Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК 004.83

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: “Розробка мультимедійного додатку з використанням штучного інтелекту”

Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ – 44.00.00.000

Студент
ІПЗ-43 _____ /Олена ТИЩЕНКО/

Науковий керівник
к.т.н., ас. _____ /Максим ТКАЧЕНКО/

Консультант
з питань нормоконтролю
фахівець _____ /Тамара ЧАПОВСЬКА/

Допускається до захисту
Завідувач кафедри
д.т.н., проф. _____ /Олексій БИЧКОВ/

Київ – 2021

Рішенням Екзаменаційної комісії

випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії

професор, доктор техн. наук Віктор ВИШНІВСЬКИЙ

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:
Завідувач кафедри програмних систем і технологій
_____/Олексій БИЧКОВ/
(підпис) (розшифровка підпису)
„___”_____2021р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ
СТУДЕНТУ

Тищенко Олені Сергіївні
(прізвище, ім’я, по-батькові)

1. Тема випускної кваліфікаційної бакалаврської роботи “Розробка мультимедійного додатку з використанням штучного інтелекту”

керівник проекту (роботи) Ткаченко Максим Васильович, к.т.н.,асистент

затверджені наказом вищого навчального закладу від „___” _____ 20__ р. № ___

2. Строк здачі студентом закінченої роботи _____

3. Вихідні дані до роботи Створений мультимедійний додаток за допомогою використання штучного інтелекту.

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)

1. Аналіз предметної області створення мультимедійних додатків;
2. Аналіз інструментальних засобів розробки;
3. Проектування мультимедійного додатку. Визначення: структури, жанру, сетингу, правил, інтерфейсу;
4. Реалізація проекту. Опис алгоритмів (штучного інтелекту), компонентів, матеріалів, які застосовувалися при реалізації;

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)

1. Прототип визначеного сетингу (Рис. 2.3.1, ст. 27);
2. Прототипи сцен додатку: головного меню (Рис. 2.5.1., ст. 30), меню опцій (Рис. 2.5.2, ст. 30), вигаданого всесвіту (Рис. 2.5.3, ст. 31);
3. Реалізація сцени віртуального світу (Рис. 3.1.4.2, ст. 39);
4. Графічне зображення об’єкту ворог (Рис. 3.2.6.4, ст. 50);
5. Блок-схема алгоритму роботи програми (додаток Б).

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____ /Максим ТКАЧЕНКО/
(підпис) *(розшифровка підпису)*

Завдання прийняв до виконання _____/Олена ТИЩЕНКО/
(підпис) *(розшифровка підпису)*

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір та вивчення літератури за обраними напрямками	28.12.2020	виконано
2	Аналіз концепцій та засобів розробки	15.01.2021	виконано
3	Проектування мультимедійного додатку	30.01.2021	виконано
4	Створення алгоритмів та план дій реалізації	20.02.2021	виконано
5	Реалізація мультимедійного додатку. Аналіз результатів	30.03.2021	виконано
6	Оформлення дипломної роботи	20.04.2021	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри		виконано

Студент – бакалавр _____/Олена ТИЩЕНКО/
(підпис) *(розшифровка підпису)*

Керівник роботи _____/Максим ТКАЧЕНКО/
(підпис) *(розшифровка підпису)*

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 71 сторінки, 21 рисунків, 3 додатків, 26 джерела.

Тема: Розробка мультимедійного додатку з використанням штучного інтелекту.

Об’єкт дослідження: приклади розробки штучного інтелекту в сучасних мультимедійних додатків (3D-шутерах).

Мета роботи: проаналізувати сучасні підходи до створення мультимедійних додатків, сценаріїв штучного інтелекту, алгоритмів пошуку, внаслідок реалізувати мультимедійний додаток .

Предмет дослідження: алгоритми створення штучного інтелекту, сценарії використання штучного інтелекту, методи реалізації алгоритмів пошуку.

Результати дослідження:

Досліджено можливості застосування засобів розробки та сценарію штучного інтелекту для реалізації мультимедійного додатку. Обрано актуальний жанр додатку - 3D-шутер, який користується попитом в сучасній індустрії.

Висновок

В результаті виконання роботи було описано алгоритми для створення додатку: поведінки користувача, ворогів та взаємодія з віртуальним світом на основі яких було реалізовано додаток.

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 71 страницы, 21 рисунков, 3 дополнений, 26 источника.

Тема: Разработка мультимедийного приложения с использованием искусственного интеллекта.

Объект исследования: примеры разработки искусственного интеллекта в современных мультимедийных приложений (3D-шутерах).

Цель работы: проанализировать современные подходы к созданию мультимедийных приложений, сценариев искусственного интеллекта, алгоритмов поиска, вследствие реализовать мультимедийный приложение.

Предмет исследования: алгоритмы создания искусственного интеллекта, сценарии использования искусственного интеллекта, методы реализации алгоритмов поиска.

Результаты исследования:

Исследованы возможности применения средств разработки и сценария искусственного интеллекта для реализации мультимедийного приложения. Избран актуальной жанр приложения - 3D-шутер, который пользуется спросом в современной индустрии.

Вывод

В результате выполнения работы были описаны алгоритмы для создания приложения: поведения пользователя, врагов и взаимодействие с виртуальным миром на основе которых было реализовано приложение.

ANNOTATION

Final qualifying bachelor's thesis: 71 pages, 21 figures., 3 additions, 26 sources.

Topic: Development of a multimedia application using artificial intelligence.

Object of research: examples of the development of artificial intelligence in today's multimedia applications (3D-shooters).

Purpose: to analyze modern approaches to the creation of multimedia applications, artificial intelligence scenarios, search algorithms, as a result of implementing a multimedia application.

Subject of research: algorithms for creating artificial intelligence, scenarios for using artificial intelligence, methods of implementing search algorithms.

Research results:

Possibilities of application of means of development and the scenario of artificial intelligence for realization of the multimedia application are investigated. The current genre of the application is chosen - 3D-shooter, which is in demand in modern industry.

Conclusion

As a result of the work, the algorithms for creating the application were described: user behavior, enemies and interaction with the virtual world on the basis of which the application was implemented.

ЗМІСТ

ВСТУП

Актуальність роботи

Індустрія з розробки мультимедійних додатків не стоїть на місці і постійно розвивається. Більшість людей використовують додатки в різних сферах життя. Розглядаючи категорію ігор та розваг, можна дійти до висновку, що вона величезна і дуже конкурентоспроможна. Такі програми популярні серед розробників, оскільки приваблюють увагу користувачів кілька разів на тиждень, іноді кілька разів на день.

У найбільш вдалих додатках як частота, так і тривалість знаходження являються великими показниками. Важливо створити додаток якомога захоплюючим, пропонуючи заохочення користувачам, які можуть повертатися щодня або певну кількість днів поспіль. Тому цей підтип зможе надати позитивний вплив на бізнес та залучить велику кількість клієнтів.

Зв'язок роботи з науковими програмами, планами, темами

Розробка та впровадження мультимедійних додатків знаходить своє застосування в різних областях, включаючи рекламу, бізнес, освіту, індустрію розваг, медицину, наукові дослідження та інші.

Тема є досить актуальною в області освіти. Певним чином навіть освіта програє розважальним додаткам. Основна проблема - відсутність інтересу до навчального матеріалу. Більшість сучасних онлайн-курсів, систем дистанційного навчання містять інформацію, але подаються без візуалізації та інтерактивності.

Мета і задачі дослідження

Метою бакалаврської роботи є проаналізувати сучасні підходи до створення: мультимедійних додатків, сценаріїв штучного інтелекту, алгоритмів пошуку, внаслідок реалізувати мультимедійний. Реалізований продукт має включати в себе задачі, які поставлені до відповідного жанру – шутеру (алгоритми та сценарії поведінки користувача, ворогів та взаємодія з віртуальним світом).

Для досягнення мети було розв’язано такі задачі:

1) аналіз предметної області. Огляд існуючих популярних жанрів мультимедійних додатків;
2) аналіз засобів реалізації додатку. Розглянуті ігрові рушії: Unity, Unreal Engine, GameMaker: Studio, Godot;
3) вивчення концепції дій та поведінки персонажів додатку;
4) формування чітких вимог до програми та визначення плану програми (що робить користувач та що робить програма у відповідь);
5) проектування структури програми;
6) аналіз існуючих алгоритмів пошуку шляху;
7) реалізація та опис використання алгоритму для пошуку шляху на основі противників у додатку;
8) створення та реалізація алгоритму роботи програми. Зображення діаграми в додатку Б.

Об’єктом дослідження є приклади розробки штучного інтелекту в сучасних мультимедійних додатків (3D-шутерах).

Предметом дослідження є алгоритми створення штучного інтелекту, сценарії використання штучного інтелекту, методи реалізації алгоритмів пошуку.

Новизна одержаних результатів

Під час розробки дипломного проекту було досліджено: сценарії створення штучного інтелекту для персонажів-противників, алгоритми роботи програми відповідно до жанру. Одержано новий алгоритм-сценарій роботи мультимедійного додатку. Досягнена програмна реалізація алгоритму (зображено в додатку Б).

Практичне значення одержаних результатів.

Одержана реалізація алгоритму згідно сценарію виконання, що дозволяє запускати мультимедійний додаток на платформі Windows. Додаток представляє собою віртуальний світ (ліс з різноманітними об’єктами та ворогами). Персонаж має знищувати або втікати від ворогів, щоб вижити зважаючи на показники здоров'я та енергії. Також присутня зброя, якою можна користуватися для захисту.

Результати виконаної дипломної роботи впроваджено на підприємстві ТОВ “МІТ”.

Особистий внесок студента.

Основним результатом є:

1. запропонований підхід до можливої реалізації додатку в розважальній предметній області;

2. реалізація алгоритму, що зображує новий віртуальний світ зі своїми: сценами, правилами, історією, сетингом та дизайном.

Публікації

Під час роботи над бакалаврською роботою було підготовлено тези для восьмої Східно-Європейської конференції на тему «Розробка мультимедійного додатку за допомогою використання штучного інтелекту та інтернету речей», що надрукована у збірнику узагальнених матеріалів восьмої Східно-Європейської конференції в секції «Математичні та програмні технології Internet of Everything» (14.05.2021, Київ).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МУЛЬТИМЕДІЙНИХ ДОДАТКІВ

1.1 Мультимедійні додатки

Мультимедіа - це інформаційне поєднання вмісту текстової, звукової, графічної та інтерактивної інформації для різних видів інформування або розваги користувачів.

Мультимедійний додаток - це інструмент створений для просування реклами товарів, послуг або розважального контенту користувачів. При якісному поєднанні всіх видів мультимедійного представлення можливо домогтися вигідного сприйняття інформації користувачем.

В мультимедійній сфері одну з важливих ролей відіграють мультимедійні комп’ютерні ігри. В таких іграх стан віртуального світу передається користувачеві за допомогою взаємодії різних способів поєднання інформації. На сьогоднішній день майже всі комп’ютерні ігри відносяться до мультимедійних ігор.

Оформлено згідно джерела [1].

1.2 Історія розвитку штучного інтелекту в мультимедійних додатках

Штучний інтелект - це відображення людського мислення.

Насправді штучний інтелект - це не що інше, як набір алгоритмів, який виконує дії за вітками умов: якщо, якщо інакше або інакше. В кінцевому випадку призводить до позитивної реакції та ілюзії користувача, але це лише реалістичний та гарно продуманий набір виконуваних команд. Найуспішніший приклад - інші люди у грі.

Поняття штучного інтелекту почало використовуватися в 1956 році, його визначення не було пов'язане безпосередньо з поняттям інтелекту. На мові оригіналу вживається, як Artificial Intillegence.

Перший мультимедійний додаток з штучним інтелектом був випущений в 1951 році до визначення самого поняття штучного інтелекту. Він створений, як противник для гри в шахи і має певний перелік алгоритмів та рівні складності. На сьогоднішній день використовуються шах-боти можна спостерігати на турнірах в різних країнах світу, де вони отримують подальший розвиток і застосовуються.

У 1960-1970-х роках більшість додатків створювались, як ігри для двох гравців (реалізація настільних ігор).

Системи штучного інтелекту можна розділити на два типи - локальний та глобальний.

Локальний – це штучний інтелект окремої одиниці (наприклад, ворог або механізм) Така система має чотири основних елементи: введення інформації, пам'ять, прийняття рішення, реакція.

Введення інформації відповідає за те, щоб об'єкт розпізнав обстановку (всі сторони навколишнього світу: цільову область, перешкоди, положення ворога та інше). Враховуючи зовнішні фактори, об'єкт визначає, що робити за допомогою наявних алгоритмів.

Глобальний штучний інтелект зазвичай використовується в жанрі стратегія. Найпоширеніший приклад - це контроль та напад армії на противника. Але незважаючи на цілісність групи об’єктів, локальні елементи можуть добровільно виконувати інші завдання, встановлені алгоритмом. Наприклад, якщо армія дістанеться до певної локації з завданням атаки, певна кількість солдат можуть втекти або загубитися через вплив інших факторів.

У 1972 році вийшла текстова відеогра полювання на Вампуса створена Георгом Йобом. В якості штучного інтелекту присутні вороги з шаблонами списків дій. Суть гри полягала в тому, щоб пробратися в лігво Вампуса і знищити.

1980 рік - виходить культова гра Pac-Man, суть якої - зібрати всі крапки в лабіринті, обходячи перешкоди у вигляді привидів. В грі присутні чотирьох різних шаблонів-алгоритмів поведінки привидів (Блінкі, Пінкі, Інкі, Клайд).

З 1990 року почалася епоха виходу відеоігор Mortal Kombat і Street Fighter, закінчуючи Soul Calibur. Головна інновація того часу для ігрового штучного інтелекту - модернізація алгоритмів в реальному часі.

Гра Golden Eye 007 вийшла у 1997 році– це шутер, в якій штучний інтелект реагує на дії та рухи гравця. Головний недолік в тому, що алгоритм побудований на точному знаходженні користувача.

Шутер Far Cry від першої особи вийшов у 2004 році продемонстрував найдосконаліший на той час штучний інтелект. Суперники реагують на стиль гравця і максимально оточують їх. У битві з гравцем вороги використовують справжню військову тактику.

Проект Left 4 Dead вийшов у 2008 році. Новизна полягала у системі глобального штучного інтелекту, під час якого присутня система адаптації складності, які впливають на кількість ворогів та вплив на користувача.

Штучний інтелект продовжує розвиватися з метою досягти такого рівня, щоб гравець був нездатний відрізнити його від людського.

Використані джерела [2-3].

