, створені звукові та візуальні ефекти в Unity 3D. Також проведене тестування роботи комп’ютерної 3D–гри.

Розроблена гра, має цікавий геймплей та приємну графіку, також передбачена можливість імпорту гри на інші платформи. Таким чином мета бакалаврської дипломної роботи досягнута.

.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Прибутки ігрової індустрії в 2021 році [Електронний ресурс]. Режим доступу: https://gamedev.dou.ua/news/games-market-200-billion-revenues-in-2022/

2. Гулько О. С., Городецька О. С., Савицька Л. А. Вибір середовища розробки комп'ютерної 3D-гри, ВНТУ — 2022

3. Комп'ютерна гра [Електронний ресурс]. Режим доступу: https://esu.com.ua/search_articles.php?id=4393

4. Жанри комп'ютерних ігор [Електронний ресурс]. Режим доступу: https://sites.google.com/site/wwwwbossfighttvua/klasifikacia-igor/klasifikacia-za-zanrami

5. Хокінг Д. М. Unity в дії. Мультиплатформенна розробка на практиці. – 2018. – 131 с.

6. Kerbal Space Program [Електронний ресурс]. Режим доступу: https://store.privatedivision.com/game/buy-kerbal-space-program-ksp

7. The Forest [Електронний ресурс]. Режим доступу: https://the-forest.fandom.com/ru/wiki/The_Forest

8. Subnautica [Електронний ресурс]. Режим доступу: https://store.epicgames.com/ru/p/subnautica

9. The Long Dark [Електронний ресурс]. Режим доступу: https://store.steampowered.com/app/305620/The_Long_Dark/c

10. Гембейк Е. В. Ігрова індустрія. Створення ігор. / Е. В. Гембейк - М. : 3DNs, 2008. - 415 с.

11. Шеннон Т. А. Unreal Engine 4 для дизайну та візуалізації. – 2020. – 26с.

12. Андерсон К. С. Professional Visual Studio. – 2016. – 341 с.

13. Мейкісон Л. К. Моделювання - це легко. – 2013. – 313 с.

14. Cinema 4D [Електронний ресурс]. Режим доступу: https://www.maxon.net/ru/cinema-4d

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н.. Азаров О.Д.

“_29_” __квітня__2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської дипломної роботи

“Програмне забезпечення комп’ютерної 3D-гри в середовищі розробки Unity”

08-23.БДР.005.00.000 ТЗ

Науковий керівник: доцент к.т.н.

_____ Городецька О. С.

Студент групи 1КІ-18б

_____Гулько О. С.

м. Вінниця – 2022

1 Підстава для використання бакалаврської дипломної роботи (БДР)

1.1 Актуальність розробки полягає у високому попиту на ринку комп’ютерних 3D ігор.

1.2 Наказ про затвердження теми бакалаврської дипломної роботи.

2 Мета і призначення БДР

2.1 Мета проекту — розробка програмного забезпечення комп’ютерної 3D-гри в середовищі розробки Unity.

2.2 Призначення розробки — виконання бакалаврського дипломного проекту із подальшим впровадженням та розвитком продукту.

3 Вихідні дані для виконання БДР

3.1 Розгляд жанрів комп’ютерних 3D ігор.

3.2 Проведення аналізу та вибору програмних засобів.

3.3 Розробка комп’ютерної 3D-гри.

3.4 Тестування комп’ютерної 3D-гри.

4 Вимоги до виконання БДР

Головна вимога — розробити функціональну комп’ютерну 3D-гру за допомогою ігрового рушія Unity 3D.

5 Етапи БДР та очікувані результати

Усі етапи роботи та очікувані результати наведено в Таблиці 1.

6 Матеріали, що подаються до захисту БДР

До захисту БДР подається пояснювальна записка, ілюстративні матеріали, протокол попереднього захисту БДР на кафедрі, відгук наукового керівника,

анотації до БДР українською та іноземною мовами, довідка про відповідність оформлення БДР діючим вимогам.

Таблиця 1 — Етапи БДР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		Початок	Кінець	
1	Аналіз завдання. Вступ	17.05	19.05	Вступ
2	Огляд жанрів комп'ютерних ігор	19.05	21.05	Розділ 1
3	Вибір засобів розробки	21.05	26.05	Розділ 2
5	Розробка та тестування гри	26.05	10.06	Розділ 3
6	Оформлення пояснювальної записки	10.06	15.05	ПЗ, презентація

7 Порядок контролю виконання та захисту БДР

Виконання етапів графічної та розрахункової документації БДР контролюється науковим керівником згідно зі встановленими термінами. Захист БДР відбувається на засіданні екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БДР

При оформлюванні БДР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— документами на які посилаються у вище вказаних.

