МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2022 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

СТВОРЕННЯ 3D ГРИ З ВИКОРИСТАННЯМ АЛГОРИТМУ ПОШУКУ В СЕРЕДОВИЩІ UNITY

Спеціальність 122 «Комп'ютерні наук»

122 – БКР – 402.21810321

Виконав студент 4-го курсу, групи 402
_____Б. О. Супрун
«___» червня 2022 р.

Керівник: ст. викладач
_____С. Ю. Боровльова
«___» червня 2022 р.

Миколаїв – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«___»_____20__ р.

З А В Д А Н Н Я

на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук Супруну Богдану Олександровичу.

1. Тема кваліфікаційної роботи «Створення 3d гри з використанням алгоритму пошуку в середовищі Unity».

Керівник роботи Боровльова Світлана Юріївна, старший викладач.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «7» грудня 2021 р. №318

2. Строк представлення кваліфікаційної роботи студентом «___» _____ 2022 р.

3. Вхідні (початкові) дані до роботи: ринок ігрової індустрії, підходи до створення 3D відеогри.

Очікуваний результат: розроблена 3D комп'ютерна гра, шляхом використання середовища розробки Unity 3D.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- розкрити теоретичні засади створення 3D відеогри;

- проаналізувати та обґрунтувати вибір алгоритму для реалізації поставленого завдання у механіці гри;
- обґрунтувати вибір інструментальних засобів розробки відеогри;
- розробити та здійснити програмну реалізацію 3D відеогри;

5. Перелік графічних матеріалів: презентація, 51 рисунок, 21 посилання.

6. Завдання до спеціальної частини: «Охорона праці на робочих місцях у відділі розробки програмного забезпечення ЧНУ ім. Петра Могили».

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Макарова О. В., ст. викладач кафедри екології Медичного інституту ЧНУ імені Петра Могили	

Керівник роботи ст. викладача Боровльова С. Ю.
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання Супрун Б. О.
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання « 23 » листопада 2021 р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: «Створення 3d гри з використанням алгоритму пошуку в середовищі Unity».

№	Найменування роботи	Початок	Закінчення	Примітки
1	Складання календарного плану роботи на весь період виконання БКР	18.03.2022	18.03.2022	виконано
2	Аналіз існуючих аналогів	21.03.2022	25.03.2022	виконано
3	Написання вступу	05.04.2022	07.04.2022	виконано
4	Вибір засобів та технологій для розробки	08.04.2022	11.04.2022	виконано
5	Написання розділу 1	28.04.2022	08.05.2022	виконано
6	Аналіз алгоритмів для реалізації поставленої задачі	10.05.2022	13.05.2022	виконано
7	Розробка основних механік відеогри	14.05.2022	18.05.2022	виконано
8	Написання розділу 2	20.05.2022	25.05.2022	виконано
9	Завершення розробки	26.05.2022	05.06.2022	виконано
10	Написання розділу 3	03.06.2022	12.06.2022	виконано
11	Подання БКР рецензенту	16.06.2022	18.06.2022	виконано
12	Захист БКР перед ЕК	27.06.2022	27.06.2022	

Розробив студент Супрун Богдан Олександрович _____
(прізвище та ініціали) (*підпис*)

Керівник роботи ст. викладач Боровльова Світлана Юріївна _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«____» _____ 2022 р.

АНОТАЦІЯ

бакалаврської кваліфікаційної роботи студента групи 402 ЧНУ ім. Петра Могили

Супруна Богдана Олександровича

Тема: «Створення 3d гри з використанням алгоритму пошуку в середовищі Unity»

Кваліфікаційна робота присвячена розробці та здійсненню програмної реалізації 3D комп'ютерної гри в жанрі шутер.

Об'єкт дослідження – процеси розробки тривимірного ігрового застосунку у жанрі шутер для ПК.

Предмет дослідження – алгоритми та підходи до розробки комп'ютерних 3D ігор в середовищі Unity.

Метою дипломної роботи є популяризація відеогри серед непрофесійних гравців, шляхом створення проєкту в Unity з простим але інтенсивним геймплеєм та атмосферним сетінгом, що задовольнить потреби таких користувачів та буде позитивно впливати на їх психологічний стан.

У першому розділі розкрито історію формування відеоігор, жанрову їх класифікацію та описан аналіз існуючих алгоритмів пошуку. У другому розділі обґрунтовано вибір інструментальних засобів розробки 3D гри. Третій розділ містить опис розробки та здійснення програмної реалізації відеогри. У четвертому розділі розкрито питання спеціальної частини з охорони праці.

Кваліфікаційна робота містить 51 рисунок, 21 джерело, 1 додаток.

ABSTRACT

to the bachelor's qualification work by the student of group 402 of Petro Mohyla Black Sea National University

Suprun Bohdan Oleksandrovych

Topic: "Creation of a 3D game using the search algorithm in the Unity environment"

Qualification work is devoted to the development of software implementation of 3D computer game in the genre of shooter.

Object of research – processes of developing a 3D game application in the shooter genre for PC.

Subject of research – algorithms and approaches to the development of computer 3D games in the Unity environment.

The purpose – is promoting video games among non-professional players by creating a project in Unity with simple but intense gameplay and atmospheric setting that will meet the needs of such users and will positively affect their psychological state.

The first section reveals the history of video games, their genre classification and describes the analysis of existing search algorithms. The second section substantiates the choice of tools for developing 3D games. The third section describes the development and implementation of software implementation of the video game. The fourth chapter reveals the issue of a special part of labor protection.

Thesis contains 51 figures, 21 sources, 1 supplement.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

UI (англ. User Interface) – засіб взаємодії користувача з інформаційною системою

VGA (англ. Video Graphics Array) – компонентний відеоінтерфейс, що використовується в моніторах та відеоадаптерах

Геймплей – компонент гри, що відповідає за взаємодію гри та гравця

Колайдер – невидима спрощена форма об'єкта, для розрахунку зіткнень з іншими об'єктами

Моб – сленгова назва рухомого об'єкта, зазвичай монстра, в комп'ютерних іграх

ПЗ – програмне забезпечення

ПК – персональний комп'ютер

Рендеринг – термін у комп'ютерній графіці, що означає процес отримання 2D зображення за 3D моделлю

Сеттінг – сукупність різнопланових елементів, які однозначно ідентифікують світ, де відбуваються події гри.

Скайбокс – об'єкт у тривимірній графіці, що грає роль неба та горизонту

Пояснювальна записка

до кваліфікаційної роботи

на тему:

«СТВОРЕННЯ 3D ГРИ З ВИКОРИСТАННЯМ АЛГОРИТМУ ПОШУКУ В СЕРЕДОВИЩІ UNITY»

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402. 21810321

Виконав студент 4-го курсу, групи 402

Б. О. Супрун

(підпис, ініціали та прізвище)

«___»_____2022 р.

Керівник: *ст. викладач кафедри ІПЗ*

(наук. ступінь, вчене звання)

С. Ю. Боровльова

(підпис, ініціали та прізвище)

«___»_____2022 р.

Миколаїв – 2022

ВСТУП

У наш час майже кожен сучасний мобільний пристрій, що має не тільки молодь, або ж звичайний стаціонарний комп'ютер чи може то ергономічний ноутбук, може бути навіть планшет, мають поширені операційні системи, чи то Android чи то IOS, та операційні системи для ПК, що включають можливість встановлення різного виду програм. Та ці програми включають в себе той різновид відеоігор, які потребує користувач. Так, не можна виключати того моменту, шо відеоігри стали частиною майже усього людства. Популярність відеоігор збільшується з кожним днем, діти та дорослі проводять все більше часу за цим заняттям, та навіть деякі розробники використтовують ігроманію, в першу чергу, для навчання.

Нейрофізіологи та психологи з Китаю провели дослідження впливу захоплення комп'ютерними іграми на мозок. За словами нейрофізіолога з Університету електроніки та технологій Китаю Тецзюнь Лю, результати дослідження говорять про те, що професійні гравці можуть краще розподіляти та перемикати свою увагу, фокусуватися на важливіших речах та моментах, тому такі ігри цілком можна використовувати для тренування уваги.

Також треба звернути увагу на те, що відеоігри допомогають скоротати час, в першу чергу це відноситься до мобільних пристроїв. Грати в телефоні дуже зручно та корисно, наприклад, відсижуючи довгу чергу, хоча це вже питання, яке давно повинно вирішуватись електронними чергами, чи знаходячись у громадському транспорті, у час далекої поїздки. Людина у цих всіх ситуаціях отримує негативні емоції та витрачає нерви, а гра у, наприклад, аркаду буде її відволікати від неприємної ситуації. Ті ж самі аркади мають перевагу у підвищенні когнітивних здібностей. Ці ігри розвивають навики багатозадачності та прийняття рішень. У грі людина має за секунди прийняти

рішення, від якого залежить результат, таким чином це розвиває рівень здібності прийняття рішень за велику швидкість та без якої шкоди.

Об'єкт дослідження – процеси розробки тривимірного ігрового застосунку у жанрі шутер для ПК.

Предмет дослідження – алгоритми та підходи до розробки комп'ютерних 3D ігор в середовищі Unity.

Метою є популяризація відеогри серед непрофесійних гравців, шляхом створення проєкту в Unity з простим але інтенсивним геймплеєм та атмосферним сеттінгом, що задовольнить потреби таких користувачів та буде позитивно впливати на їх психологічний стан.

Завдання для досягнення поставленої мети:

- аналіз сучасного стану в ігровій індустрії;
- аналіз та обґрунтування вибору інструментальних засобів для розробки 3D проєкту;
- з'ясування аспектів розробки 3D гри за допомогою двигуна;
- постановка ідеї відеогри;
- розробка та програмна реалізацій 3D гри.

1 АНАЛІЗ СФЕРИ ІГРОВОЇ ІНДУСТРІЇ

1.1 Поняття та розвиток

Відеогра – це гра з використанням зображення на дисплеї, що попередньо згенеровано цифровим пристроєм. Іншими словами, можна сказати, що відеогра являється електронною грою, що влаштована на взаємодії людини та пристрою через візуальний інтерфейс на екрані та вводом інформації через контролери.

Перші комп’ютерні ігри почали пробувати розробляти у 1950х роках. На початку сучасної комп’ютерної ігрової індустрії строять три особистості. Ральф Баєр запропонував ідею інтерактивного телевізійного каналу. У 1952 році А. С. Дуглас створив гру з назвою «OXO» (див. рис. 1.1), що є комп’ютерною інтерпретацією класичної гри «Хрестики-нулики». Третю людину вважають батьком комп’ютерних ігор – це Вільям Хігінботем. Він написав гру «Теніс» у 1958 році. В цю гру могло грати одразу дві людини.

Рисунок 1.1 – Гра "OXO"

У 1960 році було випущено перший комп'ютер серії PDP від компанії Digital Equipment, який отримав назву PDP-1. PDP являлася серією міні-комп'ютерів. Через два роки після випуску на нього було розроблено гру, якій розробник дав назву Space War.

У наш час не можливо назвати себе повноцінним комп'ютерним гравцем без наявності комп'ютерної миші в своєму арсеналі маніпуляторів. Як раз у 1970 році було видано патент на комп'ютерну мишу Дугласу Енгельбарту. До 1975 року починає проявлятись активний інтерес до комп'ютерних ігор. Вже починають з'являтись ігри не тільки казуальних жанрів. На світ з'являється гра, що є прообразом пригодницького жанру. Гра отримує назву Colossal Cave Adventure (див. рис. 1.2), її розробником є Вільям Кроутер. Ця гра з великою швидкістю розповсюджується через мережу ArpaNET. Починаючи з 1977 року, розробники активно починають створювати все більше нових комп'ютерних ігор. Вони згодом стануть поштовхом, який прискорить розвиток персональних комп'ютерів.

Рисунок 1.2 – Інтерфейс гри Colossal Cave Adventure

У 1981 році компанія IBM випускає «IBM PC», який став першим персональним комп'ютером. Завдяки цьому ринок комп'ютерних ігор став швидко зростати. Як раз у 1980-ті роки можна сказати настала епоха комп'ютерних ігор. Японська компанія Namco випускає гру під назвою «Pac-Man» (див. рис. 1.3), суть якої полягала в управлінні вигаданою істотою, яка збирає точки по лабіринту та ухиляється від ворогів. Поява відеоадаптерів VGA та SVGA у 1987 році відправили в минуле шістнадцяти-кольорове зображення, та на моніторах з'явилось аж 256 кольорів, що зробило відеоігри більш насиченими.

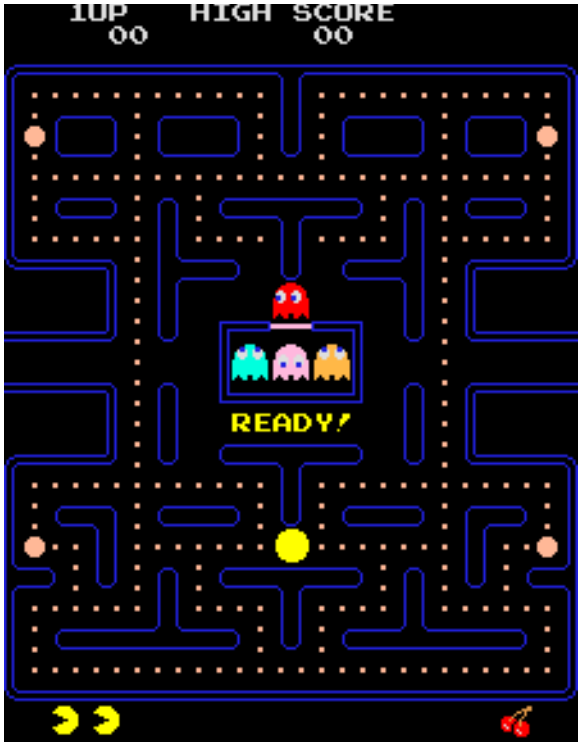

Рисунок 1.3 – Перший інтерфейс відеогри Pac-man

У цей час розвиток комп'ютерних ігор ніяк не може падати, він тільки все більше набирає обертів, та ПК стають швидше та сучасніше. 10 грудня 1993 року являється історичною датою. В цей день компанія Id Software знайомить

зі світом гру Doom, яка стала не тільки основою жанру шутер, а й однією з перших відеоігор з імітацією трьох-вимірного простору (див. рис. 1.4).

Рисунок 1.4 – Геймплей гри Doom

Наступною значною датою йде вихід першої гри з мультиплеєром – Rise of Triad у 1994 році. Перший шутер зі справжньою тривимірною графікою світу та ворогів виходить на наступний рік та під назвою The Terminator: Future Shock.

Але повноцінний та оптимізований рендеринг 3D графіки стає доступний з появою відеокарти Voodoo I у 1996 році. Слід за її появою випустили такі ігри як Duke Nukem 3D та Quake. У цьому ж році з'явилися такі ігри, як Super Mario, Command & Conquer; Red Alert, Tomb Raider, Resident Evil, Diablo тощо.

1998 рік – час появи на світ ігор Half-life і StarCraft, серії продовження яких не втрачають популярності й досі, хоча те й саме можна сказати про перші частини, які стали для багатьох гравців класикою. Така ситуація характерна для багатьох хітів минулого.

Рисунок 1.5 – Скриншот геймплею гри Half-Life

Великими кроками ігрова індустрія просувається вперед у першому десятиріччі XXI століття. Кожен рік розробники випускають тисячі відеоігор, що розлітаються по всьому світу мільйонами копій, та окрім цього копіюють та випускають диски великими кількостями так звані пірати. Такі відомі ігри як Call of Duty, Battlefield збирають сотні мільйонів з продажу, індустрія набирає нових обертів.

Все частіше останнім часом говорять про те, що розвиток комп'ютерних ігор сповільнився, або ж навіть досяг своєї межі. Наче усі можливі технології вже існують: цікаві маніпулятори, широкомасштабний кооператив, деталізована 3D графіка. І саме сьогодні може звучати питання – що далі? Деякі ігри не мають нічого нового, використовуються старі шаблони.

Комп'ютерні ігри – це така ж сама творчість, як кіно чи написання музики, не може бути меж для творчості, люди, які вважають що відеоігри зайшли в глухий кут, дуже помиляються. Навіть технології та можливості не вичерпано, ми не можемо передбачити, як може стрибнути цифровий прогрес

через пару років, чи може ми повернемося до удосконаленої аналогової техніки. Хто у 90х роках міг знати, що кожний другий матиме можливість побавитись в повноцінну комп'ютерну гру на своєму смартфоні. Декілька років назад навіть не представляли, що у світ з'явиться лінійка відеокарт з підтримкою алгоритму RTX, що є природньою генерацією тіней, освітлення та текстур, який раніше вважався неоптимізованим та грубим алгоритмом, але обчислювальні потужності змінилися. Та чи можна вважати питання «що далі?» доцільним?

1.2 Класифікація відеоігор

Тема класифікації відеоігор являється дуже великою та розглядається у низці досліджень культурологів, філософів. Нижче буде наведено приклад класифікації та представлено традиційну класифікацію відеоігор за жанрами.

Кожен автор пропонує свою класифікацію, що ділить їх за різними параметрами.

Наприклад, у роботі Сибірякова «Характери та жанри відеоекранних ігор» автор пропонує використовувати характерологічну класифікацію ігор, що ділить їх за «характерами»: активність, пошук, імітація, планування, ідентифікація, логіка. Автор підкреслює, що ці характери цілком співвідносні з традиційними жанровими іменами (див. табл. 1.1).

Таблиця 1.1 – Характеристика відеоігор за критерієм Сибірякова

Характер	Опис
Активність	Характер із вимогами до швидкості та точності реакції
Пошук	Характер із вивченням ігрового простору
Імітація	Характер з інтерактивною симуляцією реальності

Продовження таблиці 1.1

Планування	Характер із вимогами до попередньої розробки плану розвитку
Ідентифікація	Характер із ототожненням ігрового персонажа чи деякого об'єкта
Логіка	Характер із завданнями, аналогічними до шахових, карткових

Жанрова класифікація активно почала розвиватись у 90-х роках. Основою для цієї жанрової класифікації стали випуски серій ігор. Ці серії створились таким чином, що після виходу оригінальної гри як офіційні, так і сторонні розробники створювали інші ігри шляхом експлуатації рис початкової роботи, тобто створення так званих клонів. «У табл. 1.2» буде наведено жанровий розподіл відеоігор.

Таблиця 1.2 – Жанрова характеристика відеоігор

Жанр	Синоніми	Опис
Бойовик	Action, Екшен	Вимагають хорошої моторики
Симулятор	Simulation, Симуляція	Імітує певні напрямки діяльності
Стратегія	Strategy	З упором управління ресурсами
Рольова гра	Role-playing Games, RPG	З розвиненою системою зміни персонажів
Пригодницька гра	Adventure	Що передбачають подолання різних перешкод на шляху персонажа

Продовження таблиці 1.2

Головоломка	Puzzle	Основана на вирішенні логічних завдань, побудованих на загальному наборі правил

Також відеоігри можуть поділяти за кількістю гравців:

- розрахована на багато гравців;
- розрахована на одного користувача.

Ще їх часто поділяють за візуальним уявленням:

- 2D відеоігри – у яких використовується плоска графіка, що називається спрайтами, яка не має трьохмірної геометрії;
- 3D відеоігри - зазвичай використовується трьохмірний простір, де матеріали та текстури відрисовуються на поверхні ігрових об’єктів, формується ціле оточення, персонажі та об’єкти ігрового світу.

Також можна відзначити, що комбінації жанрів формують окремі змішані жанри, які можна вважати повноцінними окремими жанрами, наприклад Action RPG. Ще мають мають місце у списку піджанрів жанри, відносно нові, які формуються ігровою аудиторією та отримують назву виходячи зі скороченої назви гри, яку було покладено у розвиток цього жанру та постфікс «-like». Наприклад, існує жанр Soulslike, що зародився із серії ігор Dark Souls (див. рис. 1.6), основними характеристками якого є сеттінг темного фентезі, та стосовно геймплея, маємо систему босів і дуже високий рівень складності.

Рисунок 1.6 – Скриншот геймплею гри Dark Souls

1.2.1 Action

Екшен (action у перекладі з англ. – «дія») або бойовик (за аналогією з кіножанром) — жанр комп'ютерних ігор, у якому йде наголос на фізичні можливості гравця, у тому числі координації очей і рук, швидкості реакції. Жанр представлений у безлічі різновидів, тобто має піджанри:

- shoot’em up;
- arena shooter;
- scroll shooter;
- platformer;
- fighting;
- 3D shooter;
- quick time;
- rhythm games.

