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

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

РОЗРОБКА 2D-ГРИ В ЖАНРІ ACTION ADVENTURE НА РУШІЇ UNITY

Спеціальність 122 «Комп'ютерні науки»

122 – КРБ – 402.22010222

Виконав студент 4-го курсу, групи 402
_____ М. М. Самойленко
«17» червня 2024 р.

Керівник: д-р техн. наук, доцент
_____ О. В.Козлов
«17» червня 2024 р.

Миколаїв – 2024

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

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

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2024 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук Самойленку Максиму Миколайовичу.

1. Тема кваліфікаційної роботи «Розробка 2D – гри в жанрі action adventure на рушії Unity». Керівник роботи Козлов Олексій Валерійович, д-р техн. наук, доцент. Затв. наказом Ректора ЧНУ ім. Петра Могили від «28» грудня 2023 р. № 271
2. Строк представлення кваліфікаційної роботи студентом «17» червня 2024 р.
3. Вхідні (початкові) дані до роботи: тренди ігрової індустрії, системи та інструменти для створення ігрових застосунків.
Очікуваний результат: 2D – гра в жанрі action adventure на базі ігрового рушія Unity.
4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):
   - аналіз вже створених проектів для кращого розуміння завдання;
   - розробка основних механік застосунку;
   - аналіз графічного інтерфейсу в іграх;

– розробка графічного інтерфейсу для застосунку.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: «Оцінка умов праці фахівців з ІТ-технологій у робочому приміщенні»

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Алексєєва А. О., канд. техн. наук, доцент	

Керівник роботи ___д-р техн. наук, доцент Козлов О. В._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Самойленко М.М._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_29_» _____січня____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН
Виконання кваліфікаційної роботи

Тема: Розробка 2D – гри у жарні action adventure на рушії Unity

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРБ	10.11.2023	15.11.2023	Виконано
2	Отримання завдання на виконання КРБ	10.01.2024	15.01.2024	Виконано
3	Складання календарного плану роботи на весь період виконання КРБ	16.01.2024	30.01.2024	Виконано
4	Отримання завдання на переддипломну практику	15.04.2024	29.04.2024	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до КРБ	29.04.2024	11.05.2024	Виконано
6	Розробка звіту з переддипломної практики	12.05.2024	15.05.2024	Виконано
7	Виконання КРБ: аналіз сучасного стану розробки ігор на unity, огляд існуючих технологій, розробка ПЗ	13.05.2024	22.06.2024	Виконано
8	Перший попередній захист КРБ на засіданні комісії кафедри	27.05.2024	27.05.2024	Виконано
9	Доробка та остаточне оформлення КРБ	28.05.2024	09.06.2024	Виконано
10	Другий попередній захист КРБ на засіданні комісії кафедри	10.06.2024	10.06.2024	Виконано
11	Подання КРБ рецензенту	13.06.2024	13.06.2024	Виконано
11	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	17.06.2024	21.06.2024	Виконано
12	Захист КРБ перед екзаменаційною комісією (ЕК)	24.06.2024	28.06.2024	Виконано

Розробив студент __Самойленко М.М._____ _____
(прізвище, ім'я, по батькові студента) (*підпис*)

Керівник роботи __ д-р техн. наук, доцент Козлов О. В.__ _____
(*посада, прізвище, ім'я, по батькові)* *(підпис)*
«_29_» _____01_____ 2024 р.

АНОТАЦІЯ

кваліфікаційної роботи студента групи 402 ЧНУ ім. Петра Могили
Самойленка Максим Миколайовича
Тема: «Розробка 2D- гри в жанрі action adventure на рушії unity»

Актуальність. Action adventure – один з найпопулярніших жанрів ігор, що поєднує в собі елементи бою, дослідження та розгадки головоломок. Цей жанр пропонує динамічний геймплей, захопливі сюжети та різноманіття ігрового процесу.

Об'єкт роботи є процеси створення ігрових 2D-застосунків.

Мета роботи – розробка та впровадження 2D гри на рушії Unity, яка сприятиме розвитку моторики, покращенню швидкості прийняття рішень та когнітивних навичок користувачів через інтерактивні сценарії та завдання.

Предмет роботи – методи та технології розробки ігрових застосунків на рушії Unity.

Робота складається із фахового розділу та спеціальної частини з охорони праці.

Пояснювальна записка кваліфікаційної роботи бакалавра складається із вступу, 3 розділів, висновку, списку використаних джерел та додатків.

У першому розділі розкрита теоретична частина роботи. Проведений аналіз ігрової індустрії та її трендів. Проаналізовано особливості жанрів та їх відповідних механік. Досліджені методи та інструменти для створення ігрових застосунків.

У другому розділі проведено моделювання та проектування досліджуваної системи, а саме ігрового застосунку.

У третьому розділі описана розробка відеогри.

У четвертому розділі розкрито питання спеціальної частини.

В результаті розроблено 2d гру на мові програмування C# з використанням Tiled2Unity на рушії Unity.

Бакалаврська кваліфікаційна робота містить 57 сторінок, 19 рисунків, 0 таблиць, 25 використаних джерел та 1 додаток.

Ключові слова: C#, Unity, Unreal Engine, PhotoShop, Visual Studio Code, 3D Max, GameMaker: Studio.

ABSTRACT

To the qualification work of a student of group 402 of

ChNU named after Petro Mohyla Black Sea National University

Samoylenko Max Mykolayovych

Topic «Development of 2D - games on the Unity platform»

Object: processes of creating 2D game applications;.

The goal of the work is the development and implementation of a 2D game on the Unity engine, which will promote the development of motor skills, improve the speed of decision-making and cognitive skills of users through interactive scenarios and tasks.

The subject of research is the methods and technologies of developing game applications on the Unity engine.

The work consists of a professional section and a special part on labor protection.

The explanatory note of the thesis consists of an introduction, 3 chapters, a conclusion, a list of used sources and appendices.

The theoretical part of the work is disclosed in the first chapter. An analysis of the gaming industry and its trends has been carried out. The features of genres and their respective mechanics are analyzed. Researched methods and tools for creating game applications.

In the second chapter, modeling and designing of the studied system, namely the game application, is carried out.

The third chapter describes the development of the video game.

The issue of the special part is revealed in the fourth chapter.

As a result, a 2d game was developed in the C# programming language using Tiled2Unity on the Unity engine.

Bachelor thesis contains 57 pages, 19 figures, 0 tables, 25 used sources and 1 appendices.

Keyword: C#, Unity, Unreal Engine, PhotoShop, Visual Studio Code, 3D Max, GameMaker: Studio.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 3
ВСТУП 4
1.1.Жанр ігрового застосунку 7
1.2 Ігрові механіки, притаманні 2D гри 8
1.3 Проєкти, які стали основоположниками жанру action-adventure 9
1.4 Візуальна частина 11
1.5 Анімація 15
Висновки до першого розділу 29
2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ 31
2.1 Системи гравця 32
2.2 Планування першого рівню 35
Висновки до другого розділу 36
3 ІГРОВИЙ РУШІЙ У ДІЇ: ВІЗУАЛЬНА РЕАЛІЗАЦІЯ 2D ЗАСТОСУНКУ 38
3.1 Втілення задуму в життя завдяки рушію Unity 38
3.2 Розробка механік гравця 39
3.3 Розробка графічних елементів для гри 41
3.4 Робота з постпроцесингом 45
Висновки до третього розділу 46
ВИСНОВКИ 48
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 50
ДОДАТОК А ЛІСТИНГ КОДУ ЗАСТОСУНКУ 53

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – Програмне Забезпечення

ПК – Персональний комп’ютер

NPC – Non palyer character

RPG – Role playing game

URP – Universal Render Pipeline

ВСТУП

Ігри завжди були важливим елементом розвитку людей, та організмів. Люди, як і тварини, в дитинстві через гру з однолітками пізнають навколишній світ, розвивають мислення та вчаться виконувати різні завдання, такі як рахування і письмо.

З появою перших комп'ютерів та мов програмування людство навчилося створювати комп'ютерні ігри. Спочатку це було просто розвагою, але поступово це перетворилося на повноцінну індустрію.

На сьогодні, завдяки потужним комп'ютерам, здатним обробляти інформацію за лічені мілісекунди, ігрова індустрія знаходиться на піку розвитку. Деякі сучасні ігри мають таку реалістичну графіку, що їх легко сплутати з реальними фотографіями.

Раніше кар'єра в ігровій індустрії вважалася нетрадиційною, бо мало хто розумів, як можна заробляти на створенні ігор. Зараз ситуація кардинально змінилася. Великі компанії вкладають сотні мільйонів доларів у розробку ігор, і під час пандемії популярність відеоігор зросла, адже багато людей змушені були залишатися вдома.

Процес створення відеоігор аналогічний до розробки інших програмних продуктів. Він включає роботу програмістів, менеджерів, дизайнерів, звукових інженерів та інших фахівців. Для цього потрібні люди з різноманітними навичками: економісти, менеджери, художники (2D і 3D), музиканти та фахівці зі звукових ефектів.

Розвиток ігрової індустрії сприяв появі нових жанрів: шутери, пригоди, головоломки, навчальні, симулятори, платформери, RPG, стелс тощо. Навчальні ігри особливо популярні серед дітей, бо допомагають вивчати нові мови, рахувати, писати та креативно мислити. Симулятори використовуються в освітніх установах для навчання водінню автомобіля чи літака, що значно дешевше, ніж купівля реального обладнання. Темою розроблюваного застосунку було обрано 2D гру у жанрі action adventure – для більшого ознайомлення з рушієм Unity, також для

розробки був проведений аналіз ігрової індустрії та жанру action adventure в цілому, також способи використання різних обчислювальних алгоритмів в ігрових проектах.

Для розробки проєкту була обрана 2D гра в жанрі action adventure, щоб краще ознайомитися з рушієм Unity. Проведено аналіз ігрової індустрії та жанру action adventure, а також методів використання обчислювальних алгоритмів у ігрових проєктах.

Характеристики розроблюваного проекту:

Платформою для розробки було обрано ігровий рушій Unity, інструменти якого підходять для ігор різного масштабу, від інді-розробників до великих студій.

- мова програмування – C#, яка використовується при роботі з Unity.
- враховані критерії відчуття від гри;
- різні сценарії поведінки у ворога;
- призначення для одного гравця;
- захоплюючий контент у жанрі action adventure 2d.

Об’єкт роботи – процеси створення ігрових 2D-застосунків.

Предмет роботи – методи та технології розробки ігрових застосунків на рушії Unity.

Мета кваліфікаційної роботи - розробка та впровадження 2D гри на рушії Unity, яка сприятиме розвитку моторики, покращенню швидкості прийняття рішень та когнітивних навичок користувачів через інтерактивні сценарії та завдання.

Для досягнення мети необхідно виконати такі завдання:

- проаналізувати жанр action adventure та існуючі 2D проекти на рушії Unity;
- розробити основні механіки застосунку та різні сценарії поведінки;
- Провести дослідження створення візуальну для ігор;
- створити візуальну частину застосунку та розробити графічний інтерфейс для застосунку;

- провести дослідження створення візуалу для ігор;
- створити візуальну частину застосунку;
- аналіз графічного інтерфейсу в іграх;
- розробити графічний інтерфейс для застосунку.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ ІГРОВОГО ЗАСТОСУНКУ

1.1.Жанр ігрового застосунку

Але навіть сьогодні, коли існують ігри з реалістичною графікою, 2D проекти все ще створюються, і деякі з них здатні конкурувати з великими 3D іграми. Наприклад, Hollow Knight, Ori and The Blind Forest, Dead Cells, Katana Zero, Celeste, Cuphead, та Stardew Valley.

Жанр action-adventure один із самих давніх жанрів комп'ютерних відеоігор.

