Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ КОМП'ЮТЕРНОЇ ГРИ «THE CAVE OF EVIL»

Виконав: студент 4 курсу, групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Ярмощук Д.Р.
(прізвище та ініціали)

Керівник: д.т.н., проф. каф. КН
Іванчук Я. В.
(прізвище та ініціали)
«07» 06 2024 р.

Рецензент: д.т.н., проф., зав. каф. КСУ
Ковтун В. В.
(прізвище та ініціали)
«07» 06 2024 р.

Допущено до захисту
Зав. кафедри проф., д.т.н. Яровий А.А.
«07» 06 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А. А.
«07» 03 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Ярмощуку Дмитру Руслановичу

1. Тема роботи: Програмний модуль комп'ютерної гри " The Cave of Evil"
керівник роботи: Іванчук Я. В., проф
затверджені наказом вищого навчального закладу "11" 03 2024 року №80
2. Строк подання студентом роботи 27 05 2024 р.
3. Вихідні дані до роботи: кількість гравців - 1 чол.; роздільна здатність дисплею – 1920x1080 пікс.; частота кадрів – 60 кадр./с; розмір карти – 10000x10000 пікс.; кількість рівнів – 5; кількість видів ворогів – 3 од.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): аналіз сучасної ігрової індустрії та класифікація комп'ютерних відеоігор, аналіз проблеми графічного дизайну у комп'ютерних іграх у жанрі 2d
платформер, аналіз сучасних відеоігор у жанрі 2D платформер, розробка архітектури програмного модулю відеогри, розробка ігрової механіки, розробка алгоритму функціонування програмного модулю комп'ютерної гри обґрунтування вибору рушія та мови програмування для програмної реалізації, програмна реалізація модулю комп'ютерної гри, тестування програмного модулю комп'ютерної гри.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): діаграма основних компонентів архітектури програмного модулю відеогри, діаграма принципу взаємодії різних шарів архітектури, діаграма процедури генерації для створення ландшафту, діаграма модульного дизайну у розробці відеоігор, схема алгоритму функціонування програмного модулю комп'ютерної гри, схема алгоритму зіткнень об'єктів в комп'ютерні грі, загальний вигляд інтерфейсного вікна типу «головне меню» комп'ютерної гри, загальний вигляд інтерфейсного вікна типу «локація».

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завданн прийня
1-3	Іванчук Я. В., д.т.н., проф. каф. КН		

7. Дата видачі завдання 07.03.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Термін виконання		Прим
		початок	закінчення	
1	Аналіз сучасної ігрової індустрії та класифікація комп'ютерних відеоігор	06.05.24	07.05.24	
2	Аналіз проблеми графічного дизайну у комп'ютерних іграх у жанрі 2D платформер	08.05.24	10.05.24	
3	Розробка архітектури програмного модулю відеогри	11.05.24	16.05.24	
4	Розробка методів створення ігрової механіки	17.05.24	20.05.24	
5	Розробка алгоритму функціонування програмного модулю комп'ютерної гри	21.05.24	25.05.24	
6	Програмна реалізація модулю комп'ютерної гри	26.05.24	31.05.24	
7	Тестування програмного модулю комп'ютерної гри	01.06.24	02.06.24	
8	Розробка інструкції користувача.	03.06.24	04.06.24	
9	Оформлення матеріалів до захисту БКР	05.06.24	06.06.24	

Студент _____ **Ярмощук Д.Р.**
(підпис) *(прізвище та ініціали)*

Керівник роботи _____ **Іванчук Я. В.**
(підпис) *(прізвище та ініціали)*

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 81 сторінок формату А4, на яких є 27 рисунки, список використаних джерел містить 21 найменувань.

Дана бакалаврська робота присвячена розробці та впровадженню програмного модулю комп'ютерної гри «The Cave of Evil». Основною метою роботи є створення оптимізованої 2D графіки із зручним та інтуїтивно зрозумілим управлінням. Була розроблена структурна схема для опису функціонування модулю, схема алгоритму роботи програмного модулю, UML-діаграма алгоритму модулю комп'ютерної гри у жанрі 2D платформер. Unity забезпечує розробку основної частини гри (інтерфейс, анімації, дизайн тощо), а C# відповідає за створення скриптів для комп'ютерної гри. Архітектура модулю реалізується за підходом, що дозволяє забезпечити розділення логіки, представлення та обробки даних.

Результатом роботи є функціональний і надійний програмний модуль, який дозволяє гравцю керувати головним героєм на 2D локації, та взаємодіяти з навколишніми об'єктами.

Ключові слова: комп'ютерна гра, 2D платформер, комп'ютерна графіка.

ABSTRACT

Bachelor's thesis consists of 81 pages of A4 format, which contains 27 figures, the list of used sources contains 21 items. This bachelor's thesis is devoted to the development and implementation of the software module of the computer game "The Cave of Evil". The main goal of the work is to create optimized 2D graphics with convenient and intuitive controls. A structural diagram was developed to describe the functioning of the module, a diagram of the algorithm of the software module, a UML diagram of the algorithm of the computer game module in the genre of 2D platformer. Unity provides the development of the main part of the game (interface, animations, design, etc.), and C# is responsible for creating scripts for the computer game. The architecture of the module is implemented according to an approach that allows to ensure the separation of logic, presentation and data processing.

The result of the work is a functional and reliable software module that allows the player to control the main character in a 2D location, and interact with surrounding objects.

Keywords: computer game, 2D platformer, computer graphic

ЗМІСТ

ВСТУП

Актуальність досліджень. У сучасному світі комп'ютерних ігор, що стрімко розвивається, індустрія розваг постійно прагне нових та захоплюючих ігрових вражень. Особливо популярним жанром серед геймерів залишаються 2D-платформери [1], які поєднують у собі простоту геймплею з цікавими механіками, а також особливу комп’ютерну графіку.

Створення успішної 2D гри вимагає не лише креативного підходу до дизайну рівнів, персонажів та сюжету, але й глибокого розуміння технічних аспектів розробки. Одним з ключових елементів є програмний модуль, який забезпечує функціональність гри, взаємодію з користувачем та реалізацію ігрових механік. Він є своєрідним "серцем" гри, що відповідає за її логіку, фізику, штучний інтелект та інші важливі аспекти.

Актуальність даної роботи зумовлена постійним зростанням інтересу до 2D платформерів серед геймерів, а також необхідністю розробки якісних та конкурентоспроможних ігрових продуктів. Створення програмного модуля для гри "The Cave of Evil" є важливим кроком у цьому напрямку, оскільки він дозволить реалізувати унікальні ігрові механіки та візуальний стиль, що виділить гру серед інших представників жанру.

Проте, існують деякі основні проблеми, з якими зіштовхуються користувачі при грі в комп’ютерні ігри жанру 2D-платформер:

- низька продуктивність та оптимізація;
- неякісна графіка та анімація;
- недосконалий дизайн рівнів та ігровий процес.

Дослідження цих проблем та розробка програмного модуля допоможуть підвищити рівень оптимізації комп’ютерної графіки та зручності керування головним героєм для взаємодії з навколишніми об’єктами.

Оскільки, комп’ютерна індустрія в сучасному світі стає тільки

популярнішою серед людей, то розробка програмного модулю комп’ютерної гри «The Cave of Evil» є актуальною задачею.

Метою дослідження є реалізація дозвілля людини на основі розробленої комп’ютерної гри комп’ютерної гри «The Cave of Evil», яка дозволяє нормалізувати психоемоційний стан і підвищити інтелектуальний рівень людини.

Задачі дослідження:

1. Провести аналіз сучасної ігрової індустрії та класифікації комп’ютерних відеоігор
2. Провести аналіз проблеми графічного дизайну у комп’ютерних іграх у жанрі 2D платформер .
3. Розробити архітектуру програмного модуля комп’ютерної гри.
4. Розробити ігрову механіку комп’ютерної гри «The Cave of Evil».
5. Розробити алгоритм функціонування програмного модулю комп’ютерної гри.
6. Здійснити програмну реалізацію модулю комп’ютерної гри.
7. Протестувати програмний модуль комп’ютерної гри.
8. Розробити інструкцію користувача програмного модулю комп’ютерної гри «The Cave of Evil».

Об’єктом дослідження є процес розробки комп’ютерної гри «The Cave of Evil».

Предметом є алгоритми та програмні засоби комп’ютерної гри «The Cave of Evil».

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасної ігрової індустрії та класифікації комп’ютерних ігор

Традиційно комп’ютерні ігри – це програмне забезпечення, призначене для забезпечення ігрового процесу. Під час гри використовуються спеціальні програми для імітації взаємодії у віртуальному просторі між ігровими персонажами та користувачами (або групами користувачів) відповідно до певних алгоритмів. Для ігор використовуються різні пристрої введення, такі як комп’ютерна миша, клавіатура, камера, джойстик тощо [3].

Комп’ютерні ігри, як і традиційні ігри, є моделюванням реальності, в якій гравці свідомо взаємодіють із віртуальним світом. Однак комп'ютерні ігри обмежені за простором, часом і можливостями. Ця відрізняється від традиційних ігор. Візуальні ефекти комп'ютерної гри створені розробником і є результатом творчої діяльності розробника, а не власної уяви гравця. Комп'ютерні ігри також використовують правила , вбудовані в їхні алгоритми.

Індустрія відеоігор зазнала стрімкого розвитку в останні десятиліття. Стати однією з найприбутковіших і найдинамічніших індустрій розваг шляхом комбінування елементів технологій, мистецтва та дизайну. Відіграє важливу роль у культурі, освіті, науці та, звісно, у сфері розваг.

Комп'ютерні ігри є предметом наукових досліджень, а спеціальні комп'ютерні ігри дозволяють гравцям проводити дослідження. Виявилося, що деякі комп’ютерні ігри завдяки своїй високій популярності та можливості одночасної гри кількома людьми стали основою для появи нового спортивного руху: кіберспорт, змагання з комп’ютерних ігор. Комп’ютерні ігри все більше впливають на сучасну людину, а через них і на суспільство в цілому. Неігрове програмне забезпечення зазвичай містить елементи гри. Цей процес називається гейміфікацією [4]. Комп'ютерні ігри працюють за допомогою

комп'ютера або мультимедійного пристрою (замість комп'ютера виступає ігрова консоль). Зараз існують комп’ютерні ігри для різних типів платформ. для різних типів комп'ютерів (планшети, ПК, ноутбуки, нетбуки), для спеціальних консолей (ігрові приставки: Playstation, XBOX360, Nintendo Wii, Sony PSP та ін.), мобільних пристроїв (КПП, комунікатор, мобільний телефон).). Комп’ютерні ігри також включають ігри, ініційовані на ігрових автоматах та подібних пристроях.

Слід зазначити, що комп'ютерні ігри пройшли досить тривалий період розвитку. З моменту своєї появи в 1942 році вони поступово почали завойовувати сферу дозвілля людства. Більше того, цікавим явищем стає залучення дітей до цього процесу. Комп’ютерні ігри були призначені лише для дітей на момент їх створення, але поступово розвивалися з гравцями та набували все більш дорослих рис. У зв'язку з розширенням ареалу поширення комп’ютерних ігор перед людьми постає питання морального вибору і свободи альтернатив.

Важливо відзначити, що під час гри людина може «проживати» кілька реальностей паралельно. Наприклад, я тепер супергерой і рятую людство. За кілька хвилин я вже був пілотом Формули-1. Головний автомобіль прибув на перше місце , і за кілька хвилин мій віртуальний улюбленець виріс. Швидкість змін у цьому випадку майже миттєва, стереотипи соціальних ролей і поведінки змінюються миттєво. Іншими словами, людина , яка грає в комп’ютерну гру, апріорі змушена негайно прийняти і відтворити новий ігровий образ, стати «новим Я» і знайти «іншого Я». Свобода проявляється в здатності вибирати екстремальні шляхи. Наприклад, дозволяє вам померти героя кілька разів в одній грі, увімкнути збереження, а потім повернутися до попереднього моменту . Збереження також можна трактувати як «порятунок», і таке тлумачення стає набагато легітимнішим. Після смерті ви можете повернутися до того моменту, коли ви ще були живі.

Жанр визначається метою гри. Гра може належати до одного або кількох жанрів, іноді відкриваючи нові жанри або перебуваючи за межами всіх

жанрів. Існує вісім основних жанрів відеоігор, які далі поділяються на кілька піджанрів [5].

1) Екшен — жанр комп'ютерних ігор, в якому успіх гравця багато в чому залежить від швидкості реакції і вміння швидко приймати тактичні рішення. Сюжет таких ігор розвивається дуже динамічно, що вимагає концентрації уваги і швидкої реакції на події в грі (рис. 1.1). При цьому кожна зброя зазвичай використовується як основний засіб прогресу в грі.

Рисунок 1.1 – Загальний вигляд комп’ютерної гри у жанрі Екшен

Поділяються на стрілялки, аркади, файтинги та стелс-ігри. Цей жанр є найпопулярнішим, на нього припадає приблизно 70% ринку відеоігор.

2) Симулятор – машинний або комп’ютерний інструмент імітації, який імітує керування процесом, пристроєм або транспортним засобом. Сьогодні словом «симулятор» найчастіше називають комп’ютерну програму, як правило, гру. За допомогою комп’ютерних механічних симуляторів з абсолютною точністю відтворюються інтер’єри салонів літаків для підготовки пілотів, космонавтів і машиністів швидкісних поїздів. Симулятори — це програмно-технічні засоби, які створюють враження реальності та відображають деякі явища та характеристики дійсності у віртуальному середовищі (рис. 1.2).

Рисунок 1.2 – Загальний вигляд комп’ютерної гри у жанрі симулятор

3) Стратегія (рис. 1.3) – захоплюючий жанр відеоігор, який вимагає від гравців глибокого аналітичного мислення, планування та розробки ефективних стратегій для досягнення перемоги. Замість керування одним персонажем, гравці беруть на себе відповідальність за цілі організації, імперії, цивілізації або навіть цілі всесвіти. У цьому жанрі існує два основних типи ігор: покрокові стратегії (TBS) та стратегії в реальному часі (RTS). TBS дозволяють гравцям ретельно обдумувати кожен крок, аналізуючи ситуацію та приймаючи зважені рішення, в той час як RTS вимагають швидкого прийняття рішень та ефективного управління ресурсами в динамічному ігровому процесі, де час не зупиняється. Ключовими елементами стратегічних ігор є управління ресурсами, розвиток бази або цивілізації, військові дії та дипломатія. Гравці повинні збирати, розподіляти та використовувати різноманітні ресурси, такі як золото, дерево, їжа, мінерали та енергія, для розвитку своїх володінь, будівництва будівель, дослідження нових технологій та створення армії.