Яскравим прикладом жанру Action може стати широковідома відеогра-шутер Counter-Strike: Global Offensive, що зображена «на рис. 1.7», випуск якої був призначений 21 серпня 2012 року компанією Valve, що випустили, можна сказати, продовження легендарної класичної серії комп'ютерних ігор Counter-Strike. Гра, як і її попередники являється трохмірним мультиплеєрним шутером вигляду від першої особи, в якій гравці однієї локації діляться на дві команди та борються між собою у різних режимах. Має реалістичний сеттінг спецназу та терористів.

Рисунок 1.7 – Скриншот геймплею гри CS:GO

Різниця сучасної відеогри від її попередників полягає у достатньо покращеній графіці, тобто хоч і попередня класична версія CS 1.6 була стандартом серед гравців та кіберспортсменів, реалії та розвиток технологій вимагали нової версії гри більш сучасної розробки, потрібно вже було відмовлятись від «піксельної» графіки.

1.2.2 Simulator

Симулятор, можна навіть сказати імітатор, його задача в тому, щоб імітувати управління якимось процесом, апаратом або ж транспортним засобом. За допомогою комп'ютерно-механічних симуляторів, що точно відтворюють інтер'єр кабіни апарату, тренуються пілоти, космонавти, машиністи високошвидкісних поїздів, та як же без ігроманів, які хочуть себе відчути кимось із них та не мають можливості а мають під рукою комп’ютер. Існують і цього жарну піджанри, але їх появлення виходить з того, імітацію чого відтворено, чи то авіасимулятори чи то автосимулятори, перелік може бути довгим.

Серед даного жанру яскравим прикладом може стати серія відеоігор Ship Simulator від нідерландської компанії VSTEP (див. рис. 1.8).

Рисунок 1.8 – Скриншот геймлею Ship Simulator 2006

Відеогра у повній мірі може продемонструвати жанр симулятору. Геймлей представляє собою виконання різного виду завдань від

перешвартування невеликого судна до перетину Тихого Океану на пасажирському лайнері. Наведена гра, як і більша кількість симуляторів, має повну деталізацію кабін чи кают з усіма її елементами управління, що не просто текстури, а інтерактивні важелі. Також вона славиться своєю різноманітністю та деталізацією суден, наприклад судно Titanic гри версії 2006 року, по якому гравець в повній мірі може прогулятися та завітати в деякі зали чи каюти.

1.2.3 Adventure

Пригодницька гра являє собою інтерактивну історію з головним героєм, яким керує гравець. Важливими елементами гри є оповідь та дослідження ігрового світу та, у процесі, вирішення головоломок та задач, які вимагають розумових зусиль від користувача. Має основні піджанри:

- interactive fiction;
- візуальний роман;

  графічний квест.

1.2.4 Strategy

Стратегія являється жанром відеоігор, у якому , як би то не було очевидно, гравцю потрібно застосовувати стратегічне мислення. Основа таких ігор в тому, що гравцю не потрібно керувати одним якимось персонажем, а їх великою кількістю, так званими масами.

Класифікація:

- стратегії реального часу (RTS, от англ. Real Time Strategy);
- покрокові стратегії (TBS, от англ. Turn Based Strategy);
- тактичні стратегії;
- симулятори будівництва та управління.

Даний жанр має достатню кількість якісних представників, але можна зупинитись на серії відеоігор RTS - Warcraft. Третю частину якої зображено «на рис. 1.9». Ігри даної серії також мають елементи RPG, так як персонажі мають деякі власні здібності та характеристики, іноді потрібно керувати одним головних персонажем.

Рисунок 1.9 – Інтерфейс Warcraft III

Сутність гри полягає у проходженні деякого сюжету, та при цьому гравець починає керувати військом, що постійно росте чи несе втрати, яке повинно відбудовувати поселення, добувати ресурси чи вбивати мобів для нагород. Гра відзначається динамічним та цікавим сюжетом, який проявляється шляхом ігрових постановок, моделей персонажів, та їх великою кількістю реплік.

1.2.5 Puzzle

Головоломка – жанр відеоігор, основою якого є вирішення відповідних логічних задач, що потребують від користувача таких навиків, як логіка, навики стратегії та інтуїції.

Якщо говорити в контексті відеоігор, чому б нам не згадати гру, яку не може не знати будь-який користувач цифровим пристроєм. Мова йде про широковідому відеогру Тетріс, що була розроблена радянським вченим Олександром Пажитновим ще у 1985 році (див. рис. 1.10).

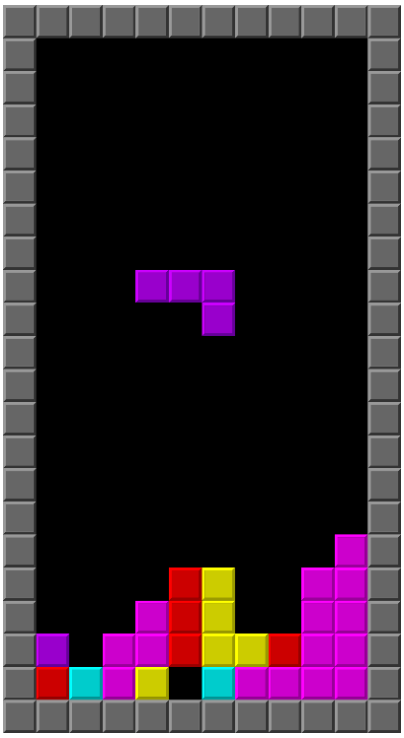

Рисунок 1.10 – Проста інтерпретація гри Тетріс

Тетріс являється головоломкою, у якій потрібно використовувати геометричні фігури різних форм, основа яких – вони складаються з чотирьох квадратів («тетраміно»). Випадкові тетраміно падають з верхньої частини ігрового інтерфейсу, гравець повинен перевертати їх у польоті таким чином

щоб задовольняти правило зникнення квадратів, що опинилися унизу, тобто які відтворили суцільну горизонтальну лінію. Швидкість постійно зростає, гравець програє коли фігури наскладаються до стелі. Тетріс також є прикладом потенційно безкінечної гри, у якій користувач тільки отримує очки, або може позмагатись з іншим.

1.2.6 RPG

Ролева гра являється високорозвиненим та популярним жанром, який також є основою великої кількості змішаних жанрів. Жанр оснований на елементах гри традиційних настільних рольових ігор. В основу покладено управління одним або декількома персонажами, де кожен має свої різні характеристики, здібності, зброю та власні очки здоров'я – HP, та часто очки мани. Характеристиками традиційно у багатьох іграх виступають сила («strenght»), спритність («agility»), інтелект («intelligence»). У рольових іграх ухил йде на великий ігровий світ та сюжет, що виражений у системі квестів, хоча це мала частина від створеного різноманіття у RPG.

Основні піджанри:

- narrative RPG;
- sandbox RPG;
- dungeon crawler.

Серію ігор Diablo можна виставити як представника піджанру dungeon crawler (див. рис. 1.11). Ця серія, також як серія Warcraft, розробляється компанією Blizzard Entertainment. На даний час серія містить 3 частини основних ігор, кожна с яких має свої доповнення з рівнями чи можливостями.

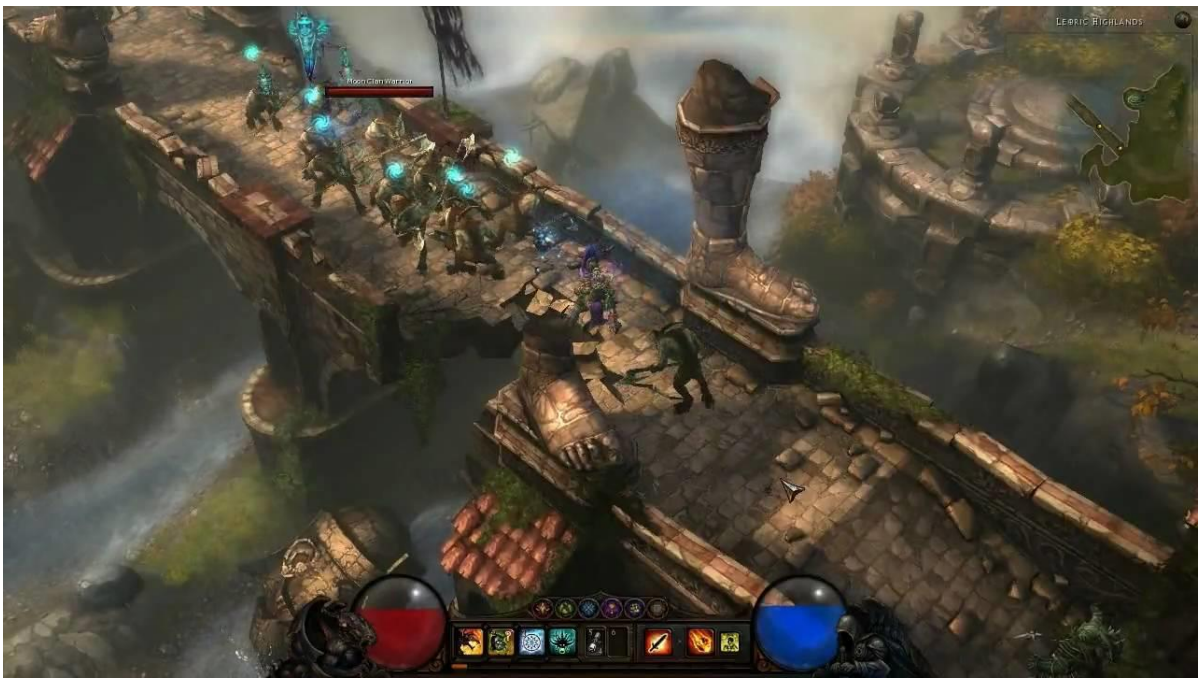

Рисунок 1.11 – Скриншот геймплею Diablo III

Дія даних ігор відбувається у вигаданому світі Санктуарій, його сеттінгу притаманний стиль дарк фентезі. Світ нагадує середньовікову Європу, також персонажу доводиться відвідати населене демонами Палаюче Пекло. Гра має свій продуманий сюжет, коротко: персонаж повинен вбивати демонів та чудовиськ, що походять від головного демона Діабло, а потім розправитися і з ним.

1.2.7 Arcade

Цей термін є поширеним та популярним в індустрії відеоігор, він означає клас відеоігор, що мають навмисно примітивний геймплей. Термін з’явився завдяки іграм для ігрових аркадних автоматів. Багато користувачів та ресурсів про комп’ютерні ігри виділяють цей термін як окремий жанр відеоігор, та в цьому є правда, але спочатку цей термін був відомий як напрям відеоігор, які

характеризувались схожістю по концепції з іграми для автоматів. До них можна і віднести приклади проєктів жанрів “гонки”, “файтинг” тощо.

Як сформований жанр, аркади можна охарактеризувати коротким по часу та простим, але інтенсивним геймплеєм. Розроблена відомою студією Blizzard Entertainment.

В першу чергу користувач може швидко навчитись грати у проєкт даного жанру, та йому притамманий простий ігровий процес. Майже всі такі відеоігри мають ігровий рахунок, гравець отримує очки за виконання навіть простих задач. Таким чином формується принцип гри, при якому гравець починає змагатись сам з собою та ставити локальні рекорди, або й навіть при глобальній базі гравці можуть змагатись онлайн у таблиці рекордів. Також важливою характеристикою деяких ігор вважається безкінечна гра. Користувач потенційно може грати безкінечно та не може перемогти. Щоб гра була не фактично безкінечною а тільки потенційно, створена логіка, при якій з часом звичайно збільшується складність гри, таким чином при безкінечній грі формується безкінечна складність.

На прикладі існує доволі популярний жанр для гравців на смартфонах – безкінечні ранери. Це раніше описана безкінечна гра, але якій притаманний рух уперед через перешкоди, чи то бігом чи на автомобілі, або ж на човні. Ціль одна – дістатись якомога далі, тим самим отримати відповідну кількість очків.

Рисунок 1.12 – Скриншот геймплею Subway Surfers

Subway Surfers – відносно стара та дуже популярна гру у будь-які часи (див. рис. 1.12). Думаю буде дуже незручно, якщо зіставляти список найкращих безкінечних ранерів, та не включити у нього цю гру. В даній користувач має управляти персонажем, що тікає від охоронця та має можливість наткнутись на різного виду перешкоди та програти. Персонаж доведеться бігати по потягам, метро, по залізницям та навіть по лініям електромережі. Гра не потребує яких-небудь зусиль для старту, але все далі потрібно проявляти високу реакцію та за короткий час приймати правильні рішення, від яких залежить майбутнє на декілька кроків уперед.

1.3 Алгоритм пошуку

У багатьох відеоіграх постає задача вирішення шляху з однієї точки в іншу. Ця проблема часто виникає у стратегіях реального часу, в яких гравець задає юнітам рухатись через ігровий рівень через перешкоди та нерівності, також проблема постає у іграх з мобами, що повинні при певній ситуації

пройти до кінцевої точки, що була запланована у ігровому процесі, знову ж таки через перешкоди та нерівності.

Дану проблему класифікують, як задача про найкоротший шлях, яка є частиною задач з теорії графів, що полягає у знаходженні найоптимальнішого шляху, тобто сума ваг ребр, з яких складається цей шлях є мінімальною.

Так, як існує велика кількість постановок даної задачі, є достатня кількість алгоритмів для вирішення задачі пошуку шляху, наведемо найпопулярніші:

- алгоритм Дейкстри;
- алгоритм Беллмана – Форда;
- алгоритм A*;
- алгоритм Флойда — Уоршелла;
- алгоритм Джонсона;
- алгоритм Лі.

1.3.1 Алгоритм Дейкстри

Алгоритм з теорії графів, що розробив Нідерландський вчений Едсгер Дейкстра у 1959 році. Суть полягає у тому, що він знаходить найкорочші шляхи від однієї вершини графа до усіх інших. Даний алгоритм дуже широко використовується в технологіях та програмуванні. Наприклад, даний алгорит використовують у протоколах маршрутизації OSPF та IS-IS. Головний недолік представленого алгоритму у тому, що він не працює для графа з від'ємними значеннями ребер графів.

Коротко про алгоритм: кожній вершині зіставимо мітку - відома мінімальна відстань від цієї вершини до початкової вершини. Алгоритм працює покроково - на кожному кроці він "відвідує" одну вершину і

намагається зменшувати мітки. Робота алгоритму завершується коли всі вершини відвідані.

1.3.2 Алгоритм A*

Алгоритм пошуку A* – у інформатиці алгоритм, що знаходить найоптимальніший маршрут, тобто маршрут з найменшим значенням суми ваг ребер, від початкової до кінцевої точки.

В онові лежить порядок обходу вершин, що визначається евристичною функцією. Ця функція собою становить суму двох інших функцій. Це звичайно вартість досягнення поточної вершини, та так звана функція евристичної оцінки відстані від поточної вершини до кінцевої. Часто в прикладних задачах функцією евристичної оцінки є відстань від поточної точки в алгоритмі до кінцевої по умовній прямій лінії.

Цей алгоритм по суті можна вважати розширенням класичного алгоритму Дейкстри, що отримав, можна сказати, апгрейд з продуктивності завдяки евристиці.

Очевидно, алгоритм A* покроково передивляється усі шляхи, що ведуть до кінцевої точки, тільки у пріоритеті ті шляхи, що можна сказати «здається» більше ведуть до кінцевої точки, тобто використання евристичної оцінки.

1.4 Двигуни для відеоігор

Серед двигунів у ринку інді-ігор явно домінують два найпопулярніших двигуна для розробки відеоігор – це Unity та Unreal Engine 4. Чому серед інді-ігор, достатньо розібратись у понятті, шо значить комп’ютерну гру, що створена окремим розробником чи невеликою групою без фінансової підтримки, тобто треба ураховувати, що двигун має задовольняти потреби одиночної розробки а не корпоративної. Їх переваги серед інших двигунів

досить очевидні, вони у комплексі мають майже усе, чого потребує розробник, у той час, як у інших двигунів чи недостатня кількість потрібних технологій, чи технології, що розроблені під окремі задачі. Приклад рендерингу зображено «на рис. 1.13».

Щодо Unity, можна сказати, що цей продукт відкрив завісу перед повноцінною безкоштовною середою розробки для багатьох потенційних розробників відеогігор. На той час, коли двигуни мали обмежені можливості та високу вартість, Unity зломив страх багатьох бажаючих зайнятися розробкою відеоігор.

Але наразі ми маємо найбільшого конкурента Unity – Unreal Engine 4, який перетворився у ще одну безкоштовну та зручну середу розробки комп’ютерних ігор.

Обидва двигуна мають схожий набір інструментів розробки: декоратор ландшафту, інструменти симуляції фізики, робота з анімаціями, редагування шейдерів, коригування освітлення та ін.

Гра у Unity будується на основі редактора та скриптових файлів, написаних на мові програмування C#. У той час, як Unreal Engine 4 використовує за основу високооптимізований C++. Хоча, важливо відмітити, що C++ становить основу двигуна, та користувач має справу далеко не з класичною мовою програмування, а майже новою скриптовою мовою, задля створення якої, розробники постаралися, щоб мати об’єктноорієнтовану скриптовану мову програмування зі швидким ядром.

Рисунок 1.13 – Рендер лісу у Unreal Engine 4

Хоча в теперішній час Unity досягли рівня графіки, як Unreal Engine 4, але останній надає більшого результату при менших зусиллях, у той час в Unity потрібно добре постаратись, щоб отримати схожий результат, Потрібно відмітити що велика частина розробки відходить на технологію BluePrints, що нібито і є цією простотою, але все одно без будь-яких знань не можливо що-небудь створити.

Висновки до першого розділу

Спираючись на дану класифікацію відеоігор було прийнято рішення розробити 3D гру жанру шутер та з напрямком безкінечної гри. Ця гра повинна мати для звичайного користувача простий інтерфейс та високу інтенсивність з ростом часу, також ефектності додають елементи шутера. Було вирішено розробляти так званих мобів, що потребують штучного інтелекту, але для простої гри достатньо звичайних алгоритмів для вирішення конкретних задач.

Для пошуку шляху до гравця мобами було проаналізовано алгоритми пошуку шляху теорії графів та обрано оптимальний алгоритм A*, що явно підходить для заданих цілей. Вибір алгоритму обґрунтовується швидкістю, що у наш час є одним з найважливіших аспектів розробки гри попри кількість пам’яті, що використовує застосунок, яка не є дуже великою проблемою у порівнянні з кількістю часу, використаного на рендеринг одного кадру. Швидкість алгоритму обумовлюється присутністю евристичних функцій, які можна використовувати при конкретних задачах, як у нашому випадку ми маємо 2D мапу деякого розміру клітинок, відстань між якими по діагоналі може дати змогу знайти більш вірогідні напрямки шляху.

Щодо двигуна для розробки відеогри, то вибір, хоч і з перевагами Unreal Engine 4, впав на розробку у середовищі Unity. Для невеликого проєкту цього двигуна достатньо стосовно графіки, та розробка на мові C# у цьому середовищі декілка простіша, є зрозуміла документація та велика кількість ресурсів для навчання. Але, при розробці великого 3D проєкту з ціллю на високу якість графіки, вибір, очевидно, впав би на Unreal Engine 4.

2 ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РОЗРОБКИ КОМП'ЮТЕРНОЇ ГРИ

2.1 Платформа .NET та мова програмування C#

.NET – це платформа від Microsoft, або ж програмна технологія, що є основою для створення як звичайних програм, так і веб-застосунків. Випуск першої версії .NET Framework відбувся у 2002 році. Вважається, що .NET Framework було створено як альтернативу платформі Java від компанії Sun. Головна відмінність у тому, що .NET Framework офіційно розрахована працювати саме з операційними системами сімейства Microsoft Windows, також однією з ідей .NET є сумісність служб, написаних різними мовами. З того часу технологія застала великої кількості змін та покращень, пройшла шлях від від версії 1.0 до 4.8. Хоча, на сьогоднішній день, передовою технологією нового покоління від Microsoft являється кросплатформна платформа .NET Core, технологія .NET Framework ніяк не втратила популярності серед розробників. Існує безліч ПЗ, бібліотек та фреймворків, які написані та розвиваються під .NET Framework.

Рисунок 2.1 – Логотип .NET

Було згадано про відносно молоду платформу .NET Core, яку випущено було у 2016 році на додаток до .NET Framework, та указано, що вона кросплатформна, тобто вона сумісна з різними ОС. .NET Core відкрила нові сценарії та можливості застосування. Це значно вплинуло на просування .NET серед розробників.