Технічне завдання до виконання отримав____ Гулько О. С.

ДОДАТОК Б

Лістинг скрипта TankControllerBase

```
using System;
using System.Collections;
using UnityEditor;
using UnityEngine;
public class TankControllerBase: MonoBehaviour
{
    [Header("Tank parts")]
    [SerializeField]
    private GameObject _tower    = null;
    [SerializeField]
    private GameObject _gun      = null;
    [Header("Bullet")]
    [SerializeField]
    private GameObject _bullet = null;
    [Header("Animation gun")]
    [SerializeField]
    private AnimationCurve _gunAnimationCurve = null;
    [Header("Sounds")]
    [SerializeField]
    private AudioSource _audioShoot = null;
    [SerializeField]
    private AudioSource _audioEngine = null;
    [SerializeField]
    private AudioSource _audioMove = null;
```

```
[Header("Particle Systems")]
[SerializeField]
private ParticleSystem _shootParticles = null;
[SerializeField]
private ParticleSystem _smokeParticles = null;
public  const float BULLET_SPEED  = 300f;
private const float MAX_SPEED     = 5f;
private const float ANGULAR_SPEED = 60f;
private const float MIN_ANGLE_GUN = -10f;
private const float MAX_ANGLE_GUN = 30f;
private float ANGLE_GUN = 0f;
private int _health = 100;
private Transform _placeSpawn = null;
private Vector3   _spawnBullet = Vector3.zero;
private Rigidbody _tankBody = null;
private delegate void EventDamage(int a);
private event EventDamage onDamage;
private void Awake()
{
    _tankBody = GetComponent<Rigidbody>();
    _tankBody.maxAngularVelocity = Mathf.Deg2Rad * ANGULAR_SPEED;
    _tankBody.centerOfMass = Vector3.zero;
    _placeSpawn = _gun.transform;
    _audioEngine.loop = true;
    _audioMove.loop = true;
    _audioEngine.Play();
    _audioMove.Play();
    onDamage += ControlSmokeParticles;
```

```
    }
    private void OnDisable()
    {
    onDamage -= ControlSmokeParticles;
    }
    private void ControlSmokeParticles(int health)
    {
        ParticleSystem.MainModule smokeParticlesMain = _smokeParticles.main;
        Color color = smokeParticlesMain.startColor.color;
        int alpha = Mathf.Clamp(30 + (100 - health), 30, 130);
        color.a = alpha / 255f;
        smokeParticlesMain.startColor = color;
    }
    protected void MoveTank( Vector3 direction, float MAX_SPEED =MAX_SPEED)
    {
        _tankBody.velocity = direction * MAX_SPEED;
    }
    protected void RotateTank( Vector3 direction )
    {
        _tankBody.AddTorque(direction * Time.deltaTime * 100,
ForceMode.VelocityChange);
    }
    protected void RotateTower(float sensitivity = 1f)
    {
        _tower.transform.Rotate(Vector3.up * sensitivity);
    }
    protected void RotateGun(float sensitivity = 1f)
    {
```

```
        ANGLE_GUN += sensitivity / 5;
        ANGLE_GUN = Mathf.Clamp(ANGLE_GUN, MIN_ANGLE_GUN, MAX_ANGLE_GUN);
        _gun.transform.localRotation = Quaternion.Euler(Vector3.left * ANGLE_GUN);
    }
    protected void Shoot()
    {
        StartCoroutine(ShootAnimation(_gun, _gunAnimationCurve, _tower));
        OnShoot();
        _spawnBullet = _placeSpawn.position + _placeSpawn.forward * 3;
        GameObject currentBullet = Instantiate(_bullet, _spawnBullet, _placeSpawn.rotation);
        currentBullet.GetComponent<Rigidbody>().velocity = currentBullet.transform.forward * BULLET_SPEED;
        IEnumerator ShootAnimation(GameObject gun, AnimationCurve animationCurve, GameObject parentObject)
        {
            Vector3 relativeVector;
            float time = 0;
            while (time <= animationCurve.keys[animationCurve.length - 1].time)
            {
                time += Time.deltaTime;
                relativeVector = parentObject.transform.position + parentObject.transform.forward * 35 / 20 + parentObject.transform.up * 5 / 20;
                gun.transform.position = relativeVector + gun.transform.forward * animationCurve.Evaluate(time);

                yield return null;
```

```
            }
        }
        void OnShoot()
        {
            _audioShoot.Play();
            _shootParticles.Play();
        }
    }
    protected void ControlMoveVolume()
    {
        float move_audio   = Mathf.Clamp(_tankBody.velocity.magnitude, 0, 0.5f);
        float rotate_audio = Mathf.Clamp(_tankBody.angularVelocity.magnitude, 0, 0.5f);
        _audioMove.volume = move_audio + rotate_audio;
    }
    public void ChangeHealth(int value)
    {
        _health += value;
        _health = Mathf.Clamp(_health, 0, 100);
        onDamage?.Invoke(_health);
    }
}
```