Рисунок 1.3 – Загальний вигляд комп’ютерної гри у жанрі стратегія

4) Пригоди — це гра, в якій герой, яким керує гравець, розвиває історію, використовуючи предмети, спілкуючись з іншими персонажами та вирішуючи логічні задачі (рис. 1.4). Це сюжетна гра, у якій ви взаємодієте зі світом.

Рисунок 1.4 – Загальний вигляд комп’ютерної гри у жанрі пригоди

5) Музичні ігри – жанр комп’ютерних ігор, який зосереджується на музичних елементах і вимагає від гравця почуття ритму (рис. 1.5). Ігри цього жанру засновані на танцях і виконанні пісень групами. Гравці повинні натискати певні кнопки або виконувати танцювальні рухи залежно від того, що відображається на екрані. За успіх нараховуються бали. Багато музичних ігор також пропонують багатокористувацькі режими, де гравці можуть змагатися один з одним за бали або грати в командах і позувати як група. Однак для більшості з них потрібні спеціальні контролери у вигляді музичних інструментів.

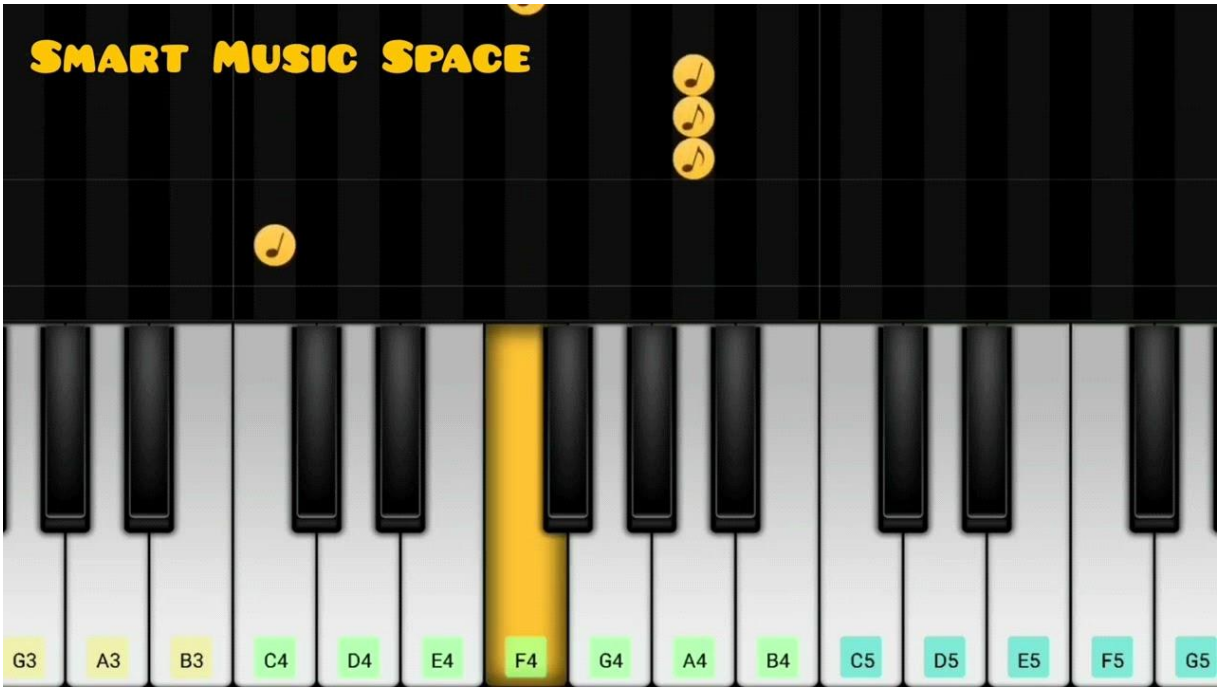

Рисунок 1.5 – Загальний вигляд комп’ютерної гри у жанрі музика

6) CRPG або RPG – жанр комп’ютерних ігор, заснований на елементах ігрового процесу традиційних настільних рольових ігор (рис. 1.6). У рольових іграх гравці керують одним або кількома персонажами. Кожен персонаж описується набором числових рис, переліком умінь і навичок. Приклади таких рис включають показники здоров’я (англ. hitpoints, HP), силу, спритність, захист, показники ухилення та рівень розвитку певних навичок. Вони можуть змінюватися під час гри.

Рисунок 1.6 – Загальний вигляд комп’ютерної гри у жанрі РПГ

7) Комп’ютерні головоломки (рис. 1.7). У некомп'ютерних головоломках роль судді, який стежить за дотриманням правил, виконують самі гравці (пасьянс) або механічний пристрій (чарівний куб). Поява комп’ютерів розширила можливості для головоломок, оскільки легше писати комп’ютерні програми, ніж проектувати механічні пристрої.

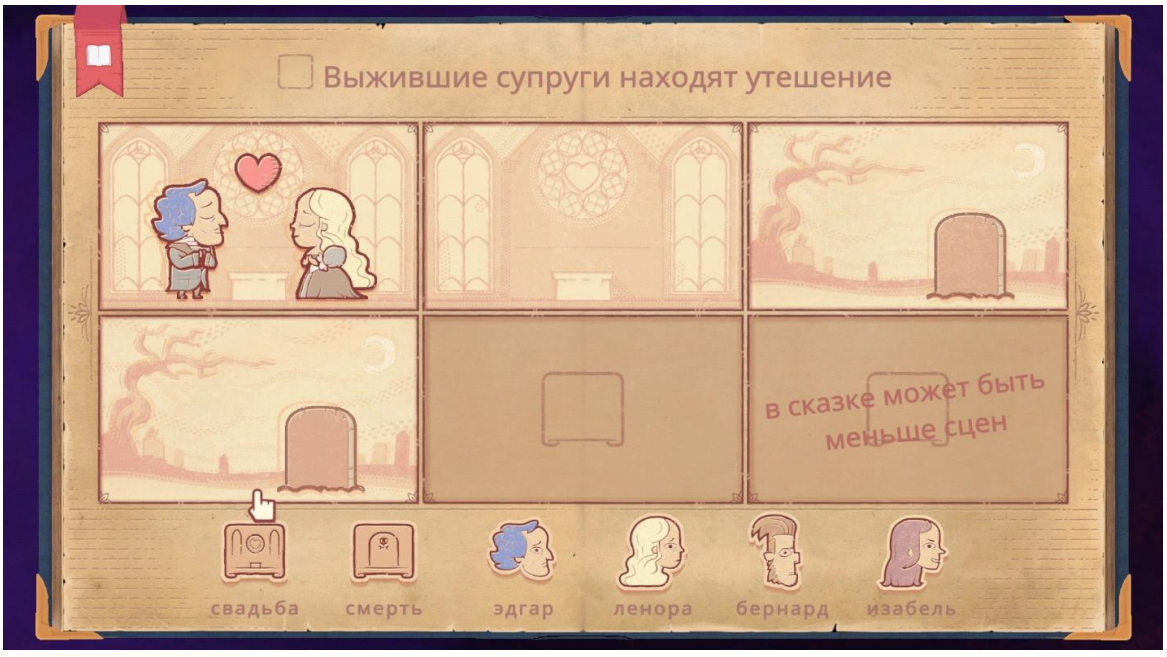

Рисунок 1.7 – Загальний вигляд комп’ютерної гри у жанрі комп’ютерні головоломки

8) Interactive fiction (IF, дослівний переклад — інтерактивна література; текстовий квест; adventure — пригодницька гра) — жанр комп’ютерних ігор, у яких спілкування з гравцем відбувається за допомогою текстової інформації. Розвиток цього жанру почався давно через низьку кількість ресурсів і не припинився з появою графічних ігор (рис 1.8). Існує два типи інтерфейсів: ті, що вводять текст з клавіатури, і інтерфейси, керовані меню (Choose Your Own Adventure), де гравець вибирає дію з кількох запропонованих дій.

Рисунок 1.8 – Загальний вигляд комп’ютерної гри у жанрі інтерактивна література

Також, важливо відзначити, що існують комп’ютерні ігри з мультиплеєром, де гравці змагаються один проти одного, а також ті, де гравці співпрацюють для досягнення спільної мети. Крім того, з'явилися ігри з елементами масових онлайн битв (MOBA), де гравці об'єднуються у команди для боротьби за контроль над територією або ресурсами. Така класифікація дозволяє краще розуміти різноманітність відеоігор та вибирати ті, що відповідають конкретним вподобанням гравців.

1.2 Аналіз проблеми графічного дизайну у комп’ютерних іграх у жанрі 2D платформер

Протягом всієї історії відеоігор різні техніки комп’ютерної графіки використовувалися для відображення вмісту відеоігор. Перевага окремих методів з часом змінилася, головним чином через удосконалення апаратного забезпечення та обмеження, такі як потужність процесора та графічного процесора.

2D комп’ютерна графіка (комп’ютерна генерація цифрових зображень) в основному складається з двовимірних моделей (2D геометричних моделей, тексту, цифрових зображень тощо) і методів, визначених для них [6]. Термін також може стосуватися галузі інформатики, яка включає як методи, так і самі моделі.

Комп’ютерна двовимірна графіка в основному використовується в програмах, розроблених на основі традиційних методів друку та малювання, таких як: В. Друк, картографія, креслення, реклама тощо. У цих додатках двовимірні зображення не лише представляють реальні об’єкти, але й є незалежними культурними експонатами з власним семантичним значенням. Тому 2D-моделі є кращими перед 3D-комп’ютерною графікою, оскільки вони дозволяють безпосередньо контролювати зображення (цей підхід більше схожий на фотографію, ніж на типографіку).

У багатьох галузях, таких як комп’ютерний фотонабір, інженерія та бізнес, розмір зображень у документах, створених за допомогою техніки 2D комп’ютерної графіки, може бути значно більшим (у 1000 разів або більше), ніж їхні цифрові аналоги. Це представлення може бути візуалізовано з різними значеннями роздільної здатності, що робить його більш гнучким і придатним для різних пристроїв виводу. З цієї причини документи та ілюстрації часто зберігаються або передаються як файли 2D-графіки.

Ера двовимірної комп'ютерної графіки почалася в 1950-х роках і базувалася на пристроях векторної графіки. У наступні десятиліття вони були

в основному замінені пристроями на основі растрової графіки. Мова PostScript і протоколи X-Window System були віхами в цій галузі.

Двовимірна графіка — це область комп'ютерної графіки, яка займається створенням і обробкою цифрових зображень, які мають лише два виміри: ширину і висоту [7]. Двовимірна графіка, незважаючи на свою «площинність», широко використовується в різних сферах людської діяльності. Приклад, що ілюструє вигляд піксельної графіки зображено на рисунку 1.9.

Рисунок 1.9 – Загальний вигляд піксельної 2D графіки

Основою двовимірної графіки є піксель, найменший елемент зображення з власним кольором. Растрове зображення (рис. 1.10) складається з сітки пікселів, кожен піксель має своє розташування та колір. Розмір растрового зображення визначається кількістю пікселів по горизонталі і вертикалі. На відміну від растрових зображень, векторна графіка базується на математичних рівняннях, які описують лінії, криві та форми. Векторні зображення можна масштабувати без втрати якості, що робить їх ідеальними для логотипів, значків та інших елементів дизайну. Колірна модель визначає, як кольори представлені та інтерпретуються в цифровому середовищі. Найпоширенішими моделями є RGB (червоний, зелений, синій) для відображення та CMYK (блакитний, пурпуровий, жовтий, чорний) для друку [8]. Шрифти відіграють важливу роль у 2D-графіці, особливо в дизайні та макеті. Правильний вибір шрифту та його оформлення впливає на

читабельність і сприйняття інформації.

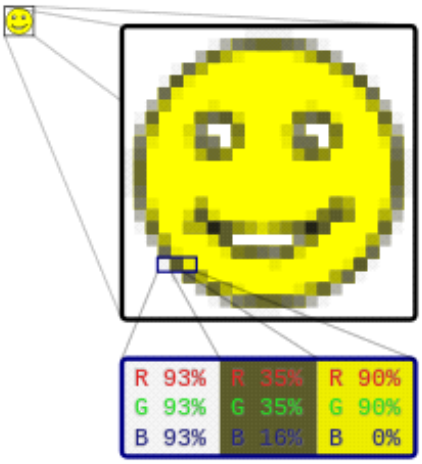

Рисунок 1.10 – Типова схема растрового зображення

2D графіка активно використовується при розробці відеоігор різних жанрів, таких як платформери, головоломки, рольові ігри. Створіть яскравий, виразний візуальний стиль, який запам'ятається вашим гравцям. Двовимірна графіка є важливою частиною веб-дизайну. Використовується для створення логотипів, іконок, ілюстрацій, банерів, інфографіки та інших елементів, які роблять ваш сайт привабливим та інформативним. Приклад 2D графіки у грі в жанрі платформер зображено на рисунку 1.11.

Рисунок 1.11 - Загальний вигляд 2D відеогри в жанрі платформер

2D-анімація широко використовується для створення мультфільмів, рекламних роликів, навчальних відео та інших анімаційних проектів. Це дозволяє оживити персонажів та історії та зробити їх цікавими та захоплюючими. 2D-графіка використовується для оформлення журналів, книг, брошур, плакатів та інших друкованих матеріалів. Це допомагає візуалізувати інформацію та зробити її більш зрозумілою та привабливою для читача.

Двовимірна графіка відіграє важливу роль у створенні зручних та інтуїтивно зрозумілих інтерфейсів програмного забезпечення та веб-сайтів. У міру розвитку технологій двовимірна графіка стає дедалі реалістичнішою та деталізованішою. Це відкриває нові можливості для створення візуально приголомшливих ігор, анімації та дизайну. Двовимірна графіка все частіше використовується для створення інтерактивних елементів на веб-сайтах і в програмах. Це дозволяє користувачам активно взаємодіяти з вашим вмістом, роблячи його більш цікавим і привабливим. 2D-графіку можна ефективно використовувати у віртуальній і доповненій реальності для створення інтерактивних елементів, інтерфейсів і візуальних ефектів.

Візуальна складова 2D-платформних ігор відіграє важливу роль у створенні захоплюючого ігрового досвіду. Використовуючи різноманітні прийоми та методи, ви зможете значно покращити графіку, надати їй унікальності та глибини, а також зробити ігровий світ більш живим та динамічним. Давайте детальніше розглянемо деякі з них:

Паралакс (прокручування паралакса): Ця техніка створює ілюзію глибини та перспективи, розділяючи фон на кілька шарів, які рухаються з різною швидкістю [9]. Шари, розташовані найближче до камери, рухаються швидше, ніж шари, розташовані далі, створюючи відчуття об’єму та простору. Parallax дозволяє створювати ефекти руху камери та виділяти окремі елементи фону. У двовимірних платформерах паралакс особливо ефективний для створення динамічних рівнів із змінним рельєфом і перешкодами.