Cave Story, Hollow Knight, Shovel Knight, Axiom Verge, The Legend of Zelda, - це все назви ігор - у жанрі action-adventure. Деякі назви цих ігор дуже знайомі багатьом споживачам ігрового контунту, якщо не всім гравцям, які полюбляють цей жанр 2D ігор.

Усі ці ігри мають схожі геймплейні та геймдизайнерські елементи, такі як вид збоку, управління та архітектура загального завдання. З розвитком технологій цей жанр еволюціонував і перейшов із 2D простору до 3D. Наприклад, The Legend of Zelda: Ocarina of Time – ця гра для консолі Nintendo 64 була першою 3D-інкарнацією легендарної серії The Legend of Zelda. Вона відзначилася величезним відкритим світом, захоплюючими головоломками та епічними боями з босами.

Tomb Raider – серія ігор Tomb Raider перейшла з 2D до 3D з випуском першої гри в 1996 році. Гравець керує Ларою Крофт, археологом-авантюристом, яка вирушає на пошуки стародавніх артефактів.

Dark Souls – це гра для різних платформ, включаючи PC, PlayStation та Xbox, змішує елементи action-adventure з RPG та викликаючим геймплеєм. Вона відома своєю високою важкістю та темною фентезійною атмосферою.

Деякі зарекомендували себе більш успішно ніж інші.

Жанр 2D action-adventure розпочав свій розвиток наприкінці 1970-х і на початку 1980-х років з появою перших відеоігор. Одним із найраніших і найважливіших представників цього жанру є гра "Adventure" (1979) для ігрової консолі Atari 2600, створена Уорреном Робінеттом. Ця гра значною мірою

визначила структуру та механіку жанру action-adventure, включаючи дослідження світу, бої з ворогами та вирішення головоломок.

Наступним важливим етапом стало з'явлення таких класичних ігор, як "The Legend of Zelda" (1986) на консолі Nintendo Entertainment System (NES) і "Metroid" (1986) на тій же консолі. Ці ігри не лише утвердили жанр action-adventure, але й внесли свої власні інновації, такі як відкритий світ, нелінійний геймплей і елементи RPG.

З плином часу жанр розвивався, приймаючи різноманітні форми і включаючи різні елементи геймплею. Однак його коріння можна простежити до цих ранніх класичних ігор кінця 1970-х – початку 1980-х років.

Згодом розробники із досвідом почали додавати до таких ігор нові механіки. Біг по стінам(вертикальний та горизонтальний), акробатичні елементи та багато іншого. Все це стало можливим із переходом до тривимірного простору. Але навіть у наш час коли, як вже згадувалось раніше існують ігри із реалістичною графікою, 2D проєкти досі створюються.

1.2 Ігрові механіки, притаманні 2D гри

Ігрові механіки можуть складатись з різних речей, але можна взяти саме просте наприклад, ігрові елементи – гравець може пересуватися вліво та вправо, стрибати, карабкатися по сходах та плейнах для дослідження гри.

Бойова система – зазвичай включає в себе використання зброї або бойових навичок для перемоги ворогів або босів. Це може бути ближній бій або дальній бій з використанням різноманітної зброї.

Розв'язання головоломок може включати знаходження ключів, взаємодію з об'єктами на рівні, використання предметів або здібностей героя для вирішення складних головоломок.

Збір предметів та апгрейдів, гравець може збирати предмети, які допомагають у подоланні перешкод або відкритті нових можливостей. Це можуть бути валюта, яка може бути використана для покупки предметів або апгрейдів, або

просто корисні предмети, які підвищують здібності героя. Дослідження світу, гравець може вивчати різні локації, знаходити та розблоковувати секрети, взаємодіяти з NPC (неперсонажні персонажі) та розвивати історію гри. І напевно останнє, це боси та епічні битви. Часто гравцю доводиться зустрічатися з могутніми ворогами або босами, з якими він повинен боротися в епічних сутичках для продовження історії гри.

1.3 Проєкти, які стали основоположниками жанру action-adventure

Super Mario – це гра, відома кожному. Випущена в 1985 році компанією Nintendo, вона завоювала серця мільйонів гравців. Основною метою гри є порятунок принцеси, яку захопив злий лиходій. Гравці стикаються з різними ворогами та викликами, що постають на шляху головного героя. Зображення гри Super Mario можна побачити на (рис. 1.1).

Бойова задача гри є надзвичайно простою: щоб перемогти ворога, Маріо повинен стрибнути йому на голову. За цими правилами гравець долає всі перепони. Проте, щоб уникнути одноманітності сутичок, розробники додали різноманітних ворогів. Наприклад, черепаху необхідно спочатку притиснути стрибком, а потім стрибнути на її панцир, щоб запустити її вздовж платформи. Це перетворює черепаху на своєрідну зброю, оскільки вона збиває інших ворогів з платформи, але й сама може зашкодити Маріо. Окрім звичайних викликів та сутичок, на рівнях розкидано різні бонуси та монети, які надають гравцеві тимчасову перевагу, змінюють темп гри та додають очки для змагань з іншими гравцями.

Окрім стрибків на ворогів, Mario може використовувати різні бонуси, які тимчасово змінюють його зовнішність та здібності. Наприклад, збивши гриб, він стає "Super Mario", збільшуючись у розмірах та отримуючи можливість розбивати цеглою головою. Інші бонуси дають можливість літати, кидати вогняні кулі або кидатися бомбами. Ці трансформації не лише роблять гру більш динамічною, але й відкривають нові можливості для проходження рівнів та пошуку секретів. На багатьох рівнях Super Mario можна знайти секретні місця, куди не так просто

дістатися.

Рисунок 1.1 – Гра Super Mario

У грі *The Legend of Zelda: Ocarina of Time*. равцеві надається більше варіантів управління персонажем. Крім базових дій, таких як ходьба та стрибки, введено додаткові можливості, що розширюють можливості героя. Наприклад, головний герой, Лінк, може взаємодіяти з навколишнім середовищем більш динамічно.

Лінк може взяти на озброєння різні предмети, які допомагають йому в різних ситуаціях. Наприклад, він може використовувати меч для боротьби з ворогами, лук та стріли для атаки з відстані, гак для зачеплення за ворожі об'єкти або перекинення через прірви, або молот для руйнування перешкод (рис. 1.2).

Крім цього, Лінк має можливість взаємодіяти з різними об'єктами в ігрі. Наприклад, він може підійти до дверей і відкрити їх, переміщати важкі предмети для розв'язання головоломок, або навіть використовувати свою окарину для виклику різних мелодій, що мають різні ефекти на світ навколо.

Такі можливості дозволяють гравцю досліджувати світ гри більш глибоко, вирішувати головоломки та подолати різноманітні виклики, що створюють більш насичений та захоплюючий геймплей. Це робить гру не лише розважальною, але й стимулюючою, адже вона кидає виклик гравцю на інтелектуальному та

стратегічному рівні Гравцю надається свобода досліджувати світ гри в будь-якому порядку, шукати альтернативні шляхи досягнення цілей.

Рисунок 1.2 – Гра The legend of Zelda: Ocarina of Time

1.4 Візуальна частина

Візуальна складова є невід'ємною частиною гри. Основною причиною цього є те, що привабливий вигляд гри привертає увагу потенційних покупців, збільшуючи її шанси на успіх. Пригодницькі ігри можуть бути виконані в різноманітних стилях, таких як піксель-арт, традиційна 2D-анімація або сучасне 3D завдяки досягненням у технологіях.

Піксель-арт – це стиль графіки, в якому зображення створюються шляхом ретельного редагування окремих пікселів. Цей стиль виник на зорі розвитку відеоігор, коли обчислювальні можливості комп’ютерів були досить обмеженими, і піксельна графіка була єдиним можливим способом відображення об’єктів на екрані.

Та незважаючи на такі обмеження художники, які працювали над створенням спрайтів для тогочасних ігор навчились і за допомогою декількох пікселів створювати образи необхідні для розуміння.

Попри такі обмеження, художники, що працювали над створенням спрайтів для ігор того часу, змогли навчитися створювати зрозумілі образи навіть за

допомогою обмеженої кількості пікселів. З розвитком технологій обчислювальні потужності комп’ютерів почали зростати, і кількість допустимих пікселів збільшувалася, що призвело до появи десятків тисяч пікселів для редагування. Через це попит на традиційний піксель-арт почав спадати.

Проте мода на речі є циклічною, тому інтерес до піксель-арту знову почав зростати. У 2010 році почали з'являтися ігри, виконані у цьому ретро стилі, що сподобалося як людям, які пам’ятали ту епоху, так і новому поколінню гравців. Нові шанувальники також знайшли в цьому стилі свою привабливість. робота в піксель - арт стилі (рис. 1.3).

Рисунок 1.3 – Робота в піксель - арт стилі

Зараз випуск ігор у стилі піксель-арт є досить поширеним явищем. Особливо це стало помітним з різким збільшенням кількості інді-проєктів на ринку відеоігор, більшість з яких обирають ретро-стиль через його ностальгічний вигляд та простоту виконання.

Для створення зображень у цьому стилі необхідно знати спеціальні прийоми, оскільки недостатньо просто відкрити будь-який графічний редактор, створити файл малого розміру та почати малювати як звичайне зображення. Процес

створення зображень у піксель-арті значно відрізняється від звичайного, адже розташування кожного окремого пікселя може суттєво змінити зміст зображення. Наприклад, у випадку анімації зміна положення одного пікселя може бути досить помітною. Тому існують окремі техніки, які необхідно знати для створення відповідних зображень у цьому стилі. Серед таких технік можна виділити дітеринг, створення градієнтів та кластерів.

Піксель-арт часто обирають інді-розробники, оскільки його значно простіше створювати, ніж звичайне 2D. При бажанні та наявності додаткового часу сам розробник може створити графіку для своєї гри, не витрачаючи додаткових коштів. Якщо ж приймається рішення найняти когось для створення графіки, це також виходить дешевше, оскільки робота в цьому стилі є досить простою і дозволяє пробачати більшість помилок, які були б неприпустимими в інших стилях.

2D стиль – має певну схожість із піксель-артом, але вимагає додаткових знань та навичок, оскільки в цьому випадку потрібно працювати з повноцінними зображеннями великих розмірів. Це означає, що для створення якісної картинки або персонажа з відповідними анімаціями необхідно мати досвід у малюванні, знати пропорції людського тіла, а також володіти навичками створення дизайну персонажів і фонів. Цей стиль не передбачає легкої роботи з ним, оскільки він не прощає помилок.

Гра Star Renegades (рис. 1.4).

Рисунок 1.4 – Star Renegades

Першою тривимірною грою була Maze Wars, Star, створена в 1973 році. Цей стиль надає можливість створювати тривимірні зображення, що дозволяє гравцям зануритися у віртуальний світ з усією його глибиною та деталями. Тривимірна графіка відкриває безліч можливостей для розробників, але також потребує значних знань і ресурсів. Для створення 3D-ігор необхідні навички роботи з моделями, текстурами, освітленням та анімацією. Крім того, 3D-ігри потребують потужних обчислювальних ресурсів для обробки і відтворення складної графіки в режимі реального часу.

Геймплей Maze Wars був простим, але захоплюючим. Гравці блукали лабіринтом, шукаючи один одного та стріляючи з пістолетів. Гра не мала сюжету, але пропонувала захоплюючий мультиплеєрний досвід, який був революційним для того часу. Maze Wars мала значний вплив на розвиток 3D-ігор. Вона продемонструвала потенціал тривимірної графіки та геймплею, надихнувши розробників на створення нових та інноваційних ігор.

Гра Maze runner (рис. 1.5).