1.3 Огляд та аналіз ігрових жанрів

В сучасній індустрії комп’ютерних ігор існує велика кількість жанрів. В рамках дипломної робити були розглянуті найпопулярніші жанри, використавши літературні джерела [4,5]:

- Action (поведінка) - основою гри є взаємодія гравця з іншими предметами. Успіх гравця у таких іграх залежить від швидкості реакції та здатності швидко приймати відповідні рішення. Переходячи від комп’ютерних ігор до реального світу, цей жанр дуже схожий на ігри в футбол, волейбол, баскетбол тощо. Єдина відмінність полягає в тому, що непотрібно напружувати м’язи;

• RPG (рольові ігри) – жанр має добре структуровані сюжетні лінії та велику кількість взаємодій із зовнішнім світом. Знову ж таки, в реальному світі ці жанри схожі на читання книг чи перегляд хороших фільмів. Найголовнішою цілю є створення атмосфери, де гравці можуть спробувати у ролі персонажа все, що відбувається;

• Strategy (стратегія) – цей жанр вимагає гравців мати такі рисих характеру, як: спокій, логіку та розум. Граючи в таку гру, гравцям можуть запропонувати взяти на себе роль командира, містобудівника або правителя своєї власної країни. Існує два підтипи стратегій:

- Стратегія в режимі реального часу - такий тип жанру означає, що ви не можете змінювати час під час гри (крім меню пауз). Всі рішення гравець має приймати швидко і без можливості на роздуми;
- Покрокова стратегія - гра поділяється на такти, в кожному такті гравець може зробити обмежений набір дій, після якого хід переходить опонентові. Такий підхід схожий з грою в настільні ігри, наприклад в шахи.

• Mass Multiplayer (багатокористувацька гра) - цей жанр взаємодіє з великою кількістю гравців в межах віртуального світу чи ігровій сесії розташованого на ігровому сервері;

• Sport game (спортивні) – жанр полягає в реалізації будь-якої спортивної гри, найбільшого поширення набули імітації: футболу, хокею, баскетболу, тенісу, гольфу та більярду;

• Adventure (пригоди) - гра-розповідь в якій герой просувається по сюжету і взаємодіє з ігровим світом за допомогою застосування предметів, спілкування з іншими персонажами і рішення логічних задач. Приклади: Siberia, Myst, The Longest Journey;

• 3D Shooter (3D-шутери) – в цьому жанрі гравець, як правило, діє поодинці, в більшості випадків повинен знаходити та знищувати ворогів за допомогою знайденої або наданою грою зброєю, для досягнення цілей на певному рівні, після досягнення всіх заданих цілей гравець, може переходити на вищий рівень або завершити гру;

- Fighting (бійки) – основний сюжет гри (геймплей) складається виключно з поєдинків двох і більше противників із застосуванням рукопашного бою. Приклади: Mortal Kombat, Street Fighter;
- Racing (гоночна гра)- жанр комп'ютерних ігор від першої або від третьої особи, в яких гравець бере участь в гоночному змаганні серед наземних, водних, повітряних або космічних транспортних засобів;
- Simulators (симулятори) – жанр гри, завдання якого полягає в імітації управління будь-яким процесом, апаратом або транспортним засобом. За допомогою комп'ютерно-механічних симуляторів, можна абсолютно точно реалізувати відтворення графічне відображення деякого апарату або процесу, завдяки цьому можна здійснювати тренування: хірургів, пілотів, космонавтів, машиністів автомобілів та поїздів;
- Arcade (аркадні) – жанр гри, який в більшій мірі пішов від симуляторів, точніше являється більш спрощеною версією технічних симуляторів, нерідко з альтернативної фізикою. Найчастіше з подібною фізикою робляться симулятори космічних кораблів і автомобілів. Приклади: X-Wing, TIE-Fighter, Wing Commander, Need for Speed;
- Puzzle (головоломки) – жанр ігор, головною цілю яких є вирішення логічних завдань, що чекають від користувача задіяння інтелектуальних рис характеру : кмітливість, логіка, винахідливість та ерудицію. Найчастіше використовується як включення до ігор інших жанрів як відволікаючі елементи від основного ігрового процесу або для урізноманітнення гри.

На сучасному ринку ігрової індустрії нечасто зустрічаються додатки конкретного жанру. Всі перераховані жанри можуть переплітатися між собою від проекту до проекту. Таким чином, майже всі сучасні комп’ютерні ігри вміщають в собі декілька ігрових жанрів.

1.4 Аналіз ігрових рушіїв

Ігровий рушій – це програмна платформа, яка створена для розробки ігор, як і будь-яке інше інтегроване середовище розробки орієнтоване на конкретні мови

програмування [6]. Компоненти ігрового рушія побудовані для підтримки та розвитку гри. Більшість рушіїв мають вбудовані компоненти, які можуть суттєво допомогти користувачеві в розробці такі, як:

- Графічний движок – програмний модуль, який відповідає за візуалізацію двомірної або тривимірної графіки;
- Фізичний движок – програмний модуль, який відповідає за моделювання фізичних законів у віртуальному світі;
- Звуковий движок – програмний модуль, який відповідає за відтворення звуків – музичний супровід, голоса персонажів та інші можливі звукові ефекти;
- Ігровий штучний інтелект – методи або алгоритми, які можна використати для створення ілюзії наявності поведінки інтелекту в певних явищах або персонажів.

Аналіз ігрових рушіїв виконано на основі літературних джерел [7-9]:

1.4.1 Unity

Unity - це програмний інструмент для розробки двовимірних та тривимірних додатків та ігор під Windows, Linux та OS X. Створені за допомогою Unity додатки функціонують на Windows, OS X, Windows Phone, Android, Apple iOS, Linux, а також на ігрових консолях для Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One та MotionParallax3D. Можна створити програми з використанням браузерів за допомогою спеціального модуля Unity (Unity Web Player) або за допомогою реалізації технології WebGL.

Середовище розробки Unity Editor має дуже простий інтерфейс Drag&Drop, який можна легко налаштувати і складається з великої кількості різних вікон, так що ви можете налагоджувати гру безпосередньо в редакторі. Ігровий рушій підтримує три сценарні мови програмування: C#, JavaScript. Раніше була підтримка Boo (Python dialect), але її було видалено у 5-й версії. Розрахунки фізики здійснює рушій PhysX від компанії NVIDIA.

Кожен Unity-Project поділяється на сцени (рівні) - окремі файли, що містять власні ігрові світи із власними об’єктами, сценарієм та налаштуваннями. Сцени можуть містити як об'єкти (моделі), так і порожні ігрові об’єкти - об’єкти без моделей. Об'єкти, в свою чергу, містять набори компонентів, з якими взаємодіють сценарії (відповідні скрипти). Також у кожного об'єкта є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Таким чином, об'єкти на сцені обов'язково мають компонент Transform - зберігає координати положення на сцені, координати обертання та розміру об'єкта вздовж усіх трьох осей. В об'єктах із видимою геометрією за замовчуванням присутній стандартний компонент Mesh Renderer (для сітчастої візуалізації), що робить модель видимою.

Також Unity підтримує фізику твердих тіл, фізику типу Ragdoll та тканини. У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

1.4.2 Unreal Engine

Unreal Engine є одним з найпопулярніших та широко розповсюджений з ігрових рушіїв випущений компанією Epic Games. Оригінальна версія була випущена в 1998 році, і через 17 років вона все ще щорічно використовується в деяких великих іграх. Відомі назви ігор, які створені в двигуні Unreal, включають серії War Gear, Mass Effect, Bioshock та інші.

Потужність движка полягає в здатності створення різноманітних деталей, що робить додатки унікальними. Остання версія, Unreal Engine 4, вважається найпростішою у використанні в професійних ручних інструментах.

Рушій підтримує мову програмування C++. За допомогою движка можна створювати ігри для таких операційних систем та платформ: Linux, Mac OS, Microsoft Windows, Mac OS X, також для консолей PlayStation 2, PlayStation Portable, 31 PlayStation 3, Wii, Xbox, Xbox 360. Використовує модульну систему

підлеглих компонентів: підтримує системи комп’ютерної візуалізації : Direct3D, OpenGL, Pixomatic; компонент відтворення звуку : EAX, OpenAL, DirectSound3D; модуль голосового відтворення тексту, компоненти для роботи з мережею (технології Windows Live, Xbox Live, і GameSpy) й підтримки пристроїв вводу та виводу. Більшість елементів ігрового рушія надаються у формі об'єктів з певними характеристиками, і клас, який визначає доступність цих характеристик. Всі класи значяться «дочірніми» класом Object.

Незважаючи на все движок має багато плюсів та мінусів. Плюси: професійні засоби та інструменти для моделювання рівнів, велике ком'юніті і велика кількість навчальних матеріалів, можливе безкоштовне використання, значний набір готових ігрових ресурсів. Мінуси: має вибагливі вимоги до характеристик розроблювального пристрою (займає багато місця сам движок та створені додатки); інтерфейс громіздкий, вимагає додаткової уваги розробника.

1.4.3 GameMaker: Studio

GameMaker: Studio – популярний ігровий рушій, який використовується для професіональної розробки додатків. Підтримує кросплатформну розробку під платформи: Windows, macOS, IOS, Ubunto, Android, Windows Phone, Xbox One, Tizen, PlayStation.

На відміну від інших ігрових рушіїв, GameMaker: Studio використовується з багатьох причин спрощення розробки, насамперед не вимагає програмування. Таким чином користувачі зробити розробку додатку простішою та швидшою, ніж програмування певною мовою.

Найвідоміші додатки розроблені за допомогою GameMaker: Spelunky, Super Crete Boxing та Miami Hotline.

GameMaker: Studio дуже популярний, тому що ви можете розроблювати ігрові додатки додатково не вивчаючи мови програмування, також можуть використовувати вже досвідчені люди з використанням написання програмного коду для більшого вдосконалення та урізноманітнення додатку. Движок має власну

спрощену мову програмування GML (Game Maker Language). Проблема GameMaker: Studio полягає в обмеженості такого виду розробки на відміну від інших ігрових рушіїв.

Також безкоштовна версія надто обмежена в можливостях, тому для повноцінної розробки, можливо знадобиться придбання платної версії.

1.4.4 Godot

Godot – це багатоплатформений ігровий рушій, який був розроблений спільнотою Godot Engine Community. Середовище розробки може працювати на платформах: Windows, Linux, OS X, BSD, Android OS, macOS, Haiku, також існує можливість експорту готового проекту на інші різні платформи.

Ігровий рушій підходить розробникам для створення 2D та 3D додатків. Движок надає багато різних базових інструментів, тому користувачам непотрібно створювати вже існуючі концепції власноруч, а навпаки можуть зосередитися на реалізації власного унікального додатку.

Кожний додаток рекомендуються створювати з використанням скриптової мови програмування GDScript, який був створений спеціально для движка. GDScript - це мова програмування високого рівня, з динамічною типізацією, синтаксис якої дуже нагадує мову Python. Метою створення цієї мови програмування була оптимізація та щільна інтеграція з двигуном, що буде забезпечувати більшу гнучкість для можливих розробників. Також движок підтримує можливість використовувати інші мови через систему GDNative, наприклад Rust, C++, D та інші.

Godot являється програмним забезпеченням з відкритим кодом, яке доступне безкоштовно за ліцензією MIT. Тому розробникам непотрібно плати за користування та всі розробки залишаються у власності користувачів.

У Godot наявна активна спільнота, яка постійно виправляє помилки та розробляє новий функціонал рушія. Також підтримка в спільноті може відповісти на ваші запитання про Godot. Це можна зробити за допомогою: форума Reddit, групи Facebook, форума Godot та іншої контактної інформації.

1.4.5 CryEngine

Ігровий рушій CryEngine був створений німецькою компанією Crytek в 2002 році та планувався для внутрішнього використання в компанією, але після успіху гри «Far Cry» всі права на движок придбала компанія Ubisoft.

Використання платформи CryEngine для розробників є безкоштовним. Користувач отримує повний вихідний код рушія та всі функціональні можливості.

Головна перевага CryEngine – це професійні компоненти: вишукана графіка, фізична система віртуального світу та звукова система. Розроблювані приклади додатків дійсно виглядають конкурентоспроможними порівнюючи з іншими рушіями.

CryEngine може використовуватися для розробки під різні платформи: Xbox One, Oculus Rift (окуляри віртуальної реальності), Playstation 4 та Windows.

Розробнику для початку необхідний певний досвід або попередній процес навчання. Спільнота підтримки рушія також пропонує безліч безкоштовних інструментів для навчання : посібники, форуми та базова документація. Цей матеріал є необхідним для початку роботи та подальшого розвитку навичок.

Існують додаткові ресурси для проектів, які можна встановити з marketplace. Деякі набори пакетів коштують грошей, але є і безкоштовні пропозиції, наприклад: CryEngine V Beginners Pack та Explosives.

1.5 Засоби розробки додатку

1.5.1 Microsoft Visual Studio

Microsoft Visual Studio – це продукт фірми Microsoft, який може включати в себе інтегроване середовище розробки програмних засобів та інших інструментів в залежності від версії. Використовується для створення консольного програмного забезпечення або з графічним інтерфейсом користувача, включаючи веб-сайти, Windows Forms, WPF, веб-додатки тощо. Середовище підтримує кросплатформену

розробку таких, як: Microsoft Windows, Windows Phone, Windows CE, .NET Framework, консолей Microsoft Silverlight та XBOX.

Microsoft Visual Studio включає інструмент - редактор вихідного коду, який підтримує технологію IntelliSense і може легко трансформувати код. Інтегрований відладчик може виконувати функцію відладчика на рівні джерела (вхідного коду) або налагоджувача на машинному рівні. Інші інтегровані інструменти включають в себе: редактор форм, веб-редактор, дизайнер класів та конструктор схем баз даних. В більшості їх використовують для спрощення створення програм GUI.

Visual Studio побудована в архітектурі, що підтримує можливість використання доповнень. Тому за допомогою Microsoft Visual Studio можна створювати та підключати сторонні плагіни для розширення функціональних можливостей. Сюди входить підтримка систем управління версіями для вихідного коду (наприклад, Subversion та Visual SourceSafe) та додавання нових наборів інструментів (для редагування та графічного представлення коду, найчастіше використовується в об’єктно-орієнтованих мовах програмування) або інструментів для інших аспектів програмного забезпечення, наприклад інструмент для відстеження процесів розробки (клієнт Team Explorer для Team Foundation Server).

Для дипломного проекту Microsoft Visual Studio буде використовуватися, як допоміжний інструмент, який допомагає створювати велику кількість залежностей та спрощує взаємодію між ними. Оформлено на основі джерела [10].

1.5.2 Мова програмування C#

C# (Сі-шарп) – це сучасна об'єктно-орієнтована мова програмування, яка була розроблена корпорацією Microsoft під керівництвом Андерсом Гейлсбергом, Скотом Вілтамутом та іншими членами команди для платформи .NET.

C# вважається об'єктно-орієнтованою мовою, але підтримує також і компонентне-орієнтоване програмування. В сучасній розробці додатків все більше набуває популярність створення програмних компонентів у формі автономних пакетів, що в результаті реалізують самостійні та унікальні функціональні

можливості. Головною особливістю компонентів є модель програмування на основі методів, властивостей та подій. Кожен компонент має властивості, що забезпечують описову інформацію про елементи, також присутні вбудовані елементи документації компонентів. C# забезпечує конструкціями, яка безпосередньо підтримує цю концепцію. Це робить C# дуже зручним для створення та використання програмних компонентів.

Найбільш популярні функцій мови C#, що допомагають додаткам забезпечити надійність, стабільність і стійкість: обробка виключень надає розробнику можливість в структурований і розширюваний спосіб виявляти і обробляти помилки; прибирання сміття дозволяє автоматично звільняти пам'ять, використовуючи зайняту видаленими і незастосованими об'єктами; сувора типізація мови може запобігти ненавмисному редагуванню невизначених змінних, фіксує межі індексованих масивів та виходи з них, також контроль приведення типів.