C# – це об'єктно-орієнтована мова програмування, що достатньо перейняла у Java та C++. C# вже також не молода мова, та пройшла такий самий шлях що й .NET. У наш час вважається дуже потужною мовою, популярною в IT, зі швидким розвитком. Наразі на цій мові також завдяки .NET пишуть різного виду проєкти: від невеликих застосунків для ПК до великих веб-порталів чи веб-сервісів, якими користуються мільйони користувачів щодня.

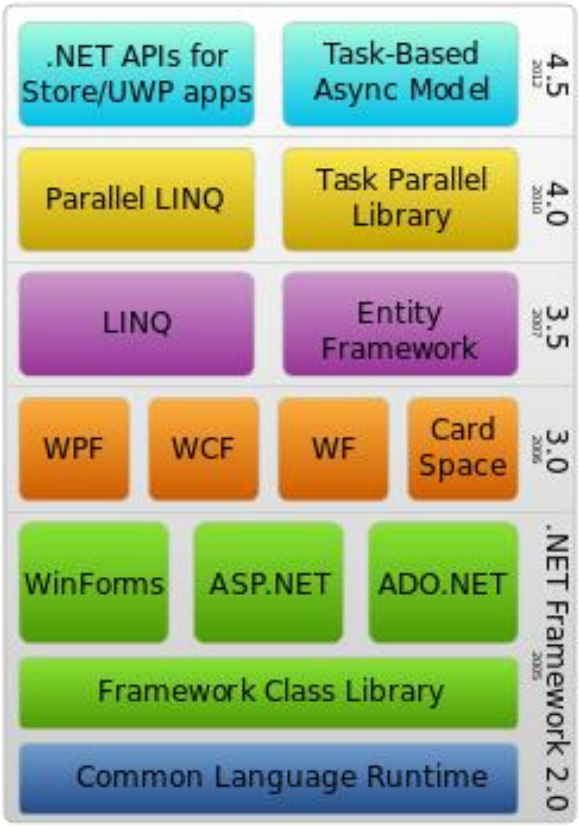

Рисунок 2.2 – Стек технологій .NET Framework

Мова C# підтримує поліморфізм, наслідування, перевантажені оператори та статичну типізацію. Об'єктно-орієнтованість дозволяє вирішувати завдання з побудови великих, але в той же час гнучких, масштабованих додатків, які з часом розширюються. C# продовжує активно розвиватися і з кожною новою версією з'являється все більше цікавих функціональностей.

Як і технологія Java, середовище розробки .NET створює байт-код, що призначений для виконання VM (Virtual Machine). Вхідна мова цієї машини в .NET називається CIL (Common Intermediate Language), також вона відома як MSIL (Microsoft Intermediate Language). Застосування байт-коду дозволяє отримати крос-платформність на рівні скомпільованого проєкту (в термінах .NET: збірка), а не на рівні серцевого тексту, як, наприклад, в C. Перед запуском збірки в середовищі виконання (CLR) байт-код перетворюється вбудованим в середовище JIT-компілятором (just in time) в машинні коди цільового процесора.

Коли ми говоримо про C# також мають на увазі технології .NET, так само навпаки, ці поняття зв'язані та не можна цього заперечувати.

Основні риси .NET.

1) **Підтримка кількох мов**. Основою платформи є загальномовне середовище виконання Common Language Runtime (CLR), завдяки чому .NET підтримує декілька мов програмування: поряд із C# стоять VB.NET, C++, F#, а також діалекти інших мов, прив'язані до .NET, наприклад, Delphi. NET. При компіляції код, будь-якою з цих мов компілюється у код мови CIL (Common Intermediate Language) – свого роду асемблер платформи .NET. Тому за певних умов можливо зробити окремі модулі однієї програми різними мовами програмування.

3) **Кросплатформність.** .NET є платформою, що переноситься. Наприклад, остання версія платформи на даний момент – .NET 6 підтримується на більшості сучасних Linux, MacOS та ОС Windows. Використовуючи різні технології платформи .NET, можна розробляти програми на мові C# для різних платформ – Linux, Android, Windows, MacOS, iOS, Tizen.

3) **Потужна бібліотека класів.** .NET представляє єдину для всіх підтримуваних мов бібліотеку класів. І яку б програму розробник не збирався написани на мові C# – текстовий редактор, месенджер або складний веб-сайт – так чи інакше використовується бібліотека класів .NET.

4) **Різноманітність технологій.** Середовище виконання CLR та базова бібліотека класів є основою цілого стеку теханологій, які розробники можуть задіяти при побудові тих чи інших додатків (див. рис. 2.2). Наприклад, для роботи з БД у цьому стеку технологій призначено технологію ADO.NET та Entity Framework Core. Також для побудови графічних програм з багатим насиченим та грамотним інтерфейсом – технологія WPF і WinUI, для створення більш простих програм з UI – Windows Forms. Для розробки кросплатформних мобільних застосунків – Xamarin. Для створення веб-сайтів – ASP.NET. Ще варто згадати активний Blazor – фреймворк, що розвивається і набирає популяність, який працює поверх .NET. Він дозволяє створювати застосунки як на стороні сервера, так і на стороні клієнта.

5) **Продуктивність.** Відповідно до тестів веб-програм на .NET 6 сильно випереджають веб-програми, побудовані за допомогою інших технологій. Програми на .NET у принципі відрізняються високою продуктивністю.

2.2 Середовище розробки Visual Studio

Microsoft Visual Studio – лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення та інші інструменти.

Інтегроване середовище розробки (IDE) – це багатофункціональне ПЗ (програмне забезпечення), що використовується саме для масштабної розробки ПЗ.

Visual Studio не тільки має у свому арсеналі стандартний редактор коду та відладчик, які мають більшості IDE, а й спрощує процес розробки ПЗ за допомогою компіляторів, великої кількості графічних конструкторів для розробки UI, засобів виконання коду тощо.

Серія даних продуктів дозволяють розробляти як консольні програми, так і програми з UI, у тому числі з підтримкою технологій Windows Forms, UWP а також веб-сайти, веб-програми, веб-служби. Програми можливо розробляти на багатьох платформах, таких як Windows, Windows Mobile, Windows CE, .NET Framework, .NET Core, .NET, MAUI, Xbox, Windows Phone .NET Compact Framework та Silverlight. Не так давно з'явилася можливість розробляти мобільні кросплатформені застосунки за допомогою .Net Core під Android та IOS, після придбання Xamarin компанією Microsoft. «На рис. 2.3» зображено інтерфейс середи розробки Visual Studio.

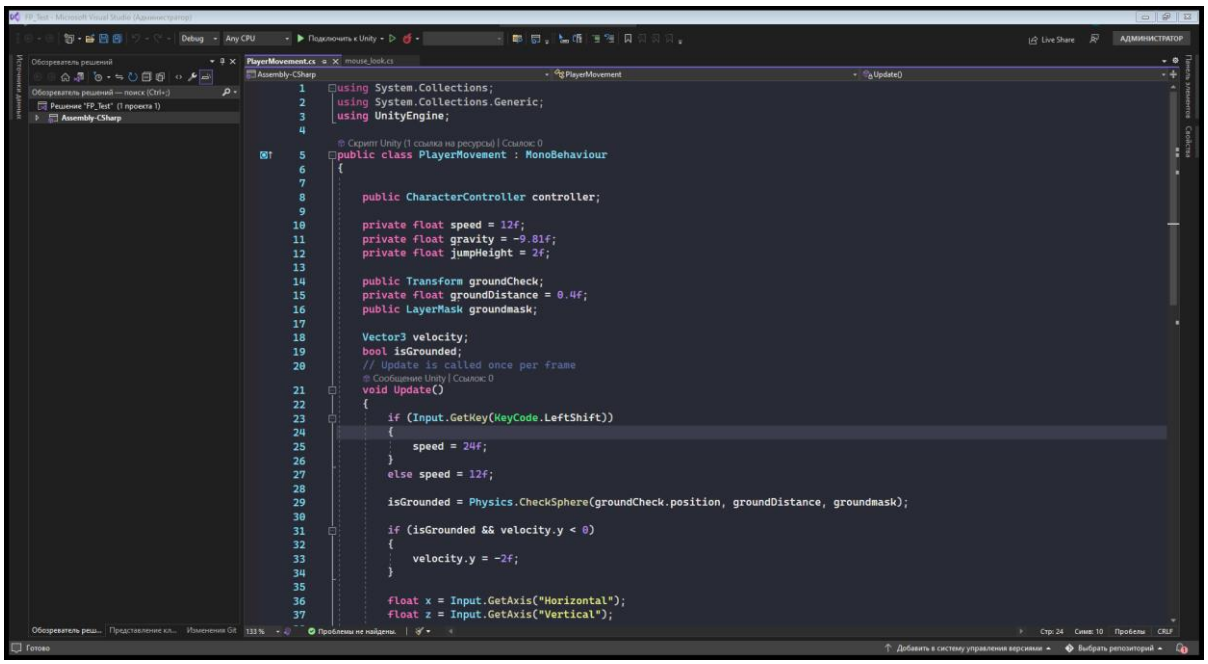

Рисунок 2.3 – Основний інтерфейс середовища Visual Studio 2022

IDE Visual Studio має редактор вихідного коду, який підтримує технологію IntelliSense та має можливість простого рефакторингу коду. Налагоджувач, що вбудований в IDE, може працювати як на рівні вихідного коду, так і на машинному рівні. Вбудованими інструментами являються: редактор форм UI для спрощення його створення, веб-редактор, редактор класів та схем баз даних. Як і більшість середовищ розробки, Visual Studio має можливість підключати та навіть створювати плагіни (сторонні доповнення), що можуть розширити функціональність середовища на будь-якому рівні. Плагіни можуть дати можливість використовувати системи контролю версій (як, наприклад, Subversion та Visual SourceSafe) у середовищі. Також можливо додати нові набори інструментів (наприклад, для редагування та візуального проектування) або інструменти для інших аспектів процесу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server).

2.3 Двигун для розробки відеоігор Unity

Як відомо з аналітичної частини даної роботи, для розробки 3D гри було обрано двигун для розробки відеоігор Unity від американської компанії Unity Technologies, перший випуск якої відбувся 8 червня 2005 року.

Основний редактор має простий Drag&Drop інтерфейс, він, як і більшість інтерфейсів двигунів, складається з декількох вікон, які можна налаштовувати, видаляти та додавати, це веде за собою можливість компіляції та налагодження під час розробки, можливо наживо регулювати деякі параметри під час виконання результуючої програми. Вже відомо, що двигун за основу використовує написання скриптів на мові програмування C#, але також відомо, що раніше підтримувалась мова Boo та JavaScript, як зараз пам’ятаю, що були навіть мануали по розробці скриптів JavaScript для Unity, але підтримка була припинена у 2017 році. Варто відзначити, що для проведення розрахунків 3D фізики використовується технологія від Nvidia – PhysX. Графічним чіпом являється DirectX 11ї версії та при можливості 12ї.

Рисунок 2.4 – Логотип Unity

Уся розробка ігрового процесу у Unity будується на трьох основних частинах. Ними являються: об’єкти GameObject, компоненти та змінні. Як не було б то очевидним, але кожний об’єкт у розробці являється GameObject, чи то 3D фігура, спецефект, чи персонаж, декорація, світло тощо.

Проєкт у Unity ділиться на сцени – окремі файли, що містять свої свої сгенеровані чи розроблені світи, об’єкти та налаштування. Самі по собі об'єкти представляють собою оболочку та не мають своєї поведінки. Для додаваня деяких атрибутів, щоб об'єкт став працювати як того забажає розробник, об'єкт може мати певну кількість компонентів. Компоненти мають низку властивостей чи змінних, а у корі скриптів, що дають змогу отримати деяку поведінку об'єкта. Наприклад, такі прості параметри, як позиція, розміри та поворот об'єкта містяться у компоненті Transform.

Для створення своїх компонентів для реалізації власних алгоритмів та механік, користувач повинен створювати власні скрипти, в яких звичайно описує свою ігрову логіку та закріплює ці скрипти за об’єктами, як компоненти. Кожен скрипт пов'язується із внутрішніми механізмами Unity шляхом реалізації класу, похідного від вбудованого класу MonoBehaviour.

Об’єкти мови C# так само, як і в C++ можуть знаходитись на різних ділянках пам’яті. При кожному завантаженні застосунку процесор повинен, для обробки об’єкта, збирати його в кучу з різних ділянок пам’яті. Така проблема може суттєво знизити швидкість завантаження.

Для вирішення цих проблем команда Unity почала переробляти базові системи Unity на основі високопродуктивного, багатопоточного стеку інформаційно-орієнтованих технологій або DOTS (нині у статусі попередньої версії).

DOTS дозволяє застосунку ефективно використовувати можливості багатоядерних процесорів нинішнього покоління, що є високим плюсом, так як у деяких навіть сучасних відеоіграх, можна і не отримати суттєвого приросту за рахунок багатоядерного процесору, бо вони не використовують повною мірою їх можливості.

Технологія DOTS:

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

- Entity Component System (ECS) для розробки високопродуктивного коду;

- компілятор Burst для компіляції скриптів у оптимізований нативний код.

Багатопоточні системи DOTS допомагають створювати ефективні ігри для широкого обсягу пристроїв з величезними згенерованими світами, зі складними симуляціями, великою кількістю елементів. Продуктивна система, як наслідок дає менше тепловиділення елементів пристою, що веде за собою ефективніше використання мобільних пристроїв та комп'ютерів у цілому.

Розташування вікон за замовчуванням дає практичний доступ до найпоширеніших вікон. Якщо розробник не знайомий з деякими вікнами в Unity, він з легкістю може визначити їх можливості за назвою на вкладці. Нижче буде наведено найпоширеніші серед розробників вікна.

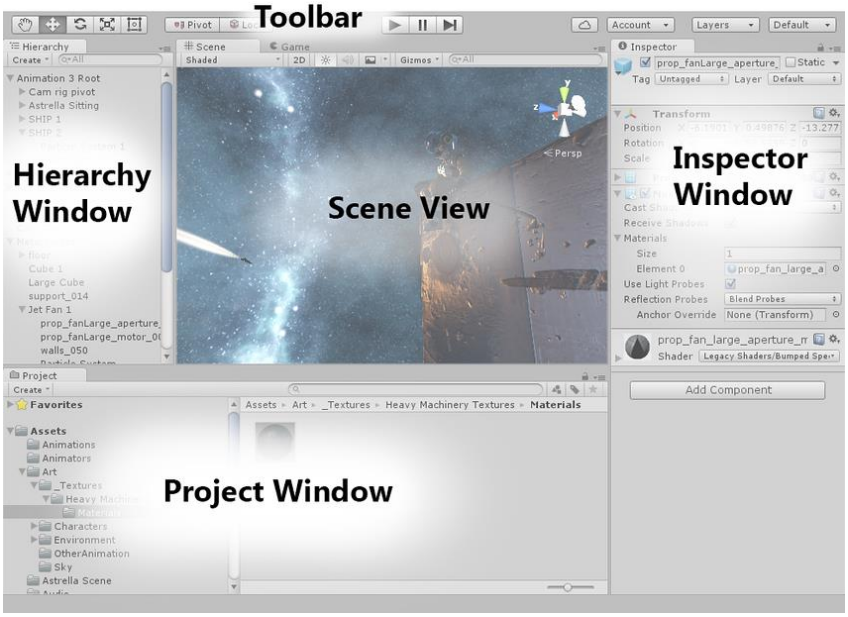

Рисунок 2.5 – Структура вікна по дефолту середовища розробки Unity

Вікно проєкту (Project Window) відображає розробнику усю бібліотеку ресурсів цього проєкту, з можливостями фільтрування та сортування файлів. Коли користувач імпортує асети з магазину Unity, вони з’являються у корені, що відображається у цьому вікні (див. рис 2.6).

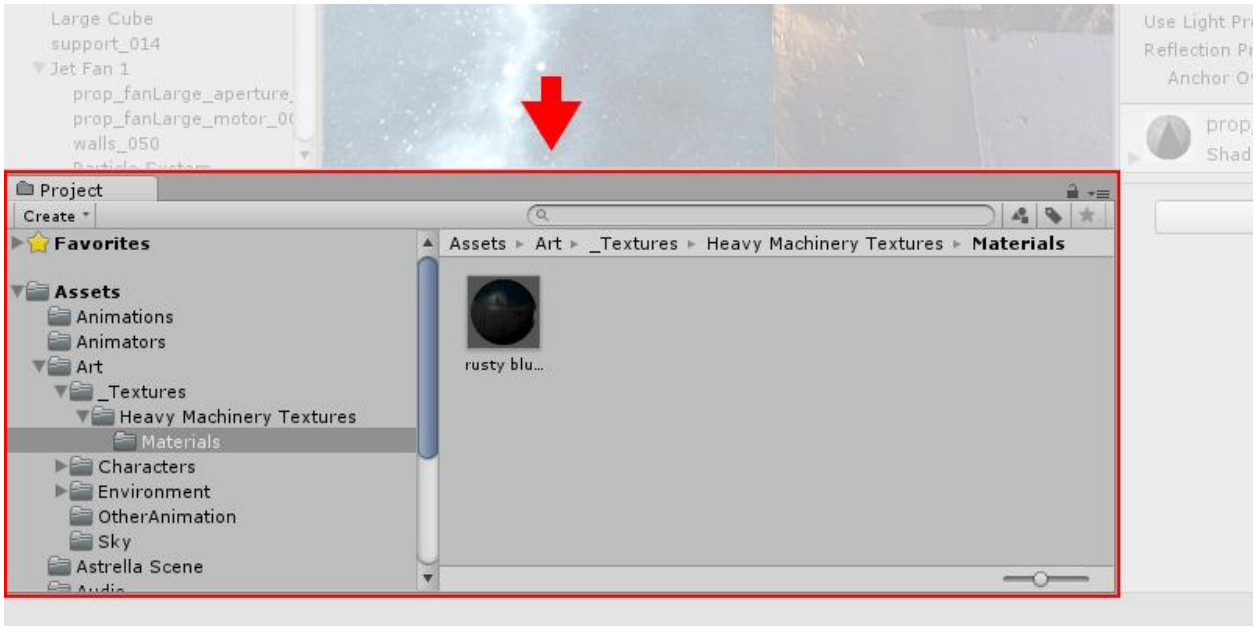

Рисунок 2.6 – Компонент інтерфейсу Unity Project Window

Перегляд сцени (The Scene View) – вважається головним вікном в редакторі, без якого сам редактор двигуна не може існувати. Вікно дозволяє редагувати 3D чи 2D сцену, переміщуватись по ній (див. рис 2.7).

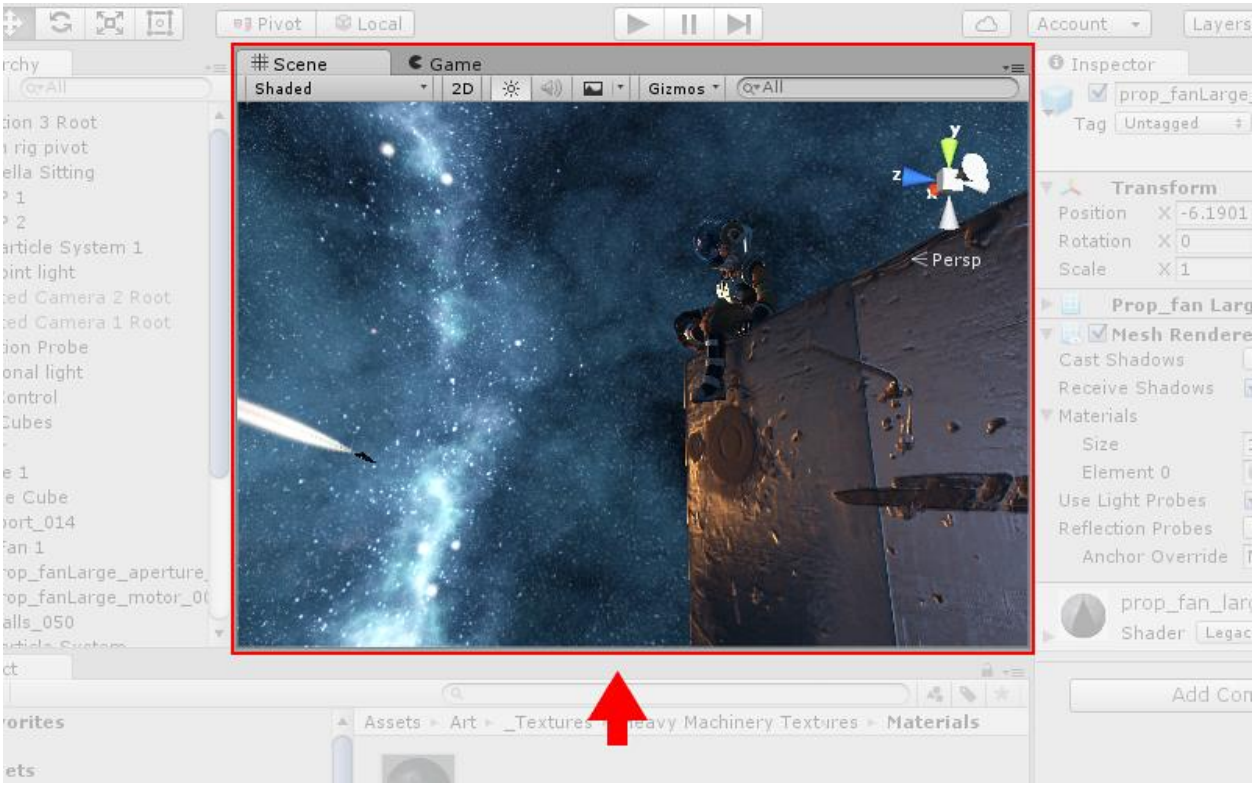

Рисунок 2.7 – Компонент інтерфейсу Scene View

Вікно ієрархії (The Hierarchy Window) — це ієрархічне текстове представлення кожного об’єкта в сцені. Кожний об’єкт на сцені має свою ієрархію. Вона розкриває структуру об’єкта, які дочірні об’єкти він має та їх упорядкованість, що також впливає на їх розположення чи відображення (див. рис 2.8).