Динамічне освітлення: Додавання динамічних джерел світла та тіней

може значно покращити візуальне сприйняття вашої гри, зробивши її більш реалістичною та атмосферною. Ви можете використовувати динамічне освітлення для створення різноманітних ефектів, зокрема: В. Зміна часу доби, мерехтіння полум'я, відблиски на поверхнях тощо. 2D-платформери можуть використовувати динамічне освітлення, щоб підкреслити важливі елементи рівня, створити небезпечні зони або просто зробити гру привабливішою.

Ефекти частинок: Частинки — це маленькі графічні елементи, які використовуються для створення різних візуальних ефектів, таких як вибухи, дим, пил, дощ, сніг та іскри. Ефекти частинок можуть бути простими або складними, використовуючи різні форми, кольори, розміри та швидкості частинок. У двовимірних платформних іграх ви можете використовувати ефекти частинок для створення динамічних і захоплюючих сцен, таких як вибух ворогів, руйнування перешкод і застосування магічних заклинань.

Постобробка зображень (ефекти після обробки): Постобробка зображень — це набір прийомів, які застосовуються до готових зображень для покращення якості, створення особливої атмосфери або їх стилізації. Поширені ефекти постобробки включають: Bloom: створює ефект світіння навколо яскравих об’єктів. Blur: розмиває зображення для створення глибини різкості або ефекту руху. Color Correction: змінює кольорову гаму зображення, щоб створити певну атмосферу чи настрій. Vignette: затемнює краї зображення та привертає увагу до центру. В іграх на платформі 2D ви можете використовувати постобробку зображень, щоб створити унікальні візуальні стилі, підкреслити певні елементи гри або покращити загальний вигляд.

Високоякісні текстури та спрайти: Використання детальних текстур і спрайтів високої роздільної здатності є одним із найпростіших і найефективніших способів покращити візуальну якість ваших 2D-ігор. Високоякісні текстури містять такі деталі, як нерівності поверхні, тріщини та подряпини, що робить зображення більш реалістичними та цікавими. спрайти з високою роздільною здатністю дозволяють створювати більш плавні та детальніші анімації для персонажів і об’єктів.

Застосовуючи ці прийоми та хитрощі під час розробки двовимірних платформних ігор, ви можете створювати візуально привабливі та захоплюючі ігри, які надовго залишаться в пам’яті ваших гравців.

Двовимірна графіка – це важлива галузь комп'ютерної графіки, яка динамічно розвивається, і широко використовується в різних сферах людської діяльності. Його основні принципи, різноманітні застосування та перспективи розвитку роблять його цікавою та актуальною темою дослідження.

1.3 Аналіз сучасних комп’ютерних ігор у жанрі 2D платформер

Сучасні ігри включають багато типів і форматів, починаючи від простих незалежних проектів до складних багатокористувацьких онлайн-ігор. Основні з них Тренди в ігровій індустрії включають поширення технологій VR/AR [10], Застосування штучного інтелекту та машинного навчання, розробка хмарних ігор, і поява нових форм монетизації, таких як NFT і блокчейн.

Що стосується ринку 2D-платформерів, то він як і раніше актуальний. Жанр, який сягає корінням у класичні ігри 80-х і 90-х років, повертається популярний своєю простотою, зручністю та ностальгією. Демонстрація ігор "Magic Celeste", "Hollow Knight" та "Shovel Knight" 2D-платформери можуть бути не тільки комерційно успішними, але й дуже високими.

Однією з характеристик сучасного ринку 2D-платформерів є його різноманітність: від класичних бігалок до складніших головоломок, від ретро-піксельної графіки до сучасних арт-стилів, від казуальних мобільних ігор до складних проєктів для консолей і ПК. Крім того, 2D платформери часто використовують як основу для експериментів з інноваційними механіками та геймплеєм, тому розробники можуть зосередитися на створенні унікального ігрового досвіду замість того, щоб витрачати час і ресурси на якісну 3D-графіку.

Комп’ютерні ігри стали невід'ємною частиною сучасного суспільства, і їх використання значно зросло за останні роки. Сьогодні люди

використовують ігри не лише для розваги та відпочинку, як це було спочатку, але й для навчання та наукових досліджень.

Аналіз та вивчення схожих застосувань комп'ютерних ігор у жанрі платформер є важливим кроком у процесі розробки власної гри. Це дозволяє переглянути існуючі ігри цього жанру, доступні на ринку, оцінити їхні можливості та успіхи, а також визначити будь-які прогалини чи недоліки, яких може уникнути у своїх власних проектах.

Ігри на платформі стали добре відомим жанром, який привів багатьох людей у світ відеоігор. Коли була випущена перша гра Super Mario Bros., одних захопила швидкість Соніка, а інших — подолання перешкод Crash Bandicoot. Жанр все ще дуже популярний, і на ПК є багато варіантів платформи [11]. Виконуючи сертифікаційну роботу , враховує широкий спектр проектів та ігор в Інтернеті, виділяючи деякі з найбільш успішних програм.

Hollow Knight було обрано першою грою в аналізі ігрового ринку. Це гра для ПК у жанрі Metroidvania, розроблена інді-студією Team Cherry. Ця гра була випущена 24 лютого 2017 року для операційної системи Windows (рис. 1.12) [12].

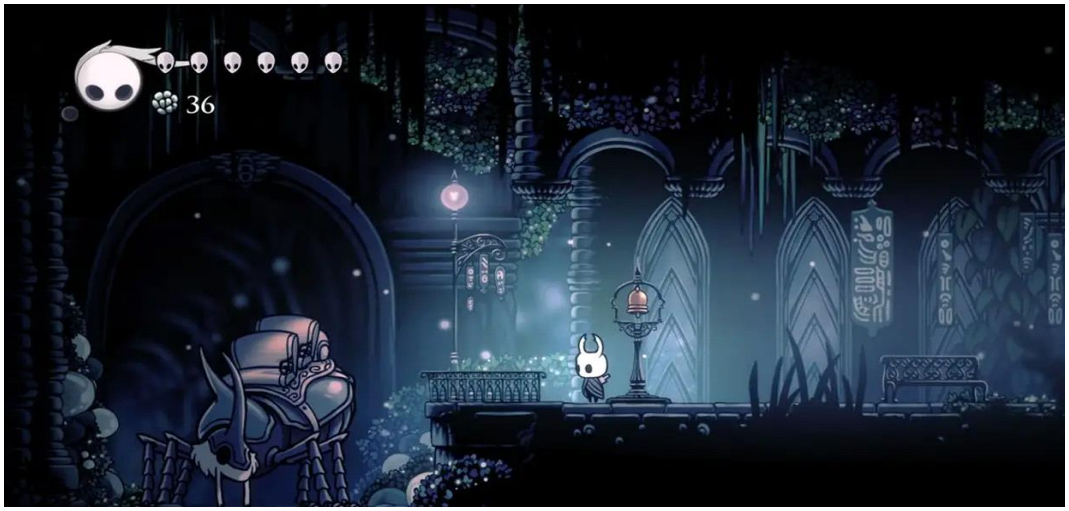

Рисунок 1.12 – Загальний вигляд комп'ютерної гри «Hollow knight»

Hollow Knight – це захоплююча пригодницька гра, яка запрошує вас у подорож величезним, таємничим підземним світом, сповненим небезпек та загадок. Ви будете досліджувати різноманітні локації, від давніх руїн до занедбаних міст, долаючи перешкоди, розгадуючи головоломки та зустрічаючись з різноманітними мешканцями цього світу.

Вашим головним інструментом у боротьбі з ворогами стане цвях – особливий меч, який можна покращувати у коваля, знаходячи нові матеріали та креслення. Битви з ворогами та могутніми босами вимагатимуть від вас спритності, реакції та вмілого використання навичок. За кожну перемогу ви будете отримувати душу, яку можна витратити на лікування або потужні прийоми.

Смерть у Hollow Knight не є кінцем шляху. Ви втратите частину душі та гео, місцевої валюти, але зможете повернути їх, перемігши свою тінь на місці загибелі. Це додає грі елемент ризику та винагороди, спонукаючи вас обережно досліджувати світ та вдосконалювати свої бойові навички.

Hollow Knight вражає своєю мальованою графікою, яка створює неповторну атмосферу таємничості та занепаду. Інтуїтивне керування дозволяє легко освоїти ігровий процес, а захоплюючий сюжет та цікаві персонажі змусять вас зануритися у світ Hollow Knight з головою. Ця гра – справжній шедевр, який поєднує в собі елементи дослідження, бою та розгадування головоломок, пропонуючи незабутній ігровий досвід.

Наступна відеогра, яку ми розберемо, це Dead Cells. Це «roguelike» платформер з елементами «Metroidvania», розроблений французькою студією Motion Twin. Ця гра доступна для Windows, MacOS, Linux і новіших операційних систем, а також для наступних консолей: Nintendo Switch, PlayStation 4 і Xbox One (рис. 1.13) [13].

Рисунок 1.13 – Загальний вигляд комп’ютерної гри «Dead Cells»

У подібних іграх персонажі Dead Cells мають лише одне життя. Якщо персонаж помре, гравець повинен перезапустити гру. Замість того, щоб постійно повертатися до місця розташування, ваш персонаж гине тут, але знання про місцезнаходження залишається згодом, дозволяючи вам швидко відкривати потрібні проходи та секретні місця.

Гра проводить гравців через темний фентезійний світ без безпечних зон або точок збереження. Протягом гри головний герой досліджує різних підземелля, стикається з різними ворогами та перемагає їх, а також збирає додаткові предмети. Він має можливість використовувати різні види зброї, знайдені або отримані протягом гри, від мечів і металевих ножів до гранат, пасток і веж. Характеристики зброї та предметів у грі є випадковими та змінюються залежно від рівня складності.

Усі рівні та всі підземелля генеруються випадковим чином. Гра автоматично створює складний лабіринт із заздалегідь підготовлених елементів і випадковим чином розміщує об’єкти та ворогів. Гравці мають багато шансів померти, здобуваючи досвід і навчаючись на своїх помилках.

Dead Cells отримала визнання критиків за приголомшливу графіку , складний геймплей і загальну якість гри.

Нарешті, ми поговоримо про гру під назвою Mark of the Ninja. Це комп’ютерна гра в жанрі стелс-екшен , розроблена канадською студією Klei Entertainment для Xbox 360 і ПК і опублікована на сервісах Xbox Live Arcade і Steam (рис. 1.14) .

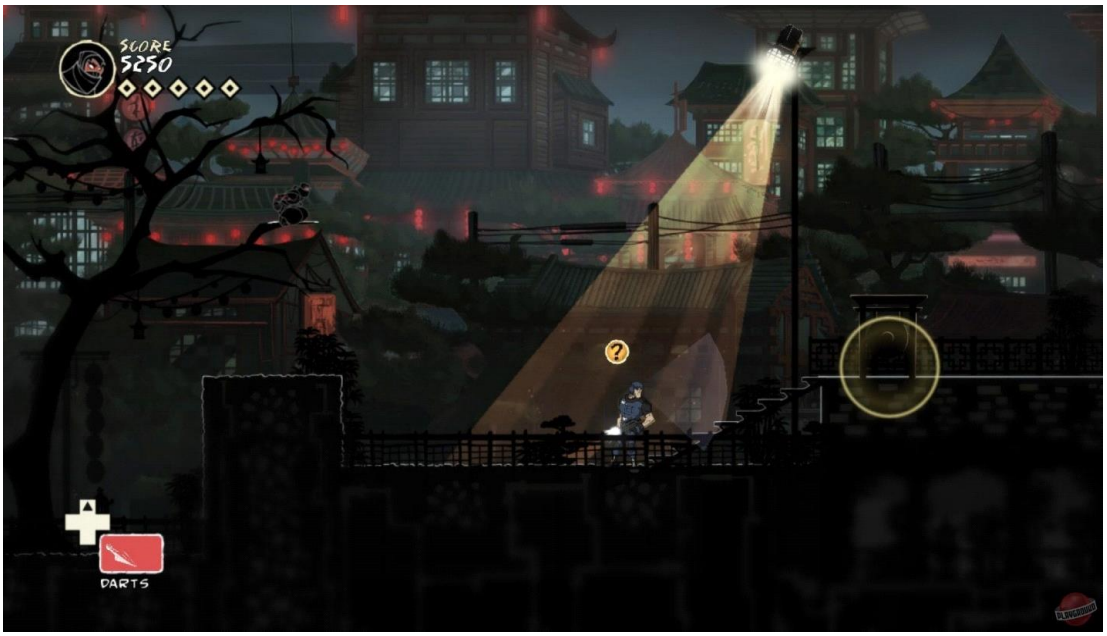

Рисунок 1.14 – Загальний вигляд комп’ютерної гри «Mark of the Ninja»

Mark of the Ninja – це захоплююча 2D-гра в жанрі стелс-екшн, де ви втілюєтесь у роль ніндзя, який використовує свої унікальні здібності для виконання різноманітних місій. Ви будете проникати на ворожі території, уникати пасток, ховатися в тінях та безшумно знешкоджувати охоронців.

Гра пропонує різні шляхи проходження рівнів: ви можете діяти максимально приховано, використовуючи навколишнє середовище для маскування та відволікання уваги ворогів, або ж вдатися до більш агресивного стилю, вступаючи у відкритий бій. Вибір за вами!

У вашому арсеналі будуть різноманітні інструменти та прийоми ніндзя: від традиційних мечів та сюрикенів до димових бомб та крюків-кішок. Ви також зможете покращувати свої навички та спорядження, використовуючи очки, отримані за успішне виконання завдань.

Mark of the Ninja вражає своєю стильною графікою, атмосферною

музикою та захоплюючим ігровим процесом, який триматиме вас у напрузі до самого кінця. Якщо ви любите стелс-ігри та хочете відчути себе справжнім ніндзя, то ця гра точно для вас!

1.4 Висновок до розділу 1

У першому розділі проведено ґрунтовний аналіз 2D графіки, її історії, застосування у комп’ютерних іграх, зокрема у 2D платформерах.

Розглянуто ключові поняття, такі як піксельна та векторна графіка, кольорові моделі, шрифти, а також еволюцію 2D графіки від початку до сучасних технологій. Особливу увагу приділено аналізу успішних ігор (Hollow Knight, Dead Cells, Mark of the Ninja), що дозволило виявити сучасні тенденції у візуальному дизайні та ігровій механіці. Крім того, розглянуто проблеми розробки 2D графіки та методи їх вирішення.