Мова програмування C# має систему єдиного типу. Усі типи C#, включаючи традиційні типи (наприклад, int та char), успадковуються від одного кореневого типу object. Отже, усі типи використовують загальний набір операцій, і дані будь-якого типу можна зберігати, передавати та обробляти подібним чином. Крім того, C# підтримує посилальні типи (наприклад, object, string, delegate та інші), також є можливість використовувати типи значень, виділяючи динамічно пам’ять для об’єктів або зберігати їх в інших структурах даних. Використано джерело [11].

РОЗДІЛ 2
ПРОЕКТУВАННЯ МУЛЬТЕМЕДІЙНОГО ДОДАТКУ

Перш за все, щоб почати створювати мультимедійний додаток необхідно створити чіткий план дій, який буде включати жанр, сценарій, механіку та сетинг (обстановка). Це важливо, тому що створення гри процес творчий і без точного уявлення кінцевого результату реалізація може затягнутися надовго, або навіть так ніколи і не закінчитися.

2.1 Розробка структури додатку

Структура проекту буде складатися з безлічі файлів та папок, але найважливіші з них: Assets і ProjectSettings. Зазвичай всі інші папки будуть генеруватися на основі цих двох папок.

Огляд структурних файлів і папок на основі літературного джерела [12].

Assembly-CSharp-vs.csproj - файли скриптів з закінченням - vs на кінці, створені для інтеграції з Visual Studio. Assembly-CSharp.csproj - файли проекту, призначені для роботи в MonoDevelop створені для C# скриптів.

Файли проекту testproject.sln і testproject-csharp.sln використовуються для інтегрованих засобів розробки, перший файл включає в себе підтримку всіх мов програмування движка таких, як : C#, JavaScript. Другий файл - тільки для проектів з мовою програмування C#. Visual Studio підтримує тільки скрипти на мові C#, тому я використовувала мову C#.

Файли testproject.userprefs і testproject-csharp.userprefs використовується для конфігурації, де зберігаються поточні відкриті файли, точки зупинку, час та інші інформація.

Усі перелічені файли, крім .userprefs, відтворюються постійно, коли користувач обирає ресурси проекту (Assets), потім пункт синхронізації проекту в меню.

Структура папок проекту в Unity:

Assets (ресурси) - папка, яка буде застосовується для зберігання ігрових ресурсів. Для проекту буде використана для структурного зберігання скриптів, текстур, аудіо-ресурсів, текстової інформації, анімації та інше. Безумовно це основна та найголовніша папка проекту.

ProjectSettings (проектні налаштування) - в цій папці зберігаються всі налаштування проекту Unity, такі як фізика, теги, ігрові налаштування. Користувач може налаштувати цю папку проекту через пункт меню Edit → Project Settings (в результаті всі налаштування будуть зберігаються).

Library (бібліотеки) - використовується, як локальний кеш для імпортованих Assets (ресурсів), при використанні зовнішньої системи контролю версій (VCS) ця папка повинна бути виключена зі списку VCS та повністю ігноруватися.

OBJ і Temp - папки для зберігання тимчасових файлів, що створюються в процесі побудови (або під час збірки) проекту, OBJ використовується в MonoDevelop, а Temp використовується в Unity.

2.2 Жанр

Після аналізу ігрових жанрів в першому розділі дипломної роботи, було прийнято рішення для мультимедійного додатку обрати жанр 3D-шутер.

Гравець буде перебувати в 3D-просторі (вигаданому всесвіті) та матиме деякі функції пересування. Ці функції можуть обмежуватися простою ходьбою (пересування від одної перешкоди до іншої) або мати додаткові можливості, як біг чи можливість перестрибувати через об’єкти. На шляху персонажа можуть з’являтися перешкоди у вигляді ворогів, які потрібно знищити наданою зброєю за відповідний час, щоб можна було просуватися далі по вигаданому простору.

2.3 Сетинг

Сетинг (обстановка) - середовище, в якому відбувається дія мультимедійного додатку. Також, сетинг визначають, як набір правил деякого оточення. Оскільки

дане слово є запозиченим з англійської мови, його значення варіюється в залежності від автора та сфери використання. Найчастіше визначається, як місце, час та умови, в яких розвиваються події. Окремим випадком поняття «сетинг» є «вигаданий всесвіт». Такий сетинг може включати неіснуючу або вигадану власну історію, географію, культуру, фізичні явища та навіть істоти (наприклад, ельфи, хобіти, орки, дракони та інші). Однак сетинг може включати не тільки вигадані середовища, а й відтворювати реальні географічні положення або події.

Для реалізації мультимедійного додатку було визначено сетинг, який за своїми характеристиками наближається до реалістичного світу із справжніми фізичними явищами та часом, як в реальному житті.

Віртуальний світ скрадатиметься з одного середовища, в я кому будуть відбуватися події додатку. Також будуть присутні персонажі-вороги зі штучним інтелектом, які вважатимуться основними противниками в грі. Художнє зображення середовища - ландшафт, який складається з однієї кліматичної зони, для якої характерно гори та ліси (дерева, чагарники, кущі). Прототип зображено на Рис. 2.3.1.

Рис. 2.3.1 Прототип сетингу

2.4 Планування механік

В додатку присутні два фактори, які визначають будь-який поточний стан гри: механіка (правила) і гравці, які взаємодіють з ним.

В кожній грі існує процес (хід гри), який поділяється на такти. Далі наведені такти додатку:

- Такт 1 - визначення стану гри. Завжди є поточний стан гри (наприклад, початок дій). Зазвичай визначається на першому такті послідовністю ініціалізації, коли гравці «завантажують» гру, переходять на вкладку «грати» (почати гру), розподіляються початкові ресурси в грі (наприклад, зброя гравцеві буде надана автоматично) і так далі. Потім стани будуть змінюватися в залежності від того, що відбувається в додатку.
- Такт 2 - гравці оцінюють стан для прийняття рішень. Після поновлення стану гри є такт, під час якого гравці приймають рішення на основі нової інформації (наприклад, появлення перешкод або ворогів на шляху під час якого гравець приймає рішення, як далі діяти).
- Такт 3 - гравці виконують дії. На цьому кроці гравці змінюють стан гри, роблячи щось відповідно до правил (наприклад, переплигуючи через перешкоди або ліквідуючи ворогів користувач може далі просуватися по ігровому світу.)
- Такт 4 - зворотний зв'язок гри. Найчастіше використовується, як генерація кількох випадкових станів (наприклад, на сцені моделюються незалежні вороги та перешкоди, які мешкають в вигаданому світі). Вихід з циклу робиться по відповідності одному з критеріїв закінчення алгоритму, тобто, наприклад, вашу поразку (смерть персонажа гравця або вихід з гри за допомогою кнопок).

Отже, гравці на основі аналізу станів приймають рішення про свої майбутні дії, виконують ці дії, взаємодіючі з іншими об’єктами світу (вороги, перешкоди) і

правилами, а потім гра переходить в новий стан. Додаток може виконувати частину робити частину сам, незалежно від дій гравців. Наприклад, якщо ваш персонаж зупинився або натрапив на перешкоди з якої не може вибратися, інші персонажі-вороги будь продовжувати напади та свою початкову поведінку незалежно від дій гравця.

Після планування правил (механік), які будуть розроблятися на останньому етапі реалізації, були визначені основні елементи додатку:

Система взаємодії з грою (в реалізації: з головним меню, меню опис дій, віртуальний світ):

1) Перехід між вкладками додатку: головне меню, опис дій (правил), ігровий всесвіт;
2) Перегляд правил додатку;
3) Рух персонажа (ходьба) за допомогою клавіш;
4) Біг персонажа за допомогою клавіш;
5) Стрибок персонажа за допомогою клавіш;
6) Взаємодія з об’єктами навколишнього світу (перешкоди, інші персонажі);
7) Відлік часу в грі;
8) Показники персонажа – шкала здоров’я та енергії;
9) Вибір зброї персонажа;
10) Використання зброї за допомогою клавіш.

Після виконання визначення з попередніми етапами проектування: структура, жанр, сетинг і з механіками, наступним етапом буде перехід до планування сцен мультимедійного додатку.

2.5 Планування сцен

Сцени (рівні) додатку складаються з окремих файли, що містять власні ігрові світи із унікальними об’єктами, сценарієм та налаштуваннями. Готовий мультимедійний додаток - це набір сцен, які з'єднані та взаємодіють між собою.

Перш ніж переходити на етап реалізації проекту було вирішено визначити сцени, які він повинен в себе включати. Було складено список сцен, обмірковано, зміст і значення кожної сцени. Головний принцип створення сцен – це зв’язний розподіл функціональних можливостей за певним критерієм без переборщування з кількістю сцен, коли це надлишково (розділяти сцени на підсцени), адже пам’ять проекту завжди є обмеженою.

Список сцен з графічними прототипами:

1. Головне меню. Прототип сцени зображено на Рис. 2.5.1;

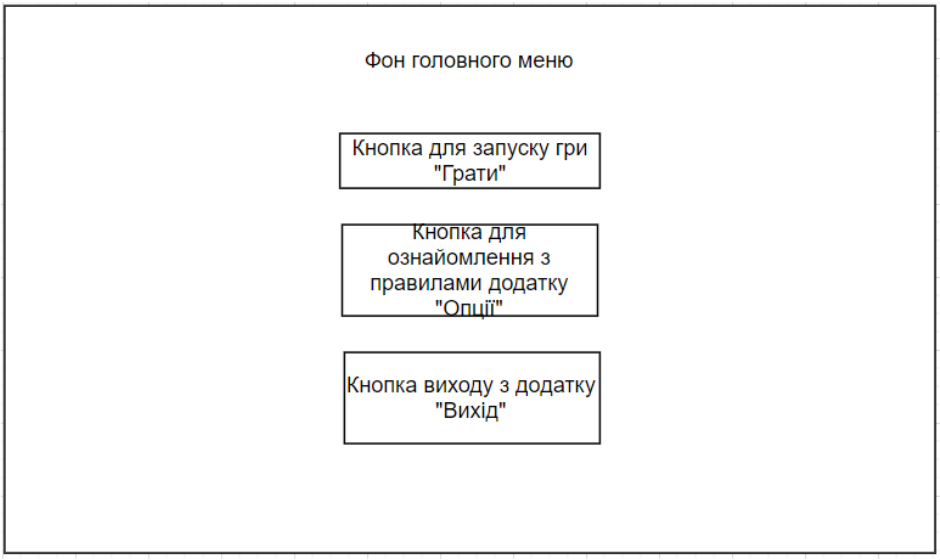


Рис. 2.5.1 Прототип сцени головного меню

2. Меню опцій додатку. Зображено на Рис. 2.5.2;

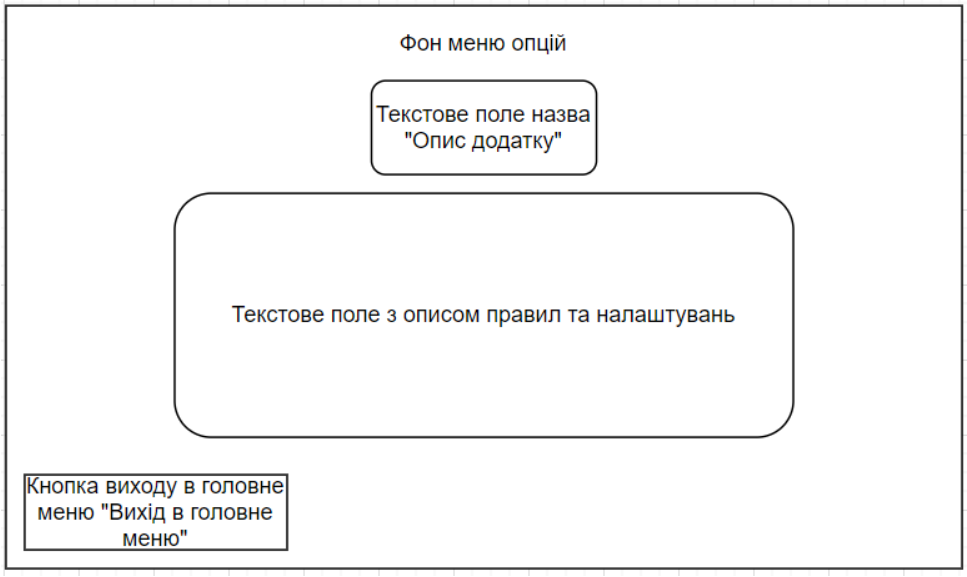


Рис. 2.5.2 Прототип сцени меню опцій

3. Ігровий світ «Ліс». Зображено на Рис. 2.5.3.

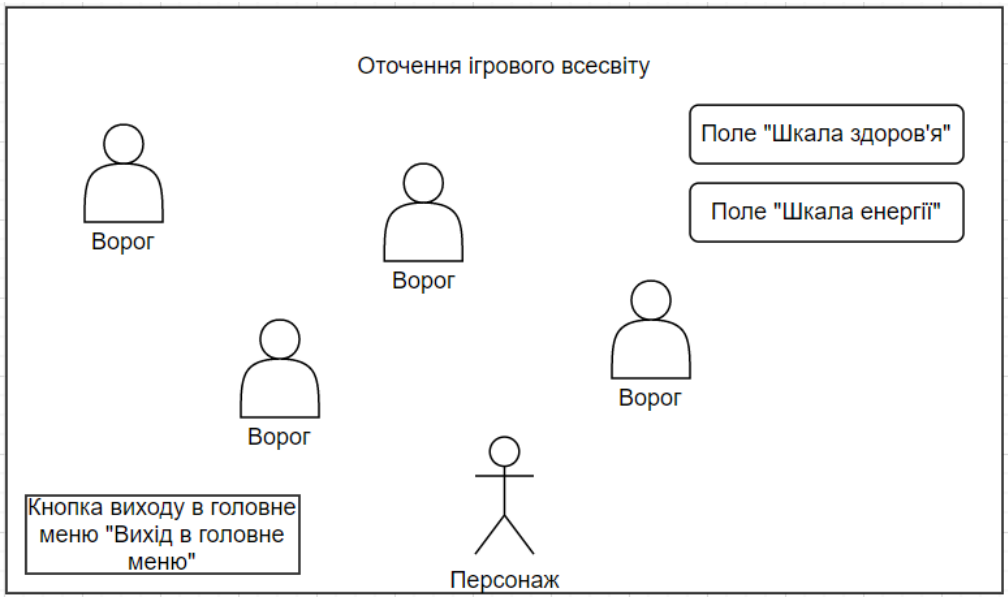


Рис. 2.5.3 Прототип ігрового світу

Наступним етапом є реалізація проекту за планом. Починаючи із створення самих сцен-локацій (головне меню, ігровий світ), на яких буде відбуватися подальша дія.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Дизайн віртуального світу (сцени)

Для подальшої реалізації проекту в першому розділі було обрано ігровий рушій Unity 3D. У редакторі Unity присутній досить зручний інтерфейс для створення різного дизайну локацій. Основна складність, це підібрати або створити власноруч вдалі та відповідні текстури для всіх об’єктів, моделей та ландшафтів.

3.1.1 Terrane (ландшафт)

Terrane (ландшафт або земля) - це об'єкт, що являє собою величезне полотно, зовнішній вигляд та висоту кожної точки якого, можна налаштувати за допомогою спеціального редактора. За допомогою того ж редактора наносяться текстури і різні об’єкти (наприклад, висаджується рослинність).