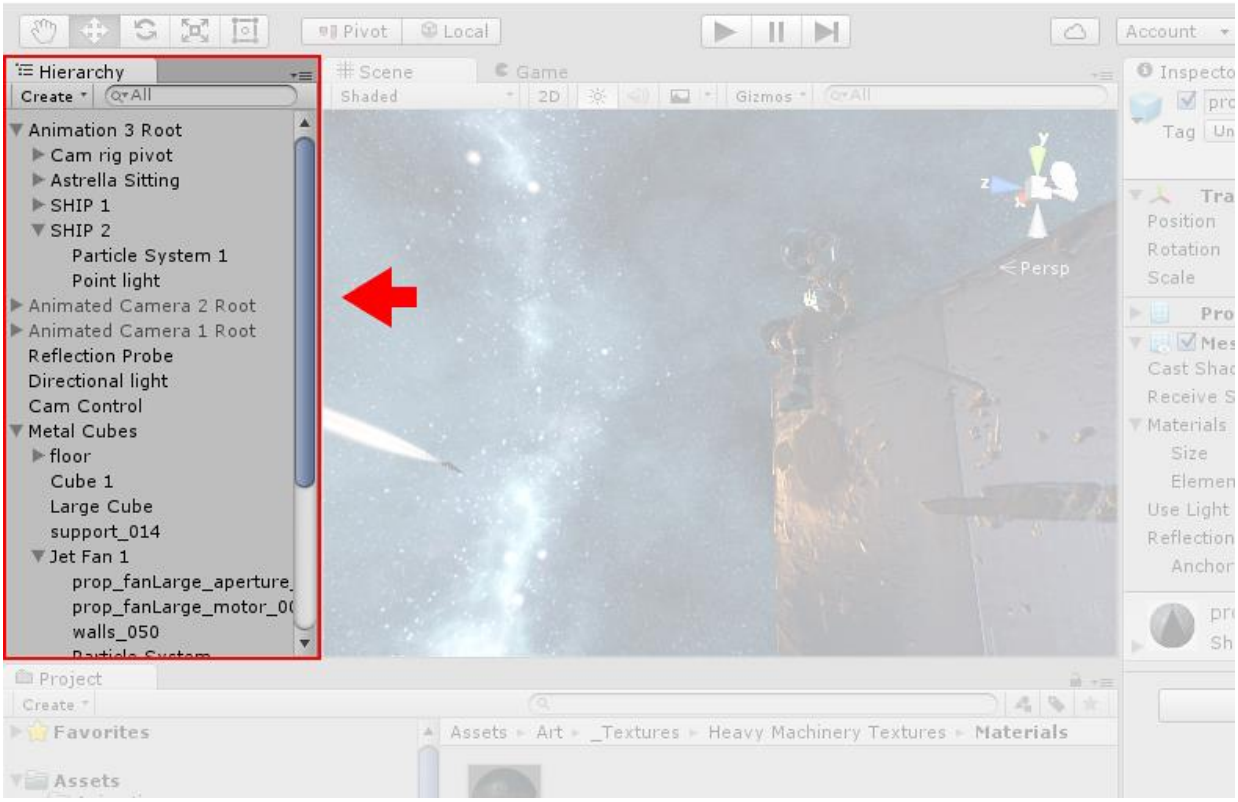

Рисунок 2.8 – Компонент інтерфейсу Hierarchy Window

Вікно інспектора (The Inspector Window) дозволяє переглядати та редагувати усі компоненти обраного поточного об'єкта в Unity (див. рис 2.9). Кожен об’єкт представляє якусь сутність, тому вони можуть мати різну кількість компонентів та налаштувань, але які не дублюються. Самим компонентом в редакторі може бути написаний скрипт, та public поля класу цього скрита відображаються в налаштуваннях компонента, які можна змінювати в реальному часі роботи програми, що є дуже зручним.

Панель інструментів надає доступ до найважливіших інструментів керування сценою, редагування її та об’єктів, що вона містить. Головними елементами на панелі інструментів, якими користується кожен розробник, є кнопки режиму Debug, тобто відтворення, пауза чи перехід на наступний крок. Праворуч на панелі розположені елементи для доступу до хмарних служб

Unity. Далі за ними йде меню макета редактора. Воно дозволяє користувачу створити макети розположення вікон редактора та переключатися між ними.

Панель інструментів не є вікном і являється єдиною частиною інтерфейсу Unity, яку не можна змінити (див. рис. 2.10).

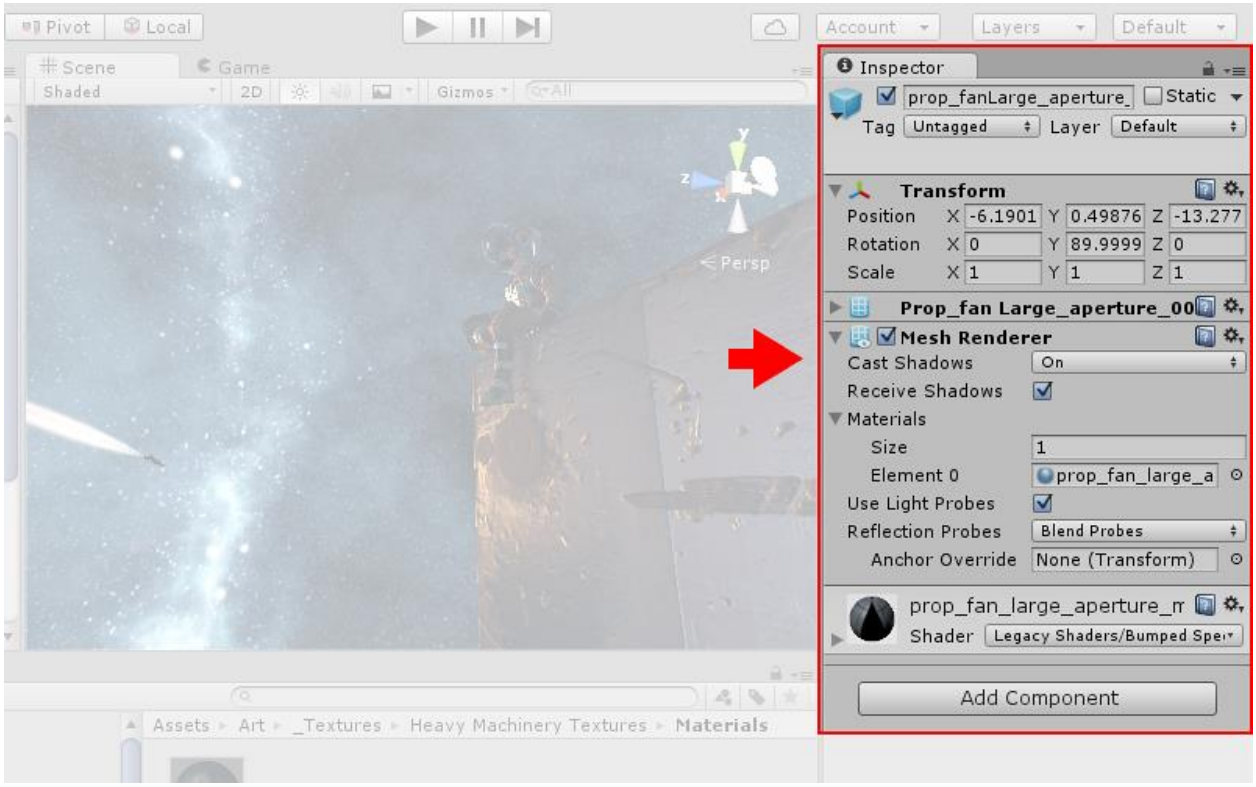

Рисунок 2.9 – Компонент інтерфейсу The Inspector Window

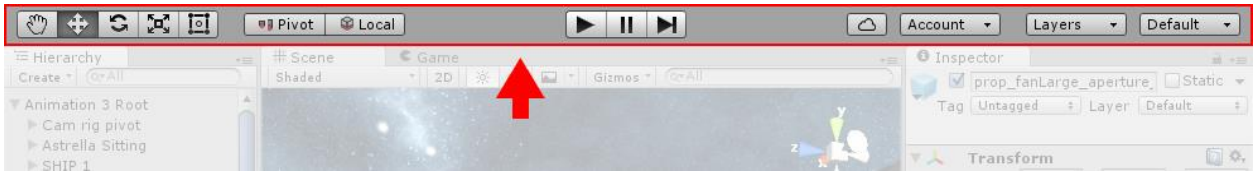

Рисунок 2.10 – Панель інструментів в Unity

При імпорті текстури в Unity є можливість згенерувати alpha-канал, mip-рівні, normal-map, light-map, карту відображень. Не можна текстуру безпосередньо кріпити на 3D модель. Спочатку потрібно створити об’єкт типу

Material, якому призначається шейдер для рендерингу, який можна написати самому при вмінні створювати шейдери, шляхом написання коду мовами шейдерів. Матеріалу призначається текстура й після є сенс прикріпити матеріал до моделі. Редактор Unity має можливість завантажувати чи редагувати анімації. Вони попередньо повинні бути створені в 3D-редакторі та завантажені разом з моделлю.

Важливою технологією для розробки 3D гри являється система LOD (Level Of Detail), яка присутня у Unity. Її суть полягає у тому, що при збільшенні відстані від камери до об’єктів, їх деталізація зменшується і навпаки. Також Unity має систему Occlusion culling, яка зменшує навантаження на центральний процесор та оптимізує роботу програми. Суть системи полягає в тому, що візуалізація, геометрія та колізія об’єктів, які не потрапляють в поле зору камери, не здійснюються. Ця технологія є дуже вагомою частиною оптимізації 3D проєкту.

«На рис. 2.11» зображено інтерфейс зв’язаного з двигуном внутрішнього магазину Unity Asset Store, який розробники використовують для завантаження та викладання в загальний доступ чи то безкоштовно, чи за гроші своїх моделей, текстур, сцен тощо. Усі файли, формат яких підтримує двигун можна запакувати у формат .unitypackage, який і використовує магазин для передачі файлів між користувачами.

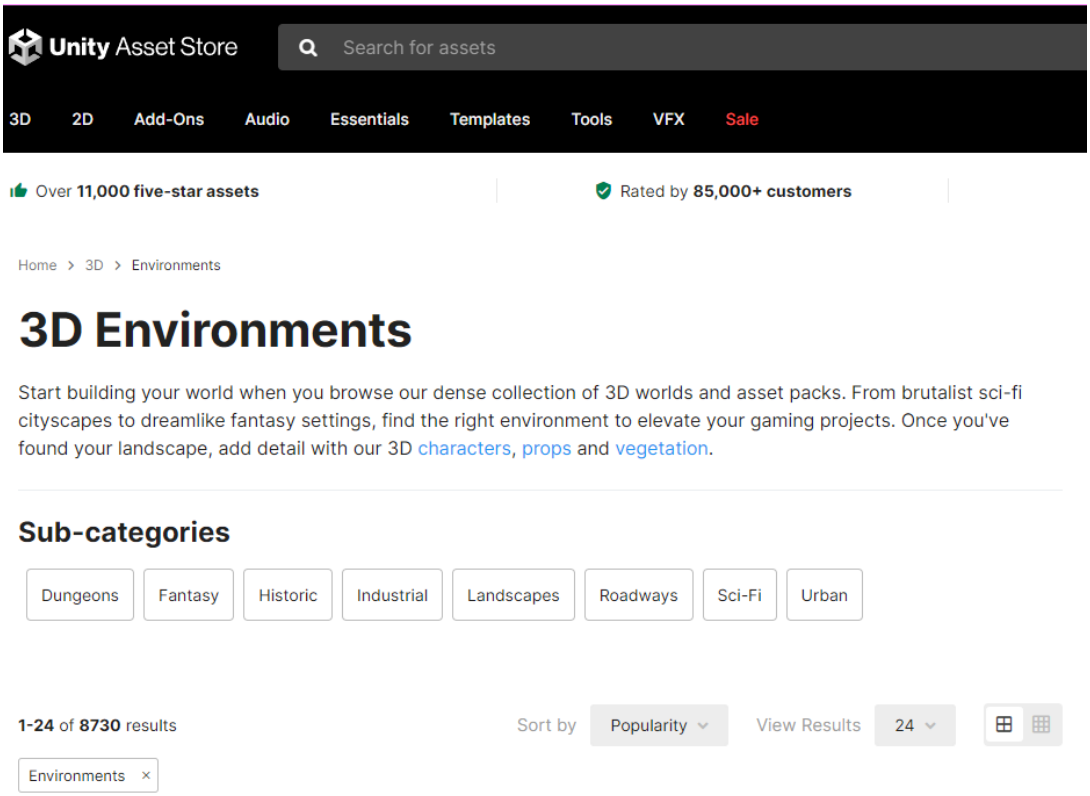

Рисунок 2.11 – Частина інтерфейсу сервісу Unity Asset Store

Висновки до другого розділу

Двигун Unity є ідеальною середою розробки для не масштабної 3D гри з усіма її тонкощами. Редактор має досить простий та гнучкий інтерфейс. Unity має підтримку багатьох форматів файлів та якісну фізику від Nvidia. Написання скриптів на мові C# тільки спрощує розробку проєктів. C# є об’єктно-орієнтованою мовою зі статичною типізацію що є плюсами при створенні комп’ютерної гри. Visual Studio являється ідеальним середовищем розробки або ж просто редактором коду з IntelliSense для мови програмування C#.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ 3D ГРИ

3.1 Опис ідеї та створення плану розробки

Для виконання поставлених вимог потрібно було придумати безкінечну 3D гру з алгоритмом пошуку шляху. Для реалізації алгоритму було вирішено додати мобів з влаштованим ШІ для слідування за головним персонажем. Сутність гри являє собою спавн персонажа у локації зимового гірського типу середніх розмірів з достатньою кількістю дерев, з глибокою нічною атмосферою. Гравець являє собою персонажа, що пересувається по локації та захищається від мобів способом метання у них снарядів. Для цього було вирішено створювати гру з FPP режимом, щоб було легше концентруватися на метанні, та взагалі, потрібність у виді від третьої особи відпадає, так як інших дій у персонажа не буде. Моби слідкують за персонажем та при досягненні його наносять удар, після якого гра завершується. Персонаж захищається від них способом метання снарядів, що при попаданні у моба, наносять йому деяку шкоду, при достатній кількості влучень моб гине. Кількість вбитих мобів зберігається у виді очків, що відображаються на UI.

Для реалізації задуманої ідеї було створено план розробки гри:

1) створення локації гри;
2) додавання механіки пересування персонажа;
3) написання алгоритму для ШІ мобів;
4) додавання мобів та створення логіки слідування за персонажем;
5) додавання моделей та анімацій мобам;
6) створення взаємодії персонажа та моба;
7) створення UI та додавання логіки рахунку очків із завершенням гри.

3.2 Створення локації гри

3.2.1 Створення рельєфу місцевості

Для створення рельєфу у Unity було додано об’єкт Terrain, що дозволяє створювати реалістичні та детальні ландшафти за допомогою великої кількості інструментів. Основними інструментами являються пензлі, що мають великий список функцій, такі як скульптування, згладжування та різні ефекти (див рис. 3.1). За допомогою пензлей з різними параметрами та масками було відредактовано ландшафт для отримання потрібної поверхні.

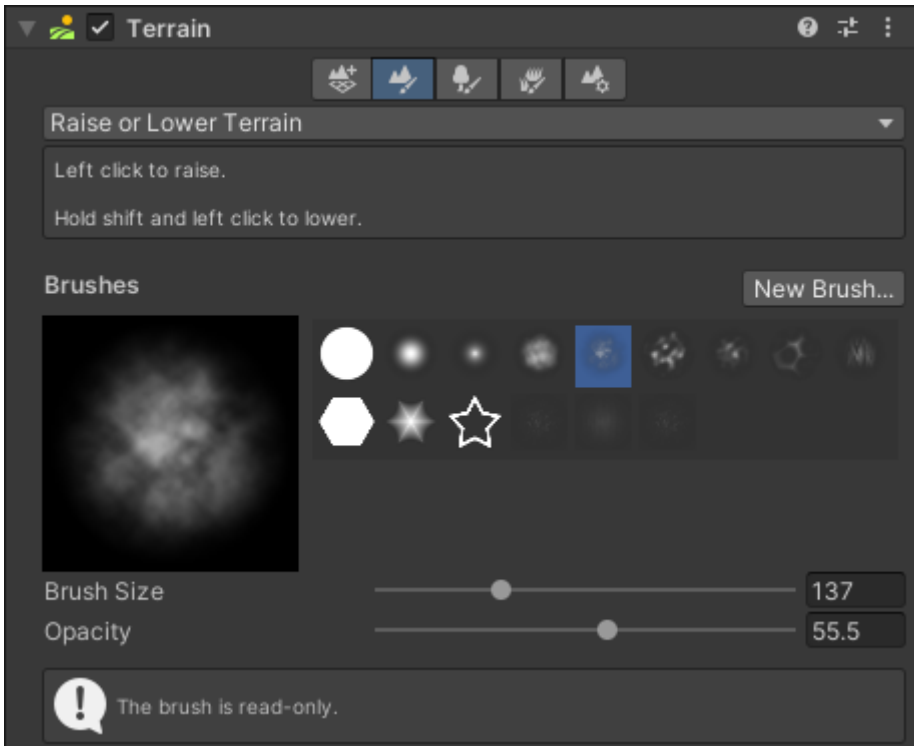

Рисунок 3.1 – Скриншот вікна редагування ландшафту

Далі для проєкту було обрано декілька текстур, що відображали задуману атмосферу локації. Текстури у редакторі можна накласти на весь рельєф, їх було намальовано також за допомогою пензлів, але які вже

призначені для текстур, що давало змогу контролювати місця, на які вони наносились та розмір нанесення.

3.2.1 Додавання дерев

Для додавання об'єктів таких, як дерева, трава та каміння, у редакторі ландшафту існує відповідний пензлик. Так як наведені об'єкти звичайно додаються у великих кількостях, для них існує окремий інструмент, що збільшує продуктивність створення ландшафту, у противному випадку окремо би додавався кожний об'єкт. 3D моделі, що завантажуються для додавання наведеним інструментом повинні мати певну структуру з каркасом. Даний пензель має такі параметри як розмір області додавання, густоту об’єктів, випадковість розташування, діапазон випадкових розмірів об’єктів тощо. До проєкту було додано чотири моделі хвойних дерев.