Проведений аналіз підтверджує актуальність та перспективність 2D графіки, що є важливим підґрунтям для подальшої розробки власної 2D гри в рамках дипломної бакалаврської роботи.

2 РОЗРОБКА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ КОМП’ЮТЕРНОЇ ГРИ

2.1 Розробка архітектури програмного модулю комп’ютерної гри

Архітектура програмного забезпечення комп’ютерної гри "The Cave of Evil" є основою для її функціональності, продуктивності та масштабованості. Вона визначає структуру, взаємозв'язки та принципи роботи різних компонентів гри, забезпечуючи їхню ефективну взаємодію та узгоджену роботу [15].

Ядро гри (Game Core) є центральним компонентом, відповідальним за основний ігровий цикл, управління станами гри, обробку вхідних даних від гравця та оновлення стану світу гри. Воно містить основний ігровий цикл (Game Loop), який постійно оновлює стан гри, обробляє вхідні дані від гравця, оновлює позиції об'єктів, виконує ігрову логіку та рендерить зображення на екран. Крім того, ядро гри відповідає за управління станами гри, перемикаючись між різними станами (меню, завантаження рівня, ігровий процес, пауза тощо) та управляючи їхньою логікою. Також воно зчитує та обробляє вхідні дані від гравця (натискання клавіш, рухи миші, дії геймпада) та передає їх відповідним компонентам гри, а також розподіляє події, що відбуваються у грі (зіткнення об'єктів, активація пасток, завершення квестів), між відповідними компонентами для їхньої обробки.

Рендерер (Renderer) відповідає за візуалізацію ігрового світу, включаючи моделі персонажів, об'єкти, ефекти та інтерфейс користувача. Він використовує OpenGL або DirectX для створення тривимірного зображення ігрового світу на основі моделей об'єктів, текстур, освітлення та ефектів, а також відображає на екрані елементи інтерфейсу, такі як меню, індикатори здоров'я та енергії, міні-карту тощо. Рендерер також застосовує різні техніки оптимізації для забезпечення плавної та швидкої роботи гри навіть на слабких комп'ютерах.

Звуковий двигун (Sound Engine) забезпечує відтворення звукових ефектів, музики та діалогів, створюючи атмосферу та підсилюючи емоційне занурення гравця у гру. Він завантажує та відтворює звукові ефекти, пов'язані з діями гравця, подіями у грі та іншими елементами ігрового світу, а також забезпечує фонову музику. Звуковий двигун може використовувати технології просторового звуку для створення більш реалістичного звучання, що дозволяє гравцеві краще орієнтуватися у просторі.

Фізичний двигун (Physics Engine) моделює фізичну взаємодію об'єктів у грі, враховуючи такі фактори, як гравітація, зіткнення, тертя тощо. Він використовує PhysX або інший фізичний двигун для реалістичного моделювання руху об'єктів, зіткнень, гравітації та інших фізичних явищ. Фізичний двигун також визначає, коли об'єкти стикаються один з одним, та передає цю інформацію іншим компонентам гри для відповідної реакції, а також забезпечує виконання фізичних обмежень, таких як зв'язки між об'єктами, шарніри, мотузки тощо.

нтерфейс користувача (UI) відіграє ключову роль у забезпеченні комфортної та ефективної взаємодії гравця з віртуальним світом гри. Він слугує мостом між гравцем та грою, надаючи доступ до різноманітних інструментів та інформації, необхідних для успішного проходження.

UI відповідає за візуальне представлення всіх елементів керування, таких як меню, інвентар, карта, вікна діалогів, індикатори здоров'я та ресурсів, кнопки дій та інші інтерактивні об'єкти. Він забезпечує зручну навігацію по меню, дозволяючи гравцеві швидко знаходити та використовувати необхідні функції, переглядати карту, відстежувати прогрес та взаємодіяти з іншими персонажами.

UI також зчитує та обробляє вхідні дані від гравця, такі як натискання кнопок, вибір елементів меню, переміщення курсору та інші дії, пов'язані з інтерфейсом. Це дозволяє гравцеві керувати своїм персонажем, вибирати предмети з інвентарю, приймати рішення у діалогах та виконувати інші дії, необхідні для проходження гри.

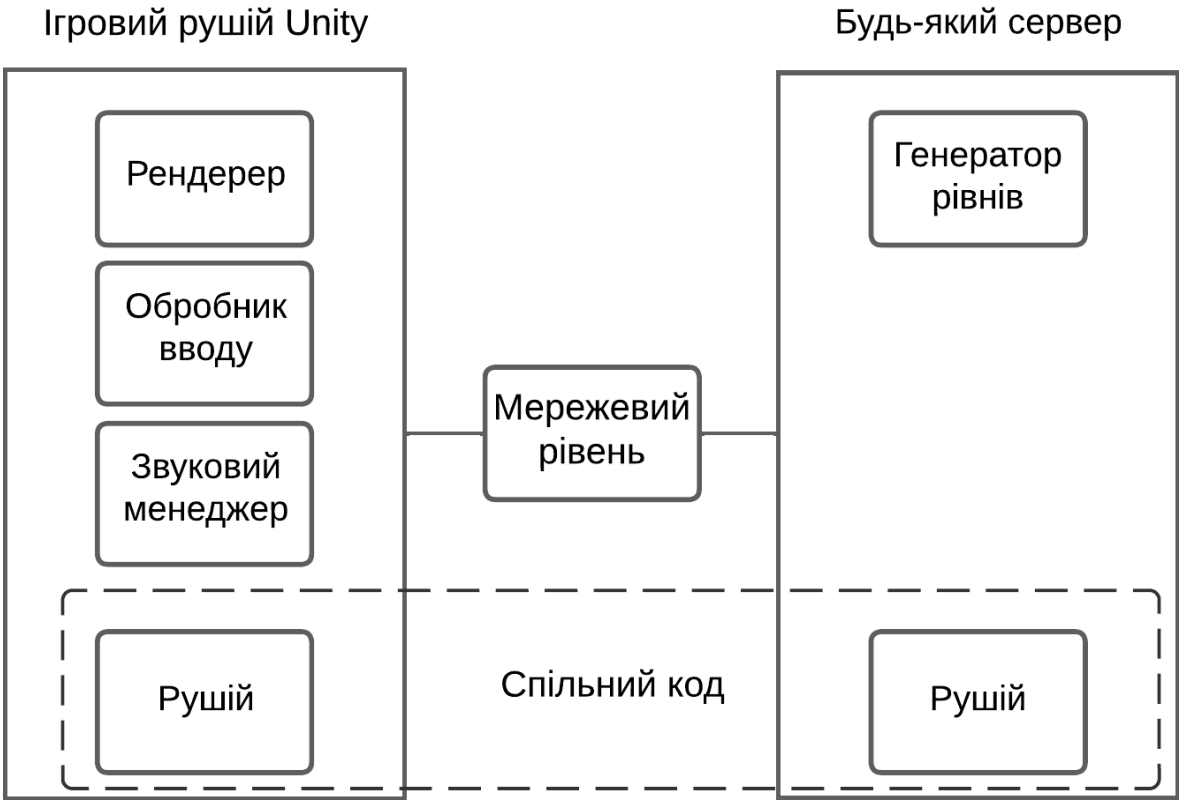

Рисунок 2.1 – Конструктивна схема основних компонентів архітектури програмного модулю комп’ютерної гри «The Cave of Evil»

Архітектура програмного забезпечення гри "The Cave of Evil" побудована на основі модульного дизайну, що передбачає розбиття гри на окремі компоненти або модулі, кожен з яких відповідає за певну функціональність, таку як рендеринг графіки, обробка звуку, фізична симуляція, штучний інтелект та мережева взаємодія. Модулі розробляються та тестуються незалежно, що дозволяє різним членам команди розробників працювати паралельно та забезпечує легкість внесення змін до окремих компонентів без впливу на інші. Чіткі інтерфейси між модулями забезпечують їхню слабку зв'язність, що дозволяє змінювати один модуль без необхідності змінювати інші [16].

Ця модульна архітектура забезпечує гнучкість, дозволяючи легко розширювати та модифікувати гру шляхом додавання нових ігрових механік, рівнів, персонажів та об'єктів без значних змін у коді. Вона також адаптивна, дозволяючи легко налаштовувати графічні параметри, роздільну здатність

екрана та схему керування для різних платформ та пристроїв.

Масштабованість архітектури дозволяє легко збільшувати обсяг контенту гри, додаючи нові рівні, персонажів, об'єкти та інші елементи без втрати продуктивності, а також розширювати її для підтримки багатокористувацької гри. Це досягається за рахунок використання ефективних алгоритмів та структур даних, а також оптимізації коду.

Переносимість гри на різні платформи, такі як ПК, консолі та мобільні пристрої, забезпечується за рахунок використання кросплатформенних бібліотек та інструментів розробки, а також абстрагування від апаратних особливостей конкретних платформ. Модульний дизайн та чіткі інтерфейси між модулями дозволяють мінімізувати кількість змін, необхідних для адаптації гри до нової платформи, роблячи процес портування максимально простим та швидким.

Дотримання цих принципів при розробці архітектури (рис. 2.2) гри "The Cave of Evil" дозволяє створити гнучку, масштабовану та переносиму гру, яку можна легко розширювати та адаптувати до різних платформ, забезпечуючи високу якість графіки, звуку та ігрового процесу.

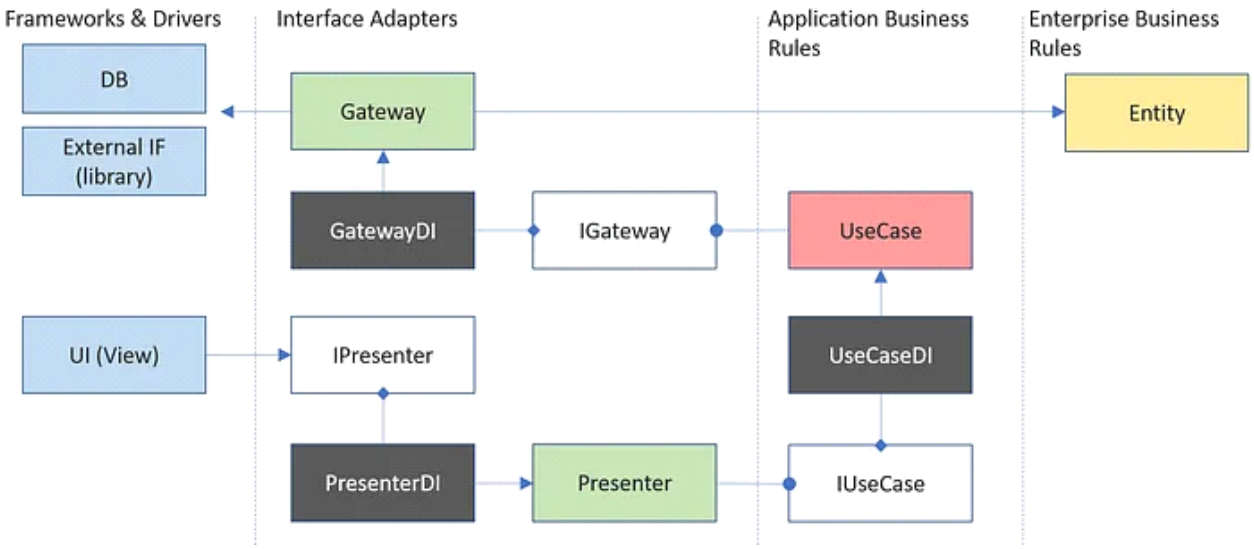

Рисунок 2.2– Діаграма принципу взаємодії різних шарів архітектури

Система збереження та завантаження у відеогрі "The Cave of Evil" відіграє важливу роль у забезпеченні комфорту гравця та збереженні його прогресу. Вона дозволить гравцеві продовжити гру з того місця, де він зупинився, не втрачаючи досягнутих результатів.

Для забезпечення повноцінного відновлення ігрового процесу необхідно зберігати такі ключові дані:

1) Позиція персонажа: Координати (x, y) персонажа на поточному рівні, щоб гравець міг продовжити з того ж місця.

2) Стан здоров'я та інші характеристики: Поточний рівень здоров'я, енергії, магії (якщо є), наявність спеціальних здібностей або бонусів.

3) Інвентар: Перелік предметів, що знаходяться в інвентарі персонажа, їх кількість та стан (наприклад, знос зброї).

4) Прогрес гри: Інформація про пройдені рівні, відкриті області, виконані завдання, вбиті вороги, знайдені секрети тощо.

5) Стан ігрового світу: Стан перемикачів, дверей, пасток та інших інтерактивних об'єктів на поточному рівні, щоб світ відновився у тому ж вигляді, як і під час збереження.

6) Налаштування гри: Збереження вибраних гравцем налаштувань (роздільна здатність, гучність звуку, схема керування) для їх автоматичного застосування при завантаженні гри.

Для збереження даних можна використовувати формати JSON або XML. Вони є зручними для читання та редагування людиною, що може бути корисним для налагодження та тестування. Крім того, вони підтримують різні типи даних (числа, рядки, списки, словники), що дозволяє легко зберігати складні структури даних.

Алгоритми збереження та завантаження:

1) Збереження: зібрати всі необхідні дані з різних компонентів гри (персонаж, ігровий світ, інвентар тощо); структурувати дані у вигляді об'єкта або словника; серіалізувати дані у вибраний формат (JSON або XML);

записати серіалізовані дані у файл або відправити їх на сервер для зберігання у хмарі.

2) Завантаження: прочитати дані з файлу або отримати їх із сервера; десеріалізувати дані у об'єкт або словник; відновити стан гри, використовуючи завантажені дані: Встановити позицію персонажа; відновити стан здоров'я та інші характеристики; заповнити інвентар предметами; встановити прогрес гри; відновити стан ігрового світу; застосувати налаштування гри.

Для забезпечення зручності гравця можна реалізувати кілька слотів збереження, автозбереження у ключових моментах гри та можливість збереження у хмарі для доступу до прогресу з різних пристроїв.

Оптимізація продуктивності програмного модуля відеогри "The Cave of Evil" є критично важливим аспектом розробки, оскільки вона безпосередньо впливає на плавність ігрового процесу, особливо на менш потужних пристроях. Для досягнення високої продуктивності можна застосувати такі методи:

Кешування ресурсів: Завантаження графічних ресурсів (текстур, спрайтів), звукових файлів та інших даних з диска може займати значний час. Щоб уникнути повторних завантажень, можна використовувати кешування. При першому зверненні до ресурсу він завантажується з диска та зберігається в оперативній пам'яті. При наступних зверненнях ресурс береться з кешу, що значно прискорює процес.