Графічні об’єкти, як: гори, дерева, чагарники створюються за допомогою 3D-об'єктів. Також автоматично оптимізується відображення об'єктів щодо гравця (поступово завантажуючи в пам'ять далекі об'єкти, а найближчі моделюються з найвищою якістю). На рисунку 3.1.1.1 проілюстровано спосіб створення ландшафту для віртуального світу та відображення первинних об'єкт на сцені.

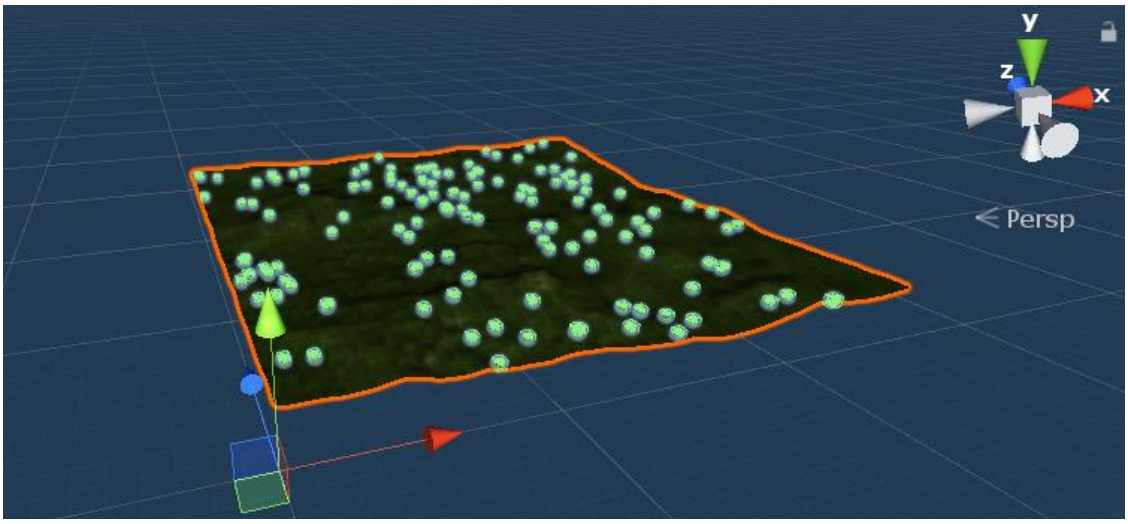


Рис. 3.1.1.1 – Створення ландшафту для віртуального світу

Для створення ландшафту використовувалась панель в правій частині редактора Unity. На ній розміщені налаштування для різних операцій: кисті малювання, збільшення або зменшення висоти (розмірів), вирівнювання по висоті, нанесення текстур на об’єкти і розміщення рослинності (зображено на Рис. 3.1.1.2). Згідно аналізу літературного джерела [13].

Таким чином, попрацювавши з різними налаштуваннями, було створено невеликий рельєф, впадини, бугори на рівному ландшафті, також на цьому етапі було створено різноманітні текстури, які будуть накладатися на різні об’єкти. В результаті отримано головну основу для майбутньої локації (світу виконання основної дії), яка буде вдосконалюватися на наступних етапах.

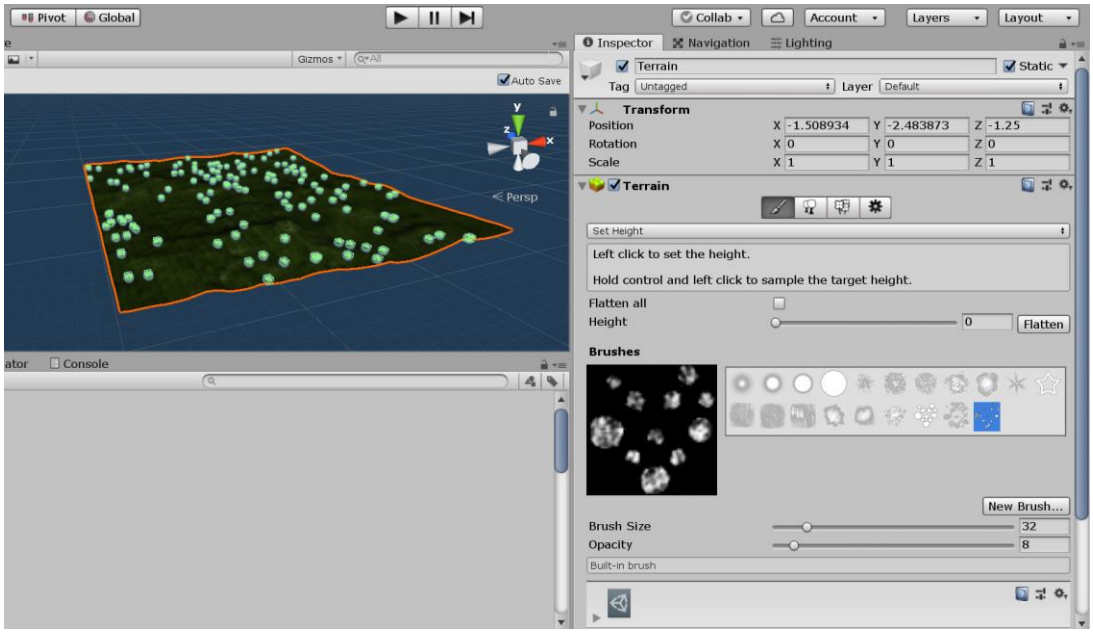


Рис. 3.1.1.2 - Інструменти зміни ландшафту

Після першого етапу створення ландшафту місцевість виглядає голою і текстури, нанесені на неї виглядають одноманітно та понуро. Це була всього лише перша ітерація на шляху до кінцевої експозиції. Виходячи з цього було виділено наступні, підзавдання, які не мають великий впливати на додаток, але створюють набагато приємніший і деталізований фон, що на сьогоднішній день є важливим для користувачів. Прийнято рішення в віртуальному світі створити різноманітний ліс, також додати текстури неба з хмарами (skybox).

Отже, для поліпшення зовнішнього вигляду віртуально світу, необхідно виконати наступний план:

- Створити кілька видів дерев та кущів і розташувати їх на місцевості;
- Знайти або створити більш різноманітні текстури скель, землі, трави;
- Створити чагарники і траву;
- Створити об’єкт каміння з відповідною текстурою;
- Додати пагорби та впадини на ландшафті;
- Створити небо (skybox);
- Наповнити місцевість деталями.

3.1.2 Створення об’єктів рослин

В Unity присутній вбудований редактор створення та редагування дерев і чагарників. Спершу було вирішено створити нові дерева для майбутнього лісу.

Перш за все, щоб створити дерево, потрібно створити відповідний об’єкт, таким чином, на сцені з'явиться стовбур дерева і надалі до нього, за допомогою редактора дерев, можна буде додати гілки, суцвіття та листя. Процес створення дерева зображено на Рис. 3.1.2.1.

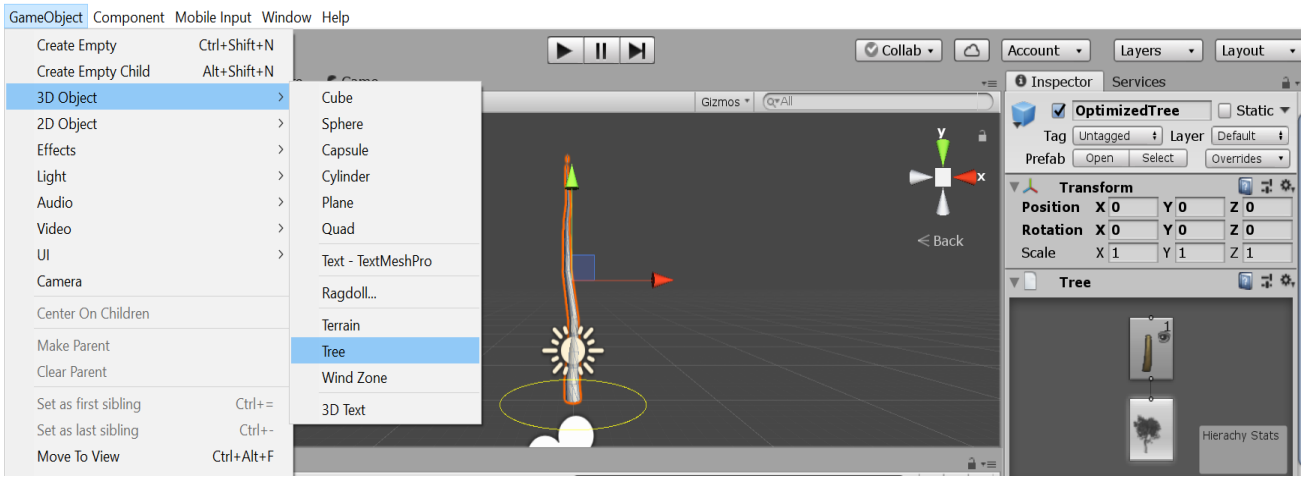


Рис. 3.1.2.1 – Створення дерева (етап №1)

Редактор дозволяє якісно налаштувати розташування довжину, вигин і багато

іншого для гілок. Основні параметри налаштування:

- Group seed - число яке визначає розташування кількості гілок або листя в групі;
- Frequency - кількість в групі;
- Distribution – забезпечує різні способи розташування гілок в групі. Основні варіанти:
  - Random - хаотичне розміщення;
  - Alternate – дзеркальне розташування по всій довжині попередньої групи в одній площині;
  - Opposite - розташовуються дзеркально по всій довжині попередньої групи в двох площинах;
  - Whorled - гілки розміщені по колу (наприклад, ялинка).
- Twirl - кут розвороту гілок в горизонтальній площині, використовується тільки для конкретних типів розподілу: alternate, opposite, whorled;
- Whorled Step - кількість гілок в підгрупі для типу розподілу whorled;
- Growth Scale - відповідає за параметри (ріст) гілок, довжину гілки в залежності від її позиції на гілки попередньої групи. Можна використовувати для всіх типів;
- Growth Angle - кут повороту по вертикальній осі;
- Branch Material - текстура кори;
- Break Material - текстура на місці зрубу або для зламаної гілки;
- Frond Material - текстура листя;
- Length - довжина гілки, позначається, як інтервал від найменшої до найбільшої;
- Relative Length - залежність довжини гілки від місця росту;
- Radius - радіус гілки;
- Crinkliness - скрученість гілки;
- Seek Sun - ступінь вигину гілок до світла;

- Break Chance - шанс, що згенерувала гілка буде зламаною;
- Break Location - інтервал на якому гілка може бути зламана.

Приклад створення дерева зображено на Рис. 3.1.2.2.

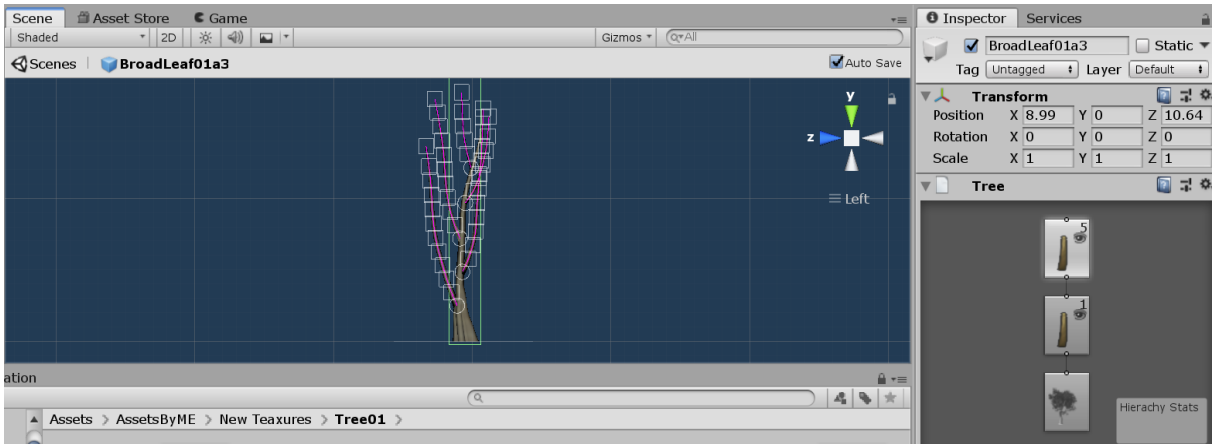


Рис. 3.1.2.2 Створення дерева (етап №2)

За допомогою вище перерахованих параметрів було створено кілька варіантів рослинності (зображено на Рис. 3.1.2.3). На кінцевому етапі планується покращити створені дерева та кущі, і створити розмаїтні варіанти.

Рис. 3.1.2.3 Результат створеної рослинності

Використані літературні джерела [14,15]

3.1.3 Текстури проекту

Один з найважливіших елементів в мультимедійному додатку є набір текстур. Чим більше текстур застосовується дизайнером проекту, тим різноманітнішим

вважається атмосфера та обстановка додатку, але перш за все кожна текстура на об’єкті має обґрунтовано перебувати на своєму місці.

В документації Unity [16] для розробки та нанесення текстур присутня панель редагування текстур, яка зображена на Рис. 3.1.3.1.

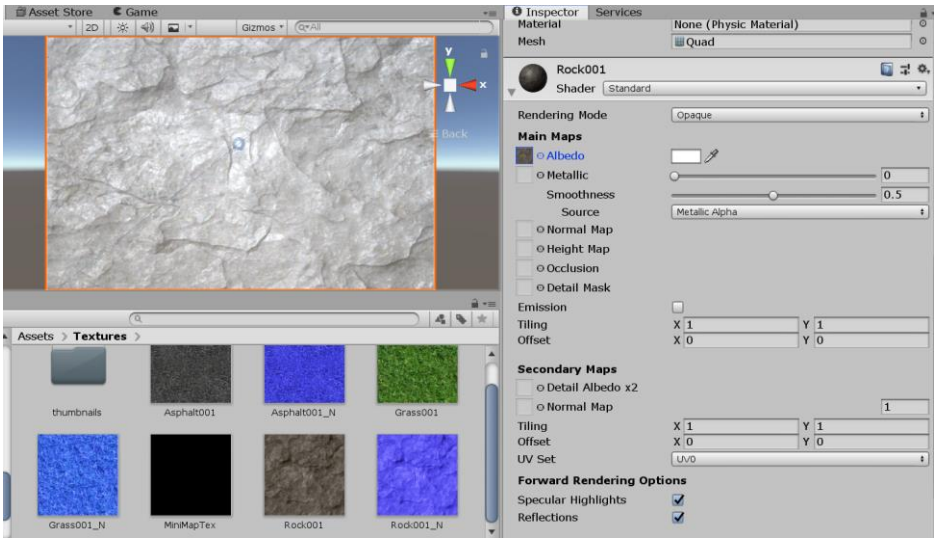


Рис. 3.1.3.1 Панель текстур

Найголовніші елементи редактора:

- Brushes - кисті для нанесення текстур;
- Textures - відображаються всі текстури проекту, обрану текстуру можна накладати на об’єкти;
- «Edit Textures» надає доступ для використання опцій: «Add Texture» (додати текстуру), «Edit Texture» (змінити текстуру), «Remove Texture» (видалити текстуру);
- Settings – налаштування опцій кисті;
- Brush Size - розмір кисті;
- Opacity – прозорість або непрозорість текстури;
- Target Strenght - опція схожа opacity, але на відміну від попередньої опції це максимальна непрозорість, яка може досягатися для текстури (жорстко задає положення на поверхні текстури і її прозорість);

Для реалістичного відображення об’єктів використовується рельєфне текстурування, ця технологія дозволяє створити ілюзію нерівності, об’ємності та шорсткості, таким чином дивлячись на рослинність під різними кутами для користувача буде створюватися враження, що це не плоска поверхня, а реалістичні 3D-об’єкти. Текстури проекту зображені на Рис. 3.1.3.2.