Рисунок 1.5 – Гра Maze runner

У ті часи тривимірні ігри виглядали зовсім не так, як зараз. Об'єкти найчастіше нагадували абстрактні образи, схожі на режим wireframe у сучасних 3D-редакторах. З розвитком комп'ютерних технологій розробники отримали можливість використовувати повноцінні 3D-моделі у своїх проєктах, що дозволило додати глибини до ігрового процесу. Проте навіть ігри, в яких основою було

переміщення лише у двох площинах, почали використовувати 3D-графіку для створення більш реалістичних і захоплюючих візуальних ефектів.

Гра Trine (рис. 1.6).

Рисунок 1.6 – Гра Trine

1.5 Анімація техніки

Після візуальної частини наступним аспектом, на який звертає увагу гравець при запуску гри, є анімація. Оскільки ігри є імерсійним продуктом, важливо, як виконані рухи персонажів і наскільки вони виглядають живими.

Наприклад, гра *Marvel's Spider-Man*, ексклюзив для PlayStation 4 та 5, сподобалась гравцям не тільки завдяки захоплюючому геймплею, а й завдяки анімаціям рухів головного героя. Під час польоту відчувається, що ти керуєш справжньою людиною. Це, звісно, заслуга не тільки аніматорів, оскільки над створенням рухів працювали також каскадери в костюмах захоплення руху, використовувалися спеціальні технології, інверсна кінематика тощо. Весь цей комплекс робіт дав результат, що не випробувавши повноцінний ігровий процес, гравець вже хоче грати в цей продукт.

2D анімація знайома кожному, адже багато хто з нас дивився мультфільми в дитинстві. Цей спосіб створення ілюзії руху повністю застосовується і в ігровій індустрії з усіма його правилами та особливостями. Єдине, що аніматорам більше не потрібно малювати все вручну, а можна використовувати спеціальне програмне забезпечення.

2D анімація створюється шляхом відтворення серії зображень з певним інтервалом, що створює ілюзію руху об’єкта. Однак цей спосіб створення анімації потребує певних знань і навичок.

По-перше, аніматор повинен вміти малювати. Якщо аніматор працює з персонажами, він також повинен знати анатомію та пропорції людського тіла або створінь, для яких створюється анімація. До цього додається знання перспективи, форми, щоб вміти «крутити об'єкти в просторі», і, найголовніше, принципи анімації, оскільки без них рухи будуть не реалістичними і в деякій мірі нагадуватимуть рухи роботів.

Дана анімація створюється за допомогою програвання серії зображень з певним інтервалом, що в свою чергу створює ілюзію руху об’єкту.

Але даний спосіб створення анімації потребує деяких знань та навичок.

По – перше аніматор повинен вміти малювати. Якщо це аніматор який працює із персонажами, то додатково він має знати анатомію та пропорції людського тіла або ж створіння для якого буде створюватись анімація. До цього потрібно знати перспективу, форму для того, щоб вміти “крутити об'єкти в просторі” та саме головне – принципи анімації, оскільки без них рухи будуть не реалістичними та схожі в деякій мірі на робота.

Анімації 3D ігор – створення анімації в тривимірному просторі дещо схоже на створення анімації у двовимірному, але має свої особливості. Перед тим як почати створювати 3D анімацію, аніматор повинен отримати змодельований об’єкт і, якщо це персонаж, він має пройти стадію рігінгу.

Рігінг – це створення скелетної структури для моделі створіння, щоб аніматор міг створювати анімаційні кліпи, рухаючи її частинами (рис. 1.7).

Рисунок 1.7 – Було розроблено скелет моделі у середовищі 3D Max для прикладу

Завдяки цій технології можна заощадити час на створенні нових анімацій. Якщо новий персонаж має схожі пропорції з тим, для якого анімаційні кліпи вже були створені, то їх можна використати і на новому об’єкті.

З часом з'явилися технології захоплення рухів (рис. 1.8), які можна використовувати як для тіла, так і для обличчя. При застосуванні цієї технології, як і при класичній анімації, має бути готова модель зі скелетом. Далі необхідно знайти актора з пропорціями тіла, схожими на пропорції моделі, або ж придумати спосіб наблизити його пропорції до необхідних. Наприклад, у фільмі «Повстання планети мавп» 2011 року використовували цю технологію. Актори грали мавп, і для того, щоб їх пропорції тіла співпали з пропорціями моделі мавпи, використовували додаткові палиці.

Рисунок 1.8 - Вбрання для захоплення рухів

В сучасності – майже всі великі компанії в розробці 3D ігор використовують прилади та вбрання для захоплення руху для кращого позиціонування моделі, та віддачі рушія у самій грі.

Інді-проєкт "Hellblade: Senua’s Sacrifice" також повністю застосовував технологію захоплення рухів. Це було важливо не лише для того, щоб створити гарні анімації, але й для того, щоб передати емоційну глибину історії героїні. Розробники прагнули, щоб гравці могли не лише спостерігати за емоціями на обличчі Сенуа, а й відчувати її стан через рухи тіла. Технологія захоплення рухів дозволила досягти цього високого рівня деталізації.

У грі є багато сцен з крупними планами обличчя головної героїні, де важливо точно передати емоції та зробити їх максимально реалістичними для гравця. Використання цієї технології дало змогу аніматорам точно відтворити кожен нюанс міміки та рухів, що значно підвищило рівень занурення у гру та емоційний зв'язок між гравцем та персонажем.

Ще однією перевагою цієї технології є можливість створити систему, яка

дозволяє в режимі реального часу бачити змодельовану анімацію. Це означає, що на знімальному майданчику актори виконують рухи, а в камері відображаються персонажі, на яких ці рухи проектуються. Такий підхід дозволяє аніматорам і режисерам миттєво оцінити результат і внести необхідні корективи на місці, що значно підвищує ефективність і точність роботи.

Штучний інтелект в іграх – Набір програмних методик, які використовуються розробниками відеоігор для створення ілюзії наявності інтелекту в неігрових персонажах, є складним та різноманітним. Ігровий ШІ включає в себе методи, притаманні традиційному штучному інтелекту, але не є повноцінним ШІ в звичайному розумінні.

Розробка ШІ для гри тісно пов'язана з бюджетом проекту, системними вимогами та впливом на ігровий процес. Розробники повинні збалансувати всі ці фактори, щоб створити просту імітацію ШІ, яка буде невимогливою до ресурсів. Через це підходи до створення ігрового ШІ відрізняються різноманітними спрощеннями та хитрощами, які дозволяють досягти бажаного результату – створення ілюзії інтелекту у неігрових персонажів.

Наприклад, у шутерах NPC мають інформацію про точне положення гравця на мапі, що дозволяє їм слідкувати за гравцем крізь стіни. Це може призвести до миттєвої смерті гравця, оскільки у нього немає часу на реакцію. Щоб вирішити цю проблему, розробники штучно знижують влучність NPC.

Ще одним прикладом є гра *Doom*, також у жанрі шутера. У цьому проекті гравцеві надано більше можливостей для переміщення, зокрема, окрім звичайних стрибків та бігу, він має можливість бігати по стінах, робити ривки та ковзання. Таким чином, у грі поєднано горизонтальний та вертикальний геймплей.

Щоб гравець користувався перевагами вертикального геймплею, розробники модифікують деякі параметри ШІ NPC залежно від дій гравця. Якщо гравець малорухливий, то влучність NPC збільшується, але якщо він починає активно рухатися та збільшує швидкість переміщення, влучність NPC знижується.

Ігрових персонажів зі штучним інтелектом поділяють на кілька категорій:

– неігрові персонажі;

– боти NPC;

– моби.

Ігрові персонажі зі штучним інтелектом поділяються на кілька категорій:

– Неігрові персонажі (NPC): Персонажі, які є дружніми або нейтральними до гравця. Це можуть бути торговці, квестові персонажі або напарники

– Ворожі NPC: Персонажі, які ворожо налаштовані до гравця. Вони мають можливості, наближені до можливостей гравця, і зазвичай використовуються в невеликій кількості або у боях один на один для ознайомлення гравця з новим типом ворога. Пізніше їх кількість може збільшитися.

– Моби - Ворожі персонажі з низьким рівнем ШІ, які з'являються у великих кількостях.

Перші спроби створення ігрового ШІ були проведені в іграх 60-х років, але вони були досить обмежені у своїх можливостях. У 1990-х роках, з появою нових ігрових жанрів, розробники почали використовувати скінченні автомати. Проте цього було недостатньо, і деякі ігри мали проблеми з пошуком шляху та керуванням персонажами, що призводило до невірного їх функціонування.

З часом розробники почали збільшувати кількість методів для написання ШІ. Наприклад, у грі GoldenEye 007, випущеній у 1997 році, NPC реагували на дії гравця, виконували перекати та використовували укриття. Проте вороги постійно знали точне місцезнаходження гравця.

У грі Halo 2001 року розробникам вдалося покращити поведінку NPC. Вони могли використовувати транспортні засоби, діяти в команді, розпізнавати загрози, такі як гранати, і робити переміщення, щоб уникнути небезпеки.

Першим проєктом із просунутим ШІ була гра F.E.A.R. 2005 року. Тут NPC могли використовувати укриття, гранати та різні тактики ведення бою, такі як штурм, відступ, виклик підкріплення та спроби оточення гравця. Вони також перекликалися між собою під час бою, що давало гравцеві підказки про їхні дії.

Гра Left 4 Dead, розроблена в 2008 році компанією Valve, включала інструмент процедурного наративу під назвою The Director. Ця система ігрового ШІ аналізувала дії гравців і створювала для них відповідні сценарії. Якщо гравці добре справлялися з завданнями, система ускладнювала ігровий процес, додаючи більше NPC і змінюючи їх типи.

Фінальною метою розробки ігрового ШІ є досягнення рівня, при якому гравцеві буде складно відрізнити справжню людину від NPC. Проте в ігровому ШІ завжди будуть присутні штучні спрощення його поведінки для забезпечення комфорту гравця.

Штучні нейронні мережі являють собою систему з великою кількістю простих паралельних процесів, між якими налагоджена значна кількість зв’язків. Коли така система вперше була створена, люди вважали, що вони майже створили повноцінний штучний інтелект, але помилилися. Незважаючи на схожість архітектури мережі з людським мозком, вони були обмежені у використанні та здатні виконувати лише специфічні завдання.

Перед початком роботи з нейронною мережею її потрібно навчити. Процес навчання відбувається на певній вибірці, враховуючи відповідні парадигми та правила, які обираються залежно від поставленого завдання. Процес навчання включає визначення архітектури мережі та ітеративне налаштування вагових коефіцієнтів.

Парадигми навчання штучної нейронної мережі

До основних парадигм навчання нейронних мереж належить:

- навчання під наглядом;
- навчання без нагляду;
- навчання з підкріпленням.

Навчання під наглядом

Навчання під наглядом означає, що нейромережа має доступ до заздалегідь відомих правильних відповідей. Із кожною ітерацією навчання коригуються вагові коефіцієнти, щоб наблизити результати наступних ітерацій до правильних. На

початку навчання вагові коефіцієнти визначаються випадково. Кількість ітерацій та змін коефіцієнтів визначається розробником. Навчання вважається завершеним, коли мережа починає видавати результат, наближений до бажаного.

Після успішного навчання нейронної мережі, тобто коли вона починає видавати правильний результат на навчальній множині, важливо перевірити її роботу на даних, які не використовувалися в навчанні, тобто на тестовій множині. Якщо результати на нових даних незадовільні, необхідно продовжити навчання.

Навчання без нагляду

Цей метод навчання не потребує зовнішнього впливу для коригування ваг, а використовує внутрішній контроль ефективності через пошук тенденцій у вхідних сигналах. Оскільки мережа не має доступу до заздалегідь відомих правильних відповідей, вона повинна мати інформацію про свою організацію та набір правил.