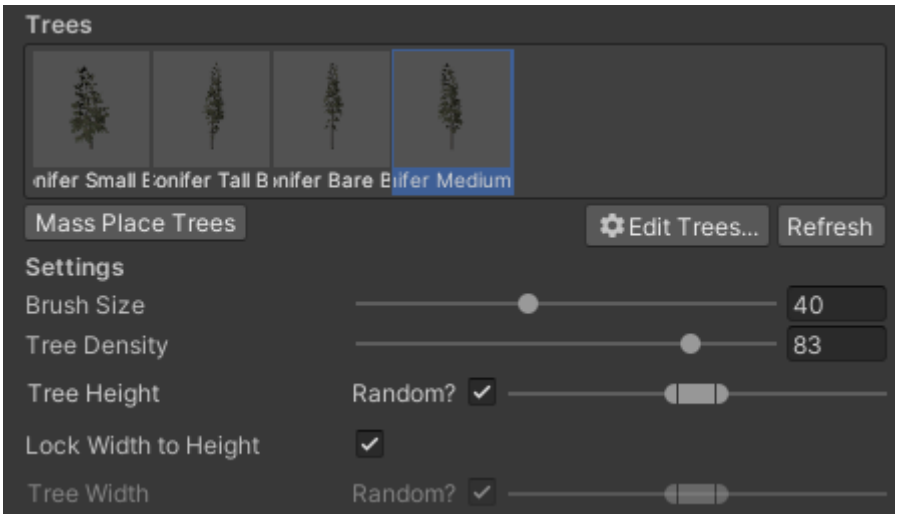

Рисунок 3.2 – Вікно пензля дерев

«На рис. 3.2» можна побачити вікно додавання дерев до ландшафту за допомогою пензлів, також присутні усі дерева, що були використані у моделюванні.

3.2.1 Регулювання світла та створення скайбоксу

Скайбоксом у 3D іграх являється шестисторонній куб, який знаходиться завжди позаду усієї сцени. Кожна грань має текстуру для імітації звичайно неба. За замовчуванням у Unity 3D є простий скайбокс. Для його кастомізації було створено матеріал з шейдером скайбоксу, та для кожної грані завантажено відповідну текстуру. Для реалізації ідеї було використано текстури темно-синього нічного неба з наявністю місяця.

Сама локація являється ніч, але для освітлення було додано направлене джерело світла, що імітує світло від місяця. Це джерело було розташоване відносно далеко від мапи та на малій висоті з достатнім радіусом освітлення. Було розташоване зі сторони малюнку місяця на текстурі скайбоксу, що додало мапі атмосферного та реалістичного місячного світла.

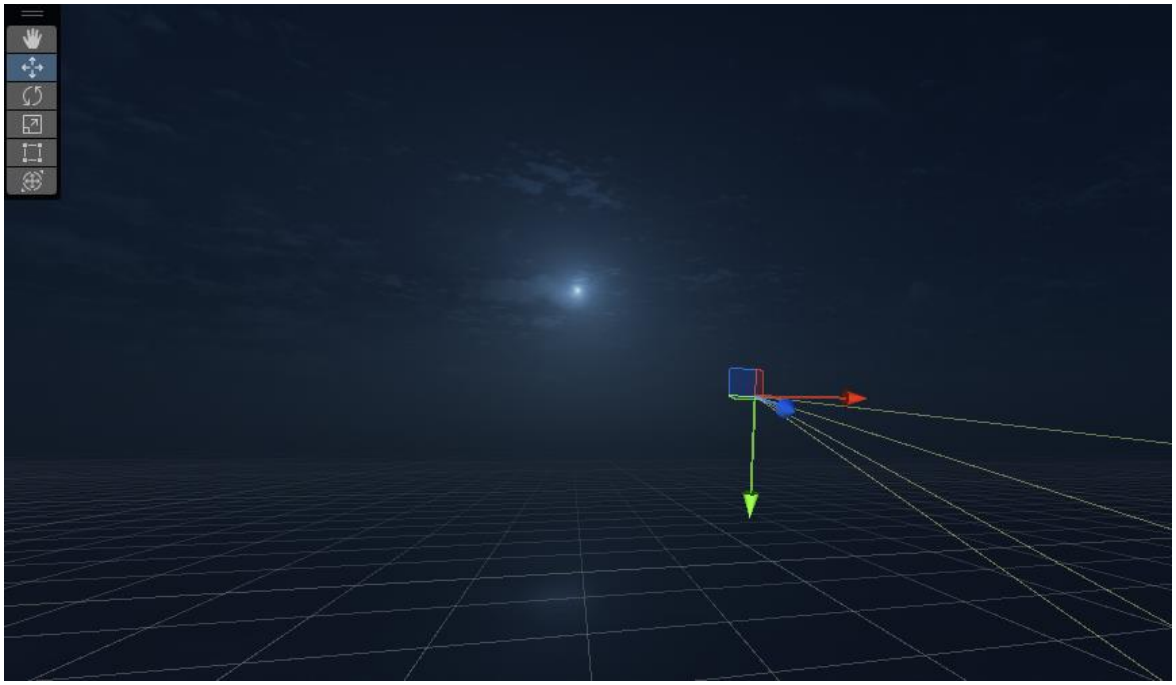

Рисунок 3.3 – Скайбокс та світло сцени

«На рис. 3.3» зображено сторону скайбоксу з малюнком місяця та направлення джерела світла.

3.2.1 Додавання сніжної бурі

Для імітації сніжної бурі Unity було вирішено використати генератор частинок – Particle System. Це компонент, який імітує рідинні чи газові субстанції, у вигляді, наприклад, вогню, газів тощо. Particle System використовує метод генерації анімацій з великої кількості 2D текстур. Цей компонент має велику кількість налаштувань для генерації будь якого виду систем. На це також вплинула можливість використання будь-якої текстури у вигляді частинки. Компонент має групи налаштувань, які мають флаги для виключення усієї групи. Наприклад група Color by speed, яка відповідає за налаштування графіку залежності відтінку та прозорості кольору від швидкості частинки. Налаштування мають широкі можливості, та розглядати усі функції немає сенсу.

Було створено об’єкт Particle System та налаштовано параметри для отримання потрібного результату (див. рис. 3.4). Спочатку було завантажено текстуру невеликого об'єму туману, відкореговано розміри самої системи та час життя частинки. Також було потрібно налаштувати діапазон випадкових швидкостей частинок та додати функцію повороту частинок. Останнім кроком стало налаштування зміни кольору та прозорості частинки відносно часу. З появою, частинки мають велику прозорість, потім набирають колір до середини часу життя, та до зникнення поступово його втрачають.

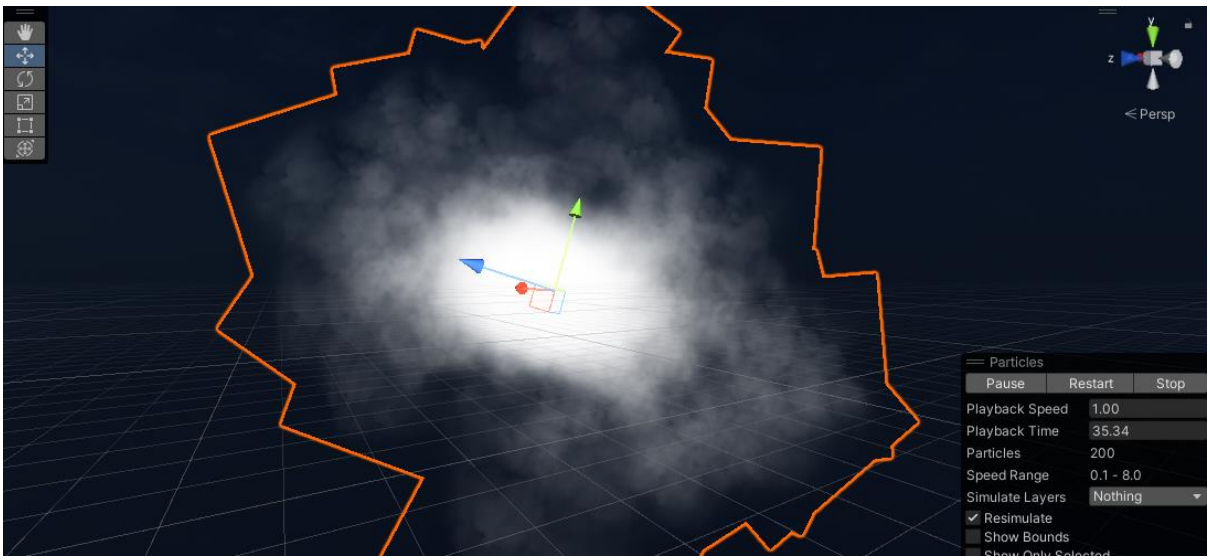

Рисунок 3.4 – Система частинок

Кінцевим результатом розробки локації для 3D гри стала засніжена нічна гірська місцевість з невеликим місячним сяйвом, невеликими лісами чи поодинокими деревами та місцями зі сніжною бурею. Кінцевий результат можна побачити «на рис. 3.5», де зображено невелику частину локації з бурею.

Рисунок 3.5 – Частина завершеної локації

3.3 Механіка пересування персонажа

Було задумано реалізувати персонажа, що пересувається по місцевості за допомогою звичайного для користувача набору клавіш мувменту «W», «A», «S», «D» та «SPACE» - для стрибка. Для заданих потреб доцільно було використовувати існуючий компонент Character Controller у Unity.

Character Controller являє собою компонент у Unity, що використовується для управління персонажем від першої чи третьої особи та має вбудований колайдер для детекції зіткнень з іншими об’єктами на сцені. Основними його параметрами являються:

- Slope Limit – можна сказати висота кроку, що обмежує персонажа при пересуванні його на височини;
- Step Offset – розмір кроку, який дає змогу рухатись між поверхностями, відстань між якими не більше ніж цей параметр;
- Center, Radius, Height – розміри та позиція колайдера у просторі відносно батьківського об'єкта.

За замовчуванням колайдер Character Controller має форму капсули, що вважається розповсюдженою формою колайдера персонажа для оптимальних розрахунків.

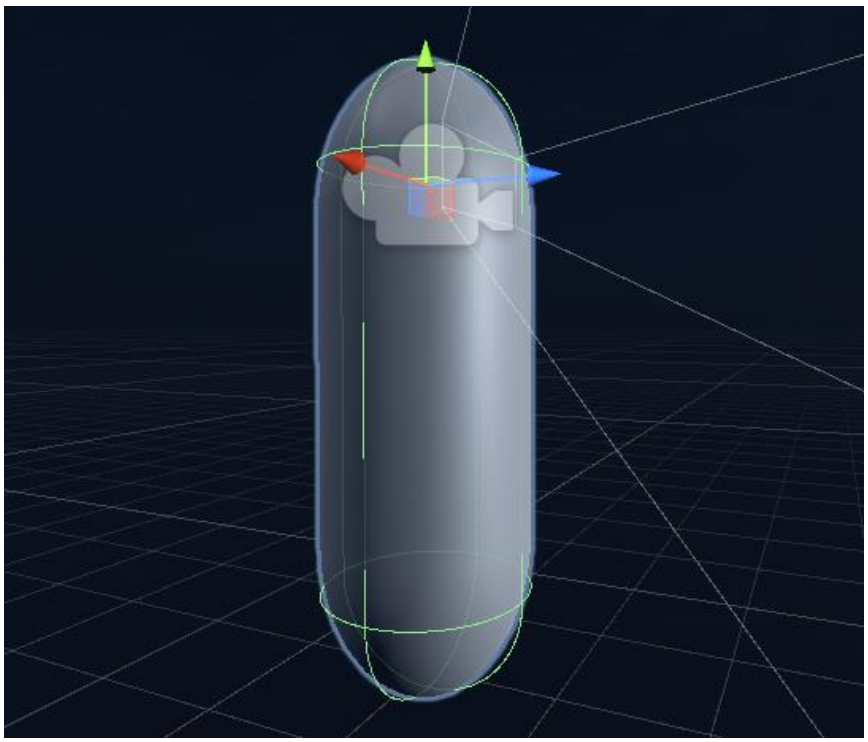

Рисунок 3.6 – Об'єкт Player

«На рис. 3.6» показано вигляд об’єкта Player на сцені, що має компонент Character Controller, колайдер якого має вигляд світло-зеленої сітки. Дочірнім об’єктом було додано капсулу розмірів колайдера для помітного відображення у просторі. Усі дочірні об’єкти у Unity мають значення положення у просторі відносно батьківського об’єкта, та змінюють положення й поворот разом з його переміщеннями. Тож доцільним було додати об’єкт камери дочірнім об’єктом для Player та розмістити її приблизно на висоті очей.

3.3.2 Написання скрипту повороту персонажа

Для повороту камери потрібно було написати скрипт повороту усього персонажа з його дочірніми об’єктами, одним з яких була камера. Було написано скрипт, що став компонентом для Player.

При написанні коду спочатку було задано флаг у методі, який спрацьовує при створенні об’єкта, що визначає центрування курсора з кожним кадром рендерингу.

```
void Update()
{
    float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
    float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;

    xRotation -= mouseY;
    xRotation = Mathf.Clamp(xRotation, -90f, 90f);

    transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
    playerBody.Rotate(Vector3.up * mouseX);

}
```

Рисунок 3.7 – Частина коду скрипта mouse_look

У методі Update() створюються дві змінні для обчислення нових координат на екрані для повороту камери. Вони отримуються з добутку значень положення курсора по відповідній осі, константи, що задає швидкість повороту камери та та часу, що було затрачено на обчислення та рендерингу кадру. Методом Clamp() визначається діапазон повороту по заданій осі. Далі виконуються методи для застосування змін (див. рис. 3.7).

3.3.3 Написання скрипту переміщення персонажа

Для переміщення персонажа в основі використовується метод Move() компонента Character Controller який дає змогу не просто перемістити персонажа, а перемістити його по нерівній поверхні або ж між поверхнями з попередньо заданою висотою кроку та його довжиною (див. рис. 3.8).

```
float x = Input.GetAxis("Horizontal");
float z = Input.GetAxis("Vertical");

Vector3 move = transform.right * x + transform.forward * z;

controller.Move(move * speed * Time.deltaTime);
```

Рисунок 3.8 – Частина коду скрипта player_movement

Перед написанням логіки було створено деякі змінні зі значеннями швидкості переміщення, висоти стрибка та прискоренням вільного падіння для подальших розрахунків.

Для зчитування натискання на клавіші для переміщення використовується метод GetAxis() з параметром назви осі. Цей метод зчитує не задані клавіші, а традиційно призначені для управління переміщенням, тобто клавіші «W», «A», «S», «D», стрілки та осі на ігрових маніпуляторах.

```
if (Input.GetButtonDown("Jump") && isGrounded)
{
    velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravity);
}

velocity.y += gravity * Time.deltaTime;

controller.Move(velocity * Time.deltaTime);
```

Рисунок 3.9 – Частина коду скрипта player_movement

Для реалізації стрибка було додано змінну типу вектора з трьома значеннями, що містить значення положення персонажа у просторі – velocity, (див. рис. 3.9). Координата осі «OY» змінної velocity кожний кадр змінюється

за допомогою обчислення вільного падіння. При натисканні гравцем на кнопку типу «Jump», що на звичайній клавіатурі відповідає кнопці «Space», виконується переміщення персонажа на висоту, зазначену у змінній jumpHeight. Також стрибок виконується тільки при позитивному значенні змінної isGrounded, яка означає положення персонажа на рельєфі та обчислюється спеціальним методом. Ще у гру було додано режим бігу, чи прискорення, що притаманний багатьом іграм з управління головним персонажем чи то у FPP чи TPP. При натисканні клавіші «LShift» значення змінної швидкості персонажа збільшується на зазначену константу.

3.4 Алогоритм пошуку шляху

Як відомо, основним алгоритмом для AI моба було обрано алгоритм A*. Спочатку для його виконання потрібно було підготувати деякі вхідні дані, тобто структуру даних з існуючими вершинами та їх інформацією.

Головною інформацією вершини є її висота, від значення якої залежить результат алгоритму. Створена мапа у редакторі була розміром 250 на 250 умовних одиниць редактора Unity, тому було вирішено зчитувати значення вершин з кроком в одну умовну одиницю.

Першим етапом створення вхідних даних стало формування двомірного масиву індексами якого стали двомірні координати вершини на площині, а значенням елементу стала висота вершини.

```
for (int i = 0; i < size; i++)
{
    for(int j = 0; j < size; j++)
    {
        Vector3 pos = new Vector3(i * scale + offset, 0.0f, j * scale + offset);
        Vector3 genPos = pos;
        genPos.y = Terrain.activeTerrain.SampleHeight(pos);
        map[i, j] = genPos.y;
    }
}
```

Рисунок 3.10 – Скриншот коду зчитування висот

Для отримання значення висоти вершини – Y за значеннями позиції (X, Z) було використано просту та досить важливу функцію SampleHeight() (див. рис. 3.10). Вона повертає значення Y об'єкта типу Terrain із заданими значеннями координат X та Z. Приймає параметром вектор позиції. Також плюсом методу є отримання висоти тільки рельєфу, як раз те, що потрібно, тобто хоч дерева та трава є частиною Terrain, але їх висоти не будуть враховуватись у результаті повертаємого значення. Також потрібно відмітити, що були додані глобальні змінні: scale, offset. Якщо уявити розбиття мапи на менші частини, то це представляє собою велику кількість квадратів на площині. Значенням scale являється розмір квадрату, позиція його лівої верхньої вершини буде знаходитись за добутком індексів на scale. Та для центрування вершини в умовному квадраті, використано змінну offset, яка є половиною від scale. Координатою вершини у двовимірному просторі є:

$$y = i * x + k, \qquad (3.1)$$

де y – координата вершини;

i – індекс у масиві;

x – значення scale;

k – значення offset.

Наступним кроком стало створення класу, який стане представленням вершини з усіма її потрібними для розрахунків полями (див. рис. 3.11).

```
public int Index { get; set; }
public Point Coord { get; set; }
public Vertex PathFrom { get; set; }
public List<Vertex> AdjVerts { get; set; }
public float GX { get; set; }
public float HX { get; set; }
public float Height { get; set; }
public bool InOpenList { get; set; }
```

Рисунок 3.11 – Список властивостей класу Vertex

Можна не говорити про значення поля Index у структурі класу, хоч клас вершини і має поле типу Point, що є унікальним, але для швидкості обчислень використано індекс для пошуку потрібної вершини. Як раніше сказано, існує поле типу Point, що містить 2 значення, якими є координати вершини. Поле PathFrom являє собою вказівник на об’єкт цього типу, що значить вершину з якої йде шлях в поточну, яку було отримано при виконанні алгоритму. Height – з назви, висота вершини. AdjVerts – колекція, яка містить усі вершини, суміжні з поточною, вміст колекції розраховується на етапі створення об’єктів Vertex зі сформованого масиву значень вершин.

GX – результат функції g(x) алгоритму A*, що представляє собою розрахунок довжини вже відомого шляху від початкової точки до поточної у алгоритмі.

HX – евристична функція, результатом якої є евристичне наближення від поточної точки до кінцевої. Для її розрахунку було вирішено використовувати метод Манхеттена, який представляє собою обчислення відстані між точками по прямій.

Наступним етапом стало написання функції для творення кожної вершини типу Vertex. Функція обходить у циклі кожний елемент двовимірного масиву та викликає конструктор класу Vertex для створення об’єкта зі значеннями координат, висоти вершини та додає контейнер із суміжними вершинами.

Для реалізації самого алгоритму було створено клас Pathfinder, конструктор якого приймає тільки дві вершини типу Vertex, чого достатньо і усі вершини не потрібно зберігати у загальному контейнері, тому що всі вершини мають посилання на суміжні. «На рис. 3.12» зображено поля класу.

```
private List<Vertex> openList;
private Vertex startVertex;
private Vertex endVertex;
```

Рисунок 3.12 – Поля класу Pathfinder

Про два поля типу Vertex вже згадано раніше, та з назви зрозуміло, що вони представляють собою стартову та кінцеву вершини. Змінна OpenList по ходу алгоритму буде наповнюватись вершинами, які обходить алгоритм, та впорядковуватись за пріоритетом – сумою значень властивостей GX та HX вершини. Клас має два методи : findPath() та getPath(). Перший запускає сам алгоритм, результатом якого є правильно задані вказівники PathFrom для кожної вершини Vertex, які беруть участь у побудові шляху, метод повертає ціле число, що сповіщає чи був знайдений шлях. Останній повертає значення послідовно розташованих вершин шляху у контейнері. «На рис. 3.13» показано створену блок-схему головного метода.