Рис. 3.1.3.2 Використані текстури

3.1.4 Skybox сцени віртуального світу

Наступним етапом є створення skybox (небо). Skybox – це певна обгортка навколо сцени що показує, як виглядає світ за межами вашого ландшафту (terrane) та інших об’єктів вашої сцени, згідно документації [17]. Для реалізації було обрано вечірній час доби приблизно 16-17 годин. Результат зображено на Рис. 3.1.4.1.

Рис. 3.1.4.1 Skybox віртуального світу

Кінцевим етапом з дизайну є розміщення створених об’єктів на сцені віртуального світу. Хаотично розташувавши кілька видів об’єктів дерев ліс став виглядати різноманітніше та цікавіше. Результатом став невеликий ліс. В підсумку об’єкти були розміщені на сцені, зображено на Рис. 3.1.4.2.

Рис. 3.1.4.2 Результат створення лісу

Інші сцени: головне меню, опції були створені аналогічним способом і будуть представлені в іншому розділі.

3.2. Опис та реалізація функціонала додатка

В будь-якому мультимедійному додатку існують взаємодії і чим більше їх буде, тим більше інтересу буде виникати у користувачів. За час розробки дипломного проекту в основному упор робився на проектування та вивчення методів і засобів, якими заплановане можна реалізувати. Тому кінцевим кроком є реалізація функціоналу, який був запланований.

Основні функціональні можливості додатку:

- Керування напрямком огляду камери;
- Пересування камери уздовж поверхні із застосуванням фізики;
- Пересування персонажа по вигаданому всесвіту (пішки, біг, стрибки);
- Імітація застосування зброї;
- Створення ворогів;

- Створення звукових ефектів та накладання музики;
- Створення вибору в грі (різні варіанти зброї);
- Створення функцій головного меню та меню опцій;
- Відображення інформації: кількість ресурсів, що залишилися шкала енергії та здоров’я персонажа.

3.2.1 C# скрипт

Для реалізації функціональних можливостей додатку було створено власні компоненти, які називаються скриптами (використано аналіз джерела [18]). Вони дозволяють активувати події, змінювати параметри певних компонентів, і взаємодіяти з користувачами зважаючи на їх дії в додатку.

Програмний інструмент Unity може підтримувати мови програмування: JavaScript , Boo, C#. Як вже описувалось раніше, для реалізації проекту було обрано мову програмування C#.

На відміну від ресурсів, які створювалися в попередньому розділі, скрипти зазвичай створюються безпосередньо в середовищі розробки Unity. Основний спосіб створити скрипт це використати пункт меню Create в лівому верхньому кутку панелі Project або обравши відповідні вкладки: Assets -> Create -> C# Script за допомогою головного меню.

Новий скрипт буде створений в папці, яку можна обрати за допомогою панелі Project. Ім'я нового ресурсу буде згенеровано автоматично, пропонуючи змінити його на більш зручне для розробника.

Краще одразу ввести нове унікальне ім'я скрипта після створення ніж змінювати його потім (може призвести до помилок в проекті). Ім'я файлу буде використано, щоб створити початковий код в скрипті, опис цього етапу описано в наступному пункті.

3.2.2 Структура скрипта

Для роботи з скриптом потрібно подвійно натиснути на скрипті в Unity, тоді він буде відкритий за допомогою текстового редактора. За замовчуванням Unity використовує MonoDevelop, але було обрано більш знайомий та звичний для себе редактор - Microsoft Visual Studio (вибір з панелі External Tools в налаштуваннях).

Виглядати змісту файлу:

```
using UnityEngine;
using System.Collections;
public class Main: MonoBehaviour {
// Буде використовуватися для ініціалізації
void Start () {
    }
// Буде викликатися один раз за оновлення екрану
void Update () {
     }
}
```

Для взаємодії з внутрішніми механізмами потрібно створити клас, який буде успадкований від вбудованого класу MonoBehaviour. Можна сприймати створений клас як про свого роду створення нового типу компонента, який може бути прикріплений до ігрового об'єкту (GameObjects) і через нього взаємодіє з ним. При приєднанні нового скриптового компоненту до ігрового об'єкту, створюється новий екземпляр об'єкта, визначений планом. Ім'я класу дублюється з імені, яке вказано користувачем при створенні файлу. Щоб приєднати компонент до GameObject, ім'я класу та ім'я файлу повинні бути однаковими.

Найголовніше в скрипті - це дві функції, які визначені усередині класу. Перша функція Update (оновлення) - це місце для написання коду, який буде обробляти оновлення кадру для об'єкта додатку. Найчастіше використовується, як: рух, спрацьовування дій і відповідна реакція на дії користувача (все те, що повинно вирішуватися з часом, під час процесу роботи додатку). Щоб налаштувати функцію

оновлення (Update) для виконання заданого завдання, необхідно визначити змінні, зчитувати властивості, параметри і здійснювати зв'язок з іншими об'єктами до того, як будуть починатися будь-які дії. Друга функція Start (початок) викликається до початку процесів (тобто перед першим викликом функції Update), в більшості випадках використовується для ініціалізації. Використано літературне джерело [19].

3.2.3 Звукові ресурси

Для забезпечення звуковими ефектами мультимедійний додаток, використовуватиметься Audio Source, який буде відтворювати аудіо кліп на сцені вигаданого всесвіту. Присутні декілька різновидів аудіо кліпів, наприклад 3D-кліп, відтворює звук в залежності від положення об’єкта в просторі (збільшує або зменшує гучність звуку виходячи з параметрів відстані). Також аудіо можуть бути розподіленими по різних колонках за допомогою властивості Spread (встановлення куту розповсюдження багатоканального або стерео звуку в просторі) або властивості PanLevel (встановлення показника наскільки впливатиме на джерело звуку).

Є можливість контролю залежності цих ефектів від відстані із підтримкою кривих затухань. Так само, якщо слухач знаходиться зонах реверберації («післязвуччя» у закритих приміщеннях, утворюється внаслідок багатократного відбиття від поверхонь та одночасного поглинання звукових хвиль. Характеризується проміжком часу у секундах, протягом якого сила звуку зменшується на 60 дБ), то для джерела застосовується реверберації (Unity Pro). Для урізноманітнення аудіо ефектів, можна застосувати окремі аудіо фільтри.

Інформація про аудіоефекти:

- Audio Clip (аудіо кліп) - посилання на аудіо для відтворення заданим джерелом;
- Mute (вимкнути звук) - звук не буде чутно, але аудіо кліп буде програватися;

- Bypass Effects (ефекти обходу) - застосовується до джерела звуку для швидкого відтворення ефектів (включення або відключення всіх ефектів);
- Bypass Listener Effects (ефекти обходу слухачів) - застосовується для швидкого включення або відключення ефектів для гравця додатку;
- Bypass Reverb Zones (обхід реверберацийних зон) - для включення або відключення всіх зон реверберації;
- Play On Awake (відтворюватися при використанні) - при включені, звук почне відтворюватися при відкритті заданої сцени. Функція Play() використовується для включення аудіо кліпу;
- Loop (цикл) - безкінечний цикл для повторення заданого аудіо кліпу;
- Priority (пріоритет) – використовується для розподілу пріоритетів між джерелами звуку на сцені. Показники: знаходяться в діапазоні від 0 (найважливіший) до 256 (дріб'язковий). Показник 128 встановлюється автоматично;
- Volume (гучність) - гучність звуку на відстані однієї світової одиниці виміру від користувача;
- Pitch (крок) - показник зміни тону звуку при уповільненні або прискоренні відтворенні (міра 1 означає нормальну швидкість відтворення);
- Min Distance (мінімальна відстань) - відтворення гучності звуку буде максимальною, якщо не виходити за межі мінімальної дистанції. Поза межами гучність буде поступово знижуватися;
- Max Distance (максимальна відстань) - відстань, де звук перестає зменшувати гучність. За межами максимальної дистанції гучність звуку залишиться на рівні від гравця і далі не зменшується;
- Rolloff Mode (режим відключення) - швидкість зменшення (згасання) звуку. При високому значенні слухач ймовірно знаходиться близько до джерела звуку до його відтворення. Можливі способи виключення:

- Logarithmic Rolloff (логарифмічне відключення) - висока гучність звуку при близькому знаходженні до об’єкту, при зменшенні дистанції гучність різко падає;
- Linear Rolloff (лінійне відключення) – при наближені до джерела звуку, гучність поступово зростає, при віддалені поступово спадає;
- Custom Rolloff (користувацьке відключення) – користувач може сам налаштовувати гучність звуку, використовуючи графік затухань.

Матеріал оформлено за допомогою літературних джерел [20-21].

3.2.4 Штучний інтелект

Штучний інтелект – це певний сектор інформатики, який вирішує питання керування деякими діями та прийняттями рішеннями комп’ютером, подібним способом до вирішення інтелектом людини (джерело [22]).

Для урізноманітнення взаємодії мультимедійного додатку з користувачем було вирішено використовувати штучний інтелект.

Є два сценарії взаємодії з гравцем:

1) Сценарій допомоги. Реалізується, як підтримка гравцеві в грі (поради, вказівки, опис дій або правил, пошуку варіантів виходів та входів);
2) Сценарій ворога. Суперник, який має подібні можливості, що і гравець, але його дії направлені проти нього. Найчастіше використовують в іграх на виживання, шутерах, бійках.

Штучний інтелект для власного проекту буде використовуватися за другим сценарієм. Головною задачею якого буде - керування інтелектом ворога (його можливість рухатися по штучному світу та атакувати гравця). Реалізований в проекті, як скрипт Enemy_Control.

Можливі стани для штучного інтелекту ворогів:

1. Режим пошуку-спокою (персонаж знаходиться на великій відстані ворогів);
2. Навігація – шлях до персонажа (пошук найкоротшого шляху до гравця);
3. Напад ворога на персонажа.

Реалізація режиму пошуку-спокою відбувається в методі patrol. Завдання методу полягають в:

1) повідомлення, що вороги (navigation agent) можуть рухатися зі встановленою швидкістю приблизної до ходьби;
2) додавання таймеру відліку режиму спокою;
3) перевірка дистанції між ворогом та персонажем. Якщо дистанція менша або дорівнює заданій в атрибутах (chase_distance), ворог зупиняється рухатися і переходить в наступний стан навігації.

Режим навігації відбувається в методі chase. Сенс методу полягає в:

1) задається стан руху ворога – активний із заданою швидкістю приблизної до бігу;
2) встановлюємо позицію персонажа, як пункт призначення для ворогів тому, що відбувається переслідування гравця (біжимо назустріч). На основі даних сцени обчислюються області можливих переміщень і, як наслідок формуються набори даних, за якими буде проведений пошук оптимального маршруту для переміщення ворога до персонажа з будь-якої точки сцену в точку поблизу персонажа (дані будуть зберігатися в ресурсах і в процесі використання можуть бути змінені);
3) перевірка відстані між ворогом та гравцем:
    3.1) якщо менше заданої, рахується, що ворог знайшов персонажа, здійснюється перехід до стану нападу на ворога;
    3.2) якщо відстань збільшується - це означає, що гравець втікає від ворога (рухається в протилежному напрямку). Ворог зупиняє біг,

встановлення режиму пошуку-спокою ворога. Скидаємо відстань погоні до початкової.

Режим нападу (атаки) на персонажа буде відбуватися за допомогою метода attack. Головні можливості:

1) встановлення мінімального часу перед атакою ворога на персонажа;
2) здійснюється напад;
3) якщо дистанція між ворогом та гравцем збільшується більше ніж дозволена під час нападу, тоді вороги переходять в стан навігації.

3.2.5 Аналіз та вибір алгоритмів пошуку шляху для ворогів

Пошук шляху – поняття в штучному інтелекту, яке відповідає за визначення оптимального шляху між двома позиціями.

В основному, алгоритм пошуку шляху будуються на графах починаючи з початкової точки і перевіряючи найближчий вузол, поки не буде досягнутий цільовий вузол. Більше того, у більшості випадків алгоритм пошуку маршруту спрямований на пошук найкоротшого шляху. В мультимедійних ігрових додатках жанру 3D-шутер часто можна помітити використання точок шляху (waypoints) із-за обмеженого простору, який майже неможливо поділити на вузли. Точки шляху – вузли, які містять в собі дані про спосіб знаходження шляху до інших вузлів.

Найпопулярніший алгоритми та системи пошуку шляху:

1. Алгоритм Дейкстри;
2. Пошук у ширину;
3. Хвильовий алгоритм;
4. Алгоритм A*;
5. Навігаційна сітка.

Використавши літературні джерела [23-25] проаналізовано найпопулярніші алгоритми пошуку:

1. Алгоритм Дейкстри - популярний алгоритм, який використовуються для пошуку найкоротших відстаней або мінімальних витрат залежно від графу. Починається від кінцевої вершини до початку, знаходячи кожен раз найкоротший шлях. Є можливість ставити певні пріоритети досліджених шляхів. Віддає перевагу за критерієм (наприклад, шляхах з більш низькою вартістю). Може використовуватися в ігрових світах для алгоритму вартості дороги за критеріями: складність, ціна, кількість ворогів та інше.

   Опис алгоритму:

   Позначки при переході від вузла i до наступного вузла j: qi- найкоротша відстань від початкового вузла до вузла i; dij - довжина ребра (i, j). Тоді для вузла j мітка [qj, i] наступним чином:

   $$[qj, i] = [qi + dij, i], dij> = 0 \qquad (3.2.5.1)$$

   Крок 1. Початковому вузлу присвоюється мітка [0, -]. Вважаємо i = 1;

   Крок 2. Обчислюються тимчасові мітки (можна замінити на іншу, якщо буде знайдений коротший шлях до вузла) [qi + dij, i] для всіх вузлів j, які можна досягти безпосередньо з вузла i та які не мають постійних міток. Якщо вузол j вже має позначку [qj, k], отриману від іншого вузла k, і якщо qi + dij <qj, тоді мітка [qj, k] замінюється на [qi + dij, i]. Якщо всі вузли мають постійні мітки, процес обчислень закінчується. В іншому випадку вибирається мітка з найменшим значенням відстані qs серед всіх тимчасових міток.

2. Алгоритм пошук у ширину – простий спосіб знаходження шляхів. Обирається довільна точка з якої рівномірно відстежується напрям у всіх траєкторіях. Для розробки ігрових додатків може використовуватися як генерація різних шляхів на карті.

   Опис алгоритму:

   Крок 1. Обрати довільну точку i. Тоді i=1 та заноситься у чергу.

Крок 2. Розглянути точку на початку черги (нехай це точка j). Якщо для всіх інших точок, суміжних із вершиною j, уже визначено номери в черзі, перейти до кроку 4, якщо ні - до кроку 3.