Парадигма навчання без нагляду спрямована на виявлення зв’язків між окремими групами нейронів, які працюють разом. Якщо на вхід мережі надходить сигнал, який активує будь-який із нейронів у групі, активність всієї групи зростає. І навпаки, якщо активність нейрона знижується, активність всієї групи також зменшується.

Навчання з підкріпленням

Навчання з підкріпленням є проміжним варіантом між двома попередніми парадигмами. Цей метод навчання передбачає використання іншого блоку, який називається «критиком». Цей блок відслідковує реакцію середовища на сигнали від вхідних даних та визначає евристичну похибку, яка використовується в процесі навчання.

Цей метод навчання застосовується, коли працюють із великими системами, де точні методи стають нездійсненними.

Правила навчання штучної нейронної мережі

Існує 4 основних правил за якими проводиться навчання нейронної мережі:

- правило корекції за помилкою;
- правило навчання Больцмана;

- дельта правило;
- навчання методом змагання.

Правило корекції за помилкою: Це правило передбачає зміну вагових коефіцієнтів мережі на основі різниці між очікуваним і фактичним результатом. Метою є мінімізація цієї різниці, або помилки, з кожною ітерацією навчання.

Правило навчання Больцмана: Це правило використовує ймовірнісні методи для визначення станів нейронів і базується на принципах статистичної механіки. Воно спрямоване на досягнення мінімальної енергії системи та знаходження глобального мінімуму функції помилки.

Дельта правило: Відоме також як правило Відроу-Гоффа, це правило використовує метод градієнтного спуску для корекції вагових коефіцієнтів. Метою є мінімізація середньоквадратичної помилки між виходом мережі та цільовим значенням.

Навчання методом змагання: У цьому методі нейрони змагаються за активацію під час навчання. Лише найбільш активні нейрони коригують свої ваги. Це правило часто застосовується в самоорганізованих картах і кластеризації даних.

Ігровий рушій – є центральною частиною будь-якої гри, відповідальною за її технічну реалізацію. Рушії значно полегшують роботу над розробкою проєкту завдяки наявному графічному інтерфейсу та попередньо створеним і вбудованим бібліотекам. Однією з головних переваг рушія є можливість створювати мультиплатформені застосунки, які можна запускати на таких платформах, як ПК, Xbox та PlayStation.

Ігровий рушій відповідає майже за всю основну функціональність застосунку, а саме:

- візуальна частина;
- анімації;
- звук;
- фізичні розрахунки;
- додавання скриптів;

– ігровий штучний інтелект;
– керування ресурсами.

У деяких випадках вже розроблений рушій можна використати для декількох різних проєктів. Сам термін "ігровий рушій" з’явився в 1990-х роках, коли популярність 3D ігор у жанрі шутерів почала зростати. Розробники стали економити ресурси, ліцензуючи деякі частини програмного забезпечення з вже створених проєктів та використовуючи їх у нових. Це дозволило зменшити витрати на розробку та заощадити час.

Згодом ці частини ліцензованого ПЗ почали удосконалювати та надавати іншим розробникам. Наприклад, такі рушії, як Unity, Unreal Engine, Frostbite, CryEngine, Godot та інші, стали доступними для широкого кола розробників. Це принесло додатковий дохід компаніям-розробникам рушіїв з кожного проєкту, створеного за допомогою їхнього програмного забезпечення. Наприклад, компанія Epic Games, розробники Unreal Engine, отримують 30 відсотків від продажу проєктів, розроблених з використанням їхніх технологій.

Ігрові рушії часто мають компонентну архітектуру, що дозволяє змінювати або доповнювати вже існуючі системи новими, більш ефективними. Наприклад, для фізичних розрахунків можна використовувати Havok або PhysX, а для звуку – FMOD.

Висока гнучкість ігрових рушіїв дозволяє використовувати їх не тільки в ігровій індустрії. Завдяки здатності створювати графіку в реальному часі, ігрові рушії часто застосовуються в розробці реклам, фільмів та в архітектурній візуалізації.

Ігрові рушії та системи рендерингу графіки побудовані на основі графічних API, таких як Direct3D та OpenGL. Вони також використовують низькорівневі бібліотеки, що забезпечують апаратно незалежний доступ до компонентів комп’ютера.

До найпопулярніших ігрових рушіїв належать:

– GameMaker: Studio;

– Unreal Engine;
– Unity.

GameMaker: Studio – це потужний ігровий рушій, розроблений Марком Овермарсом, який спеціалізується на створенні двовимірних ігор. Наразі він належить компанії YoYo Games. Це програмне забезпечення дозволяє кросплатформену розробку, але лише у преміум версіях; безкоштовна версія підтримує створення проєктів тільки для платформи Windows.

GameMaker: Studio підтримує роботу з різними бібліотеками та надає інструменти для роботи зі скалярними та векторними величинами, а також математичними виразами. Вбудований фізичний рушій Box2D дозволяє здійснювати маніпуляції з фізикою та фізичними об’єктами (рис. 1.9).

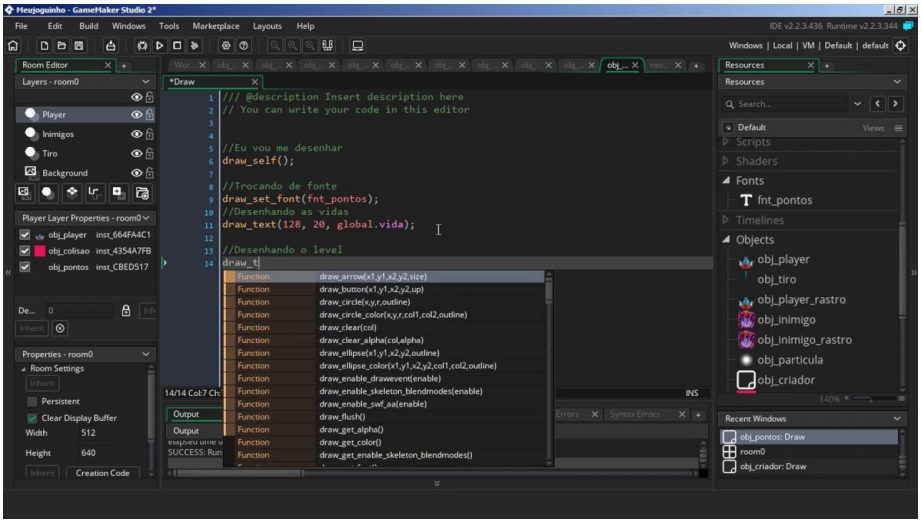

Рисунок 1.9 - Інтерфейс ігрового рушія GameMaker: Studio

Серед успішних ігрових проектів розроблених на базі GameMaker:

– Studio є;
– Hyper Light Drifter;
– Katana Zero;
– Undertale;
– Hotline Miami.

Ігровий рушій Unreal Engine – один із найдавніших ігрових рушіїв, на якому

створювалися проєкти ще з 1998 року, починаючи з гри *Unreal*, на честь якої і було названо рушій. Відтоді це програмне забезпечення використовувалося в багатьох інших проєктах. Хоча Unreal Engine спочатку був розроблений для створення шутерів, розробники знайшли способи використовувати його для інших жанрів, таких як RPG, MMORPG, файтинги та інші.

Мовою програмування для Unreal Engine є C++.

Архітектура рушія побудована на об'єктно-орієнтованому підході, де всі елементи представлені у вигляді об'єктів з набором характеристик та класів, які визначають доступні функції.

Інтерфейс ігрового рушія Unreal Engine (рис. 1.10).

Рисунок 1.10 - Інтерфейс ігрового рушія Unreal Engine До основних класів відносяться:

Основні класи в Unreal Engine

– Actor – базовий клас, який включає всі об’єкти, що мають відношення до ігрового процесу або просторові координати.

– Pawn – може бути як моделлю гравця, так і об’єктом, що керується ігровим ШІ.

– World – об’єкт, який характеризує властивості рівня, фізичну взаємодію об’єктів, тертя, силу тяжіння та інші параметри.

Ігровий рушій Unity – ігровий рушій, розроблений компанією Unity

Technologies, призначений для багатоплатформеної розробки. Інтерфейс ігрового рушія Unity (рис. 1.11). Застосунки, створювані на базі ігрового рушія Unity, можуть бути виконані як у двовимірному (2D), так і в тривимірному (3D) просторі. Крім того, Unity підтримує роботу з пристроями віртуальної реальності (VR), що дозволяє створювати імерсивні досвіди.

Основною мовою програмування для Unity є C#. Раніше рушій підтримував також мови Boo та JavaScript, але розробники вирішили відмовитися від їх підтримки, щоб спростити екосистему і зосередити зусилля на вдосконаленні підтримки C#.

Інтерфейс рушія складається з кількох основних вікон:

Оглядач ресурсів (Asset Browser): Використовується для перегляду та організації всіх ресурсів проєкту, таких як моделі, текстури, скрипти та інші файли. Інспектор (Inspector): Дозволяє проводити маніпуляції з об'єктами, змінювати їхні властивості та додавати компоненти. Вікно сцени (Scene View): Основне вікно для роботи з об'єктами на сцені, що дозволяє перемикатися між різними режимами перегляду та редагування. Оглядач ієрархії проєкту (Hierarchy View): Відображає структуру об'єктів сцени у вигляді ієрархії, що дозволяє легко керувати об'єктами та їхніми підлеглими. Ці компоненти інтерфейсу забезпечують зручне середовище для розробки та дозволяють розробникам ефективно керувати всіма аспектами створення гри.

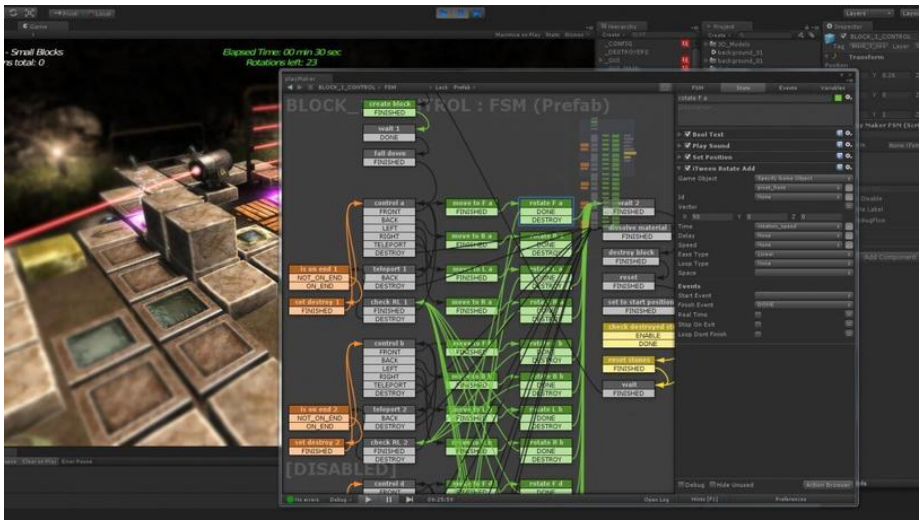

Рисунок 1.11 - Інтерфейс ігрового рушія Unity

Проект у Unity поділяється на сцени – окремі файли, що містять усю інформацію про певний рівень, який створюється вручну розробником. Файл сцени може містити об'єкти (3D моделі чи 2D спрайти). Об'єкти можуть бути пустими, але містити у своїх властивостях скрипти, виконуючи роль контролерів, тригерів, точок збереження чи переходів між сценами.