Об'єднання спрайтів (Sprite Atlasing): У 2D-іграх часто використовується велика кількість спрайтів для відображення персонажів, об'єктів та елементів оточення. Кожен спрайт вимагає окремого звернення до графічного процесора, що може знижувати продуктивність. Об'єднання спрайтів в один великий атлас дозволяє зменшити кількість звернень до GPU та прискорити рендеринг.

Використання ефективних алгоритмів та структур даних: Вибір правильних алгоритмів та структур даних може значно вплинути на продуктивність гри. Наприклад, для пошуку шляху можна використовувати

алгоритм A*, для виявлення зіткнень – квадродерева або BSP-дерева, для зберігання даних – хеш-таблиці або словники.

Оптимізація рендерингу: Використання таких технік, як об'єднання об'єктів (batching), зменшення кількості полігонів у моделях, використання рівнів деталізації (LOD) та оптимізація шейдерів, дозволяє знизити навантаження на графічний процесор та підвищити частоту кадрів.

Оптимізація звуку: Використання стиснених аудіоформатів, потокове завантаження музики та звукових ефектів, обмеження кількості одночасно відтворюваних звуків та оптимізація звукових ефектів дозволяє знизити навантаження на процесор та пам'ять.

Профілювання та аналіз продуктивності: Використання інструментів профілювання дозволяє виявити "вузькі місця" у коді, які найбільше впливають на продуктивність. Це дозволяє зосередити зусилля на оптимізації саме цих ділянок коду.

Врахування цільової платформи: При розробці гри необхідно враховувати особливості цільової платформи (ПК, мобільні пристрої, консолі) та оптимізувати гру під її апаратні можливості. Це може включати використання різних графічних налаштувань, обмеження використання ресурсів та адаптацію інтерфейсу користувача.

2.2 Розробка методів створення ігрової механіки

Комп’ютерні ігри стали невід'ємною частиною сучасної культури, пропонуючи гравцям захоплюючі пригоди, випробування та можливість зануритися у віртуальні світи. Одним з ключових елементів, що визначають успіх відеогри, є її ігрова механіка - сукупність правил, взаємодій та алгоритмів, що формують ігровий процес та забезпечують захоплюючий досвід для гравців. Цей підрозділ присвячений дослідженню та розробці методів створення ігрової механіки для програмного модуля відеогри "The Cave of Evil", яка є пригодницькою грою з елементами жахів, де гравці

досліджують таємничу печеру, наповнену небезпеками та загадками.

Створення ефективної ігрової механіки вимагає ретельного аналізу та вибору підходів, що відповідають жанру та особливостям гри. Розглянемо деякі з найпоширеніших підходів [17]:

1) Процедурна генерація: Цей підхід дозволяє створювати унікальні ігрові світи та рівні, використовуючи алгоритми та випадкові числа. Процедурна генерація забезпечує високу реіграбельність та непередбачуваність, що особливо важливо для ігор з дослідженням та виживанням.

2) Скриптування: Скриптування надає розробникам потужний інструмент для створення складних ігрових сценаріїв, подій та взаємодій між об'єктами. За допомогою скриптів можна реалізувати різноманітні механіки, такі як діалоги, квести, бойові системи тощо.

3) Модульний дизайн: Модульний дизайн передбачає розбиття ігрової механіки на окремі компоненти або модулі, які можна легко змінювати, комбінувати та розширювати. Це забезпечує гнучкість та масштабованість розробки, дозволяючи легко додавати нові функції та механіки.

4) Фізичні симуляції: Фізичні симуляції дозволяють моделювати реалістичну поведінку об'єктів у віртуальному світі, враховуючи такі фактори, як гравітація, зіткнення, тертя тощо. Це може бути використано для створення різноманітних ігрових механік, таких як платформинг, головоломки з фізикою, реалістичні бойові системи.

У контексті комп’ютеної гри "The Cave of Evil", метою якої є створення атмосферного та напруженого досвіду дослідження таємничої печери, було обрано комбінований підхід до розробки ігрової механіки, що поєднує в собі елементи процедурної генерації, скриптування та модульного дизайну. Кожен з цих підходів має свої унікальні переваги та недоліки, але їхнє поєднання дозволяє досягти оптимального балансу між гнучкістю, ефективністю та масштабованістю розробки.

Процедурна генерація, як ключовий елемент даної методології, дозволяє створювати унікальні та непередбачувані рівні печери, використовуючи

алгоритми та випадкові числа. Це забезпечує високу реіграбельність, оскільки кожне проходження гри буде відрізнятися від попереднього, що є особливо важливим для жанру пригод та жахів, де непередбачуваність та несподіванка є ключовими елементами залучення гравців. Проте, процедурна генерація вимагає ретельного балансування та тестування, щоб уникнути створення нецікавих або неможливих для проходження рівнів. Процедура генерації для створення ландшафту зображена на рисунку 2.3

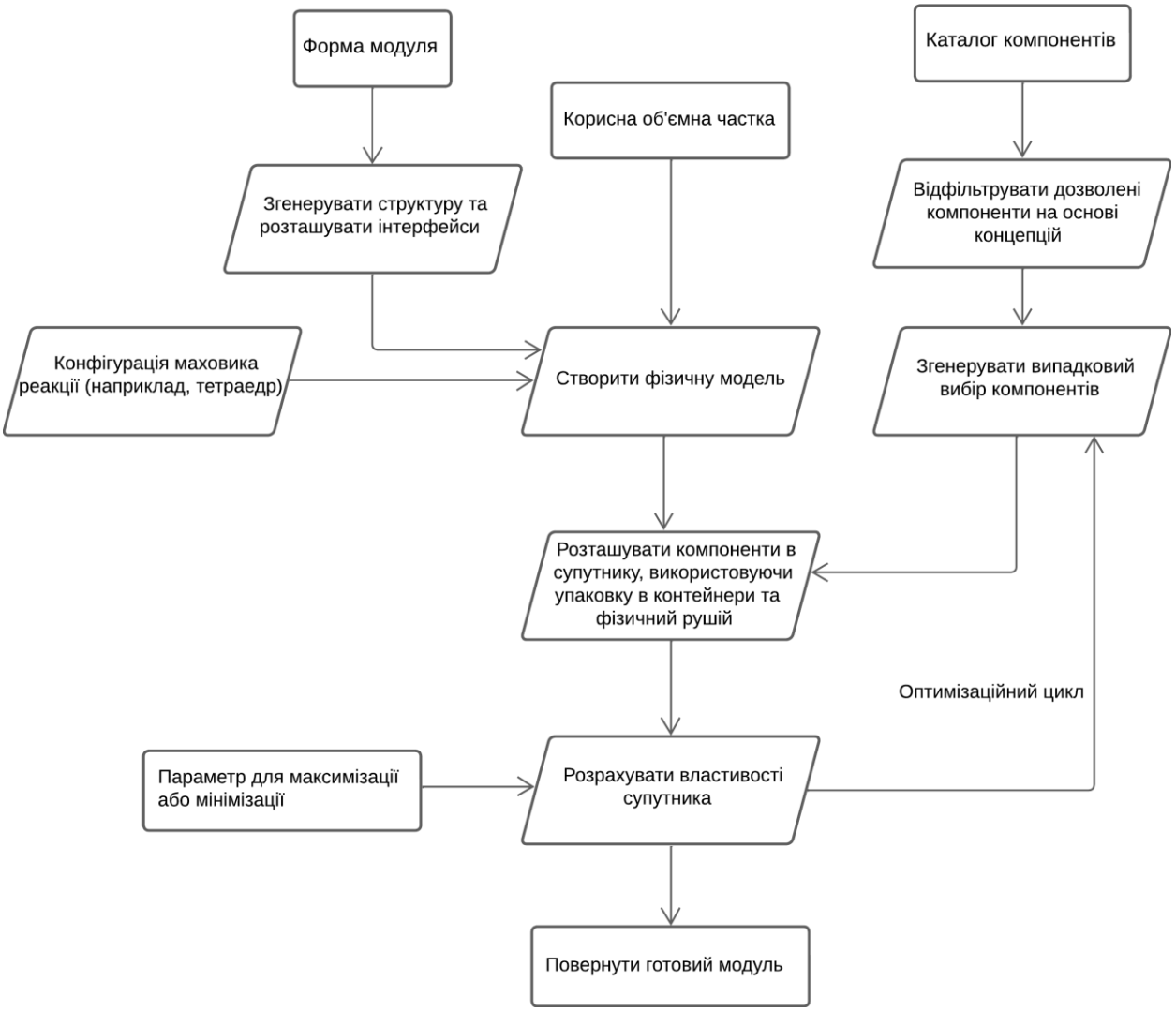

Рисунок 2.3 – Діаграма процедури генерації для створення ландшафту

Скриптування, як інший важливий компонент методології, надає розробникам потужний інструмент для створення складних ігрових сценаріїв, подій та взаємодій між об'єктами. За допомогою скриптів можна реалізувати

різноманітні механіки, такі як діалоги з персонажами, квести, головоломки, бойові системи, зміни освітлення та звукові ефекти, що значно збагачують ігровий світ та роблять його більш інтерактивним та реалістичним. Однак, скриптування може бути менш ефективним, ніж використання компільованої мови програмування, та вимагає ретельного тестування та відлагодження.

Модульний дизайн (рис. 2.4), як третій елемент методології, передбачає розбиття ігрової механіки на окремі компоненти або модулі, які можна розробляти та тестувати незалежно один від одного. Це забезпечує гнучкість та масштабованість розробки, дозволяючи легко додавати та змінювати різні ігрові системи, такі як система інвентарю, система бою, система діалогів, система розвитку персонажа тощо. Модульний дизайн також спрощує командну роботу та підтримку проекту, оскільки кожен модуль може бути розроблений окремим членом команди. Проте, модульний дизайн може ускладнити інтеграцію та оптимізацію гри, оскільки необхідно забезпечити коректну взаємодію між різними модулями.

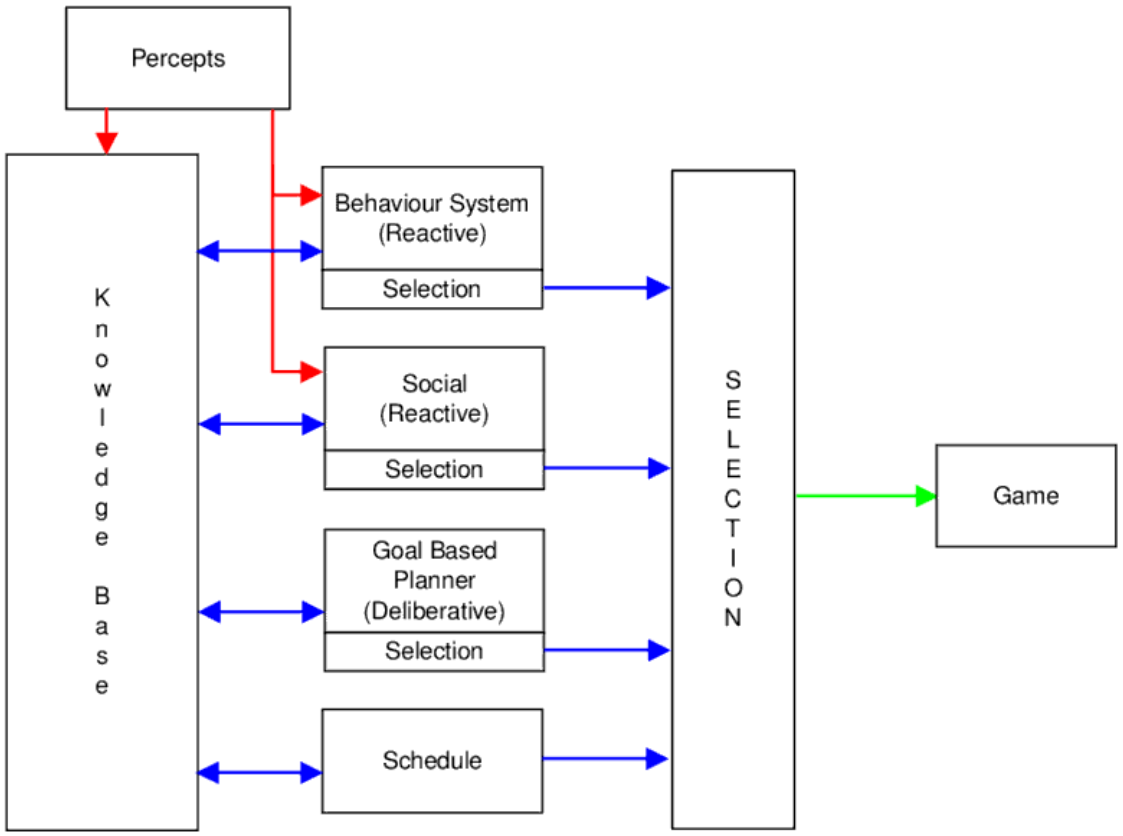

Рисунок 2.4 – Принципова схема модульного дизайну у розробці відеоігор

Застосування комбінованого підходу, що поєднує процедурну генерацію, скриптування та модульний дизайн, дозволить створити захоплюючу та атмосферну гру "The Cave of Evil", яка буде тримати гравців у напрузі, змушувати їх досліджувати кожен куточок таємничої печери та відчувати справжній страх перед невідомим. Цей підхід забезпечить не тільки унікальний ігровий досвід, але й гнучкість та масштабованість розробки, що є важливими факторами для успіху будь-якого ігрового проекту.

Розробка методів обробки зіткнень у комп’ютерної гри "The Cave of Evil" є важливим аспектом створення реалістичної та захоплюючої ігрової механіки. У цьому контексті необхідно враховувати різні типи зіткнень, які можуть виникати між ігровими об'єктами, та розробляти відповідні алгоритми для їх обробки.

Зіткнення персонажа з твердими поверхнями (платформами, стінами, стелею):

1) Виявлення зіткнення: Використовувати геометричні примітиви (наприклад, прямокутники або кола) для представлення об'єктів та перевіряти їх перетин.

2) Обробка зіткнення: зупинити рух персонажа у напрямку зіткнення; забезпечити можливість стояти на платформах та перешкоджати проходженню крізь стіни та стелю; реалізувати механіку відштовхування від стін для стрибків та інших маневрів.

Зіткнення персонажа з ворогами:

1) Виявлення зіткнення: Використовувати ті ж геометричні примітиви, що й для зіткнень з твердими поверхнями.