Крок 3. Нехай (j,k) - ребро, у якому номер точки k відсутній. Визначити номер k, як наступний, занести точку у чергу й перейти до кроку 2.

Крок 4. Виключити точку j із черги. Якщо черга порожня, алгоритм завершено, якщо ні – виконати крок 2.

3. Хвильовий алгоритм створений на основі алгоритму пошуку у ширину. Алгоритм працює на матриці з двома типами комірок на яких можна прокласти шлях та комірок обмежень руху. Алгоритм має на меті знайти найкоротший шлях (хвиля рухається від початкової точки, помічаючи доступні комірки, як пройдені та ігноруючи непрохідні та пройдені комірки, доки досягне фінальної комірки) або опублікувати повідомлення про неможливість знаходження маршруту, коли шляху немає. В розробці використовується як пошук коротшого шляху на карті.

   Опис алгоритму:

   Вхідні дані: граф G, в якому потрібно знайти шлях між точками q та z.

   Крок 1. Кожній вершині xi присвоюється число T (xi) - хвильова мітка (початкове значення = - 1);

   Крок 2. Створюються два списки OldF і NewF (фронти хвилі – старий і новий), а також змінна T (час початковий);

   Крок 3. Встановлення значень: OldF = {q}; NewF = {}; T (q) = 0; T = 0;

   Крок 4. Для кожної з вершини, що входить в OldF, розглядаються суміжні їй вершини uj, і якщо T (uj) = -1, то T (uj) = T + 1, NewF = NewF+ {uj};

   Крок 5. Якщо NewF = {}, то рішення не має, алгоритм завершено;

   Крок 6. якщо одна з вершин uj збігається z, то шлях вважається знайденим між q і z з T (t) = T + 1 проміжними ребрами. Алгоритм завершено;

Крок 7. OldF = NewF, NewF = {}; T = T + 1; Перейти до кроку 4.

4. Алгоритм A* - алгоритм, де пошук маршруту відбувається з даної початкової точки до кінцевої заданої. Головна мета якого досягти цілі - кінцевої точки. Може використовуватися як генерація найкоротшого маршруту до певної точки локації.

   Опис алгоритму:

   Вхідні дані: x – початкова точка; y - кінцева точка; g(n) - вартість шляху від початкової точки до будь-якої точки n; h(n) - вартість від точки n до цільової точки, f(n) – сусідня точка n з найменшою вартістю.

   Крок 1. При кожному вході у сусідні точки відбувається обчислення вартості, f(n), щоб дістатися до всіх сусідніх точок, а потім обирається точка з найменшим значенням f(n).

   Ці значення розраховуються за такою формулою:

   $$f(n) = g(n) + h(n) \quad (3.2.5.2)$$

   Крок 2. Безліч переглянутих точок будуть зберігатися в черзі C, а відкриті точки (можливі для шляху) в черзі з пріоритетом O. На кінцевому етапі обирається найбільш вигідний шлях.

5. Навігаційна сітка - це структура даних, що часто використовується в галузі штучного інтелекту, яка може бути застосована до будь-якої сітки, яка буде забезпечувати пошук маршрутів за алгоритмом пошуку (наприклад, A*). Завданням навігаційної сітки є переміщення об'єкта згідно навколишнього фізичного середовища та уникнення фізичних зіткнень з іншими об’єктами простору, тому вона намагається знайти найпростіший і найбільш реалістичний у пересуванні маршрут. Найпоширеніший спосіб використання даної системи є відеоігри, які вимагають реалістичного пересування та шляхів для різних персонажів, наприклад, ворог, який хоче напасти на певну ціль.

На основі проведеного аналізу було обрано два алгоритми для визначення найбільш підходящого – це алгоритм A* та алгоритм Дейкстри. Проведений аналіз залежностей часової складності.

Алгоритм A*:

Якщо евристична функція немонотонна, час роботи алгоритму буде експоненційним, оскільки точки можуть переглядатись більше одного разу. При наближенні до точної оцінки, менше точок відстежуються.

На швидкість впливають: реалізація списків досліджуваних вершин, також операції зменшення швидкодії є: додавання, вилучення і зміни елементів в списках, реалізація структур даних.

Допустимо, що V - всі вершини деякого графу; евристична функція - монотонна. Список вершин представлений у виді бінарної купи , список пройдених у вигляді масиву. Алгоритм A* має квадратичний час роботи в гіршому випадку:

$$O(|V|^2) \qquad (3.2.5.3)$$

Якщо кожна вершина зберігатиме вказівник, то загальний час роботи алгоритму зменшиться до лінійно-логарифмічного:

$$O(|V| \cdot log|V|) \qquad (3.2.5.4)$$

Алгоритм Дейкстри:

Не залежить від евристичних функції. Вибираються вершини відповідно до вартості шляху. Залежить від структури даних зберігання вершин, операцій: зміни елемента, вилучення.

Якщо кожна точка розглядається один раз, потрібно змін:

$$O(V) \qquad (3.2.5.5)$$

У гіршому випадку, може призвести до зміни одного елемента структури:

$$O(E) \qquad (3.2.5.6)$$

Якщо вершини використовують структуру даних масив і для пошуку мінімуму є алгоритм лінійного пошуку, складність становить:

$$O(V * V + E) = O(V^2) \qquad (3.2.5.7)$$

При використанні черги з пріоритетами, реалізованої на основі двійкової купи, складність наближається до:

$$O(V \log V + E \log E) = O(E \log V) \qquad (3.2.5.8)$$

Кінцевим вибором для реалізації пошуків шляхів від ворога до користувача був обраний алгоритм A*. Тому, що він більш автоматизований для знаходження від початкової до кінцевої заданої точки або області. Алгоритм Дейкстри вигідно застосувати, коли кінцева точка шляху є невідомою.

Отже, під час реалізації штучного інтелекту для дипломного проекту було використано навігаційну сітку та алгоритм пошуку шляху A*, які використовує персонажі-вороги для нападу і відстеження користувача. Щоб знайти шлях між двома точками (ворога і користувача) у віртуальному просторі (сцені), спершу було відзначено стартову і кінцеву локацію їх найближчих оточень. Потім було почато пошук з початкового оточення, відвідуючи всіх сусідів, до тих пір, поки не буде знайдено потрібний шлях. Відстеження всіх відвіданих оточень дозволяє визначити потрібний маршрут, який з'єднує потрібні точки від старту до фінішу. Для нападу більше одного ворога в той же самий час на користувача використовується можливість відстеження їх зіткнень, щоб вороги не відстежували гравця по одному маршруту, а змінювали їх.

3.2.6 Дизайн та функціональне налаштування об'єкту ворог

В цьому підрозділі буде розглядатися створення об'єктів ворогів на сцені. Після розгляду попередніх розділів (за допомогою який засобів та налаштувань був реалізований додаток) в цьому розділі, як приклад практичного застосування наводиться створений об'єкт ворога.

Алгоритм створення об'єкту та дизайн

Усередині віртуального світу (на сцені об'єкт terrain) був створений об'єкт Enemy (ворог). Поведінка та штучний інтелект, якого були реалізовані в скриптах (описані в попередньому пункті). Був обраний прототип ворога, який зображений на Рис. 3.2.6.1.

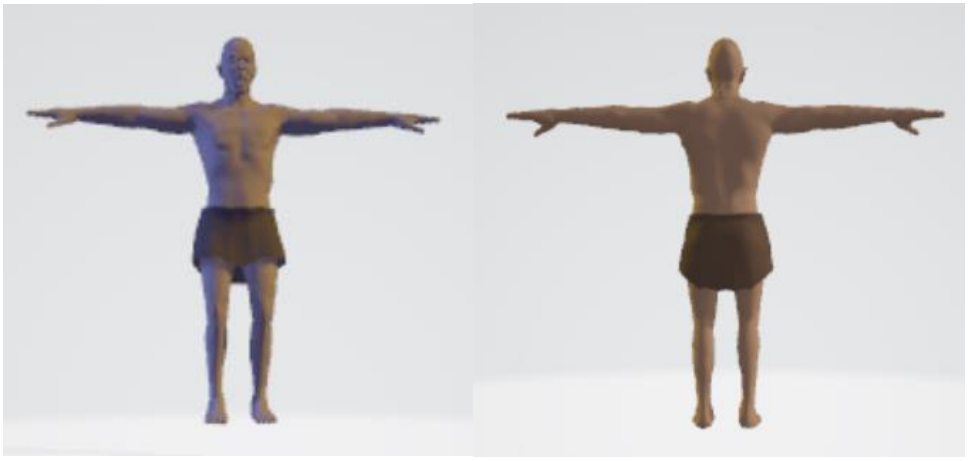


Рис. 3.2.6.1 Прототип ворога

На першій ітерації створення об’єкт являє собою голу 3D-модель. Зображено на Рис. 3.2.6.2.

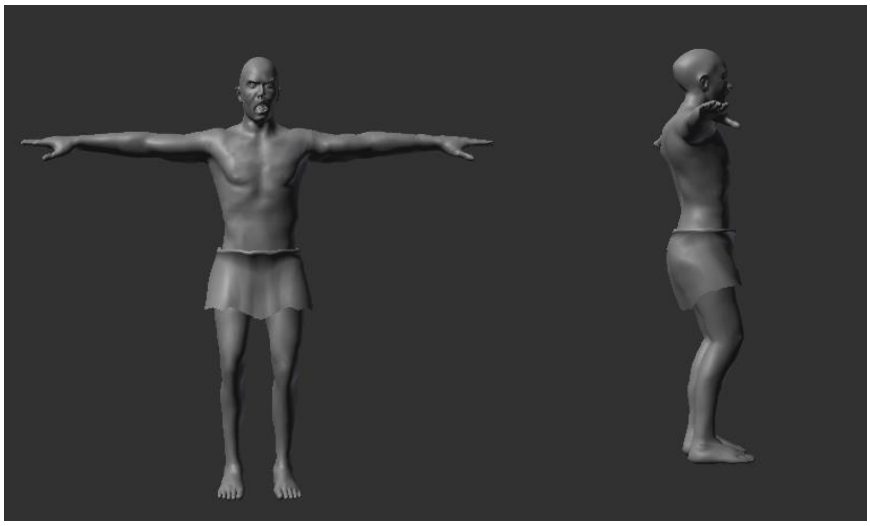


Рис. 3.2.6.2. Модель об’єкту ворог

В майбутньому ця модель буде в себе вміщати дочірні об'єкти, наприклад, таких як: очі, тіло (шкіра), одяг та інші.

Для того, щоб створити більш реальний вид (аналіз з реального світу) для всіх ворогів додатку, потрібно реалізувати зібрку об’єкту. Для цього було знайдено та створено матеріали, зображення яких можна переглянути на Рис. 3.2.6.3.

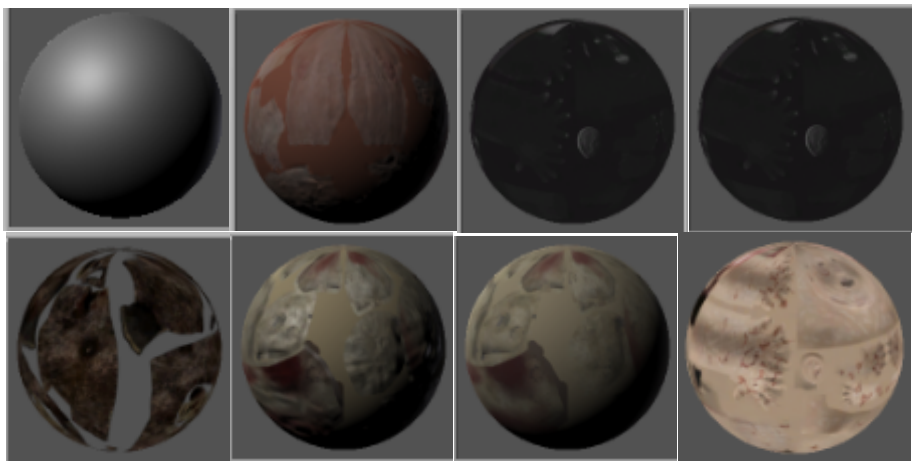


Рис. 3.2.6.3 Матеріали об’єкта ворог

Після успішної збірки об’єкту ворога, було отримано реалістичний варіант, на основі якого будуть розроблені інші можливі суперники, зображено об’єкту можна переглянути на Рис. 3.2.6.4.

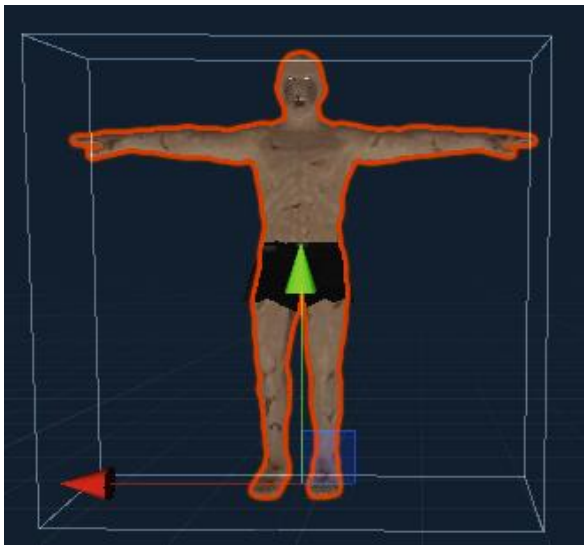


Рис. 3.2.6.4 Готовий об’єкт ворог

3.2.7 Реалізація персонажа додатку

Гравець (персонаж додатку) є найголовнішим компонентом, навколо якого відбуваються всі дії та процеси додатку. Було прийнято рішення реалізувати персонажа, який буде керуватися від першої особи. Всі процеси мультимедійного додатку будуть зображені графічно, з точки зору персонажа.

3.2.7.1 Реалізація фізичної поведінки персонажу

Щоб взаємодія персонажа з навколишнім світом була реалістичною, потрібно налаштувати основні фізичні показники (наприклад, зіткнення). Для цього використовується компонент Character Controller (джерело [26]) - для керування героєм від першої особи. Він надає користувачу простий колайдер схожий за формою на капсулу, який знаходиться у вертикальному стані.

Налаштовані характеристики:

- Slope Limit (обмеження за нахилом) - обмеження колайдеру збиратися по підйомах, значення має бути небільше вказаного – 55 градусів;
- Step Offset (крок зміщення) - перехід на іншу поверхню (наприклад, на сходи), якщо персонаж стоїть на відстані ближче ніж задана - 0.3;
- Skin width (ширина шкіри) - взаємодія двох різних тіл відбувається через зіткнення колайдерів, цей показник вказує максимальну глибину зіткнення (проникнення) між ними. Значення має встановлюватися зважаючи на радіус колайдера. Показник дорівнює - 0.08;
- Min Move Distance (мінімальна відстань переміщення) - використовується, коли персонаж прагне рухається нижче вказаного значення, як наслідок може відбуватися тремтіння. Значення показника дорівнює - 0;
- Center (центр) - зсув колайдеру в просторі віртуального світу без впливання на обертання персонажем. Встановлені показники дорівнюють: X=0, Y=1, Z=0;
- Radius (радіус, ширина) - задана ширина колайдеру (графічно зображено у вигляді капсули). Значення дорівнює - 0.4;
- Height (висота) - показник висоти колайдеру (капсули) визначеного персонажу. Змінюється в межах осі Y, можливо і в позитивному, і в негативному напрямку. Значення показника дорівнює - 2.