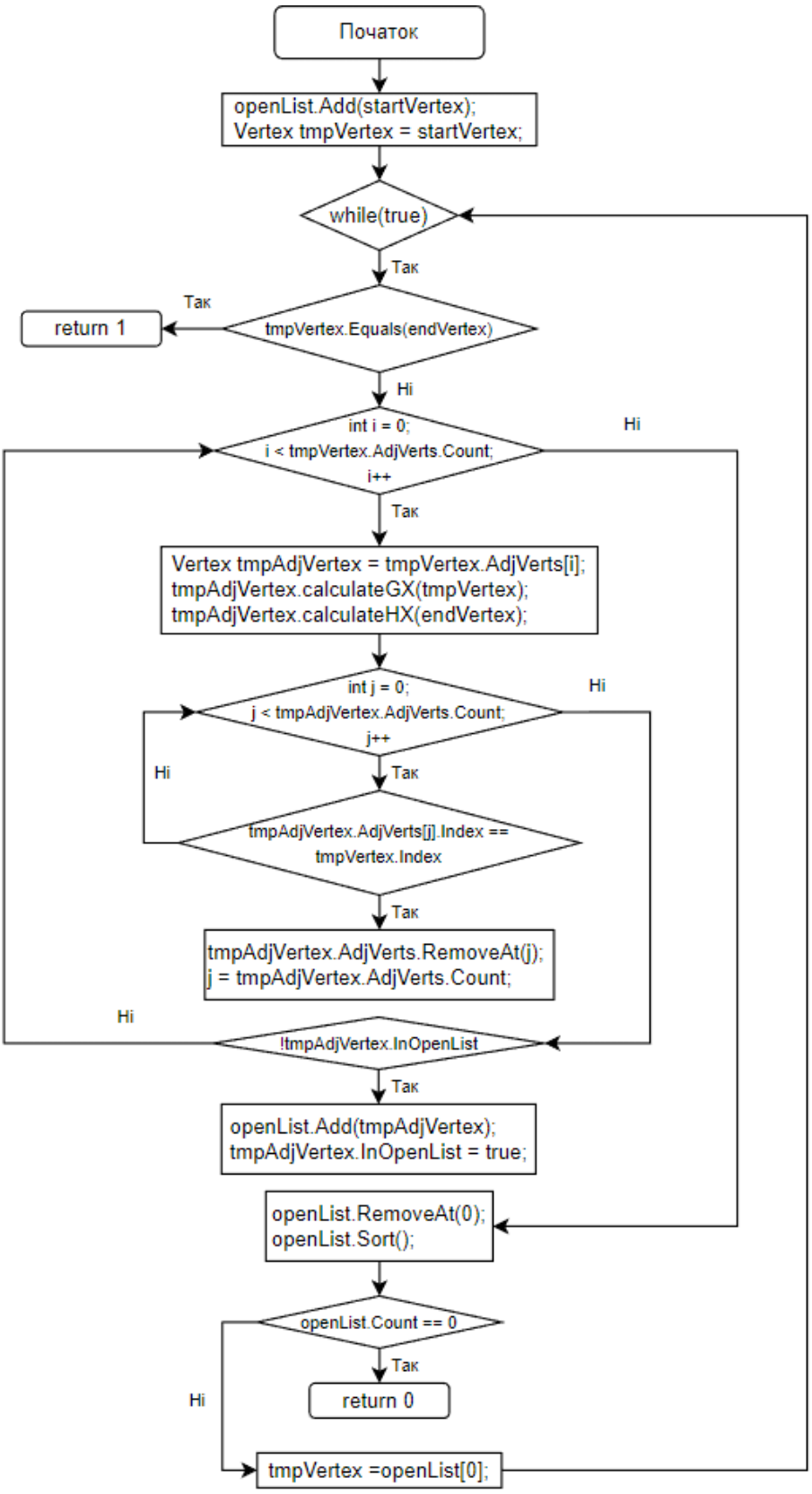

Рисунок 3.13 – Блок - схема методу findPath()

У середовищі Unity є можливість переглядати не тільки вікно гри в режимі Debug а й відобразити вікно редагування сцени та змінювати деякі параметри в реальному часі. Режим Debug дає змогу не тільки відправляти повідомлення у консоль, а й генерувати промені, лінії тощо на локації, що буде видно тільки з редактора сцени. Маючи таку можливість, було вирішено створити маску з кольорових ліній на локації, що будуть відображати вершини, з'єднані цими ж лініями, та окремим кольором відобразити знайдений шлях. Це було потрібно для переконання у роботі алгоритму та правильності генерації отриманих висот.

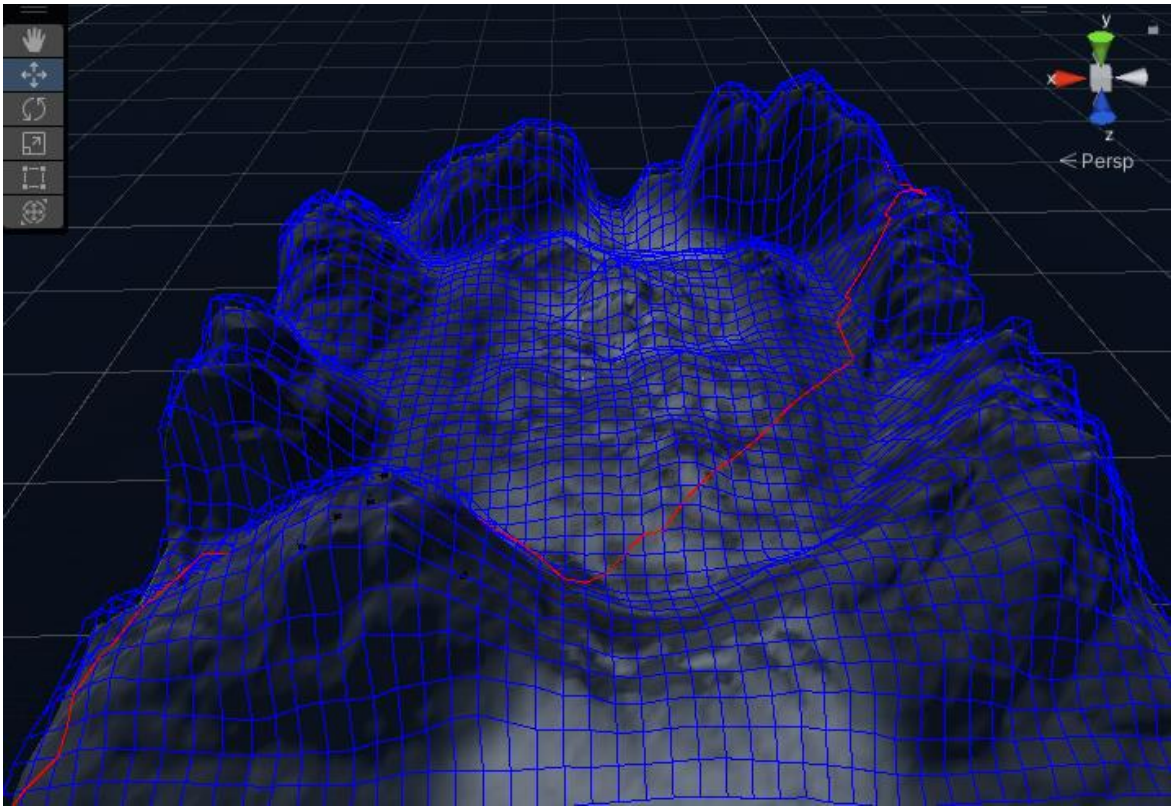

Рисунок 3.14 – Зображення сітки в режимі Debug

«На рис. 3.14» показано вікно редактора сцени зі згенерованою маскою вершин, що синього кольору та червоними лініями, що з'єднують знайдений маршрут між двома точками за допомогою алгоритму.

3.5 Створення мобів

Для створення моба постало завдання у завантаженні в проєкт його моделі та налаштування анімацій. Достатньо було створити дві анімації для бігу та нанесення удару по персонажу. Створення реалістичної анімації поведінки гуманоїда досить трудомісткий процес. Тому було прийнято рішення використати існуючий сервіс Mixamo від компанії Adobe. Даний сервіс дає змогу обрати готові моделі чи завантажити 3D модель та завантажити її з обраною анімацією та відкорегованими налаштуваннями. Сервіс має дуже великий вибір анімацій для гуманоїдів. Якщо завантажується своя модель, Mixamo автоматично накладає на неї анімацію та дає змогу завантажити готовий файл моделі з анімацією для імпорту в проєкт, але в цьому випадку в Unity потрібно буде відкоригувати скелет анімації моделі, так як сервіс міг некоректно згенерувати деякі кістки скелету.

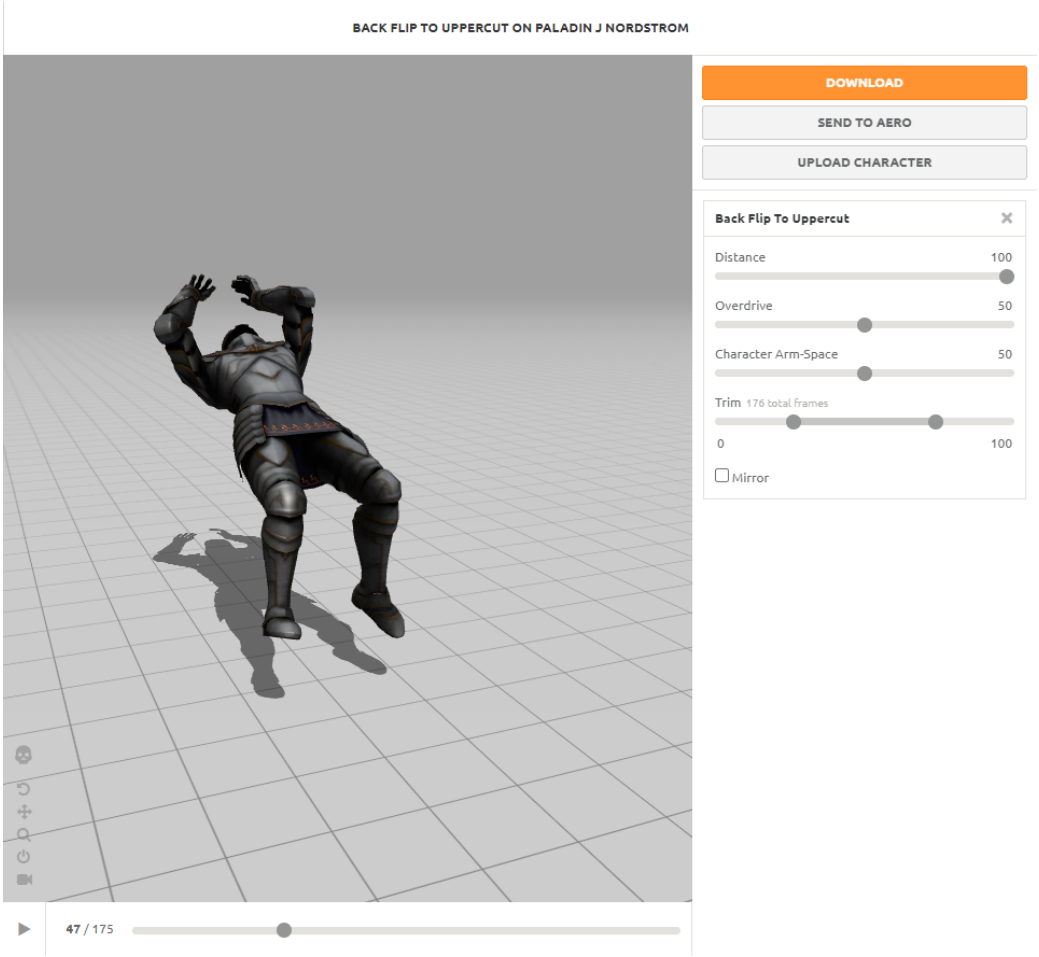

Рисунок 3.15 – Вікно редагування анімації Mixamo

«На рис. 3.15» зображено інтерфейс вікна редагування анімації сервісу Mixamo.

Для управління анімаціями було створено компонент Animator, що дає змогу контролювати та налаштовувати, які анімації та у який час будуть використовуватися (див. рис. 3.16). Також має налаштування переходу між анімаціями.

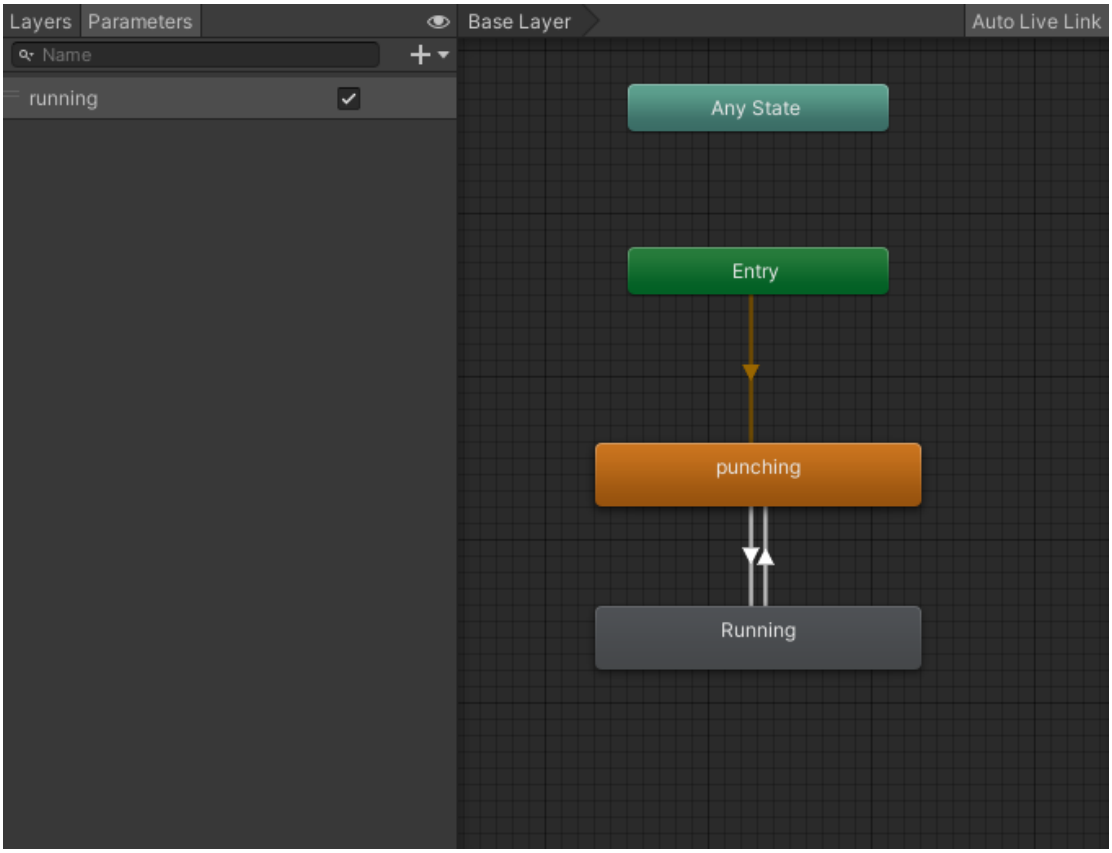

Рисунок 3.16 – Вікно компонента Animator

Як вже було згадано, для анімації моба потрібно дві анімації, тож їх було створено та додано в Animator. У аніматорі було створено зв’язки, тобто переходи між анімаціями. Для задання анімації, що програє, створено булеву змінну «running», значення якої в подальшому буде змінюватись скриптом.

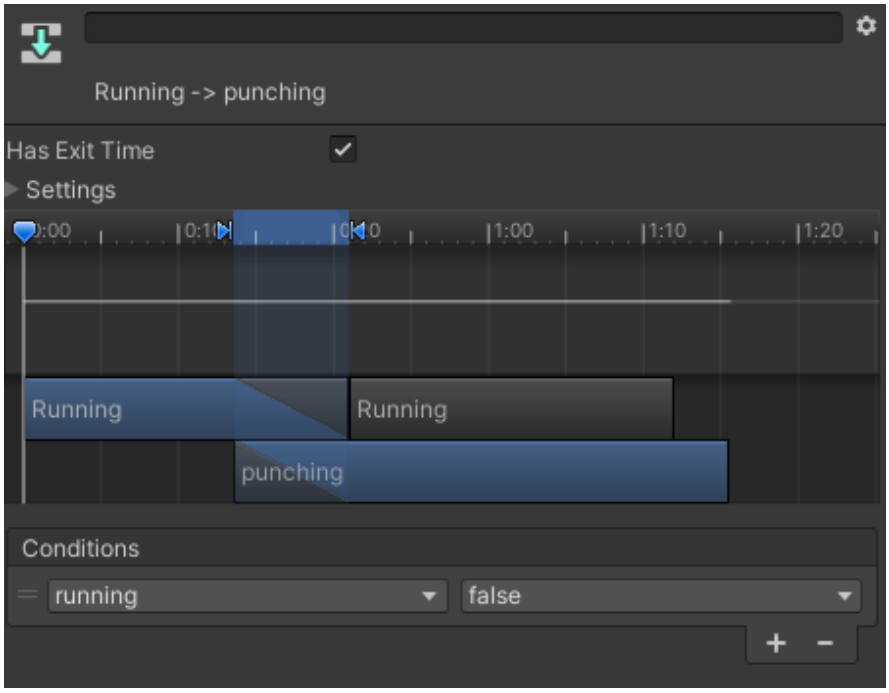

Рисунок 3.17 – Вікно налаштувань Transitions

Transitions – це переходи між анімаціями, у аніматорі вони позначаються зв'язками с направленням. «На рис. 3.17» можна побачити вікно налаштувань переходу з анімації running у punching. У графі Conditions зазначена булева змінна та при якому її значенні буде виконуватись перехід. Також Transitions має налаштування плавного переходу, тобто в момент переходу буде програватись частина першої та частина другої анімації, послідовно переростаючи повністю у другу. При значенні змінної running – false буде виконуватись перехід у анімацію punching та циклічне її виконання і навпаки.

Сама структура моба представляє собою його 3D модель, компонент колайдер для детекції взаємодії з іншими фізичними тілами та раніше створений компонент Animator.

3.6 Написання механік взаємодії

3.6.1 Механіка удару моба

Було задумано створити нанесення мобом шкоди головному персонажу шляхом удару рукою на близькій дистанції. Раніше було розглянуто саму анімацію удару моба. Для реалізації цього процесу потрібно було написати скрипт, що відслідковує момент часу самого удару в анімації та перевіряє наявність персонажа у певному радіусі від моба. Тільки при цих двох обставинах спрацьовує нанесення шкоди.

```
public Transform playerTransform;
public Transform enemyTransform;
public NavMeshAgent meshAgent;
public LayerMask whatIsPlayer;
public Animator animator;
float animationTime;
bool isHitting;
```

Рисунок 3.18 – Змінні скрипта enemy_movement.cs

Першим етапом написання скрипта логіки моба стало оголошення змінних, потрібних для обчислення (див. рис. 3.18):

- playerTransform – компонент положення у просторі гравця;
- enemyTransform – компонент положення у просторі моба;
- meshAgent – компонент для переміщення моба по рельєфу;
- whatIsPlayer – маска персонажа для його детекції;
- animator – компонент для управління анімацією моба;
- animationTime – змінна для відслідковування часу удару;
- isHitting – булева змінна для обчислення часу удару.

Далі було написано метод для отримання часу анімації.

```
public float GetCurrentAnimatorTime(Animator targetAnim, int layer = 0)
{
    AnimatorStateInfo animState = targetAnim.GetCurrentAnimatorStateInfo(layer);
    float currentTime = animState.normalizedTime % 1;
    return currentTime;
}
```

Рисунок 3.19 – Метод отримання часу анімації

Заздалегідь в редакторі анімації було виявлено момент часу нанесення удару. «На рис. 3.19» зображено кадр та час, виявлений у редакторі.