2) Обробка зіткнення: завдати шкоди персонажу або ворогу залежно від типу атаки та захисту; реалізувати різні типи реакції на зіткнення, такі як відкидання персонажа, зміна стану ворога (наприклад, оглушення) тощо.

Зіткнення персонажа з пастками:

1) Виявлення зіткнення: Використовувати тригери – спеціальні області,

які активують пастку при вході персонажа.

2) Обробка зіткнення: завдати шкоди персонажу або застосувати інший ефект залежно від типу пастки; активувати анімацію або візуальний ефект пастки.

Зіткнення персонажа з інтерактивними об'єктами (двері, перемикачі, предмети):

1) Виявлення зіткнення: Використовувати тригери або перевірку відстані між персонажем та об'єктом.

2) Обробка зіткнення: виконати дію, пов'язану з об'єктом (наприклад, відкрити двері, активувати перемикач, підібрати предмет); відобразити повідомлення або підказку для гравця.

Зіткнення між ворогами та іншими об'єктами:

1) Виявлення зіткнення: Використовувати ті ж геометричні примітиви, що й для інших типів зіткнень.

2) Обробка зіткнення: запобігти проходженню ворогів крізь стіни та інші перешкоди; реалізувати взаємодію між ворогами (наприклад, бійки між собою).

Для ефективної обробки зіткнень необхідно використовувати оптимізовані алгоритми та структури даних, такі як просторове розбиття (наприклад, квадродерева або BSP-дерева) для швидкого визначення потенційно пересічних об'єктів. Це дозволить зменшити кількість перевірок на перетин та підвищити продуктивність гри.

2.3 Розробка алгоритму функціонування програмного модулю комп’ютерної гри

Розроблений алгоритм функціонування програмного модулю комп’ютерної гри «The Cave of Evil» зображено на рисунку 2.5

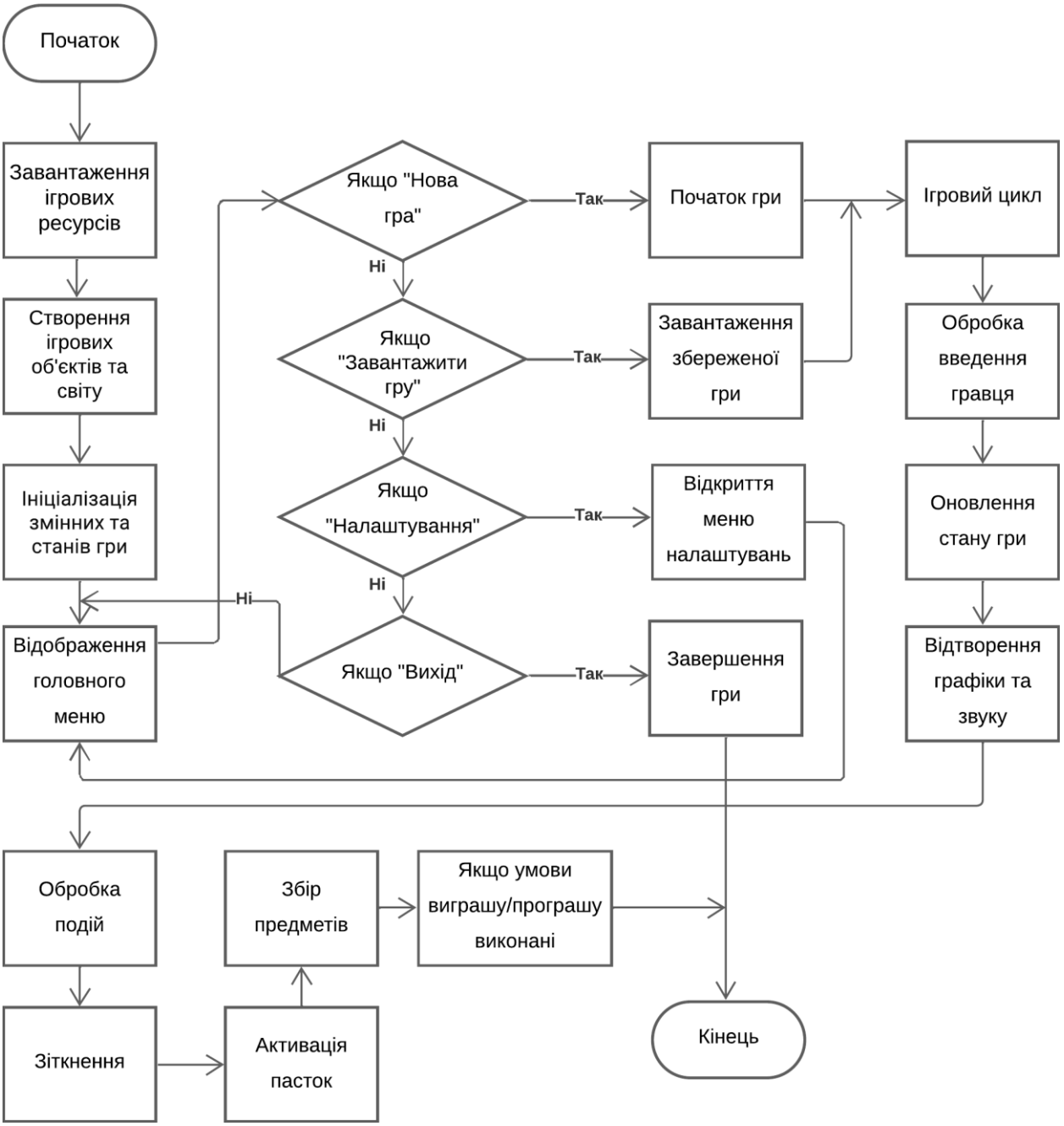

Рисунок 2.5 - Алгоритм функціонування програмного модулю комп’ютерної гри «The Cave of Evil»

Розроблений алгоритм для відеогри «The cave of evil» можна розділити на такі кроки:

Завантаження ігрових ресурсів (графіка, звуки, рівні).

Створення ігрового світу та ігрових об'єктів (персонаж, вороги, платформи).

Відображення головного меню гри, де гравець може вибрати опції "Нова

гра", "Завантажити гру", "Налаштування" або "Вихід".

Обробка вводу гравця: зчитування дій гравця (рух, стрибки, атаки); перевірка коректності вводу та виконання відповідних дій.

Оновлення стану гри; розрахунок фізики руху персонажів та об'єктів з урахуванням гравітації, зіткнень та інших фізичних параметрів; перевірка зіткнень між об'єктами, включаючи зіткнення персонажа з ворогами, платформами та іншими інтерактивними об'єктами; оновлення стану інших ігрових елементів, таких як пастки, бонуси, двері, перемикачі тощо, залежно від взаємодії з гравцем або інших умов; перевірка умов перемоги або програшу, наприклад, досягнення кінця рівня, смерті персонажа або виконання певних завдань.

Рендеринг – це процес, який перетворює програмний код у візуальне свято на екрані гравця. Він включає в себе створення та відображення різноманітних графічних елементів, таких як персонажі, оточуючий світ, спеціальні ефекти та анімації, використовуючи потужні інструменти, такі як OpenGL або DirectX. Крім того, рендеринг відповідає за відображення інтерфейсу користувача, який є важливим інструментом взаємодії гравця з грою. Він включає в себе меню, індикатори здоров'я та енергії, рахунок, міні-карту, підказки та інші елементи, які допомагають гравцеві орієнтуватися у віртуальному світі та приймати правильні рішення. Також рендеринг забезпечує відтворення звукових ефектів та музики, які створюють атмосферу та підсилюють емоційне занурення у гру.

Зіткнення – це невід'ємна частина будь-якої гри, яка надає їй динаміки та реалізму. Система зіткнень визначає, коли персонаж або об'єкт стикається з іншим об'єктом у грі, таким як ворог, платформа або предмет. Залежно від типу зіткнення, система виконує відповідні дії, такі як завдання шкоди персонажу або ворогу, відбиття атаки, збір предмета або активація механізму. Це дозволяє створювати різноманітні ігрові ситуації та виклики для гравця.

Активація пасток – це ще один важливий елемент ігрового процесу, який додає непередбачуваності та вимагає від гравця уважності та обережності.

Система пасток перевіряє, чи виконуються умови для їх активації, наприклад, натискання на кнопку або перетин певної області. При активації пастки виконуються певні дії, такі як завдання шкоди персонажу, переміщення платформ або відкриття дверей, що може суттєво вплинути на хід гри.

Збір предметів є важливою частиною багатьох ігор, оскільки дозволяє гравцеві отримувати нові ресурси, зброю, спорядження або інші корисні предмети, які можуть допомогти йому у проходженні гри. Система збору предметів перевіряє, чи відбулося зіткнення персонажа з предметом, і якщо так, то додає цей предмет до інвентарю персонажа або виконує іншу дію, пов'язану з цим предметом.

Збереження та завантаження гри – це важливі функції, які дозволяють гравцеві зберігати свій прогрес та продовжувати гру з останньої збереженої точки. Система збереження записує поточний стан гри, такий як позиція персонажа, стан здоров'я, інвентар та прогрес, у файл або хмарне сховище. Система завантаження дозволяє відновити цей стан, що дає гравцеві можливість продовжити гру з місця, де він зупинився.

Завершення гри відбувається при виконанні певних умов, таких як досягнення кінця рівня, перемога над босом або смерть персонажа. Система завершення гри відображає відповідне повідомлення (екран перемоги або поразки) та звільняє ресурси, зайняті грою.

Цей алгоритм описує основні етапи роботи програмного модуля відеогри "The cave of evil", забезпечуючи обробку вводу гравця, оновлення стану гри, рендеринг графіки та звуку, обробку подій та взаємодію з іншими компонентами гри. Він також враховує особливості жанру пригодницької гри з елементами жахів, включаючи процедурну генерацію рівнів, скриптові події та взаємодію з ігровим світом.

Оскільки персонаж гри має фізичні властивості, то він повинен взаємодіяти з навколишнім світом з дотриманням фізичних законів. Для цього буде реалізовано алгоритм зіткнень об’єктів для комп’ютерної гри "The Cave of Evil" на рисунку 2.6.

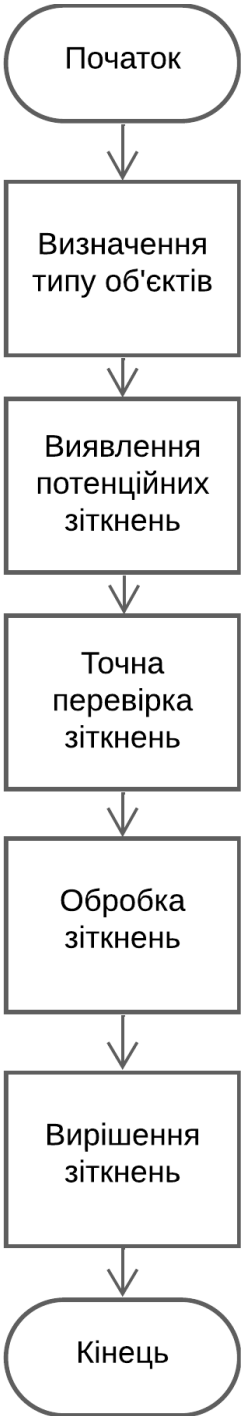

Рисунок 2.6 – Схема алгоритму зіткнень об’єктів у комп’ютерні грі "The Cave of Evil"

Алгоритм зіткнення об'єктів у 2D платформері "The Cave of Evil" можна реалізувати, використовуючи наступний підхід:

1) Визначення типу об'єктів:статичні об'єкти: платформи, стіни, стеля - нерухомі об'єкти, з якими персонаж може зіткнутися; динамічні об'єкти: персонаж гравця, вороги, предмети, що рухаються; тригери: невидимі області, що активують певні події при зіткненні з персонажем (пастки, переходи між рівнями тощо).

2) Виявлення потенційних зіткнень: використовувати метод обмежувальних рамок (Bounding Box) для попередньої перевірки зіткнень. Обмежувальна рамка - це прямокутник, що описує об'єкт. Якщо обмежувальні рамки двох об'єктів перетинаються, це означає, що можливе зіткнення; для оптимізації можна використовувати просторове розбиття (наприклад, квадродерево або грід), щоб розділити ігровий світ на області та перевіряти зіткнення тільки між об'єктами, що знаходяться в одній області.

3) Точна перевірка зіткнень; якщо обмежувальні рамки перетинаються, провести більш точну перевірку зіткнення, використовуючи геометрію об'єктів (наприклад, перевірка перетину кіл або прямокутників); для складніших форм можна використовувати метод розділення осей (Separating Axis Theorem - SAT) або інші алгоритми виявлення зіткнень.

4) Обробка зіткнень: статичні об'єкти (зупинити рух об'єкта у напрямку зіткнення. Забезпечити коректну реакцію на зіткнення (наприклад, персонаж повинен зупинитися при зіткненні зі стіною, але може стояти на платформі)); динамічні об'єкти (залежно від типу об'єктів (персонаж-ворог, персонаж-предмет) виконати відповідну дію (наприклад, завдати шкоди, підібрати предмет)); Тригери (активувати подію, пов'язану з тригером (наприклад, запустити пастку, перейти на наступний рівень)).

5) Вирішення зіткнень: у разі виявлення зіткнення необхідно змінити позицію або швидкість об'єктів, щоб вони не перетиналися; можна використовувати різні методи вирішення зіткнень, такі як відштовхування об'єктів, переміщення їх на мінімальну відстань, зміна напрямку руху тощо.

Запропоновані алгоритми детально описують структуру та логіку програмної реалізації, що сприятиме успішній розробці комп'ютерної гри "The

Cave of Evil". Чітке визначення кроків алгоритму, опис методів та структур даних, а також візуалізація у вигляді блок-схем дозволяють отримати повне уявлення про функціонування програмного модуля та його взаємодію з іншими компонентами гри. Це, у свою чергу, полегшить процес написання коду, тестування та налагодження, що зрештою призведе до створення якісного та захоплюючого ігрового продукту.

2.4 Висновок до розділу 2

У другому розділі було проведено детальну розробку та проектування програмного модуля для відеогри "The Cave of Evil". Було розглянуто різні методи створення ігрової механіки, такі як процедурна генерація, скриптування та модульний дизайн, та обґрунтовано вибір комбінованого підходу, що поєднує переваги кожного з них.

Було розроблено архітектуру програмного модуля, яка включає такі компоненти, як ядро гри, рендерер, звуковий двигун, фізичний двигун. Кожен компонент має свою чітко визначену функціональність та відповідає за певний аспект гри.