3.2.7.2 Реалізація скриптів персонажу

Наступним етапом є реалізація скриптів поведінки гравця (переміщення, керування, аудіо ефекти та інше). Для повного функціонування об’єкту «гравець» було створено з відповідними алгоритмами скрипти та прикріплені до об’єкту Player (гравець):

1. Скрипт Mouse_L. Відповідає за реалізацію методів:
    - прокручування з відповідним кутом та швидкістю графічного відображення віртуального світу за допомогою руху миші. Реалізовано в методі LookAround();
    - метод LockAndUnlockCursor() відповідає за блокування та розблокування курсору за допомогою натискання на клавішу esc;
2. Скрипт Movement_P. Цей скрипт забезпечує переміщення об’єкту гравця по сцені віртуального світу з відповідною заданою швидкістю. Реалізовано в методах:
    - метод MovePlayer() відповідає за рух гравця в просторі за допомогою векторного перетворення, використовуючи клавіши: a - ліворуч, w- вперед, s - назад, d - праворуч;
    - метод Gravity() застосовуються, як накладання гравітації на відповідний об’єкт (наприклад після стрибка персонаж має повернутися назад на поверхню);
    - метод Jump() дозволяє гравцеві перемішуватися вгору на певний час по осі Y до повернення на землю за допомогою гравітації. Для застосування використовується клавіша space;
3. Скрипт SprintCrouch. Відповідає за реалізацію методів:
    - метод Sprint(). При натисканні на клавішу Shift, швидкість переміщення збільшується приблизно до бігу людини. Функція доступна лише при наявності показнику енергії, який з часом зменшується та потім відновлюється да початкового стану;

- метод Crouch(). При натисканні на клавішу C персонаж уявно присідає, зменшується показник висоти (з 1.6 до 1), швидкість пересування в такому стані також зменшується. Вийти в звичний стан можливо при повторному натисканні на клавішу;

4. Скрипт Attack. Використовується для виживання та захисту персонажа у віртуальному світі. Реалізовані методи:
    - метод Shoot() викликається, якщо гравець обрав вогнепальну зброю та натиснув ліву кнопку миші для вистрілу в ворогів. Можливість стріляти відновлюється через певний проміжок часу;
    - метод Sight() використовується для прицілу зброї на ворогів, викликається при натисканні на праву кнопку миші та працює при її утриманні, для виходу із стану потрібно припинити натискання;
    - метод Fired() контроль вистріляної кулі у ворога, якщо позиція кулі співпадає з позицією ворога, тоді дані обробляються та використовуються з скриптом пов’язаним із здоров’ям;
5. Скрипт Health. Для контролю показників здоров’я персонажа та ворогів та дії в разі їх вичерпання (відновлення додатку). Початковий показник здоров’я дорівнює - 100%. Були реалізовані такі методи:
    - метод Damage() при нанесенні шкоди персонажу здійснюється контроль показників здоров’я (відображається відповідна шкала), відмінусовується від значення здоров’я відповідний збиток. Якщо показник стає менше або дорівнює 0, тоді персонаж переходить в стан смерті;
    - метод Died() - додаток переходить у стан відновлення. Зупиняється генерація ворогів в віртуальному світі;
    - метод Restart() - відновлення роботи додатку, персонаж буде знаходитися на початковій позиції з максимальним показником здоров’я;
6. Скрипт Statistics. Використовується для відображення відповідних показників статистики гравця:

- метод Display_health() представляє відображення шкали здоров’я відповідно до поточного значення;
- метод Display_energy() представляє відображення шкали енергії відповідно до поточного значення;

7. Скрипт AudioClip_P зчитує поточний стан гравця і відтворює відповідний аудіо кліп. Реалізовано в методі FootstepSound(). Звук відтворюється при русі персонажу, (якщо натиснути на клавіши a, w, s, d) схожий на кроки. При натисканні паралельно на клавішу Shift персонаж переходить в стан бігу та відтворення аудіо кліпу пришвидшується;
8. Скрипт ForestClip. Реалізований метод Sound() відтворює звук, як фон мультимедійного додатку, схожий до звуків лісу.
9. Скрипт WeaponClip. Відповідає за відтворення відповідних звуків під час використання зброї. Основні методи:
    - метод ShootSound() після вибору вогнепальної зброї, при натисканні на ліву кнопку миші відбувається вистріл на цьому стані розповсюджується звук аудіо кліпу, який схожий на вистріл;
    - метод reload_Sound.Play() забезпечує розповсюдження відповідного аудіо кліпу після стану вистрілу, якщо вогнепальна зброя потребує перезаряд.

Щоб реалізувати вище перераховані скрипти з використанням звуків, було знайдено та збережено підходящі аудіо кліпи в папку Audio.

3.3 Інструкція користувачу програми

Для відтворення програми необхідно запустити файл збірки проекту - Diplom.exe.

Після запуску з’явиться вікно конфігурації гри, в якому буде пропонуватися вибір графічних характеристик:

1) Розширення екрану;
2) Якість графіки;
3) Вибір монітору;
4) Запуск в віконному або повному (на весь екран) режимі.

Можливі налаштування зображені на Рис. 3.3.1.

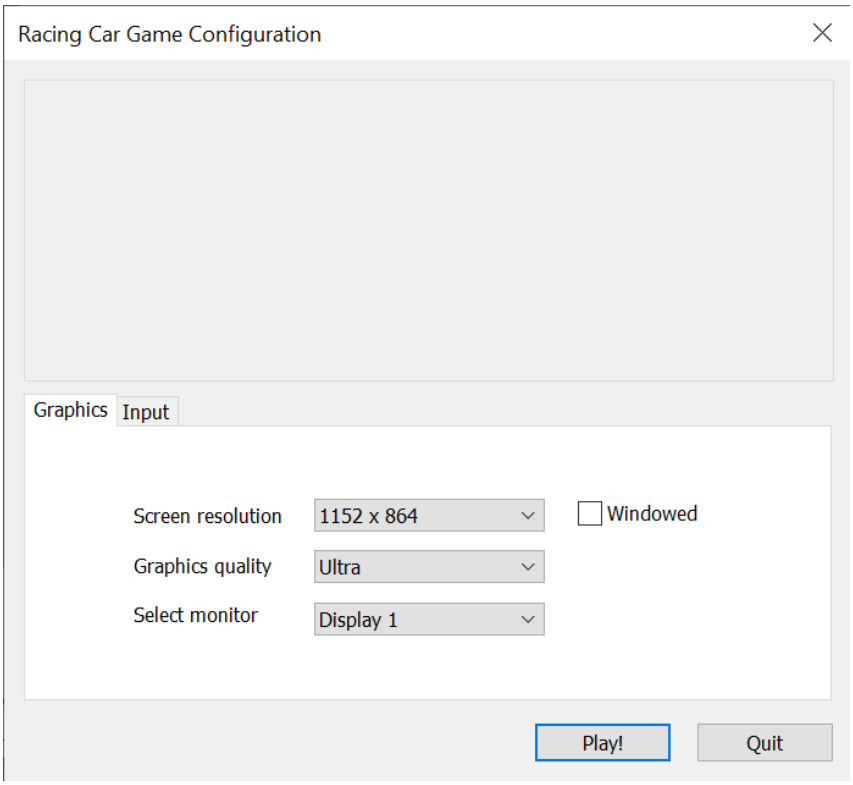


Рис. 3.3.1 Конфігурація гри

Також на вкладці Input (вхідних даних) відображено, якими саме клавішами буде керуватися мультимедійний додаток (стосується керування персонажем). Автоматично будуть стояти клавіши за замовчуванням, але в разі потреби можна

змінити подвійним натисканням лівої кнопки миші на потрібному полі. Зображено на Рис. 3.3.2

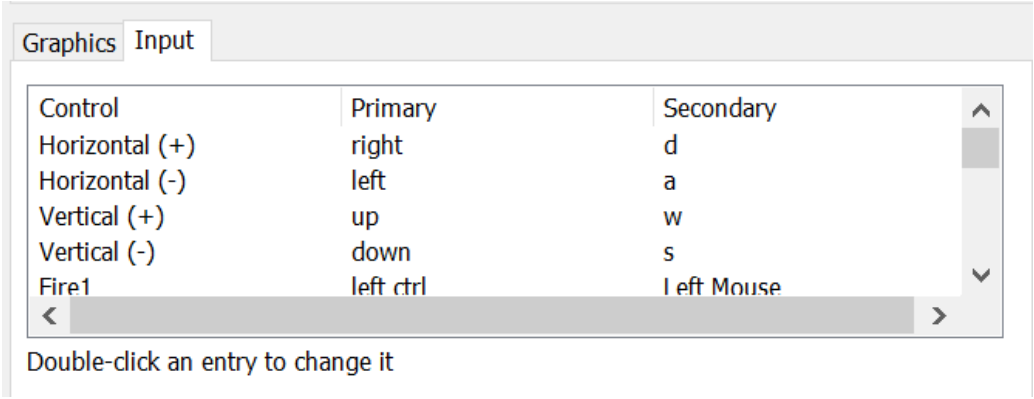


Рис. 3.3.2 Клавіши керування

Після натискання кнопки «Play!» буде здійснений перехід до головного меню гри (зображено на Рис. 3.3.3).

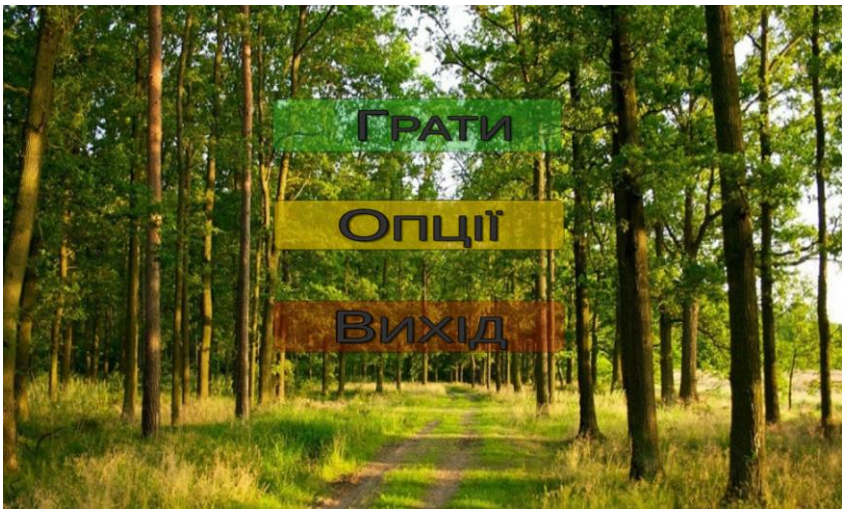


Рис. 3.3.3 Головне меню

Можливі дії користувача:

1) Почати використовувати мультимедійний додаток. У користувача є можливість почати, натиснувши на кнопку «Грати». Тоді буде завантажено сцену віртуального світу з максимальними показниками здоров’я та енергії. Результат дій зображено на Рис. 3.3.4.

Рис. 3.3.4. Перший ітерація віртуального світу

2) Натиснувши на кнопку «Опції», користувач здійснюється перехід в меню опцій, де можна почитати загальний опис сюжету та ознайомитися з можливостями керування мультимедійним додатком (зображено на Рис. 3.3.5).

Рис. 3.3.5.Меню «Опції».

3) Натиснувши на кнопку «Вихід», користувач завершує роботу з мультимедійним додатком (вікна додатка закриваються).
4) Кнопка «Вихід в меню». Якщо гравець знаходиться не в головному меню на кожному додатковому вікні можна здійснити перехід натиснувши кнопку «Вихід в меню», як наслідок гравець буде автоматично направлений в головне меню, де можна здійснити вихід з гри натиснувши кнопку «Вихід» або почати гру спочатку.

ВИСНОВКИ

В процесі роботи над дипломною роботою, в першому розділі було досліджено та проаналізовано основний перелік підготовлених теоретичних матеріалів, з них найважливіше: актуальність предметної області, історія створення комп'ютерних ігор, огляд найпопулярніших жанрів та засобів розробки мультимедійних додатків. Проведений аналіз існуючих підходів розробки мультимедійних додатків на основі аналізу ігрових рушіїв: Unity, Unreal Engine, GameMaker: Studio, Godot, CryEngine. Було обрано програмний інструмент - рушій Unity. На сьогоднішній день у колі незалежних розробників користується найбільшою популярністю із-за своїх базових характеристик, потужних можливостей, різноманітних ресурсів та фінансової підтримки серед розробників, що значно впливає на подальший розвиток інструмента (оновлення версій). Може використовуватися, як для великих проектів, так і для індивідуальної розробки.

В другому розділі на основі отриманих знань, було здійснено проектування мультимедійного додатку. Розроблені та визначені основні елементи: вимоги, структура, жанр, сетинг, механіки (правила) та зображення інтерфейсу сцен, які використовуються для подальшої реалізації проекту.

В третьому розділі були чітко підібрані засоби реалізації додатку. Першим етапом було створення дизайну віртуального світу, головні елементи: terrane (ландшафт), skybox (вигляд неба), підбір різноманітних текстур і створення об'єктів рослинності. Наступним етапом була реалізація проекту на мові програмування C# за допомого скриптів. Створено кінцеві сценарії поведінки головного героя - гравця та персонажів-ворогів. Для реалізації ворогів було використано штучний інтелект - алгоритм пошуку шляху в віртуальному світу до заданої точки (гравця). Визначено кінцеві варіанти сцен та об'єктів.