Кожен об'єкт можна переміщувати, змінювати його масштаб та обертати за допомогою відповідних інструментів або за допомогою спеціальних компонентів у вікні інспектора об'єкта. Усі об'єкти на сцені можна групувати, присвоюючи їм спеціальні теги або додаючи до спеціальних шарів, які створюються користувачем. Об'єкти підтримують систему успадкування, завдяки якій дочірні об'єкти повністю повторюють рух батьківського.

Фізика і рендеринг

Рушій підтримує фізику твердих тіл та тканин. В налаштуваннях проекту можна змінювати сценарії взаємодії тіл або силу гравітації. Рендеринг об'єктів здійснюється за допомогою віртуальної камери. На сцені можуть бути присутніми декілька камер, рух яких задається користувачем. Камера може працювати в перспективному або ортографічному режимі, що дозволяє створювати як тривимірні, так і двовимірні сцени, зберігаючи можливість роботи з глибиною.

Графічний рушій і шейдери

Графічний рушій Unity підтримує різні API для рендерингу:

- DirectX – для Windows платформ;
- OpenGL – для Windows, Mac, Linux;
- OpenGL ES – для мобільних платформ.

Для роботи із шейдерами використовується ShaderLab, який підтримує шейдерні програми, написані на GLSL та Cg.

Скрипти та інструменти

Рушій підтримує створення користувацьких скриптів, які можна додавати до об'єктів у вигляді спеціальних компонентів. Однією з основних переваг Unity є наявність візуального середовища розробки та мультиплатформеної підтримки. Це

не тільки дозволяє запускати проекти на різних платформах, але й забезпечує необхідні інструменти для розробки під них.

Переваги та недоліки

До переваг рушія можна віднести:

– Наявність візуального середовища розробки;
– Мультиплатформену підтримку;
– Велика кількість навчальних відео та гарно зібрана документація.

До недоліків можна віднести:

– Ускладнення роботи в складних сценах через обмеження візуального редактора;
– Деякі проблеми при роботі із зовнішніми бібліотеками.

Unity є досить популярним серед інді-розробників та розробників мобільних ігрових додатків.

Серед найвідоміших проектів, створених за допомогою ігрового рушія Unity, можна виділити такі:

– Genshin Impact;
– Subnautica;
– Inside;
– Cuphead;
– Superhot;
– Enter the Dungeon;
– Valheim;
– Rust;
– Серія ігор Ori.

Висновки до першого розділу

В першому розділі було проведено аналіз жанру action-adventure, який характеризується динамічним ігровим процесом, що поєднує в собі елементи бою,

дослідження та розгадування головоломок. До ключових особливостей жанру належать:

Динамічний геймплей: Action-adventure ігри пропонують швидкий та захоплюючий ігровий процес, який часто включає в себе бойові зіткнення, стрілянину, погоні та інші динамічні дії.

Дослідження: Гравці мають можливість досліджувати відкритий світ або різноманітні локації, розкриваючи таємниці, шукаючи скарби та виконуючи завдання.

Розгадування головоломок: Для просування по сюжету та досягнення успіху гравцям часто потрібно розгадувати головоломки, які можуть бути логічними, фізичними або комбінованими.

Сюжет: Action-adventure ігри, як правило, мають захоплюючий сюжет, який мотивує гравців досліджувати світ та йти далі.

Жанр action-adventure мав значний вплив на індустрію відеоігор. Він надихнув безліч розробників та породив одні з найпопулярніших та найвідоміших відеоігор в історії. Action-adventure ігри продовжують захоплювати гравців всіх віків завдяки своїй динамічній дії, захоплюючим сюжетам та безмежним можливостям для дослідження.

2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ІГРОВОГО ДОДАТКУ

Перед початком роботи необхідно було провести аналіз створюваної системи та створити її приблизну модель. Для проектування потрібно було врахувати можливі стани гравця, систему класів та їхні зв’язки, анімації та способи їх інтеграції з технічною частиною програми, а також відповідні контролери.

Скінченний автомат – це абстрактна машина, яка в будь-який момент часу може перебувати тільки в одному стані. Зміна станів відбувається за виконання певних умов, наприклад, вводу користувача. Кожному зі станів притаманні відповідні специфічні набори функцій.

Для роботи скінченного автомату необхідно визначити початковий стан та відповідні умови переходу між станами.

У розробці ігрових додатків скінченний автомат може використовуватися для різних цілей. Наприклад, для роботи з анімацією або з різними станами, в яких може перебувати головний герой. У великих проєктах, де можливості головного персонажа досить великі, скінченний автомат спрощує управління цими можливостями та полегшує впровадження нових функцій за необхідністю.

Окрім анімації головного героя, скінченні автомати можуть бути корисними для реалізації штучного інтелекту (ШІ) неігрових персонажів (NPC): Скінченні автомати можуть використовуватися для створення поведінки NPC, що реагує на дії гравця та інші події в ігровому світі. Наприклад, NPC може мати різні стани, такі як "спокійний", "тривожний", "ворожий", і переходити між ними залежно від ситуації. Загалом, скінченні автомати є потужним інструментом, який може бути використаний для вирішення різноманітних завдань в розробці ігрових застосунків.

В Unity скінченні автомати застосовуються для управління анімаційними кліпами будь-якого об'єкта, для якого вони були створені. Ці маніпуляції зі станами здійснюються за допомогою вбудованого компонента Animator (рис. 2.1).

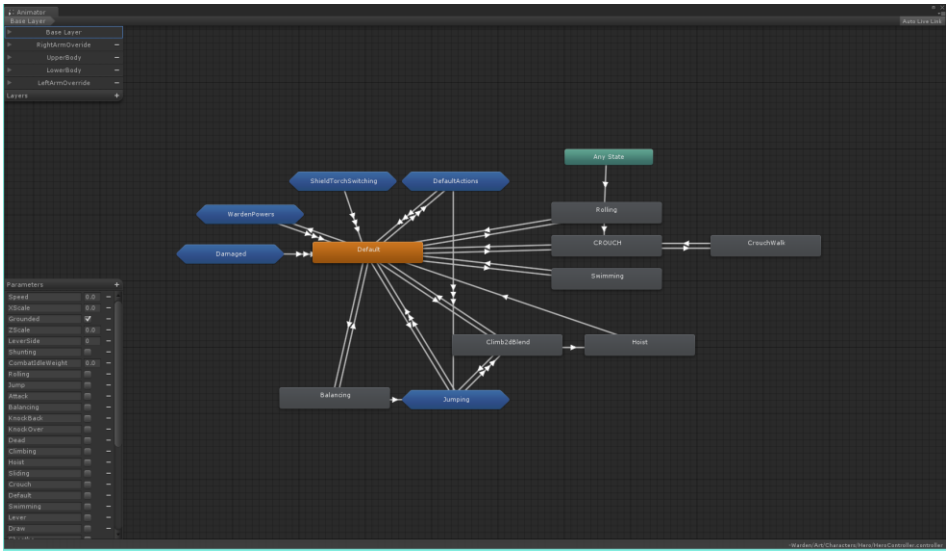

Рисунок 2.1 - Компонент Animator

У компоненті Animator можна задати початковий стан, визначити переходи між станами, створити відповідні змінні для цих переходів, а також відредагувати поведінку анімацій, включаючи швидкість переходу від однієї анімації до іншої, плавність та інші параметри. Крім того, можна створювати групи станів для полегшення роботи з анімаціями.

2.1 Системи гравця

До систем, що належать об'єкту гравця, відносяться стани та класи, притаманні лише цьому об'єкту (рис. 2.2).

Рисунок 2.2 - Схема зв'язків між класами

До систем, що належать об'єкту гравця, відносяться стани та класи, притаманні лише цьому об'єкту (рис. 2.2). Діаграма станів гравця (рис. 2.3) демонструє можливості та обмеження, які були запроваджені. Вона також допоможе створити дерево станів для анімацій в ігровому рушії.

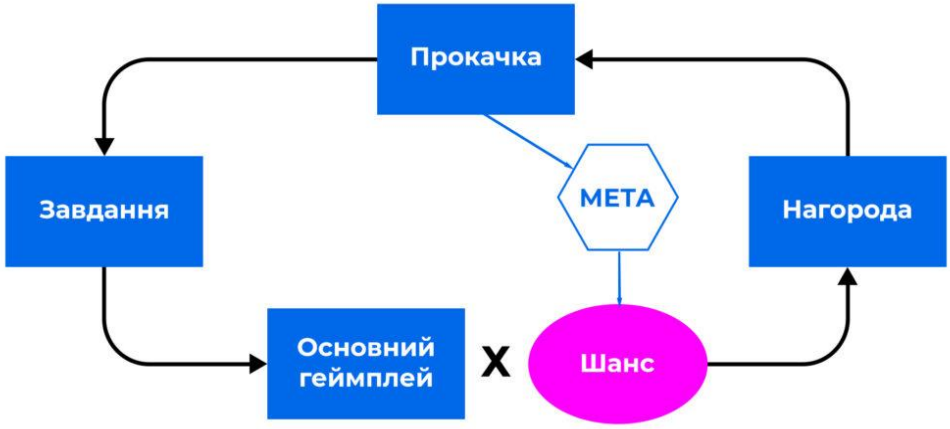

Рисунок 2.3 -Діаграма станів гравця

До систем, що належать об'єкту гравця, відносяться стани та класи, притаманні лише цьому об'єкту (рис. 2.2). Діаграма станів гравця (рис. 2.3) демонструє можливості та обмеження, які були запроваджені. Вона також допоможе створити дерево станів для анімацій в ігровому рушії.

У стані сну NPC не буде виконувати жодних дій, окрім реагування на потрапляння гравця в його зону активності. Після цього NPC перейде в активний стан і почне атакувати гравця.

Цикл ігрового процесу – важлива частина геймдизайну, за допомогою якої можна повністю заволодіти увагою гравця та зосередити його на ігровому процесі. Це досягається за рахунок правильно розставлених активностей у грі, що змінюють геймплей, щоб користувач не займався однією справою протягом всієї гри. Наприклад, якщо гра включає тільки біг та стрілянину і має великий обсяг, гравцеві швидко набридне виконувати однотипні дії протягом тривалого часу.

Ігровий дизайн – одна з основних частин розробки відеоігор, яка включає проектування, створення та контроль за ігровим процесом. На початку розвитку ігрової індустрії цими питаннями займалися самі програмісти, тому дизайнерів ігрового процесу як окремої професії тоді не існувало. З часом, із розвитком технологій, збільшенням масштабу ігор та часу, необхідного на їх створення, питання правильного ігрового дизайну стало більш актуальним.

Кількість ігрових дизайнерів у компанії може варіюватися: їх може бути як десяток, так і один, залежно від розміру компанії.

До обов'язків ігрового дизайнеру можна віднести:

- дизайн світу;
- дизайн ігрових механік;
- дизайн завдань;
- дизайн локації.

Дизайн світу – створення загальної концепції та деталей ігрового світу, включаючи його структуру, стиль та атмосферу.

Дизайн ігрових механік – Розробка та впровадження систем, які мають прямий вплив на ігровий процес. Це можуть бути окремі здібності головного героя, на яких побудована бойова система та спосіб переміщення по світу, а також системи очок, торгівлі та спілкування з неігровими персонажами.

Дизайн завдань – Створення квестів, місій та інших завдань, які гравці повинні виконувати впродовж гри.

Дизайн локації – планування локацій, на яких відбувається ігровий процес, розташування NPC, схованих точок інтересів та пазлів.

Дані обов'язки можуть бути положені на одного ігрового дизайнера або розділені між декількома, які будуть направлятись головним дизайнером.

Пейсинг – це важлива складова будь-якої гри та потужний інструмент у руках досвідченого ігрового дизайнера. Пейсинг – це набір методик, спрямованих на контроль уваги користувача шляхом підтримки відповідного темпу оповідання.