Було розроблено алгоритм функціонування програмного модуля, який описує основні етапи роботи гри, такі як ініціалізація, основний ігровий цикл, обробка подій, збереження та завантаження, завершення гри. Також було детально описано алгоритм зіткнення об'єктів, що є важливим аспектом реалістичної ігрової механіки.

Таким чином, у другому розділі було закладено міцний фундамент для подальшої реалізації відеогри "The Cave of Evil". Розроблена архітектура, алгоритми та вибір інструментів дозволять створити якісний та захоплюючий ігровий продукт, який відповідатиме сучасним стандартам та задовольнятиме потреби гравців.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО МОДУЛЮ КОМП'ЮТЕРНОЇ ГРИ «THE CAVE OF EVIL»

3.1 Обґрунтування вибору рушія для програмної реалізації

Розробка програмного забезпечення, особливо ігор, вимагає ретельного вибору середовища розробки. Існує безліч ігрових рушіїв, кожен зі своїми перевагами та недоліками. Деякі з них спеціалізуються на 2D-графіці (Construct 2, Corona, Cocos2D-X), інші на 3D (Unreal Engine, Unity 5, CryEngine). Важливим фактором є також вартість використання: деякі рушії безкоштовні, тоді як інші вимагають щомісячної або щорічної підписки.

Для полегшення вибору запропоновано розглянути порівняльну таблицю (табл. 3.1) з коротким описом плюсів та мінусів популярних рушіїв CryEngine, Cocos2D-x, Unity 3D та Unreal Engine.

Таблиця 3.1 - Переваги та недоліки рушіїв Unity 3D, Unreal Engine, CryEngine та Cocos2D–x

	CryEngine	Cocos2D–x	Unity3D	Unreal Engine
	Переваги			
1	GameSDK (бібліотека Google для ігор)	Безкоштовний	Багато користувачів	Візуальне програмування
2	realtime render	Низький поріг входження	Гарна документація	Мультизадачність та універсальність
3	Графіка	швидкість	Кросплатформ ість	Кросплатформість
4		Декілька мов програмуванн я підтримує	Низький поріг входження	Гарана візуалізація

Продовження таблиці 3.1

	Недоліки			
5	Стара документація	Тільки 2д ігри	Оптимізація	Велика ціна на контент
6	Мало користувачів		Руйнується білд через встановлення різних плагінів	Мало універсального контенту
7	Маленький магазин ассетів		Часто ігри на телефони багато займають місця	Навантаження на ПК
8	Складний процес складання білда			Високий поріг входження

Unity 3D — це чудовий вибір для тих, хто хоче створювати ігри, але не має глибоких технічних знань. Його простота використання, багата бібліотека ресурсів та потужні можливості роблять його ідеальним інструментом для розробників різного рівня.

Unity 3D, популярний ігровий движок, також підтримує MVC (Модель-Вид-Контролер) [20]. Це допомагає розробникам ігор створювати добре структуровані проекти, де логіка гри, візуальне представлення та взаємодія з користувачем чітко розділені. Такий підхід сприяє створенню якісних, масштабованих та легко підтримуваних ігор.

Для створення якісних, зрозумілих та красивих програм важливо дотримуватися архітектурних принципів. Один з найпоширеніших шаблонів - MVC. Він розділяє програму на три взаємопов'язані компоненти:

1. Модель: відповідає за дані та логіку програми. Вона зберігає та обробляє інформацію, не залежно від того, як вона відображається користувачеві.

2. Вид: відповідає за візуальне представлення даних користувачеві. Він отримує дані з моделі та відображає їх у зручному для користувача форматі.

3. Контролер: відповідає за взаємодію між моделлю та видом. Він обробляє дії користувача, оновлює модель та повідомляє виду про необхідність оновлення відображення.

MVC дозволяє розробникам розділити відповідальність між різними компонентами програми, що робить код більш структурованим, зрозумілим та легким у підтримці.

C# – потужний інструмент для створення 2D платформерів на Unity 3D. Його тісна інтеграція з рушієм забезпечує безпосередній доступ до API Unity, спрощуючи взаємодію з усіма компонентами гри. Завдяки компіляції Just-In-Time та вбудованим оптимізаціям, C# гарантує високу продуктивність гри навіть на мобільних пристроях.

C-подібний синтаксис мови, знайомий багатьом розробникам, разом з об'єктно-орієнтованим підходом, робить C# легким для вивчення та використання. Величезна спільнота розробників та велика кількість доступних ресурсів, включаючи документацію, підручники та онлайн-курси, забезпечують всебічну підтримку на всіх етапах розробки.

Unity Build Pipeline та наявність кроссплатформних бібліотек дозволяють розробникам створювати ігри для різних платформ, таких як Windows, macOS, Linux, Android, iOS та ігрові консолі, використовуючи єдину кодову базу на C#. Це значно скорочує час та зусилля, необхідні для портування гри на різні пристрої.

Вибір C# для розробки 2D платформера на Unity 3D – це запорука створення високоякісної, захоплюючої та доступної гри для широкої аудиторії. Мова поєднує в собі продуктивність, зручність використання та широкі можливості, що робить її ідеальним вибором для розробки ігор.

3.2 Програмна реалізація модулю комп’ютерної гри

Рух персонажа у грі "The Cave of Evil" є ключовим елементом геймплею, що забезпечує взаємодію гравця з ігровим світом та проходження рівнів. У цьому підрозділі буде детально розглянуто реалізацію руху персонажа, включаючи опис основних скриптів, компонентів та методів, що використовуються для досягнення плавного, реалістичного та зручного керування.

Основна логіка руху персонажа реалізована у скрипті Hero.cs. Цей скрипт містить змінні, що визначають характеристики руху, такі як швидкість (speed), сила стрибка (jumpForce) та стан персонажа (State):

```
 [SerializeField] private float speed = 3f;
[SerializeField] private float jumpForce = 15f;
private States State
{
   get { return (States)anim.GetInteger("state"); }
   set { anim.SetInteger("state", (int)value); }
}.
```

Методи Run та Jump відповідають за реалізацію бігу та стрибка відповідно. Метод Run зчитує вхідні дані від гравця (горизонтальна вісь), змінює позицію персонажа у відповідному напрямку та перевертає спрайт персонажа, якщо він рухається вліво:

```
private void Run()
{
   if (isGrounded) State = States.run;
   Vector3 dir = transform.right * Input.GetAxis("Horizontal");
   transform.position = Vector3.MoveTowards(transform.position, transform.position + dir, speed * Time.deltaTime);
   sprite.flipX = dir.x < 0.0f;
}.
```

Метод Jump додає силу до компонента Rigidbody2D персонажа, що призводить до стрибка.

```
private void Jump()
{
   rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
}.
```

Для реалізації фізики руху використовується компонент Rigidbody2D, який дозволяє застосовувати сили та контролювати рух персонажа відповідно до законів фізики. Компонент Animator відповідає за відтворення анімацій персонажа, таких як біг, стрибок, атака та очікування. Перемикання між анімаціями відбувається залежно від стану персонажа, який встановлюється у скрипті Hero.cs.

Метод CheckGround використовується для визначення, чи знаходиться персонаж на землі. Це важливо для коректної роботи стрибка та інших механік, які залежать від стану персонажа. Метод перевіряє, чи є колайдери під ногами персонажа, і встановлює значення змінної isGrounded відповідно:

```
private void CheckGround()
{
   Collider2D[] collider = Physics2D.OverlapCircleAll(transform.position, 0.3f);
   isGrounded = collider.Length > 1;

   if (!isGrounded) State = States.jump;
}.
```

Обробка зіткнень (колізій) є важливим аспектом ігрової механіки, що забезпечує реалістичну взаємодію об'єктів у віртуальному світі. У грі "The Cave of Evil" зіткнення використовуються для реалізації різноманітних ігрових подій, таких як отримання шкоди персонажем, активація пасток, збір предметів тощо.

Базовий клас Entity є основою для всіх об'єктів у грі, які можуть

взаємодіяти один з одним через зіткнення. Він містить такі ключові елементи:

1. Змінна lives: Зберігає кількість життів об'єкта.

2. Метод GetDamage(): Викликається при отриманні об'єктом пошкодження. Зменшує кількість життів на одиницю та викликає метод Die(), якщо кількість життів досягла нуля.

3. Метод Die(): Викликається при смерті об'єкта. Зазвичай призводить до знищення об'єкта або виконання інших дій, пов'язаних зі смертю:

```
protected int lives;
public virtual void GetDamage()
{
   lives--;
   if (lives < 1)
   {
      Die();
   }
}
public virtual void Die()
{
   Destroy(this.gameObject);
}.
```

Скрипти Obstacle.cs (перешкоди) та Berserk.cs (ворог) наслідуються від базового класу Entity та перевизначають метод OnCollisionEnter2D. Цей метод викликається при зіткненні об'єкта з іншим об'єктом. У випадку зіткнення з персонажем гравця, скрипти викликають метод GetDamage() у об'єкта персонажа, завдаючи йому шкоди:

```
private void OnCollisionEnter2D(Collision2D collision)
{
   if (collision.gameObject == Hero.Instance.gameObject)
   {
      Hero.Instance.GetDamage();
```

```
    }
}.
```

Скрипт Hero.cs, який керує персонажем гравця, також наслідується від класу Entity та перевизначає метод GetDamage(). У цьому методі зменшується кількість життів персонажа, і, якщо кількість життів досягла нуля, викликається метод Die(), що призводить до завершення гри або інших дій, пов'язаних зі смертю персонажа.

Бойова система у грі "The Cave of Evil" є важливою складовою геймплею, що надає гравцеві можливість взаємодіяти з ворогами та перешкодами, використовуючи різноманітні атаки та захисні маневри. Розглянемо детальніше реалізацію бойової системи, зосередившись на основних скриптах, методах та змінних, що відповідають за її функціональність.

Основна логіка бойової системи знаходиться у скрипті Hero.cs, який керує діями персонажа гравця. Ключові методи та змінні, що стосуються бою:

1. isAttacking: Логічна змінна, що визначає, чи персонаж знаходиться в стані атаки.

2. isRecharged: Логічна змінна, що вказує на готовність персонажа до наступної атаки.

3. attackPos: Трансформ, що визначає позицію, з якої відбувається атака.

4. attackRange: Радіус зони ураження атаки.

5. enemy: LayerMask, що визначає шар, на якому знаходяться вороги.

6. Attack(): Метод, що відповідає за ініціалізацію атаки. Він перевіряє, чи персонаж знаходиться на землі та чи готовий до атаки, і якщо так, то встановлює стан атаки (States.attack), запускає корутини AttackAnimation() та AttackCoolDown().

7. OnAttack(): Метод, що викликається під час анімації атаки. Він перевіряє, чи є вороги в зоні ураження атаки, та завдає їм шкоди за допомогою методу GetDamage().

8. AttackAnimation(): Корутина, що відповідає за тривалість анімації

атаки. Після завершення анімації встановлює змінну isAttacking у false.

9. AttackCoolDown(): Корутина, що відповідає за час перезарядки атаки. Після завершення перезарядки встановлює змінну isRecharged у true:

```
private void Attack()
{
   if (isGrounded && isRecharged)
   {
      State = States.attack;
      isAttacking = true;
      isRecharged = false;
      StartCoroutine(AttackAnimation());
      StartCoroutine(AttackCoolDown());
   }
}.
```

Метод GetDamage() у скрипті Entity.cs відповідає за отримання пошкоджень об'єктом. Він зменшує кількість життів об'єкта та викликає метод Die(), якщо кількість життів досягла нуля.

Скрипт Berserk.cs, що керує поведінкою ворога, також містить метод OnCollisionEnter2D, який викликається при зіткненні з персонажем гравця. У цьому методі ворог завдає шкоди персонажу, викликаючи метод GetDamage() у об'єкта персонажа.

Компонент камери у грі "The Cave of Evil" відіграє важливу роль у забезпеченні комфортного візуального сприйняття ігрового процесу. Він відповідає за позиціонування віртуальної камери таким чином, щоб гравець завжди бачив головного героя в центрі екрана, забезпечуючи плавне стеження за його рухами.

Основна логіка роботи камери реалізована у скрипті Camera.cs. Цей скрипт містить:

1. Посилання на об'єкт гравця (player): Використовується для отримання поточної позиції героя.

1. Змінна pos: Тимчасова змінна для зберігання розрахованої позиції камери.
2. Awake(): Метод, який викликається при старті гри. Якщо посилання на об'єкт гравця не задано, скрипт автоматично знаходить його за допомогою методу FindObjectOfType<Hero>().
3. Update(): Метод, який викликається щокадру. Він отримує поточну позицію гравця, коригує її (додає 1 по осі Y та встановлює -10 по осі Z для створення перспективи), а потім плавно переміщує камеру до цієї позиції за допомогою методу Vector3.Lerp.

   pos = player.position;

   pos.z = -10f;

   pos.y += 1f.

   transform.position = Vector3.Lerp(transform.position, pos, Time.deltaTime);

Використання методу Vector3.Lerp дозволяє створити плавне переміщення камери, що робить ігровий процес більш комфортним для гравця.

У грі "The Cave of Evil" використовується архітектурний шаблон MVC (Модель-Вид-Контролер), який є поширеним підходом до розробки програмного забезпечення, включаючи ігри. MVC розділяє логіку програми на три основні компоненти:

1. Модель: Відповідає за дані та логіку гри. У грі "The Cave of Evil" модель представлена скриптом Entity.cs, який містить дані про стан ігрових об'єктів (здоров'я, швидкість тощо) та методи для зміни цього стану (отримання пошкоджень, смерть).

2. Вид: Відповідає за візуальне представлення даних. У грі "The Cave of Evil" вид представлений спрайтами та анімаціями персонажів та об'єктів.

3. Контролер: Відповідає за обробку вхідних даних від користувача, оновлення моделі та взаємодію з видом для відображення змін. У грі "The Cave of Evil" контролери представлені скриптами Hero.cs, Berserk.cs, Obstacle.cs.

Використання MVC дозволяє розробникам розділити відповідальність між різними компонентами гри, що робить код більш структурованим,

зрозумілим та легким у підтримці. Наприклад, зміна візуального вигляду персонажа (виду) не вплине на логіку його поведінки (модель), оскільки ці компоненти розділені. Це також спрощує повторне використання коду та полегшує внесення змін до гри, оскільки зміни в одному компоненті не впливають на інші компоненти.

3.3 Тестування програмного модулю комп’ютерної гри

При тестуванні програмного модулю комп’ютерної гри необхідно перевірити виконання всіх ігрових механік, що були спрокетовані та розроблені. Також варто перевірити рівень оптимізації комп’ютерної графіки, а також графічний інтерфейс відеогри.