Отже, під час виконання дипломної роботи було повністю спроектовано та реалізовано мультимедійний додаток з використанням штучного інтелекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мультимедія - Wikipedia, the free encyclopedia. – [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/ Мультимедіа

2. Історія розвитку штучного інтелекту в іграх - [Електронний ресурс]. - Режим доступу: https://stopgame.ru/blogs/topic/93248

3. Game Engine and History of Game Development - [Електронний ресурс]. – Режим доступу: https://www.studytonight.com/3d-game-engineering-with-unity/game-engine

4.Жанри відеоігор - Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Жанри_відеоігор

5. The Many Different Types of Video Games & Their Subgenres - iD Tech.- [Електронний ресурс]. – Режим доступу: https://www.idtech.com/blog/different-types-of-video-game-genres

6. Ігровий рушій - Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Ігровий_рушій

7. Game Engine Analysis - Gamesparks – [Електронний ресурс]. – Режим доступу: https://www.gamesparks.com/blog/game-engine-analysis/

8. Порівняння ігрових рушіїв – [Електронний ресурс]. – Режим доступу: https://sc2tv.ru/blogs/falcon/2019/07/23/sravnenie-igrovykh-dvizhkov

9. 10 найкращих ігрових рушіїв - ulab. – [Електронний ресурс]. – Режим доступу: https://ulab.sumdu.edu.ua/uk/10-najkrashhih-igrovih-rushiiv

10. Visual Studio – Microsoft. – [Електронний ресурс]. – Режим доступу: https://visualstudio.microsoft.com/ru/vs/

11. Троелсен Ендрю, Джепікс Філіп. Мова програмування C# 7 і платформи .NET и .NET Core, 8-е видання. Довідник. Розділ.: Програмування Киев, 2020.– 672с

12. Project Architecture: Unity Project Folder Structure – Unity - [Електронний ресурс]. – Режим доступу: https://visualstudio.microsoft.com/ru/vs/https://learn.unity.com/tutorial/project-architecture-unity-project-folder-structure/?tab=overview#

13. Terrain – Unity - [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ru/current/ScriptReference/Terrain.html

14. Trees – Unity - [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/terrain-Trees.html

15. TreeInstance – Unity - [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ru/current/ScriptReference/TreeInstance.html

16. Textures – Unity - [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/Textures.html

17. Skyboxes – Unity - [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/skyboxes.html

18. Scripting concepts– Unity - [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/ScriptingConcepts.html

19. Creating and Using Scripts – Unity - [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/CreatingAndUsingScripts.html

20. Audio Source – Unity - [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ru/2019.4/Manual/class-AudioSource.html

21. Audio Clipe. – Unity - [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ru/2019.4/Manual/class-AudioClip.html

22. Штучний інтелект - Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Штучний_інтелект

23. Пошук шляху- Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Пошук_шляху

24. Просте пояснення алгоритмів пошуку шляху і A. – Habr -[Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/444828/

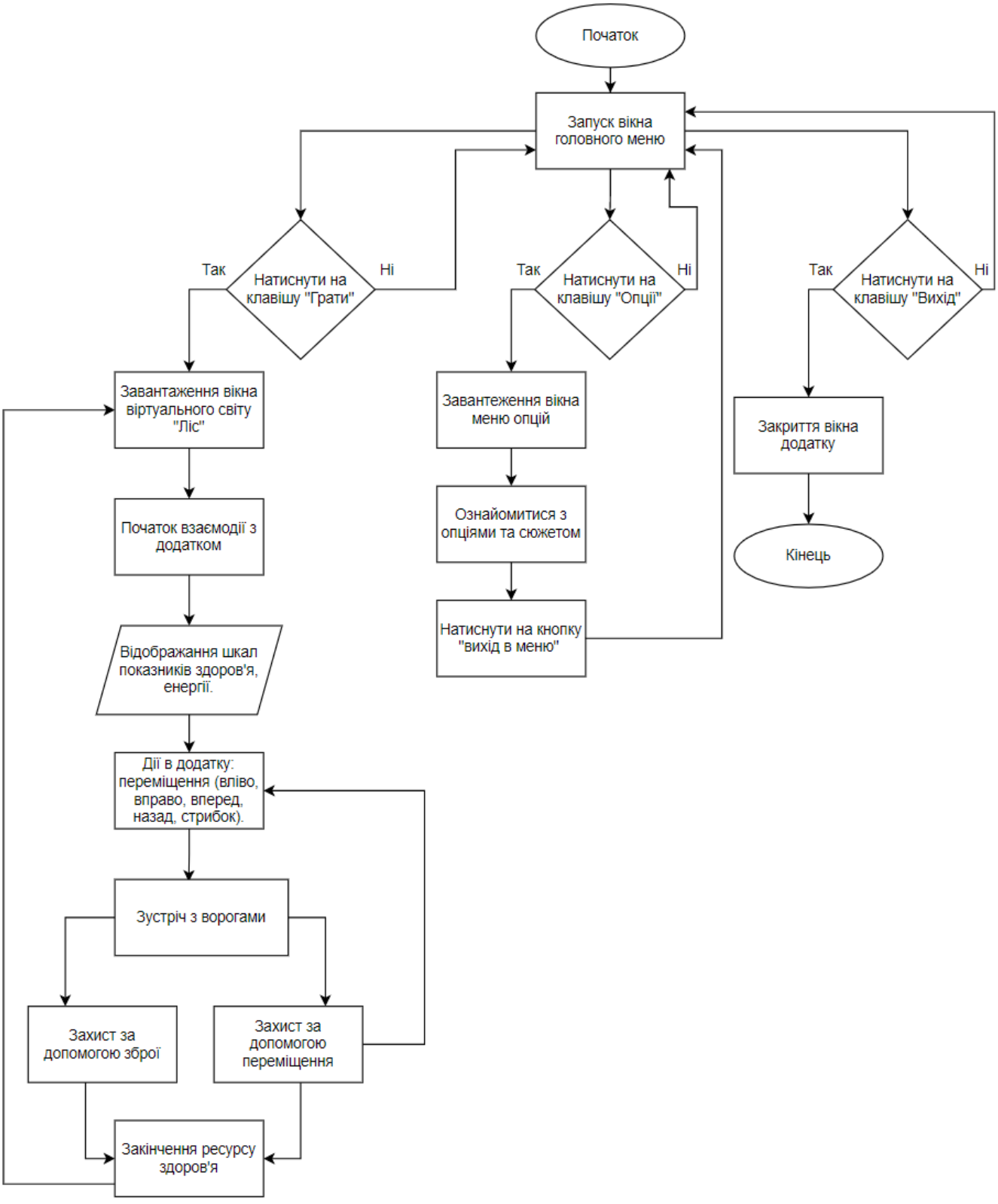
25. Navigation System in Unity– Unity - [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/Manual/nav-NavigationSystem.html

26. Character Controller – Unity - [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ru/2019.4/Manual/class-CharacterController.html

Додаток А – Алгоритм роботи програми

1. Запуск гри:
    1) Завантаження вікна головного меню.
2. Ознайомлення з мультимедійним додатком перед початком взаємодії:
    1) Натискання на клавішу «Грати» або натискання на клавішу меню «Опцій»;
    2) При натисканні на клавішу «Грати», користувач буде автоматично переправлений на сцену віртуального світу (ліс);
    3) Після натискання на клавішу «Опції» гравець може ознайомитися з сюжетом та правилами керування додатком.
3. Початок взаємодії:
    1) Встановлюється шкали здоров’я та енергії на максимальний показник, далі починається взаємодія з віртуальним світом та його об’єктами за його правилами (можливо переглянути в меню опцій).
4. Зворотній зв’язок додатку:
    1) Відображення шкали здоров’я. Шкала може тільки зменшуватися, без можливості відновлення. При вичерпанні шкали додаток відновлюється, починаючи взаємодію спочатку;
    2) Відображення шкали енергії. Шкала може відновлюватися з проходження певної кількості часу;
    3) Вороги переслідують та нападають на персонажа (користувача) за допомогою штучного інтелекту.
5. Вихід з гри:
    1) Гравець натискає на кнопку «Вихід в меню» (присутня на кожному вікні гри, крім головного меню);
    2) Натискання на кнопку «Вихід»;
    3) Закривається вікно програми, дані видаляються.

Додаток Б – Блок-схема алгоритму роботи програми



Додаток В – Software Architecture Document (SAD)

Taras Shevchenko National University of Kyiv

3D-shooter

Software Architecture Document (SAD)

CONTENT OWNER: Tyshchenko Olena

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
1	Version 1.0	March 2021

Table of Contents

1.1 Document Management and Configuration Control Information

- Revision Number: 1
- Revision Release Date: 24/05/2021
- Purpose of Revision: creating the final version of the documentation

1.2 Purpose and Scope

Purpose: to develop and implement a multimedia application (3D-shooter) using artificial intelligence.

In Scope

The project will be based on creating an 3D-shooter game for use in entertainment.

- Single Player
- 3D
- Single level
- 2D for menu systems
- Testing
- Written in C #

Out of Scope

- Will not be multiple player
- Will not be multi-level

1.3 Viewpoint Definitions

1.3.1 Algorithm of multimedia application Viewpoint Definitions

1.3.1.1 Abstract

Views that correspond to the point of view on the algorithm of the multimedia application. The procedure for determining actions, entry points, exit points, transitions.

1.3.1.2 Stakeholders and Their Concerns Addressed

Stakeholders and their concerns addressed by this viewpoint include

- system build engineers who use the elements to produce a running version of the system;
- testers need to know how the program should behave in different situations;
- configuration management specialists who are in charge of maintaining current and past versions of the elements;

1.3.1.3 Elements, Relations, Properties, and Constraints

Elements of the program algorithm are a sequence of precise commands aimed at solving a specific problem. Each action in the algorithm is connected and logical. There are different types of interactions between them: linear algorithm, algorithm with full branching, algorithm with incomplete branching, cycle with precondition, cycle with postcondition, cycle with parameter.

Requirements and restrictions

- Discretion. The algorithm consists of a set of separate commands, each of which is executed for a finite time. Only after completing the execution of one command, the executor proceeds to the execution of another.
- Unambiguity. Each command of the algorithm unambiguously defines the actions of the performer and does not allow their double interpretation. The procedure for performing operations is also unambiguous.

- Formality. Any executor who has a given system of commands of the executor, can, without knowing the essence of the task, execute the algorithm and get the same result.
- Mass. The algorithm must provide for the possibility of solving a group of typical problems, changing the input (initial) data within certain allowable limits.
- Finiteness. The algorithm consists of a finite number of actions, the knees of which are performed over a finite period of time. The execution of the algorithm cannot end in an uncertain situation or not at all.
- Effectiveness. Execution of the algorithm with valid input data will lead to the expected result.

1.3.1.4 Language(s) to Model/Represent Conforming Views

- plain text using indentation or outline form
- Formula-verbal - presentation of algorithms in educational and scientific activities using the language of mathematical formulas with verbal explanations in natural language.
- Graphic - representation of algorithms in the form of graphical schemes (block diagrams or block diagrams UML) to simplify the development and analysis of algorithms, facilitate the transition from writing algorithms to writing programs.
- Software - presentation of algorithms in a programming language for their further processing on a computer.

2. Architecture Background

2.1. Problem Background

2.1.1. System Overview

• Control the direction of the camera view;
• Moving the camera along the surface using physics;
• Movement of the character in a fictional universe (walking, running, jumping);
• Imitation of the use of weapons;
• Creating enemies;
• Creating sound effects and overlaying music;
• Creating a choice in the game (different weapons);
• Creating main menu and option menu functions;
• Information display: the number of resources remaining on the character's energy and health scale.

2.1.2. Goals and Context

In any multimedia application, there are interactions and the more they are, the more interest users will have. During the development of the project, the main emphasis was on the design and study of methods and tools by which the planned can be implemented. Therefore, the final step is the implementation of the functionality that was planned.

Project is an action based 3D-shooter game where the player takes role of the main character to destroy enemies. The gameplay is action-based with no strategic or role-playing elements; instead, the game entirely provides a rush of adrenaline.

The version in this project consists of one level. The created virtual world is a forest with enemies. The character must destroy or flee from enemies to survive in terms of health and energy. Weapons that can be used for protection are also needed.

2.1.3. Significant Driving Requirements

The Architecture Tradeoff Analysis Method (ATAM) is a method for evaluating software architectures relative to quality attribute goals. ATAM evaluations expose architectural risks that potentially inhibit the achievement of an organization's business goals.

In Bass they discuss about “Quality attributes”. These attributes are certain characteristics that the final software should possess to some extent (as given in the requirements). present six of the most common qualities; availability, modifiability, performance, security, testability, and usability.

If a traditional 3D-shooter game is used as an example, these quality attributes can be ordered according to importance. One such ordering might be (1) usability, (2) performance, (3) modifiability, (4) availability, (5) testability, and (6) security. Based on such an ordering, the developing organization can prioritize effort and other resources to achieve the results they wish.

2.2. Solution Background

2.2.1. Architectural Approaches

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Examples:

1. The structure or structures of the system. This implies that a program chiefly consists of several elements, and that the ordering between them are of importance. It also implies that this ordering can be changed as a response to different conditions.

2. Which comprise software elements. This implies that the actual elements chosen to do the job is part of the software architecture. This again implies that although pieces possibly are exchangeable, choosing one rather than the other is an architectural decision.
3. The externally visible properties of those elements. At the same time as one includes every last bit of code into the software architecture, one also abstracts to a sufficient level, looking at the observable effects the different elements of the software architecture have. This allows us to introduce constraints on the different elements, both in terms of interfaces and inter-element relations.
4. And the relationships among them. Lastly we return to the relationships among the elements. This definition clearly states that the form of the final product is important, and that the different roles elements assume, and the relationships amongst them, are also part of the software architecture.

Responsibility-Driven Design. In the article “Object-Oriented Design: A Responsibility-Driven Approach” the authors Wirfs-Brock and Wilkerson present a design method which focuses on the responsibilities of the different entities of a system. They start out by addressing some of the problems with the data-driven design method, most importantly including that it can lead to a violation of encapsulation, since the structure of an object becomes part of it’s definition. They then go on to describe the responsibility-driven approach. In responsibility-driven design, the designers inspect the relationships between the different modules as a client-server relationship. A server is an object that can perform a service (e.g., provides data) and a client is an object that requests services (e.g., requests data). By designing modules in this way, you do not tie down the actual structure of the different modules or classes in the definition, but instead produce objects which have clearly defined services which they can perform. If we look at a game engine, each main module can be viewed as a server. The rendering engine, for example, provides the service of generating an image that can be viewed by the end user. The physics engine can calculate object interactions, and the sound engine can output sound. All these are great examples of object-oriented design, as they provide an encapsulated service, and can, in theory, be exchanged by another module which performs the same services (e.g., by adding a wrapper to the new module).

2.2.2. Analysis Results

Game engine for software that is extensible and can be used as the foundation for many different games without major modification. Here certain desirable architectural traits already surface. Identified a need for the game engine to be data-driven, and that most, if not all, of the game-specific elements should be left out of the game engine. Being data-driven, the game engine should accept new data containing information about the levels, the characters in it, and the possible interactions, and render this to the user in a correct manner. This, in turn, implies that if one uses a complete game engine to create a game, you should only have to focus on the content, not the mechanics allowing you to create a game. Furthermore, a game engine is usually structured around different modules. These modules each have their own responsibility (e.g., rendering, physics, and lighting), and thus the engine itself is also a responsibility driven architecture.

2.2.3. Requirements Coverage

Development Requirements:

- Microsoft Visual Studio;
- Programming language C#;
- Unity;
- Blender;

System Requirements

- Windows 7, 8, 8.1, 10;
- 2 GB RAM or higher;

3. Referenced Materials

1	Bass, Clements, Kazman, Software Architecture in Practice, second edition, Addison Wesley Longman, 2003.
2	ANSI/IEEE-1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, 21 September 2000.
3	R. Wirfs-Brock and B. Wilkerson. Object-Oriented Design: A Responsibility-Driven Approach. SIGPLAN Not., 2010.
4	Jeff Ward. What is a Game Engine?, April 2008. URL: http://gamecareerguide.com/features/529/what_is_a_game_.php.
5	Andrew Rollings and David Morris. Game Architecture and Design: A New Edition. New Riders Games, 2003.
6	Eric Evans. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional, 2003

4. Directory

4.1 Glossary

Term	Definition
Software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
Unity	is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as a Mac OS X-exclusive game engine.
Game engine	is conceptually the core software necessary for a game program to properly run. This is not to be confused with the often incorporated software-development environment designed for people to build video games.
Single Player	is a video game where input from only one player is expected throughout the course of the gaming session. A single-player game is usually a game that can only be played by one person, while "single-player mode" is usually a game mode designed to be played by a single player, though the game also contains multi-player modes.
Software elements	components, connectors, and data--constrained in their relationships in order to achieve a desired set of architectural properties.

4.2 Acronym List

Term	Definition
AI	Artificial Intelligence
OS	Operating System
3D	Three Dimensional
SAD	Software Architecture Document