В іграх часто чергуються різні активності, якими займається гравець протягом гри. Наприклад, після активної фази, такої як сутичка з ворогами або втеча, майже завжди слідує фаза спокою, і навпаки. Оскільки ігри є інтерактивним жанром, на відміну від кіно, ігрові дизайнери не можуть тримати гравця в постійному напруженні, як це роблять у деяких фільмах. У кіно глядач лише спостерігає за головними героями, а в іграх гравець змушений не тільки спостерігати та співчувати, а й керувати персонажем, реагуючи на різні стимули, що у великій кількості може просто втомити користувача.

Стимуляція гравця поділяється на високорівневу та низькорівневу.

Високорівнева стимуляція – це спроби вплинути на емоційну складову користувача, викликаючи у нього напруження, страх, сміх та інші емоції.

Низькорівнева стимуляція – стосується механік гри, дизайну локацій, перешкод на них, розташування ворожих NPC тощо. Якщо правильно використовувати низькорівневу стимуляцію, можна зекономити час на використанні високорівневої.

В іграх від компанії FromSoftware розробники майже повністю зосередилися на низькорівневій стимуляції гравця. Тут немає чіткої історії, користувач не знає, хто він і до чого приведуть його дії. Завдяки правильному дизайну рівнів гравцеві не потрібно знати сюжет для проходження гри, він зрозуміє, куди треба йти. Проте, незважаючи на сильне зосередження на низькорівневій стимуляції, розробники не забули про високорівневу. Вони змушують гравця відчути страх перед новим босом через його дизайн та дизайн місця, в якому він знаходиться. Для допитливих гравців розробники створили історію для кожного предмета в грі. Щоб повністю зрозуміти історію, користувачеві потрібно читати описи предметів і складати історію ігрового світу наче пазл.

2.2 Планування першого рівню

Працюючи із жанром action adventure є декілька способів зберегти темп гри.

Другим – фаза спокою із простим adventure жанром.

Третім – локації, які будуть кидати виклик гравцеві.

Оскільки це перший рівень і перше знайомство гравця з грою, він є найважливішою частиною проєкту, оскільки має зацікавити користувача. Тому було вирішено підтримувати більш-менш активний темп за допомогою подій із простими боями. Повноцінні виклики поки що не кидаються на гравця, адже він лише навчається управлінню та основним механікам гри.

Бої: На рівні мають бути присутні прості бої, щоб дати гравцеві можливість випробувати бойову систему гри. Бої не повинні бути занадто складними, щоб не розчаровувати гравця, але й не занадто простими, щоб не здатися нудними.

Рівень має пропонувати гравцеві можливість досліджувати нові місця та знаходити цікаві предмети. Це може бути що завгодно, від секретних кімнат до скарбів, які можна знайти.

Фінал рівня має бути захоплюючим та залишати гравцеві бажання продовжити гру. Це може бути зустріч з босом, розкриття таємниці або ж просто динамічний епізод, який веде до наступного рівня.

Висновки до другого розділу

У другому розділі було детально описано три основні типи систем, що використовуються в грі:

Системи, притаманні гравцеві – системи визначають можливості та поведінку гравця протягом гри. До них належать системи керування персонажем, інвентарю, бою та інші.

Системи NPC – системи керують поведінкою неігрових персонажів (NPC), які населяють світ гри. Вони визначають, як NPC взаємодіють з гравцем та один з одним, а також їхні цілі.

Системи ворожих NPC: Ці системи є підмножиною систем NPC і спеціально розроблені для керування поведінкою ворогів, з якими гравець стикається протягом гри. Вони визначають бойові здібності, тактику та штучний інтелект ворогів.

У другому розділі було описано основний цикл ігрового процесу, який визначає хід гри. Цей цикл можна розбити на такі етапи:

Введення: Гравець отримує завдання або інструкції щодо того, що робити далі.

Дія: Гравець використовує системи, притаманні гравцеві, для взаємодії з ігровим світом та виконання завдань.

Результат: В залежності від дій гравця, він отримує зворотний зв'язок, який може включати в себе зміни в ігровому світі, оновлення статусу або інші винагороди/покарання.

Повторення: Цикл повторюється, поки гравець не виконає всі поставлені перед ним завдання.

У другому розділі було коротко висвітлено важливість ігрового дизайну при створенні ігрових застосунків.

Другий розділ заклав фундамент для розуміння ключових систем та ігрового процесу, що використовуються в грі. Розширений висновок доповнює цю інформацію, надаючи більш детальний опис цих аспектів, а також підкреслюючи важливість ігрового дизайну.

3 ІГРОВИЙ РУШІЙ У ДІЇ: ВІЗУАЛЬНА РЕАЛІЗАЦІЯ 2D ЗАСТОСУНКУ

3.1 Втілення задуму в життя завдяки рушію Unity

У Unity запуск програми починається зі сцени. Сцена являє собою локацію, що візуально відображається, на якій розгортається дії програми і містить об'єкти в локації та їх ієрархію. Такі об'єкти називаються GameObject, тобто ігровий об'єкт. Такі об'єкти містять інформацію про своє глобальне і локальне місцезнаходження, розмір і обертання, і, найголовніше, до них можуть кріпитися компоненти, наприклад компонент відтворення 3d моделі, спрайту, компонент колайдера, освітлення і скриптові компоненти, успадковані від класу MonoBehavior , Для якого прописаний свій інтерфейс взаємодії із середовищем та GameObject носієм в першу чергу.

Отже, всі скрипти, що відповідають за певні властивості об'єктів GameObject, будуть успадковуватися від MonoBehavior.

Скрипти, що відповідають за зберігання інформації, будуть успадковуватися від компонента ScriptableObject, який дозволяє використовувати [CreateAssetMenu()] для створення файлу з форматом .asset у папці ресурсів, - "Resources/".

Для спрощення розробки та швидших тестів працездатності використовуватимемо похідні від класу Editor.

Загалом ігрову логіку можна розбити на такі найбільш відокремлені частини як:

- бойова система;
- візуальна складова;
- звукова складова.

Основним компонентом, що бере участь у бою, є компонент Body. Він містить пасивну частину логіки кожному за конкретного об'єкта, тобто отримання втрат, кількість здоров'я, дії при “смерті”.

Body містить поля:

1) Поточне здоров'я, являє собою кількість здоров'я об'єкта;

2) public float currentHealth;

3) isAlive, що показує, чи живий об'єкт;

4) animator, Animator компонент, прив'язаний до даного GameObject;

5) defaultPushTimer, таймер тривалості поштовху за умовчанням, необхідний коректної роботи відштовхування персонажа;

6) pushTimer, таймер поштовху, необхідний для відрахування часу, коли не працює аніматор, дозволяючи штовхати персонажа по горизонталі;

7) isPushing, якщо true, то в даний момент об'єкт штовхається;

8) capsuleCollider, CapsuleCollider компонент даного GameObject-а;

9) rigidbody, Rigidbody компонент даного GameObject-а;

10) ragdollColliders, список компонентів Collider даного GameObject-а, що використовуються для реалістичної фізики ляльки;

11) ragdollRigidbodies, список компонентів Rigidbody даного GameObject-а використовуються для реалістичної фізики ляльки.

Start, успадкований метод від MonoBehavior, викликається на момент створення екземпляра класу, відразу після ініціалізації GameObject-а.

В даному випадку відбувається ланцюжок подій: у GameDataOptimizer, створений для оптимізації пошуку об'єктів, вноситься даний компонент Body; полю animator надається посилання на Animator компонент даного GameObject-а; полю rigidbody надається посилання на Rigidbody компонент даного GameObject-а; полю capsuleCollider надається посилання на CapsuleCollider компонент даного GameObject-а; поточне здоров'я стає рівним 100; полю ragdollColliders надається список Collider компонентів даного GameObject-а; полю ragdollRigidbodies надається список Rigidy компонентів даного GameObject-а; викликається метод EnableRagdoll, з параметром false, виключаючи фізику.

3.2 Створення ігрових механік для гравця

Процес створення ігрового застосунку розпочався з розробки механік гравця.

Потрібно було забезпечити наступну функціональність:

– звичайне переміщення(біг);

– атака;

– кастування;

Для більш зручної роботи було вирішено розбити функціонал гравця, процес створення ігрового застосунку розпочався з розробки механік гравця. Основні компоненти механік гравця включають:

– player – містить опис всіх методів, необхідних для переміщення гравця та його анімації;

– inputhandler – зчитує ввід від користувача та викликає відповідний метод із скрипта Player;

– animationcontroller – зчитує інформацію з InputHandler та відтворює відповідну анімацію.

Впровадження переміщення

Було прийнято рішення розробити фізично достовірну систему переміщення гравця. Ігровий рушій Unity дозволяє виконати цю задачу кількома способами: за допомогою компонента Transform або RigidBody. Перший забезпечує переміщення без врахування законів фізики, тоді як другий – з врахуванням.

Для роботи з фізичними функціями, які вже вбудовані в Unity, було додано компонент Rigidbody 2D до створеного об’єкта Player. Також додано компонент Capsule Collider 2D, щоб надати об’єкту фізичну форму певного геометричного тіла, у нашому випадку – капсули. Маючи ці два компоненти, можна почати створювати скрипти, які відповідатимуть за функціонал гравця. У середовищі Unity вони також представлені у вигляді компонентів, але створюються вони користувачем.

Впровадження переміщення

Було прийнято рішення розробити фізично достовірну систему переміщення гравця. Ігровий рушій Unity дозволяє виконати цю задачу кількома способами: за допомогою компонента Transform або RigidBody. Перший забезпечує переміщення

без врахування законів фізики, тоді як другий – з врахуванням.

Для роботи з фізичними функціями, які вже вбудовані в Unity, було додано компонент Rigidbody 2D до створеного об’єкта Player. Також додано компонент Capsule Collider 2D, щоб надати об’єкту фізичну форму певного геометричного тіла, у нашому випадку – капсули. Маючи ці два компоненти, можна почати створювати скрипти, які відповідатимуть за функціонал гравця. У середовищі Unity вони також представлені у вигляді компонентів, але створюються вони користувачем.

За основні фізичні властивості бігу відповідають 4 змінні – Run Speed, Acceleration, Decceleration, Friction Force.

Run Speed – відповідає за швидкість руху.

Acceleration та *Decceleration* – відповідають за швидкість набору максимальної швидкості та швидкість зупинки.

Friction Force – імітує силу тертя.

Завдяки цьому було створено правдиву систему переміщення гравця.

3.3 Розробка графічних елементів для гри

Графіка розроблялася паралельно з розробкою механік гри. Для цього проєкту було створено:

- дизайн головного героя;
- дизайн NPC;
- анімації головного героя;
- анімації NPC;
- задній фон гри;
- атака.

Розробка графіки персонажів

Створення візуальних референсів: Використання ескізів, мудбордів, 3D-моделей та інших візуальних матеріалів для дослідження різних ідей та створення

чіткого образу персонажа. Визначення особистості, мотивації, історії,

походження та інших важливих аспектів персонажа (рис. 3.1), які допоможуть сформувати його візуальний стиль.

Рисунок 3.1 - Розроблений дизайн головного персонажу гри

Макет головного гравця Вогник був розроблений з логотипу Chupuharik (рис. 3.2).

Рисунок 3.2 – Приклад логотипу Chupuharik з якого робився головний герой гри.

Макет дизайну розробки головного персонажу було створено у програмному забезпеченні PhotoShop.

Enemy NPC (рис 3.3).

Рисунок 3.3 – Дизайн ворожих мобів NPC

Frendly assignement NPC (рис. 3.4).