ДОДАТОК В

Лістинг скрипта Bullet

```
using System;
using UnityEngine;
public class Bullet : MonoBehaviour
{
    public delegate void EventHandler(ContactPoint point);
    public static event EventHandler onExplosion;
    public static event EventHandler onRicochet;
    private Rigidbody _rigidbody;
    private AudioSource _audioSource;
    private ContactPoint _point;
    private const int _explosive_speed = 50;
    private float _bullet_speed = TankControllerBase.BULLET_SPEED;
    private float _speed_after_collision;
    private float _speed_collision;
    private bool _explosion;
    private void Start()
    {
        _rigidbody   = GetComponent<Rigidbody>();
        _audioSource = GetComponent<AudioSource>();
    }
    private void OnCollisionEnter(Collision other)
    {
        int layer = other.gameObject.layer;
        if (isGround(layer))
        {
```

```
            Destroy(gameObject);
            return;
        }
        _speed_after_collision = _rigidbody.velocity.magnitude;
        _speed_collision = _bullet_speed - _speed_after_collision;
        _explosion = _speed_collision >= _explosive_speed;
        _point = other.GetContact(0);
        if (_explosion)
            Explosion(_point);
        else
            Ricochet(_point);
        ControlVolume(_speed_after_collision /TankControllerBase.BULLET_SPEED);
        _bullet_speed = _speed_after_collision;
    }
    private bool isGround(int layer) => layer == (int) Layers.Ground;
    private void Explosion(ContactPoint point)
    {
        onExplosion?.Invoke(point);
        Damage(point.otherCollider.transform);
        Destroy(gameObject);
    }
    private void Ricochet(ContactPoint point)
    {
        onRicochet?.Invoke(point);
    }
    private void ControlVolume(float volume)
    {
        _audioSource.volume = volume;
```

```
    }
    private void Damage(Transform transform)
    {
        do
        {
            transform = transform.parent ? transform.parent : transform;
        } while (transform.parent);
        TankControllerBase tankControllerBase =
transform.GetComponent<TankControllerBase>();
        if (tankControllerBase)
            tankControllerBase.ChangeHealth(-35);
    }
}
```

ДОДАТОК Г

Лістинг скрипта Cleaner

```
using System;
using System.Collections;
using UnityEngine;
public class Cleaner : MonoBehaviour
{
    [Header("Time to destroy object")]
    [Range(1,60)]
    [SerializeField]
    private float time = 1;
    [Header("Despawn Type")]
    [SerializeField]
    private DespawnType _despawnType = DespawnType.Destroy;
    private void Start()
    {
        if (_despawnType == DespawnType.Destroy)
            Destroy(gameObject, time);
        else if (_despawnType == DespawnType.DestroyAsSprite)
            StartCoroutine(DestroyAsSprite());
    }
    private IEnumerator DestroyAsSprite()
    {
        SpriteRenderer spriteRenderer = GetComponent<SpriteRenderer>();
        Color color = spriteRenderer.color;
        yield return new WaitForSeconds(time);
        while (color.a > 0)
```

```
        {
            color.a -= Time.deltaTime;
            spriteRenderer.color = color;
            yield return null;
        }
        Destroy(gameObject);
    }
}
public enum DespawnType: byte
{
    Destroy = 0,
    DestroyAsSprite
}
```

ДОДАТОК Д

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: _____

_____

_____

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 67,7% Схожість 32,3%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ _____
(підпис) (прізвище, ініціали)

Керівник роботи _____ _____
(підпис) (прізвище, ініціали)