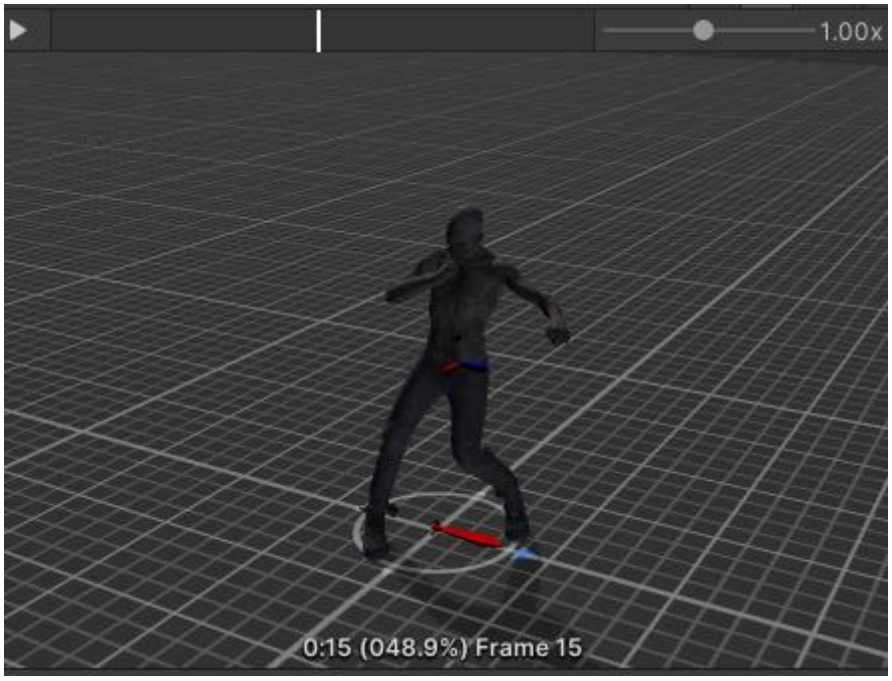

Рисунок 3.20 – Нанесення удару в анімації

«На рис. 3.20» зображено частину коду, яка відповідає за зупинення моба при достатній дистанції до персонажа та нанесенні удару. Для детекції персонажа у радіусі, викликається функція CheckSphere(), що приймає параметрами координати центу, її радіус та маску, яка повинна бути у радіусі. Далі виконується зупинення моба та задання змінній running значення false.

Також властивості enemyTransform.rotation присвоюється нове значення, що було обраховане для повороту моба за персонажем, коли він стоїть на місці. У змінну newAnimTime записано результат функції, що обраховує час анімації. Сам удар проходить коли значення newAnimTime не менше ніж момент часу удару та isHitting не має позитивне значення. Не можна порівнювати newAnimTime з моментом часу удару, так як показник кількості кадрів у секунду не стабільній та цей кадр програма може пропустити. Також при обрахуванні моменту удару isHitting становиться true до початку нового фрагменту анімації. Тобто доки повністю не закінчиться анімація удару, новий удар не може відбутися. Описаний алгоритм виконується в методі Update() (див. рис. 3.21).

```
if(Physics.CheckSphere(enemyTransform.position, 2.0f, whatIsPlayer))
{
    meshAgent.isStopped = true;
    animator.SetBool("running", false);

    Vector3 dir = playerTransform.position - enemyTransform.position;
    Quaternion lookRotation = Quaternion.LookRotation(dir);
    Vector3 rotation = Quaternion.Lerp( enemyTransform.rotation,
                                        lookRotation, Time.deltaTime * 100.0f).eulerAngles;
    enemyTransform.rotation = Quaternion.Euler(0f, rotation.y, 0f);

    float newAnimTime = GetCurrentAnimatorTime(animator);
    if(newAnimTime > 0.15f && !isHitting)
    {
        Debug.Log("HIT");
        isHitting = true;
    }
    if (newAnimTime < animationTime) isHitting = false;
    animationTime = newAnimTime;
}
```

Рисунок 3.21 – Фрагмент коду метода Update() скрипта

3.6.2 Механіка метання снарядів

Для реалізації цього процесу потрібно було створити сам об’єкт, який буде кидатись. Відповідно до існуючої локації, персонаж повинен кидати сніжки. Для імітації простої моделі снігової кульки було створено сферичну 3D модель та накладено на неї матеріал з текстурою снігу.

```
[Header("References")]
public Transform cam;
public Transform attackPoint;
public GameObject objectToThrow;

[Header("Settings")]
public float throwCooldown;

[Header("Throwing")]
public KeyCode throwKey = KeyCode.Mouse0;
public float throwForce;
public float throwUpwardForce;

bool readyToThrow;
```

Рисунок 3.22 – Поля скрипта Throwing.cs

Перед самим написанням скрипта потрібно було створити деякі параметри (див. рис. 3.22):

- cam відповідає за положення камери гравця у просторі;
- attackPoint - спеціальний об’єкт, створений у просторі, що визначає місце початку траєкторії снаряду;
- throwCooldown – змінна для управління часом перезарядки кидка;
- trowKey – клавіша, що буде зчитуватись для відслідковування початку кидка;

- throwForce, throwUpwardForce – застосовані до снаряду сили направлені вперед та вгору відповідно;
- readyToThrow – булева змінна для створення затримки між кидками.

```
readyToThrow = false;

GameObject projectile = Instantiate(objectToThrow, attackPoint.position, cam.rotation);

Rigidbody projectileRb = projectile.GetComponent<Rigidbody>();

Vector3 forceDirection = cam.transform.forward;
```

Рисунок 3.23 – Частина коду метода Throw()

На рис. «3.23» зображено спавн снаряду методом Instantiate(), що параметрами приймає об'єкт, який буде створено, координати спавну та поворот камери. Координатами спавну задано об’єкт, що було створенно на сцені та розположено його приблизно в районі випуску снаряду з руки персонажа. Змінна prjectileRb містить посилання на компонент Rigidbody снаряду.

Rigidbody – компонент об’єкта Unity, що дозволяє застосувати фізику твердого тіла об’єкта.

```
Vector3 forceToAdd = forceDirection * throwForce + transform.up * throwUpwardForce;

projectileRb.AddForce(forceToAdd, ForceMode.Impulse);

Invoke(nameof(ResetThrow), throwCooldown);
```

Рисунок 3.24 – Частина коду метода Throwing()

Було створено змінну для задання усіх сил в один параметр (див. рис. 3.24). Методом AddForce() було застосовано усі обраховані сили типом

імпульсу для снаряду. Функція Invoke() створює таймер з часом throwCooldown та при завершенні викликає метод ResetThrow(). Створений метод ResetThrow() задає значення true змінній readyToThrow. Метод Throw() виконується тільки при умові позитивного значення цієї змінної. При початку кидку їй присвоюється негативне значення. Тобто створюється так звана затримка між кидками.

Висновки до третього розділу

Здійснено розробку та програмну реалізацію 3D гри з алгоритмом пошуку A*. Було створено ігрову локацію з усіма задуманими її деталями. Додано та налаштовано персонажа з first person камерою, що пересувається по локації. Створено ворожих юнітів з усіма анімаціями, що знаходять шлях до персонажа за допомогою алгоритму A* та атакують його. Розроблено механіку кидка снаряду у ворожих юнітів для нанесення їм шкоди.

Спеціальний розділ

ОХОРОНА ПРАЦІ

до кваліфікаційної роботи

на тему:

«СТВОРЕННЯ 3D ГРИ З ВИКОРИСТАННЯМ АЛГОРИТМУ ПОШУКУ В СЕРЕДОВИЩІ UNITY»

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402.21810321

Виконав студент 4-го курсу, групи 402

Б. О. Супрун

(підпис, ініціали та прізвище)

«___» _____ 2022 р.

Консультант *ст. викладач*

(наук. ступінь, вчене звання)

Макарова О. В.

(підпис, ініціали та прізвище)

«___» _____ 2022 р.

Миколаїв – 2022

ВСТУП

Природне освітлення – це поєднання сонячного світла й дифузного світла небосхилу. Сонячне світло має величезне біологічне й гігієнічне значення, бактерицидні та оздоровчі властивості, дає змогу мати у приміщенні здоровий контакт з навколишнім світом.

Проектування природного освітлення має здатність до того, щоб раціонально використовувати природний ресурс сонячної енергії в даній місцевості.

Природне освітлення буває одно – або двостороннє бічне, що здійснюється через вікна у зовнішніх стінах, верхнє (через ліхтарі та отвори в дахах і перекриттях) та комбіноване (поєднання верхнього та бічного освітлення).

Проєктування природного освітлення має враховувати світлокліматичні особливості району будівництва, призначення будівель, характер зорової роботи.

Оскільки природне освітлення непостійне впродовж дня, кількісна оцінка цього виду освітлення проводиться за відносним показником коефіцієнтом природного освітлення (КПО).

Нормовані значення КПО визначаються «Будівельними нормами і правилами» (СНиП 11-4-79). В основі визначення КПО покладено розмір об’єкта розпізнавання, під яким розуміють предмет, що розглядається або ж його частину, а також дефект, який потрібно виявити.

Метою спеціальної частини є створення безпечних і здорових умов праці на робочому місці в процесі чого було проведено аналіз умов праці робочої кімнати, та запропоновано деякі заходи з покращення умов праці.

У даній роботі виконано перевірочний розрахунок природного освітлення для кімнати робочого місця.

4 СТВОРЕННЯ БЕЗПЕЧНИХ І ЗДОРОВИХ УМОВ ПРАЦІ НА РОБОЧОМУ МІСЦІ

4.1 Нормативні документи, які регулюють експлуатування комп'ютерів

Термінологія сфери інформаційних технологій прописана у державних стандартах серії ДСТУ ISO/IEC 2382 «Інформаційні технології. Словник термінів» (від ДСТУ ISO/IEC 2382-4:2005 «Частина 4. Організація даних» до ДСТУ ISO/IEC 2382-32:2003 «Частина 32. Електронна пошта»).

Експлуатують ЕОМ на підставі таких нормативно-правових актів:

- Правила охорони праці під час експлуатації електронно-обчислювальних машин (затверджені наказом Держгірпромнагляду від 26.03.2010 № 65; НПАОП 0.00-1.28-10; далі — Правила № 65);
- ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» (затверджені постановою Головного державного санітарного лікаря України від 10.12.1998 № 7; ДСанПіН 3.3.2.007-98).

Електронно-обчислювальна техніка — електротехнічний та електронний пристрій, який складається з ЕОМ і периферійних пристроїв, які передають, візуалізують або документують інформацію (наприклад через модем, дисплей, принтер).

ДСанПіН 3.3.2-007-98 розрізняє ЕОМ колективного використання та персональні ЕОМ. Ці правила стосуються таких професій, як програміст ЕОМ, оператор у залах обробки інформації й оператор комп'ютерного набору (дод. 1 ДСанПіН 3.3.2.007-98).

Вимоги щодо улаштування увідної мережі електроживлення ЕОМ регламентують Правила улаштування електроустановок (наприклад глава 1

«Загальні правила», глава 3 «Захист і автоматика»; затверджені наказом Міністерства палива та енергетики України від 21.07.2017 № 476; далі — ПУЕ). Правила технічної експлуатації електроустановок споживачів (затверджені наказом Міністерства палива та енергетики України від 25.07.2006 № 258; далі — ПТЕЕС) та Правила безпечної експлуатації електроустановок споживачів (затверджені наказом Держнаглядохоронпраці від 09.01.1998 № 4; НПАОП 40.1-1.21-98; далі — ПБЕЕС).

4.2 Небезпеки під час роботи з комп'ютерною технікою

На користувачів під час роботи з комп'ютерною технікою можуть діяти такі види небезпек:

- ураження електричним струмом;
- енергетична небезпека (виникає через коротке замикання: опіки, електрична дуга, викид розплавленого металу);
- небезпека загоряння;
- термонебезпека (дія високих температур через нагрівання конструктивних елементів);
- механічна небезпека (травми через падіння, дію рухомих частин, поріз за гострі частини конструктивних елементів);
- небезпека випромінювання (дія звукового (акустичного), високочастотного, інфрачервоного, ультрафіолетового й іонізуючого випромінювання, а також видимого світла когерентної високої інтенсивності (лазерного випромінювання);
- хімічна небезпека (контакт із деякими хімікатами, які використовують для того, щоб обслуговувати обладнання, або від вдихання їх парів).

4.3 Вимоги до безпеки експлуатування ІТ-засобів

Електробезпека будівель та приміщень, де розміщена комп'ютерна техніка, має відповідати вимогам ПБЕЕС (п. 1.4 розд. III Правил № 65). Базові вимоги до засобів та заходів захисту, які використовують для того, щоб запобігти ураженню електричним струмом від прямого й непрямого дотику, визначають пункти 1.7.55, 1.7.56 глави 1.7 «Заземлення і захисні заходи електробезпеки» ПУЕ.

Порядок присвоєння кваліфікаційної групи з електробезпеки на виробництві регламентують ПБЕЕС. Їх норми поширюються на працівників, які обслуговують електроустановки споживачів.

За Правилами № 65 не потрібно присвоювати групу з електробезпеки користувачам комп'ютера, який експлуатують у приміщеннях без підвищеної небезпеки, поза межами вибухо- та пожежонебезпечних зон.

ПБЕЕС та ПТЕЕС не поширюються на користувачів комп'ютера. Адже сучасний персональний комп'ютер не є електроустановкою (засобом виробництва із підвищеною небезпекою). Допуск до роботи з комп'ютером визначає Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці (затверджене наказом Держнаглядохоронпраці від 26.01.2005 № 15; НПАОП 0.00-4.12-05; далі — Типове положення № 15). Цей документ передбачає інструктаж з електробезпеки. Його реєструють у Журналі реєстрації інструктажів з питань охорони праці на робочому місці. Інструкція з електробезпеки має враховувати вимоги безпеки під час експлуатування комп'ютера (як електротехнічного пристрою). Періодичність інструктажу з електробезпеки залежить від встановленої на підприємстві періодичності повторних інструктажів із питань охорони праці: для робіт без підвищеної небезпеки — раз на шість місяців. Окрім того, у Типовому

тематичному плані і програмі навчання з питань охорони праці посадових осіб є тема 5 «Електробезпека» (дод. 4 Типового положення № 15).

Обслуговування (технічне) — комплекс робіт із підтримання працездатності обладнання в період його використання. До технічного обслуговування електрообладнання належать випробування обладнання, пристроїв, огляд обладнання, підтяжка контактних з'єднань (п. 3.1 розд. III ПТЕЕС)

До технічного обслуговування комп'ютерної техніки мають залучати обслугу — особу з відповідною технічною освітою і досвідом, що дає їй змогу усвідомлювати небезпеку, на яку вона може наразитися під час виконання робіт. Обслуга може застосовувати засоби щодо усунення небезпеки як до себе, так і до інших (п. 1.2.13.5 ДСТУ 4467-1:2005 «Апаратура обробляння інформації. Безпека. Частина 1. Загальні вимоги», затверджений наказом Держспоживстандарту від 05.10.2005 № 287). Користувачем або оператором є будь-яка особа, яка не належить до обслуги (п. 1.2.13.6 ДСТУ 4467-1:2005). Обслуговують (ремонт, перевірка технічного стану) ЕОМ електротехнічні працівники.

На підприємстві роботодавець може затвердити Перелік професій та посад працівників, яких звільняють від повторного інструктажу (далі — Перелік). До нього включають працівників, участь у виробничому процесі яких не пов'язана з безпосереднім обслуговуванням об'єктів, машин, механізмів, устаткування, застосуванням приладів та інструментів, збереженням або переробкою сировини, матеріалів тощо (п. 6.11 Типового положення № 15). Тобто до цього Переліку можуть бути включені й користувачі персональних ЕОМ.

4.4 Опис кімнати з робочим місцем

Приміщення кімнати має прямокутну форму. Воно розташоване у трьохповерховому будинку на третьому поверсі. У приміщенні два вікна, які розташовані з виходом на головну вулицю міста. На вікнах є вертикальні жалюзі. В інтер'єрі цієї кімнати переважає світлий колір. Поблизу вікна розташований світло коричневий письмовий стіл. На даному столі знаходиться комп`ютер, настільна лампа, ксерокс. Окрай стін стоїть шафа для документів. Підлога покрита лінолеумом сірого кольору. Стіни пофарбовані в зелений колір. Стеля білого кольору.

Геометрія розрахункового приміщення, розташування робочої поверхні і протилежної будівлі наведено «на рис. 4.1 – 4.3».

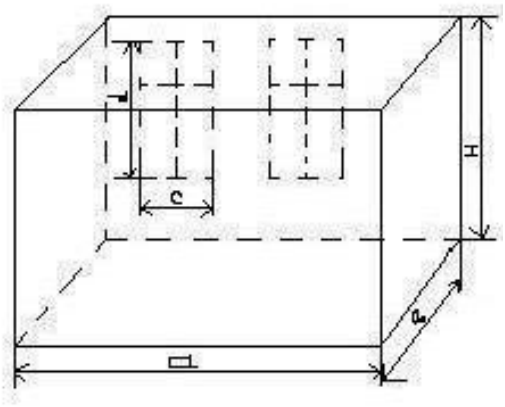

Рисунок 4.1 – Геометрія розрахункового приміщення

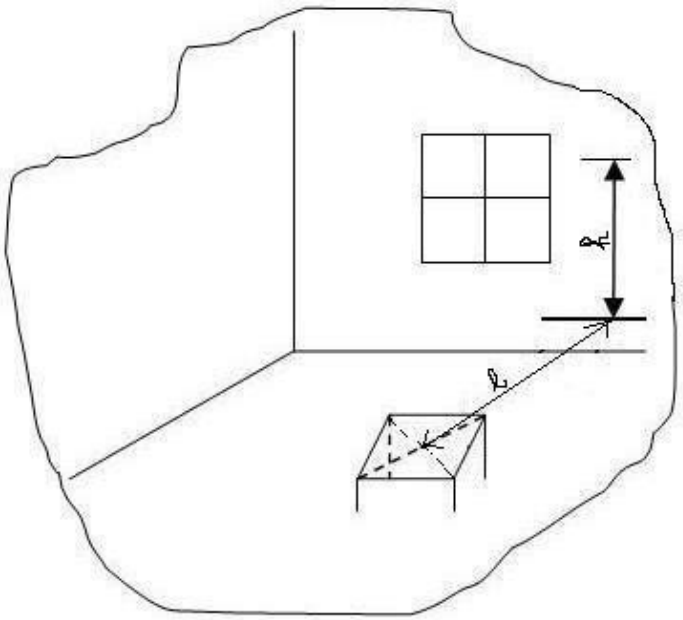

Рисунок 4.2 – Розташування робочої поверхні у приміщенні.

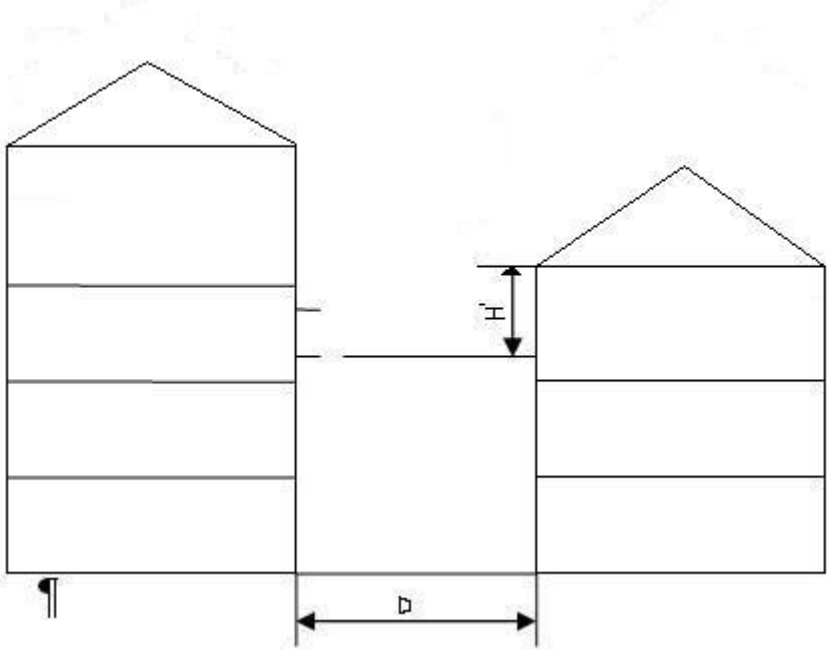

Рисунок 4.3 – Розташування протилежної будівлі

Далі «у табл. 4.1» наведемо чисельні значення геометричних параметрів представлених «на рис. 4.1 – 4.3».

Таблиця 4.1 – Значення геометричних параметрів

Величина	Значення
Довжина виробничого приміщення a, м	6.0
Ширина виробничого приміщення b, м	4.0
Висота виробничого приміщення H, м	3,2
Ширина вікна c, м	1,5
Висота вікна d, м	1,5
Висота верхньої кромки вікна відносно робочої поверхні h, м	2.0
Відстань середини робочої поверхні до зовнішньої стіни приміщення l, м	2.0
Відстань до протилежної будівлі D, м	20
Висота карнизу протилежної будівлі відносно підвіконня H', м	7.0

4.5 Перевірочний розрахунок природного освітлення для обраного робочого приміщення

Основним завданням при проектуванні природного освітлення є вибір типу та визначення розміщення і сумарної площі світлових отворів (вікон), при яких у приміщеннях забезпечується необхідний світловий режим. Для функціонуючого приміщення доцільно виконати перевірочний розрахунок природного освітлення з метою визначення відповідності існуючого рівня освітлюваності (площі вікон) вимогам нормативних документів.