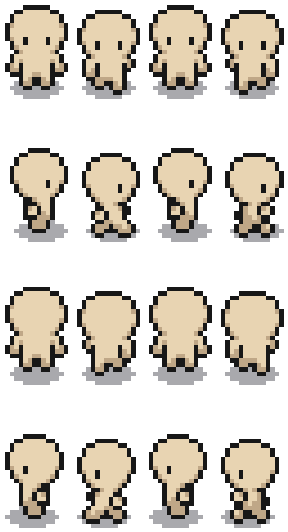

Рисунок 3.4 – Дизайн NPC у яких можливо брати квести

Було створено анімації:

– простою (idle);
– бігу (run);
– ривку (sprint);

– атаки (atack).

Після створення спрайт-листів із готовими анімаціями, вони були імпортовані до проєкту з відповідним налаштуванням. Оскільки спрайт-лист є суцільним зображенням, його необхідно було нарізати на окремі кадри, щоб анімація працювала правильно. Для цього потрібно було вказати, що спрайт-лист містить кілька зображень, задати відповідну фільтрацію та максимальний розмір файлу, щоб уникнути появи неочікуваних артефактів під час використання.

Інші персонажі були створені та анімовані за аналогічною схемою.

Розробка графіки фону

Для розробки графіки фону використовувалось програмне забезпечення Adobe Photoshop. Цей потужний графічний редактор надає широкий спектр інструментів та функцій, які ідеально підходять для створення високоякісних 2D-зображень.

Спочатку був створений базовий рівень деталізації, після чого додавалися додаткові деталі та кольори.

Розробка композиції фону, враховуючи розміщення елементів, баланс та візуальну ієрархію.

Вибір кольорової гами, яка відповідає загальному стилю гри та створює бажану атмосферу.

Додавання дрібних деталей, таких як текстури, візерунки та тіні, щоб зробити зображення більш реалістичним та цікавим. Використання освітлення та затемнення для підкреслення певних елементів та створення глибини зображення.

Застосування різних художніх прийомів, таких як фокусування, перспектива та атмосферна перспектива, для спрямування погляду глядача та створення відчуття простору.

Фінальне доопрацювання кольорів для забезпечення гармонії та цілісності зображення.

Коли фон був завершений, його також імпортували до проєкту (рис. 3.5).

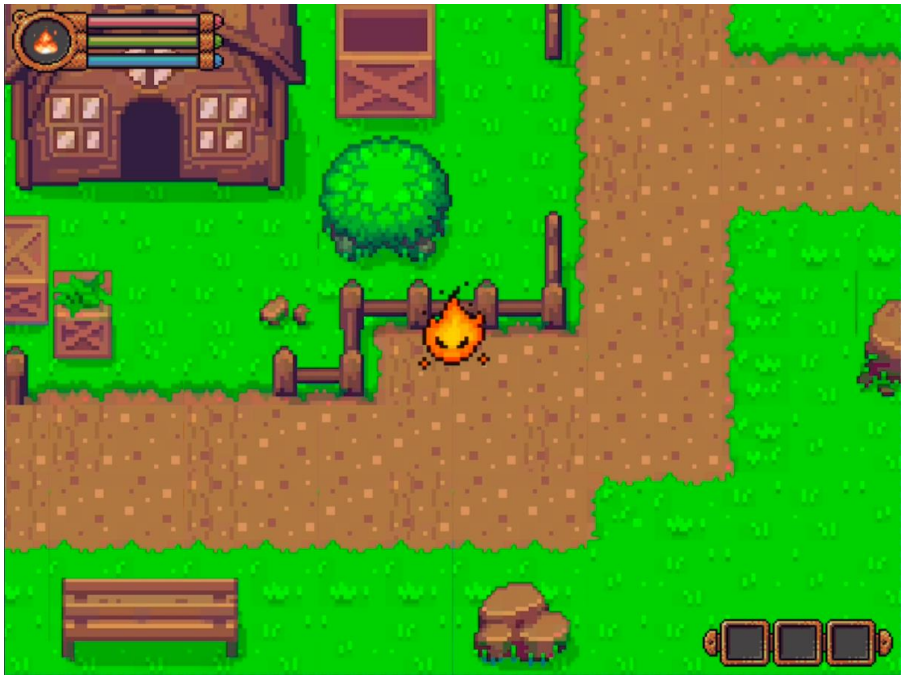

Рисунок 3.5 – Перша локація у грі. Кінцева форма фону вже з фізичним текстурами, та працюючими скриптами.

Після розробки фону для першого рівня проводилась робота над графікою інвентаря.

Для розробки були застосовані різні художні прийоми, зокрема перспектива, контрастність форм і кольорів, а також знання будови об’єктів.

3.4 Робота з постпроцесингом

Додавання пакету URP (Universal Render Pipeline) до проєкту надало можливість працювати з постпроцесингом для 2D проєктів. Постпроцесинг включає сукупність усіх додаткових ефектів, які покращують уже створене зображення. Це можуть бути додаткові тіні, джерела світла та різноманітні ефекти, пов’язані зі світлом.

Щоб об’єкти відкидали тінь при спрямуванні світла на них, до цих об’єктів потрібно додати компонент Shadow Caster 2D. Цей компонент забезпечує створення тіні відповідної форми. Наприклад, до об’єкта Player був доданий компонент Shadow Caster 2D та створена відповідна форма тіні.

Міні локація на початку гри spawn home (рис 3.6) яка дозволяє поставити чек-

поінт у пічці гравцем.

Рисунок 3.6 – Змога зайти до будинку на початку гри, та поставити точку респавну у пічці.

Пічь відіграє роль ліжка, як у багатьох іграх. Якщо гравець втратить життя, то він відразу з’явиться на точки респувна біля пічки.

Пічька була вибрана точкой респувну – бо гравень грає за маленького вогника.

Висновки до третього розділу

У третьому розділі було представлено опис основних етапів розробки ігрового застосунку:

Система переміщення: Опис реалізації фізично достовірної системи переміщення гравця з використанням компонентів Rigidbody 2D, Capsule Collider 2D та налаштуванням змінних для швидкості, прискорення та гальмування.

Система управління: Пояснення логіки роботи системи управління, яка зчитує ввід від користувача та викликає відповідні методи зі скрипта Player.

Опис основного компоненту бойової системи - Body, який містить пасивну частину логіки кожного об'єкта, включаючи отримання шкоди, здоров'я, дії при "смерті" та інші параметри.

Детальний опис процесу розробки дизайну та анімацій головного героя, NPC, а також пояснення створення спрайт листів та імпорту в проект.

Фон: Опис процесу створення фону в Photoshop з додаванням деталей, кольору та імпорту в проект.

Інвентар: Пояснення принципів розробки графіки інвентаря з використанням прийомів малювання, таких як перспектива, контрастність форм та кольору.

Були детально описані механіки для гравця, включаючи системи переміщення, управління та бою, а також процес розробки графіки персонажів, фону, інвентарю та використання постпроцесингу для покращення візуальної складової гри.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра були досягнуті наступні цілі:

– проведений аналіз жанру action adventure та доступних аналогів;

– досліджено різні методи та інструменти для створення графічного контенту та анімацій у відеоіграх;

– проведено аналіз доступних ігрових рушіїв, визначено їхні переваги та недоліки;

– розроблено механіки та графіку гри імплементовано алгоритми штучного інтелекту.

При аналізі жанру action-adventure було визначено особливості та унікальні механіки, притаманні цьому жанру, зокрема, вид збоку або top-down.

Було проаналізовано способи та інструменти створення графічного контенту для відеоігор. Досліджено наступне програмне забезпечення:

1) Photoshop – потужний графічний редактор, який дозволяє працювати з різними типами зображень, включаючи піксель-арт. Проте його інтерфейс та інструменти, орієнтовані на високоякісну графіку та фотографії, можуть бути менш зручними для роботи з піксельною графікою. Незважаючи на це, його багатофункціональність і можливості для редагування роблять його незамінним інструментом для багатьох художників і дизайнерів.

2) Tiled2Unity – спеціалізований графічний редактор, створений для роботи з піксельною графікою. Цей інструмент не тільки спрощує створення піксель-арт зображень, але й значно полегшує процес макетування локацій у Unity. Завдяки інтеграції з Unity, розробники можуть швидко та ефективно створювати та впроваджувати складні ігрові середовища, забезпечуючи високу продуктивність і гнучкість у процесі розробки.

Для вибору ігрового рушія було проведено ретельний аналіз існуючих варіантів, їхніх переваг та недоліків. Серед розглянутих рушіїв були Unreal Engine від Epic Games, Unity та GameMaker: Studio. Обраним рушієм став Unity завдяки

його доступності, гнучкості та великій кількості навчального матеріалу. Unity надає широкий спектр можливостей для розробників, дозволяючи створювати ігри різної складності з використанням потужних інструментів та бібліотек.

Для написання коду ігрового застосунку використовувалася мова програмування C#. Ця мова добре інтегрується з Unity, забезпечуючи високий рівень продуктивності та зручність написання коду. Крім того, C# є популярною мовою серед розробників, що забезпечує доступ до численних ресурсів та спільнот для обміну досвідом та вирішення проблем.

Для розробки схем та діаграм використовувався сайт app.diagrams.net. За допомогою цього інструменту було створено схеми зв’язку між класами об'єкта гравця, що дозволило візуалізувати структуру коду та взаємодію між різними компонентами. Також були розроблені діаграми станів, які відображають можливі стани, в яких можуть перебувати гравець та NPC. Це допомогло краще зрозуміти логіку роботи ігрових механік та полегшило процес їх реалізації.

Використання цих інструментів та методів дозволило забезпечити високу якість розробки ігрового застосунку, ефективно організувати робочий процес та досягти поставлених цілей у створенні гри.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Performance of neural networks in materials science / H. K. D. H. Bhadeshia et al. *Materials Science and Technology*. 2009. Vol. 25, no. 4. P. 504–510. URL: https://doi.org/10.1179/174328408x311053 (date of access: 16.06.2024).

2. Forever W. C# 4.0. Полное руководство. Герберт Шилдт | PDF. *Scribd*. URL: https://ru.scribd.com/document/421884328/C-4-0-Полное-руководство-Герберт-Шилдт (дата звернення: 16.06.2024).

3. C# Game Programming Cookbook for Unity 3D | Jeff W. Murray | Taylor &. Taylor & Francis. URL: https://doi.org/10.1201/b17100 (date of access: 16.06.2024).

4. Richter J. CLR via C#. 4th ed. Redmond : Microsoft Press, 2012. 863 p.

5. GitHub - head-first-csharp/fourth-edition: Code and graphics for the projects in the 4th edition of Head First C#. GitHub. URL: https://github.com/head-first-csharp/fourth-edition (date of access: 16.06.2024).

6. Object-oriented Game Development. *Google Books*. URL: https://books.google.com.ua/books?id=WRwFBG93tJMC&pg=PA2&hl=ru&source=gbs_toc_r&cad=1#v=onepage&q&f=false (date of access: 16.06.2024).

7. Forever W. C# 4.0. Полное руководство. Герберт Шилдт | PDF. *Scribd*. URL: https://ru.scribd.com/document/421884328/C-4-0-Полное-руководство-Герберт-Шилдт (дата звернення: 16.06.2024).

8. Solutions manual to accompany pattern classification [2nd. ed.] - DOKUMEN.PUB. *dokumen.pub*. URL: https://dokumen.pub/solutions-manual-to-accompany-pattern-classification-2ndnbsped.html (date of access: 16.06.2024).

9. Artificial intelligence: A Modern Approach. Prentice Hall, 2010.

10. Троелсен Э. Язык программирования C# 5.0 и платформа .NET 4.5 (2013) :: Книга :: Sergey Drozdov. *Sergey Drozdov*. URL: https://sd.blackball.lv/books/7278 (дата звернення: 16.06.2024).