Перевірочний розрахунок природнього освітлення виконується у такій послідовності.

Нормований коефіцієнт природного освітлення для III поясу світлового клімату $e_н^{III}$, %.

Визначається відповідно до СНиП–II–4–79 [5]. Для зорових робіт малої точності при боковому освітлені та найменшому розмірі об’єкту розпізнавання 1…5 мм (так званий IV розряд зорової роботи).

$$e_н^{III} = 1,5\% \quad (4.1)$$

Коефіцієнт сонячності клімату с. Для світлових отворів у зовнішніх стінах будівель, розташованих у IV поясі світлового клімату та зорієнтованих по азимуту в діапазоні 316...345 градусів згідно з даними [5].

$$c = 0,95 \quad (4.2)$$

Коефіцієнт світлового клімату, m. Для Миколаївської області, що належить до IV поясу світлового клімату згідно з даними [5].

$$m = 0,9 \quad (4.3)$$

Нормоване значення коефіцієнта природного освітлення для умов, що відповідають розрахунковому приміщенні.

$$e_н=e_н^{III} \cdot c \cdot m = 1 \cdot 0,95 \cdot 0,9 = 0,86\% \quad (4.4)$$

Коефіцієнт запасу, що враховується при виконанні розрахунку природнього освітлення $К_з$. Відповідно до рекомендації [5] $К_з$=1,3…1,5. Прийнято $К_з$=1,35.

Відношення довжини приміщення a до його ширини b a/b.

$$a/b=7.0/4.0=1.5 \quad (4.5)$$

Відношення ширини приміщення b до висоти верхньої кромки вікна відносно робочої поверхні h b/h.

$$b/h=4.0/2.0=2 \quad (4.6)$$

Світлова характеристика вікон $ή_в$. Визначається згідно з даними [5]. При a/b=1.5 та b/h=2.

$$ή_в=10.5 \quad (4.7)$$

Коефіцієнт світлопропускання для матеріалу вікна $τ_1$. Визначається згідно з данними [5]. Для подвійних склопакетних металопластикових вікон.

$$\tau_1=0.86 \quad (4.8)$$

Коефіцієнт, що враховує втрати світла у віконній рамі τ_2. Згідно з даними [5] для подвійних металопластикових рам вікна.

$$\tau_2=0.86 \quad (4.9)$$

Коефіцієнт, що враховує втрати світла у несучих конструкціях τ_3. При боковому освітленні.

$$\tau_3=1 \quad (4.10)$$

Коефіцієнт, що враховує втрати світла у сонцезахисних пристроях τ_4. Для внутрішніх вертикальних регульованих жалюзів.

$$\tau_4=1 \quad (4.11)$$

Загальний коефіцієнт світлопропускння $\tau_{заг}$

$$\tau_{заг} =\tau_1 \cdot \tau_2 \cdot \tau_3 \cdot \tau_4=0.86 \cdot 0.86 \cdot 1 \cdot 1=0.74 \quad (4.12)$$

Коефіцієнти відбиття внутрішніх поверхонь виробничого приміщення: стелі $\rho_{стелі}$, стін $\rho_{стін}$, підлоги $\rho_{підлоги}$, %.

Визначається в залежності від стану та кольору внутрішньої поверхні [5 табл. 3.8-3.10]:

- для стелі білого кольору $\rho_{стелі}$=65…85 % (прийнято $\rho_{стелі}$=85%);
- для стін пофарбованих у зелений колір $\rho_{стін}$=25…40 % (прийнято $\rho_{стін}$=41 %);
- для підлоги $\rho_{підлоги}$=10…40% (для лінолеуму бежевого кольору $\rho_{підлоги}$ = 38%).

Площі внутрішніх поверхонь виробничого приміщення: стелі $S_{стелі}$, стін $S_{стін}$, підлоги $S_{підлоги}$, м^2:

$$S_{стел\ і}=a \cdot b=6 \cdot 4=24 \quad (4.13)$$

$$S_{стін}=2 \cdot (a+b) \cdot H =2 \cdot (6+4) \cdot 3.2 = 64 \quad (4.14)$$

$$S_{підлоги}=24 \quad (4.15)$$

Середній коефіцієнт відбиття внутрішніх поверхонь виробничого приміщення $\rho_{сер}$:

$$\rho_{сер}= (\rho_{стелі} \cdot S_{стелі}+\rho_{стін} \cdot S_{стін}+\rho_{підлоги} \cdot S_{підл}) / (S_{стелі}+S_{стін}+S_{підл}) \times \quad (4.16)$$

$$\times 100\%=(85 \cdot 24)+(41 \cdot 64)+(38 \cdot 24) / (24+64+24) \cdot 100\%=0,49$$

Співвідношення, що визначають геометрію виробничого приміщення: a/b, b/h, l/b.

$$a/b=1.75;\ b/h=2;\ l/b=0,5; \quad (4.17)$$

Коефіцієнт, що враховує підсилення коефіцієнта природного освітлення у виробничому приміщенні завдяки світлу, що відбивається від внутрішніх поверхонь виробничого приміщення r_1.

Згідно з даними [5 табл. 3.7] $r_1=F(\rho_{сер,}\ a/b,\ b/h,\ l/b)$. При $\rho_{сер}=0,49$; a/b=1,5; b/h=2; l/b=0,5.

Відношення відстані до протилежної будівлі D до висоти карнизу протилежної будівлі відносно підвіконня H' D/ H' :

$$D/H'=20/7=2.8 \quad (4.18)$$

Коефіцієнт, що враховує вплив протилежної будівлі на освітленість у виробничому приміщенні $К_{буд.}$. Враховуючи дані [5 табл. 3.5]. При D/H'=2.8:

$$К_{буд} =1.03 \quad (4.20)$$

Площа вікон, що необхідна для забезпечення нормованого природного освітлення у розрахунковому виробничому приміщенні: $S_в$, м$^{2.}$.

$$S_в=(e_н \cdot К_з \cdot ή_в \cdot S_{підлоги} \cdot К_{буд}) / \tau_з \cdot r_1 \cdot 100 = \quad (4.21)$$

$$= (0,95*1,35*10,5*24*1,03)/1*0,8=4,24$$

У виробничому приміщенні вмонтовано два вікна розмірами с х d=1.5 х 1.5 м. Їх загальна площа складає 4,5 м$^{2.}$.

Таким чином сумарної площі вікон вистачає для забезпечення нормативного природного освітлення у розрахунковому приміщенні робочого місця.

Висновки до четвертого розділу

Було проведено розрахунки для обчислення сумарної площі вікон приміщення робочого місця, якої повинно бути достатньо для забезпечення нормативного природнього освітлення. Стало відомо, що загальна площа вікон робочого приміщення має більше значення ніж значення розрахованої допустимої площі на 0.26 м^2. Тобто розраховано, що розміри та кількість вікон у кімнаті задовольняють нормативні умови проєктування робочого приміщення.

Для створення сприятливих умов зорової роботи, які б виключали швидку втомлюваність очей, виникнення професійних захворювань, нещасних випадків і сприяли підвищенню продуктивності праці та якості продукції, виробниче освітлення повинно відповідати наступним вимогам.

1. Освітленість на робочих місцях має відповідати характеру зорової роботи (забезпечення необхідної освітленості робочих поверхонь поліпшує умови бачення об'єктів, підвищує продуктивність праці).
2. Рівномірний розподіл яскравості на робочій поверхні.
3. Відсутність різких тіней на робочих поверхнях (у полі зору людини різкі тіні спотворюють розміри й форми об'єктів розрізнення,

що додатково втомлює зір, а тіні, що рухаються, можуть призвести до травм).

4. Відсутність блисткості й засліпленості (блисткість викликає порушення зорових функцій, а засліпленість – призводить до швидкого втомлення зорового аналізатора і зниження працездатності людини).

5. Сталість освітленості в часі (коливання освітленості викликає переадаптацію ока, призводить до значного втомлення).

6. Надійність і простота в експлуатації, економічність та естетичність.

7. Правильна передача кольору (спектральний склад штучних джерел світла повинен бути максимально наближений до спектра природного освітлення).

ВИСНОВКИ

Комп’ютерні ігри можуть класифікуватися за різними критеріями. Відзначено та проаналізовано найпопулярнішу жанрову класифікацію відеоігор. Було розглянуто такі жанри, як Action, RPG, Strategy, Puzzle, Simulator та Arcade. Стало відомо, що комп’ютерні ігри поділяють за кількістю гравців та візуальним уявленням. Спираючись на жанрову класифікацію було обрано жанр та тематику гри.

Було проаналізовано алгоритми для вирішення задачі пошуку шляху та обрано алгоритм A*, який має високу обчислень завдяки евристичній функції.

Серед технологій та засобів розробки 3D комп’ютерної гри було використано гнучкий двигун для розробки відеоігор Unity. Його перевагами стали компонентно-орієнтований підхід розробки, досить простий для освоєння та функціональний Drag & Drop інтерфейс, якісна фізика твердих тіл та система LOD. Плюсом стало написання скриптів на потужній об’єктно-орієнтованій мові зі статичною типізацією C#. Для написання коду мови C# було використано відому середу розробки Visual Studio від Microsoft Corporation, яка не має конкурентів при створенні ПЗ із технологіями .Net.

Було здійснено розробку 3D гри в середовищі Unity. В ході розробки було створено програмну реалізацію алгоритму пошуку шляху A* для реалізації ШІ ворожого персонажа у грі. Було здійснено велику роботу для отримання необхідної атмосфери гри, шляхом створення деталізованого ландшафту. Для реалізації ворожих персонажів було завантажено якісну 3D модель та налаштовано анімації цього перснонажа.

Результатом розробки стала 3D коп’ютерна гра, що задовольняє поставлені до неї вимоги. Вона має безкінечний набір очок, атмосферний зимовий гірський сетінг, простий але інтенсивний геймплей.

У спеціальному розділі було проаналізовано організацію і ведення

роботи з охорони праці у загальноосвітньому закладі.

Поставлені завдання виконані повністю, але застосунок має пройти детальне його тестування для виявлення та усунення багів, які можуть виникати при незвичайних ситуаціях геймплею. Також, як будь-який продукт, гра має можливість її удосконалення. Такими моментами можуть стати:

- додавання симуляції снігопаду;
- відтворення слідів кроків на снігу;
- створення більш деталізованих 3D моделей;
- використання додаткових технологій для покращення зображення рендерингу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Хокинг, Джозеф. Unity — в действии. Мультиплатформенная разработка на C# : СПб : Питер, 2016. 336 с. ISBN 978-1617292323.

2. Документація для користувача з розробки відеоігор у середовищі Unity. Unity3d : вебсайт. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 23.04.2022).

3. Стаття про користь відеоігор. Knowable magazine: вебсайт. URL: https://knowablemagazine.org/article/mind/2019/video-games-educational-benefits (дата звернення: 24.04.2022).

4. Anguera, J. A., Boccanfuso, J., Rintoul, J. L., Al-Hashimi, O., Faraji, F., Janowich, J., et al. Video game training enhances cognitive control in older adults. Nature 501, 2013, 97–101. doi: 10.1038/nature12486.

5. Технічна документація Microsoft. Microsoft ; вебсайт. URL: https://docs.microsoft.com/ru-ru/ (дата звернення: 04.05.2022).

6. Донован Тристан. Играй! История видеоигр: Москва : Белое Яблоко, 2014. 782 с.

7. Поняття відеогра та його історія. Wikipedia : вебсайт. URL: https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%B5%D0%BE%D0%B3%D1%80%D0%B0 (дата звернення: 28.04.2022).

8. Tresca, Michael J. The Evolution of Fantasy Role-Playing Games : McFarland, 2010. P. 238.

9. Стаття про опис гри Warcraft III: Game revolution : вебсайт. URL: https://www.gamerevolution.com/review/36318-warcraft-iii-reign-of-chaos-review (дата звернення 30.04.2022).

10. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms : The MIT Pess : Massachusetts, 2009. P. 1292.

11. Стаття про оновлення версій C#. Microsoft : вебсайт. URL: https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-10 (дата звернення 03.05.22).

12. Докуметнація мови C#. Microsoft : вебсайт. URL: https://docs.microsoft.com/en-us/dotnet/csharp/ (дата звернення 04.05.22).

13. Стаття релізу змін у версії Visual Studio 2022. Microsoft : вебсайт. URL: https://docs.microsoft.com/en-us/visualstudio/releases/2022/release-notes-preview (дата звернення 04.05.22).

14. Alex Okita. Learning C# Programming with Unity 3D : A. K. Peters, : Ltd.63 South Avenue Natick : MA United States, 2014. P. 690.

15. Стаття про Unity Asset Store. Unity3d.com : вебсайт. URL: https://unity3d.com/ru/quick-guide-to-unity-asset-store (дата звернення 01.06.22).

16. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Frame work 2.0 на языке C#. Мастер класс : пособие / пер. с англ. издательство «Русская Редакция». СПб. : Питер , 2008. 656 с.

17. Правила охорони праці під час експлуатації електронно-обчислювальних машин. Будстандарт : вебсайт. URL: http://online.budstandart.com/ru/catalog/doc-page?id_doc=27405 (дата звернення 10.06.22).

18. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. Верховна рада України : вебсайт. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text (дата звернення. 11.06.22).

19. Про затвердження Правил безпечної експлуатації електроустановок споживачів. Верховна рада України: вебсайт. URL: https://zakon.rada.gov.ua/laws/show/z0093-98#Text (дата звернення 11.06.22).

20. Грибан В. Г., Негодченко О. В. Охорона праці: навч. посібник. [для студ. вищ. навч. закл.] / В. Г. Грибан, О. В. Негодченко — К.: Центр учбової літератури, 2009. — 280 с.

21. Практикум із охорони праці / В. Ц. Жидецький, В. С. Джигирей, В. С. Сторожук та ін. – Львів: Афіша, 2000. – 352 с.

ДОДАТОК А

Лістинг коду застосунку

Скрипт mouse_look:

```
using UnityEngine;

public class mouse_look : MonoBehaviour
{

    private float mouseSensitivity = 300f;
    public Transform playerBody;
    float xRotation = 0f;

    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
    }

    // Update is called once per frame
    void Update()
    {
        float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;

        xRotation -= mouseY;
        xRotation = Mathf.Clamp(xRotation, -90f, 90f);

        transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
        playerBody.Rotate(Vector3.up * mouseX);

    }
}
```

Скрипт player_movement:

```
using UnityEngine;

public class player_movement : MonoBehaviour
{
    public CharacterController controller;

    private float speed = 12f;
    private float gravity = -9.81f;
    private float jumpHeight = 2f;

    public Transform groundCheck;
    private float groundDistance = 0.4f;
    public LayerMask groundmask;

    Vector3 velocity;
    bool isGrounded;
    // Update is called once per frame
```

```
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftShift)) speed = 24f;
        else speed = 12f;

        isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundmask);

        if (isGrounded && velocity.y < 0)
        {
            velocity.y = -2f;
        }

        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");

        Vector3 move = transform.right * x + transform.forward * z;

        controller.Move(move * speed * Time.deltaTime);

        if (Input.GetButtonDown("Jump") && isGrounded)
        {
            velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravity);
        }

        velocity.y += gravity * Time.deltaTime;

        controller.Move(velocity * Time.deltaTime);


    }
}
```

Скрипт throwing:

```
using UnityEngine;

public class Throwing : MonoBehaviour
{
    [Header("References")]
    public Transform cam;
    public Transform attackPoint;
    public GameObject objectToThrow;

    [Header("Settings")]
    public float throwCooldown;

    [Header("Throwing")]
    public KeyCode throwKey = KeyCode.Mouse0;
    public float throwForce;
    public float throwUpwardForce;

    bool readyToThrow;

    private void Start()
    {
        readyToThrow = true;
```

```
    }
    private void Update()
    {
        if (Input.GetKeyDown(throwKey) && readyToThrow)
        {
            Throw();
        }
    }
    private void Throw()
    {
        readyToThrow = false;

        GameObject projectile = Instantiate(objectToThrow, attackPoint.position, cam.rotation);
        Rigidbody projectileRb = projectile.GetComponent<Rigidbody>();
        Vector3 forceDirection = cam.transform.forward;
        RaycastHit hit;
        if (Physics.Raycast(cam.position, cam.forward, out hit, 500f))
        {
            forceDirection = (hit.point - attackPoint.position).normalized;
        }
        Vector3 forceToAdd = forceDirection * throwForce + transform.up * throwUpwardForce;
        projectileRb.AddForce(forceToAdd, ForceMode.Impulse);
        Invoke(nameof(ResetThrow), throwCooldown);
    }

    private void ResetThrow()
    {
        readyToThrow = true;
    }
}
```

Клас Vertex.cs:

```
using System;
using System.Collections.Generic;

public class Vertex : IComparable
{
    public Vertex(int index, Point coordinates, float height)
    {
        Index = index;
        Coord = coordinates;
        AdjVerts = new List<Vertex>();
        InOpenList = false;
        Height = height;
    }
    public void calculateGX(Vertex newPath)
    {
        float distanceToPast;
        if ((newPath.Coord.X - Coord.X) != 0 && (newPath.Coord.Y - Coord.Y) != 0) distanceToPast =
(float)Math.Sqrt(Math.Sqrt(2) + Math.Pow(Math.Abs(Height - newPath.Height), 2));
        else distanceToPast = (float)Math.Sqrt(1 + Math.Pow(Math.Abs(Height - newPath.Height), 2));
        float newGX = distanceToPast + newPath.GX;

        if (PathFrom == null || newGX < GX)
        {
```

```
            PathFrom = newPath;
            GX = newGX;
        }
    }
    public void calculateHX(Vertex endVertex)
    {
        HX = (float)Math.Sqrt(Math.Pow(Math.Abs(Coord.X - endVertex.Coord.X) * 10.0f, 2)
                        + Math.Pow(Math.Abs(Coord.Y - endVertex.Coord.Y) * 10.0f, 2));
    }
    public int CompareTo(object obj)
    {
        if (obj == null) return 1;

        Vertex otherVertex = obj as Vertex;
        if (otherVertex != null)
            return (GX + HX).CompareTo(otherVertex.GX + otherVertex.HX);
        else
            throw new ArgumentException("Object is not a Vertex!");
    }
    public int Index { get; set; }
    public Point Coord { get; set; }
    public Vertex PathFrom { get; set; }
    public List<Vertex> AdjVerts { get; set; }
    public float GX { get; set; }
    public float HX { get; set; }
    public float Height { get; set; }
    public bool InOpenList { get; set; }
}
```

Клас Pathfinder:

```
using System.Collections.Generic;

public class Pathfinder
{
    public Pathfinder(Vertex startVertex, Vertex endVertex)
    {
        this.startVertex = startVertex;
        this.endVertex = endVertex;

        openList = new List<Vertex>();
    }

    public int findPath()
    {
        openList.Add(startVertex);
        Vertex tmpVertex = startVertex;

        while (true)
        {
            //Console.WriteLine(tmpVertex.Index.ToString());
            if (tmpVertex.Equals(endVertex)) return 1;

            for (int i = 0; i < tmpVertex.AdjVerts.Count; i++)
            {
                Vertex tmpAdjVertex = tmpVertex.AdjVerts[i];
```

```
            tmpAdjVertex.calculateGX(tmpVertex);
            tmpAdjVertex.calculateHX(endVertex);

            for (int j = 0; j < tmpAdjVertex.AdjVerts.Count; j++)
            {
               if (tmpAdjVertex.AdjVerts[j].Index == tmpVertex.Index)
               {
                  tmpAdjVertex.AdjVerts.RemoveAt(j);
                  j = tmpAdjVertex.AdjVerts.Count;
               }
            }

            if (!tmpAdjVertex.InOpenList)
            {
               openList.Add(tmpAdjVertex);
               tmpAdjVertex.InOpenList = true;
            }
         }
         openList.RemoveAt(0);
         openList.Sort();

         if (openList.Count == 0) return 0;
         tmpVertex = openList[0];
      }
   }

   public List<Point> getPath()
   {
      List<Point> path = new List<Point>();

      Vertex tmpVertex = endVertex;
      while (tmpVertex != null)
      {
         path.Add(tmpVertex.Coord);
         tmpVertex = tmpVertex.PathFrom;
      }

      return path;
   }

   private List<Vertex> openList;
   private Vertex startVertex;
   private Vertex endVertex;
}
```