11. Як був розроблений Skyrim URL: https://blackforest.su/articles/skyrim-creation

12. Час UNIX: A History and a Memoir. Санкт-Петербург, Росія : ООО Видавництво «Пітер», 2021. 224 с.

13. Unity Development Cookbook: Real-Time Solutions from Game Development to AI 2nd Edition - 263206 - Tim Nugent, Jon Manning, Paris Buttfield-Addison - Balka-URL: https://balka-book.com/ua/razrabotka_programnogo_obespecheniya-366/unity-development-cookbook-realtime-solutions-from-game-development-to-ai-2nd-edition-263206 (дата звернення: 16.06.2024).

14. GitHub - PacktPublishing/Unity-2023-Cookbook-Fifth-Edition: Unity 2023 Cookbook. *GitHub*. URL: https://github.com/PacktPublishing/Unity-2023-Cookbook-Fifth-Edition (date of access: 16.06.2024).

15. Packt+ | Advance your knowledge in tech. *Packt*. URL: https://www.packtpub.com/en-pl/product/libgdx-cross-platform-game-development-cookbook-10-9781783287291/chapter/rigid-body-physics-with-box2d/rigid-body-physics-with-box2d (date of access: 16.06.2024).

16. Pro Unity Game Development with C-sharp Book. *SlideShare*. URL: https://www.slideshare.net/slideshow/pro-unity-game-development-with-c-sharp-book/269447118 (date of access: 16.06.2024).

17. Макар О. Unity «Best» Practices. *Хабр*. URL: https://habr.com/ru/companies/mygames/articles/649687/ (date of access: 16.06.2024).

18. ST Segment Depression (4-Codes) and Negative T-Waves (5-Codes). *The Minnesota Code Manual of Electrocardiographic Findings*. London, 2010. P. 60–97. URL: https://doi.org/10.1007/978-1-84882-778-3_7 (date of access: 16.06.2024).

19. PatientZero. Поради та рекомендації у роботі з Unity3D. *Хабр*. URL: https://habr.com/ru/articles/309478/ (дата звернення: 16.06.2024).

20. Unity - *Unity - Manual: Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/ru/530/Manual/CreatingAndUsingScripts.html (date of access: 16.06.2024).

21. Unity полезные ссылки. *Gist*. URL: https://gist.github.com/Ciberusps/b509268867d097c6a47cd23a665c3bfb (дата звернення: 16.06.2024).

22. Просунення програмування та архітектура коду | Unity. *Unity*. URL: https://unity.com/ru/how-to/advanced-programming-and-code-architecture (дата звернення: 16.06.2024).

23. Починаємо роботу з віртуальной реальністью! Приклади та важливі статьї | Unity Blog. *Unity Blog*. URL: https://blog.unity.com/ru/games/get-started-with-vr-sample-pack-learning-articles (дата звернення: 16.06.2024).

24. Visual Studio не видит классы Unity. *Stack Overflow на русском*. URL: https://ru.stackoverflow.com/questions/846472/visual-studio-не-видит-классы-unity (дата звернення: 16.06.2024).

25. 7 найкращих ігор, написанних на Unity. *ITVDN*. URL: https://itvdn.com/ru/blog/article/7best-unity-games (дата звернення: 16.06.2024).

ДОДАТОК А

ЛІСТИНГ КОДУ ЗАСТОСУНКУ

Скрипт PlayerMovement

```
using System.Collections;
using System.Runtime.CompilerServices;
using UnityEngine;

public enum PlayerState {
    walk,
    attack,
    interact,
}

public class PlayerMovement : MonoBehaviour {

public PlayerState currentState;
public float speed;
private Rigidbody2D myRigidbody;
private Vector3 change;
private Animator animator;

    // Start is called before the first frame update
    void Start () {
        currentState = PlayerState.walk;
        animator = GetComponent<Animator>();
        myRigidbody = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    void Update () {
        change = Vector3.zero;
        change.x = Input.GetAxisRaw("Horizontal");
        change.y = Input.GetAxisRaw("Vertical");
        if (Input.GetButtonDown("attack") && currentState != PlayerState.attack)
        {
            StartCoroutine(AttackCo());
        }
        else if (currentState == PlayerState.walk)
        {
            UpdateAnimationAndMove();
        }
    }

    private IEnumerator AttackCo()
    {
        animator.SetBool("attacking", true);
        currentState = PlayerState.attack;
        yield return null;
        animator.SetBool("attacking", false);
        yield return new WaitForSeconds(.33f);
        currentState = PlayerState.walk;
    }
    void UpdateAnimationAndMove()
    {
        if (change != Vector3.zero)
        {
```

```
            MoveCharacter();
            animator.SetFloat("moveX", change.x);
            animator.SetFloat("moveY", change.y);
            animator.SetBool("moving", true);
        }
        else
        {
            animator.SetBool("moving", false);
        }
    }

    void MoveCharacter()
    {
        myRigidbody.MovePosition(
            transform.position + change * speed * Time.deltaTime
            );
    }
}
```

Скрипт Pot

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

public class pot : MonoBehaviour
{
    private Animator anim;
    void Start() {
        anim = GetComponent<Animator>();
    }
    void Update ()
    {

    }
    public void Smash()
    {
        anim.SetBool("smash", true);
    }
}
```

Скрипт PlayerHit

```
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using UnityEngine;

public class PlayerHit : MonoBehaviour
{

    void Start()
    {

    }
    void Update() {
    }
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("breakable"))
        {
            other.GetComponent<pot>().Smash();
```

```
        }
    }
}
```

Скрипт Permutatio

```
using System.Collections;
using UnityEngine;

public class Permutatio : MonoBehaviour
{
    public float sizeChangeSpeed = 1f;
    public float rotationChangeSpeed = 15f;

    private void Start()
    {
        StartCoroutine(AnimateTransform());
    }

    private IEnumerator AnimateTransform()
    {
        while (true)
        {
            // Сейв теперешнього масштабу та кута обертання
            Vector3 currentScale = transform.localScale;
            Quaternion currentRotation = transform.rotation;

            // Зміна розміру з часом
            float newSize = Mathf.Sin(Time.time * sizeChangeSpeed) + 1f;
            Vector3 newScale = new Vector3(newSize, newSize, newSize);

            // Зміна кута нахилу з часом
            float newRotation = Mathf.Sin(Time.time * rotationChangeSpeed) * 45f;
            Quaternion newRotationQuaternion = Quaternion.Euler(0f, 0f, newRotation);

            // Застосування змін
            transform.localScale = newScale;
            transform.rotation = newRotationQuaternion;

            yield return null; // Зачекати один кадр перед наступною ітерацією

            // Повернення до попередніх значень
            transform.localScale = currentScale;
            transform.rotation = currentRotation;
        }
    }
}
```

Скрипт Sign

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

public class Sign : MonoBehaviour {

    public GameObject dialogBox;
    public Text dialogText;
    public string dialog;
    public bool playerInRange;
```

```
    // Start is called before the first frame update
    void Start() {

    }

    // Update is called once per frame
    void Update() {
        if (Input.GetKeyDown(KeyCode.Space) && playerInRange)
        {
            if (dialogBox.activeInHierarchy)
            {
                dialogBox.SetActive(false);
            }
            else
            {
                dialogBox.SetActive(true);
                dialogText.text = dialog;
            }
        }
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            playerInRange = true;
        }
    }
    private void OnTriggerExit2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            playerInRange = false;
            dialogBox.SetActive(false);
        }
    }
}
```

Скрипт RoomMove

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

public class RoomMove : MonoBehaviour {

    public Vector2 cameraChange;
    public Vector3 playerChange;
    private CameraMovement cam;
    public bool needText;
    public string placeName;
    public GameObject text;
    public Text placeText;

    // Start is called before the first frame update
    void Start() {
        cam = Camera.main.GetComponent<CameraMovement>();
    }
```

```
    // Update is called once per frame
    void Update () {

    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            cam.minPosition += cameraChange;
            cam.maxPosition += cameraChange;
            other.transform.position += playerChange;
            if (needText)
            {
                StartCoroutine(placeNameCo());
            }

        }
    }

    private IEnumerator placeNameCo()
    {
        text.SetActive(true);
        placeText.text = placeName;
        yield return new WaitForSeconds(4f);
        text.SetActive(false);
    }
}
```

Скрипт CameraMovement

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

public class CameraMovement : MonoBehaviour {

    public Transform target;
    public float smoothing;
    public Vector2 maxPosition;
    public Vector2 minPosition;

    // Start is called before the first frame update
    void Start() {

    }

    // Update is called once per frame
    void LateUpdate () {
        if(transform.position != target.position)
        {
            Vector3 targetPosition = new Vector3(target.position.x, target.position.y, transform.position.z);
            targetPosition.x = Mathf.Clamp(targetPosition.x, minPosition.x, maxPosition.x);
            targetPosition.y = Mathf.Clamp(targetPosition.y, minPosition.y, maxPosition.y);

            transform.position = Vector3.Lerp(transform.position, targetPosition, smoothing);

        }
    }
}
```

Scrypt for heart system

```
using JetBrains.Annotations;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class HeartManager : MonoBehaviour
{
   public Image[] hearts;
   public Sprite fullHeart;
   public Sprite halfFullHeart;
   public Sprite emptyHeart;
   public FloatValue heartContainers;

   void Start()
   {
      InitHearts();
   }


   public void InitHerts()
   {
      for (int i = 0; i < heartContainers.initialValue; i++)
      {
         hearts[i].game0bject.SetActive(true);
         hearts[i].sprite = fullHeart;
      }
   }
}
```

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

 public class Body : MonoBehaviour,IHealth,IArmor,ITakeDamage
     {

             const string projectileFolder = "Prefabs/Projectiles/"; public static bool isAtacking;


             public GameObject hips;//root bone

             public Transform leftWeaponT; public Transform
             rightWeaponT;

             public Transform twoHandedWeaponT; public float currentHealth;

             public Armor Armor_ { get; } float defaultPushTimer =
             0.3f; float pushTimer;
             bool isPushing;
             CapsuleCollider capsuleCollider; Animator animator;
             bool weaponEnabled; Rigidbody
             rigidbody; Vector3 delayedForce;
             List<Collider> ragdollColliders = new List<Collider>(); List<Rigidbody> ragdollRigidbodyes = new
 List<Rigidbody>();
             int currentWeaponId; bool isArmed;
             public   bool   isDead; public event Action
```

```
onDie; void Awake()
{
        GameDataOptimizer.InsertValue(this); animator =
        GetComponent<Animator>();


        rigidbody = GetComponent<Rigidbody>(); capsuleCollider =
        GetComponent<CapsuleCollider>();


        Health_.baseValue = 100;


        /*
    var joints = GetComponentsInChildren<CharacterJoint>();
        foreach (var j in joints)
        {
                Destroy(j);
        }


        var rigB = GetComponentsInChildren<Rigidbody>(); foreach (Rigidbody r in rigB)
        {
                if (r != rigidbody)

                        Destroy(r);
        }
        */
    ragdollColliders = new List<Collider>(GetComponentsInChildren<Collider>());
    ragdollRigidbodyes = new List<Rigidbody>(GetComponentsInChildren<Rigidbody>());

        EnableRagdoll(false);
}
void A()
{

}
// Update is called once per frame

private void FixedUpdate()
{
if (delayedForce.magnitude > 0)
```

Листинг файлу Character.cs

```
using System.Collections.Generic;
using UnityEngine;
public class Character : MonoBehaviour
{
[SerializeField] float m_MovingTurnSpeed = 360;


    public float multiplier; float
    defaultMultuplier;
```

```
        [SerializeField]
        string defaultWeaponPath;

        Weapon defautWeapon;

        public Transform leftHand; public
        Transform rightHand;

Rigidbody m_Rigidbody;
```