Тестування розпочато з відкриття Unity та запуск проекту, що є розробленим програмним модулем.

При відкритті проекту з’являється головне меню комп’ютерної гри «The cave of evil», що зображено на рисунку 3.4.

Рисунок 3.4 – Загальний вигляд інтерфейсного вікна типу «Головне меню» комп’ютерної гри «The cave of evil»

Після натискання на середню кнопку, що означає «Грати», відкривається перша локація комп’ютерної гри (рис. 3.5).

Рисунок 3.5 – Загальний вигляд інтерфейсного вікна типу «Локація» комп’ютерної гри «The cave of evil»

Далі протестовано основні механіки комп’ютерної гри, а саме здатність персонажа перепригувати прірви та перешкоди на ігровій локації (рис. 3.6), здатність персонажа завадання шкоди ворогам (рис. 3.7).

Рисунок 3.6 – Загальний вигляд інтерфейсного вікна типу «Зображення персонажа, що перепригує перешкоду»

Рисунок 3.7 – Загальний вигляд інтерфейсного вікна типу «Зображення персонажа, що завдає шкоду ворогу»

Згідно з проведеними тестами, комп’ютерна гра «The cave of evil» виконує всі функції, що зазначалися в проектуванні програмного модуля.

3.4 Висновок до розділу 3

У даному розділі було проаналізовано популярні рушії для створення відеоігор такі як: Unreal Engine, Cry Engine, Unity 3D та Cocos2D–x. Серед них було визначено переваги та недоліки. Було обрано найкращий варіант для розробки комп’ютерної гри у жанрі 2D-платформер.

Обраними програмними засобами було реалізвано програмний модуль комп’ютерної гри «The cave of evil».

Було протестовано програмний модуль комп’ютерної гри «The Cave of Evil».

ВИСНОВКИ

Поставлені перед бакалаврською дипломною роботою роботою задачі виконано в повному обсязі, а саме:

1. Проведено аналіз сучасної ігрової індустрії та класифікації комп’ютерних відеоігор

2. Провдено аналіз проблеми графічного дизайну у комп’ютерних іграх у жанрі 2D платформер .

3. Розроблено архітектуру програмного модуля комп’ютерної гри.

4. Розроблено ігрову механіку комп’ютерної гри «The Cave of Evil».

5. Розроблено алгоритм функціонування програмного модулю комп’ютерної гри.

6. Здійснено програмну реалізацію програмного модулю комп’ютерної гри.

7. Протестовано програмний модуль комп’ютерної гри.

8. Розроблено інструкцію користувача програмного модулю комп’ютерної гри «The Cave of Evil».

У ході виконання дипломної роботи було досліджено сучасний стан ігрової індустрії, виявлено тенденції та особливості розвитку жанру 2D платформерів. Проаналізовано різні підходи до графічного дизайну та обрано оптимальний варіант для гри "The Cave of Evil".

Створена архітектура програмного модуля забезпечує його гнучкість, масштабованість та можливість подальшого розвитку. Реалізована ігрова механіка надає гравцям різноманітні можливості для взаємодії з ігровим світом та досягнення поставлених цілей. Розроблений алгоритм функціонування забезпечує логічну послідовність ігрових подій та коректну роботу всіх компонентів гри.

Програмна реалізація модуля виконана з урахуванням сучасних вимог до якості та продуктивності програмного забезпечення. Проведене тестування підтвердило працездатність та стабільність гри. Розроблена інструкція

користувача дозволить гравцям швидко освоїти гру та отримати від неї максимальне задоволення.

Розроблено функціональний прототип 2D гри згідно з визначеними вимогами. Гра підтримує одного гравця, має роздільну здатність 1920x1080 пікселів, частоту кадрів 60 кадрів за секунду, карту розміром 10000x10000 пікселів та містить 5 рівнів з 3 видами ворогів.

Мета роботи – реалізація дозвілля людини на основі розробленої комп'ютерної гри "The Cave of Evil", яка дозволяє нормалізувати психоемоційний стан і підвищити інтелектуальний рівень людини – досягнута. Розроблена гра має потенціал для використання в освітніх та розважальних цілях, а також для розвитку когнітивних здібностей гравців.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Valve Software [Електронний ресурс]: Режим доступу: https://developer.valvesoftware.com/wiki/Dimensions.

2. Gartner. Gamification [Електронний ресурс]: Режим доступу: https://www.gartner.com/en/information-technology/glossary/gamification.

3. The Top 50 Genres by jeff [Електронний ресурс]: Режим доступу: https://www.giantbomb.com/profile/jeff/lists/the-top-50-genres/3066/.

4. ScienceDirect. Two-Dimensional Computer Graphics [Електронний ресурс]: Режим доступу: https://www.sciencedirect.com/topics/computer-science/two-dimensional-computer-graphics.

5. Techopedia. Two-Dimensional (2-D) Graphics [Електронний ресурс]: Режим доступу: https://www.techopedia.com/definition/1786/two-dimensional-2-d-graphics.

6. Lifewire. Color Models in Graphics [Електронний ресурс]: Режим доступу: https://www.lifewire.com/color-models-in-graphics-3466875.

7. Mozilla. 2D parallax scrolling [Електронний ресурс]: Режим доступу: https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_parallax_scrolling.

8. PwC. Video games trends [Електронний ресурс]: Режим доступу: https://www.pwc.com/gx/en/industries/technology-media-and-telecommunications/technology/video-games-trends.html.

9. Steam. Platformer [Електронний ресурс]: Режим доступу: https://store.steampowered.com/genre/Platformer/.

10. Hollow Knight [Електронний ресурс]: Режим доступу: https://www.hollowknight.com/.

11. Dead Cells [Електронний ресурс]: Режим доступу: https://dead-cells.com/.

12. Steam. Mark of the Ninja [Електронний ресурс]: Режим доступу: https://store.steampowered.com/app/214370/Mark_of_the_Ninja/.

13. Game Developer. The 5 Most Common Game Mechanics [Електронний

ресурс]: Режим доступу: https://www.gamedeveloper.com/design/the-5-most-common-game-mechanics.

14. Game Programming Patterns. Game Loop [Електронний ресурс]: Режим доступу: https://gameprogrammingpatterns.com/game-loop.html.

15.Toptal. Video Game Physics, Part I: An Introduction to Rigid Body Dynamics [Електронний ресурс]: Режим доступу: https://www.toptal.com/game/video-game-physics-part-i-an-introduction-to-rigid-body-dynamics.

16. Unity. 2D games [Електронний ресурс]: Режим доступу: https://unity.com/solutions/2d.

17. Microsoft. C# documentation [Електронний ресурс]: Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/.

18. Cry Engine [Електронний ресурс]: Режим доступу: https://www.cryengine.com/4.

19. Unreal Engine [Електронний ресурс]: Режим доступу: https://www.unrealengine.com/en-US.

20. MVC [Електронний ресурс]: Режим доступу: https://ru.wikipedia.org/wiki/Model-View-Controller.

21. C# [Електронний ресурс]: Режим доступу: https://learn.microsoft.com/ru-ru/dotnet/csharp/.

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний модуль комп'ютерної гри «The Cave of Evil»

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 95,94% Схожість 4,06%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Озеранський В.С.

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

Автор роботи _____ Ярмощук Д.Р.

Керівник роботи _____ Іванчук Я.В.

ДОДАТОК Б

Інструкція користувача

1. Відкриваємо рушій Unity.
2. Запускаємо комп’ютерну гру натискаючи на проект з назвою «The cave of evil».

Рисунок Б.1 – Загальний вигляд інтерфейсного вікна типу «Головне меню» комп’ютерної гри «The cave of evil»

3. Натискаємо на середні кнопку «Грати», щоб запустити гру.

Рисунок Б.2 – Загальний вигляд інтерфейсного вікна типу «Локація» комп’ютерної гри «The cave of evil»

4. За допомогою «A», «D» персонаж переміщається вліво та вправо відповідно, «Пробіл» відповідає за стрибок персонажа, «ЛКМ» відповідає за нанесення удару.

ДОДАТОК В

Лістинг програми

Hero.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Data.SqlTypes;
using System.Xml.Serialization;
using UnityEngine;
public class Hero : MonoBehaviour
{
    [SerializeField] private float speed = 3f;
    [SerializeField] private int lives = 5;
    [SerializeField] private float jumpForce = 15f;
    private bool isGrounded = false;

    public bool isAttacking = false;
    public bool isRecharged = true;

    public Transform attackPos;
    public float attackRange;
    public LayerMask enemy;

    private Rigidbody2D rb;
    private Animator anim;
    private SpriteRenderer sprite;

    public static Hero Instance { get; set; }
    private States State
    {
        get { return (States) anim.GetInteger("state"); }
        set { anim.SetInteger("state", (int)value); }
    }
    private void Awake()
    {
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        sprite = GetComponentInChildren<SpriteRenderer>();
        Instance = this;
        isRecharged = true;
    }

    private void FixedUpdate()
```

```
    {
        CheckGround();
    }
    private void Update()
    {
        if (!isAttacking && isGrounded) State = States.idle;
        if (!isAttacking && Input.GetButton("Horizontal"))
            Run();
        if (!isAttacking && isGrounded && Input.GetButtonDown("Jump"))
            Jump();
        if (Input.GetButtonDown("Fire1"))
            Attack();
    }
    private void Run()
    {
        if (isGrounded) State = States.run;

        Vector3 dir = transform.right * Input.GetAxis("Horizontal");

        transform.position = Vector3.MoveTowards(transform.position,
transform.position + dir, speed * Time.deltaTime);

        sprite.flipX = dir.x < 0.0f;
    }

    private void Jump()
    {
        rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
    }

    private void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(attackPos.position, attackRange);
    }

    private void CheckGround()
    {
        Collider2D[] collider = Physics2D.OverlapCircleAll(transform.position, 0.3f);
        isGrounded = collider.Length > 1;

        if (!isGrounded) State = States.jump;
    }
```

```
    public void GetDamage()
    {
        lives -= 1;
        Debug.Log(lives);
    }

    private void Attack()
    {
        if (isGrounded && isRecharged)
        {
            State = States.attack;
            isAttacking = true;
            isRecharged = false;

            StartCoroutine(AttackAnimation());
            StartCoroutine(AttackCoolDown());
        }
    }

    private void OnAttack()
    {
        Collider2D[] colliders = Physics2D.OverlapCircleAll(attackPos.position,
attackRange, enemy);

        for (int i = 0; i < colliders.Length; i++)
        {
            colliders[i].GetComponent<Entity>().GetDamage();
        }
    }

    private IEnumerator AttackAnimation()
    {
        yield return new WaitForSeconds(0.4f);
        isAttacking = false;
    }

    private IEnumerator AttackCoolDown()
    {
        yield return new WaitForSeconds(0.5f);
        isRecharged = true;
    }

}
```

```
public enum States
{
   idle,
   run,
   jump,
   attack
}
```

Entity.cs

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

public class Entity : MonoBehaviour
{
   protected int lives;
   public virtual void GetDamage()
   {
      lives--;
      if (lives < 1 )
      {
         Die();
      }

   }

   public virtual void Die()
   {
      Destroy(this.gameObject);
   }
}
```

Camera.cs

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

public class Camera : MonoBehaviour
{
   [SerializeField] private Transform player;
   private Vector3 pos;

   private void Awake()
   {
```

```
        if (!player)
            player = FindObjectOfType<Hero>().transform;
    }

    private void Update()
    {
        pos = player.position;
        pos.z = -10f;
        pos.y += 1f;
        transform.position = Vector3.Lerp(transform.position, pos, Time.deltaTime);
    }
}
```

Berserk.cs

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

public class Berserk : Entity
{
    private float speed = 1.5f;
    private Vector3 direction;
    private SpriteRenderer sprite;

    private void Awake()
    {
        sprite = GetComponentInChildren<SpriteRenderer>();
    }

    private void Start()
    {
        direction = transform.right;
        lives = 3;
    }

    private void Move()
    {
        Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position +
transform.up * 0.1f + transform.right * direction.x * 0.7f, 0.1f);

        if (colliders.Length > 0) direction *= -1f;
        transform.position = Vector3.MoveTowards(transform.position,
transform.position + direction, Time.deltaTime);
        sprite.flipX = direction.x < 0.0f;
```

```
    }

    private void Update()
    {
        Move();
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject == Hero.Instance.gameObject)
        {
            Hero.Instance.GetDamage();
        }
    }
}
```

Entity.cs

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

public class Entity : MonoBehaviour
{
    protected int lives;
    public virtual void GetDamage()
    {
        lives--;
        if (lives < 1 )
        {
            Die();
        }

    }

    public virtual void Die()
    {
        Destroy(this.gameObject);
    }
}
```

Camera.cs

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

```
public class Camera : MonoBehaviour
{
   [SerializeField] private Transform player;
   private Vector3 pos;

   private void Awake()
   {
      if (!player)
         player = FindObjectOfType<Hero>().transform;
   }

   private void Update()
   {
      pos = player.position;
      pos.z = -10f;
      pos.y += 1f;
      transform.position = Vector3.Lerp(transform.position, pos, Time.deltaTime);
   }
}
```

Berserk.cs

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

public class Berserk : Entity
{
   private float speed = 1.5f;
   private Vector3 direction;
   private SpriteRenderer sprite;

   private void Awake()
   {
      sprite = GetComponentInChildren<SpriteRenderer>();
   }

   private void Start()
   {
      direction = transform.right;
      lives = 3;
   }

   private void Move()
```

```
    {
        Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position +
transform.up * 0.1f + transform.right * direction.x * 0.7f, 0.1f);

        if (colliders.Length > 0) direction *= -1f;
        transform.position = Vector3.MoveTowards(transform.position,
transform.position + direction, Time.deltaTime);
        sprite.flipX = direction.x < 0.0f;

    }

    private void Update()
    {
        Move();
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject == Hero.Instance.gameObject)
        {
            Hero.Instance.GetDamage();
        }
    }
}
```

Obstacle.cs

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

public class Obstacle : MonoBehaviour
{
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject == Hero.Instance.gameObject)
        {
            Hero.Instance.GetDamage();
        }
    }
}
```

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ КОМП'ЮТЕРНОЇ ГРИ «THE CAVE OF EVIL»

Виконав: студент 4 курсу, групи 2КН-20б
Спеціальності 122 – Комп'ютерні науки
(шифр і назва напрямку підготовки, спеціальності)
Ярмощук Д.Р.
(прізвище та ініціали)
Керівник: д.т.н., проф. каф. КН
Іванчук Я. В.
(прізвище та ініціали)

«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік

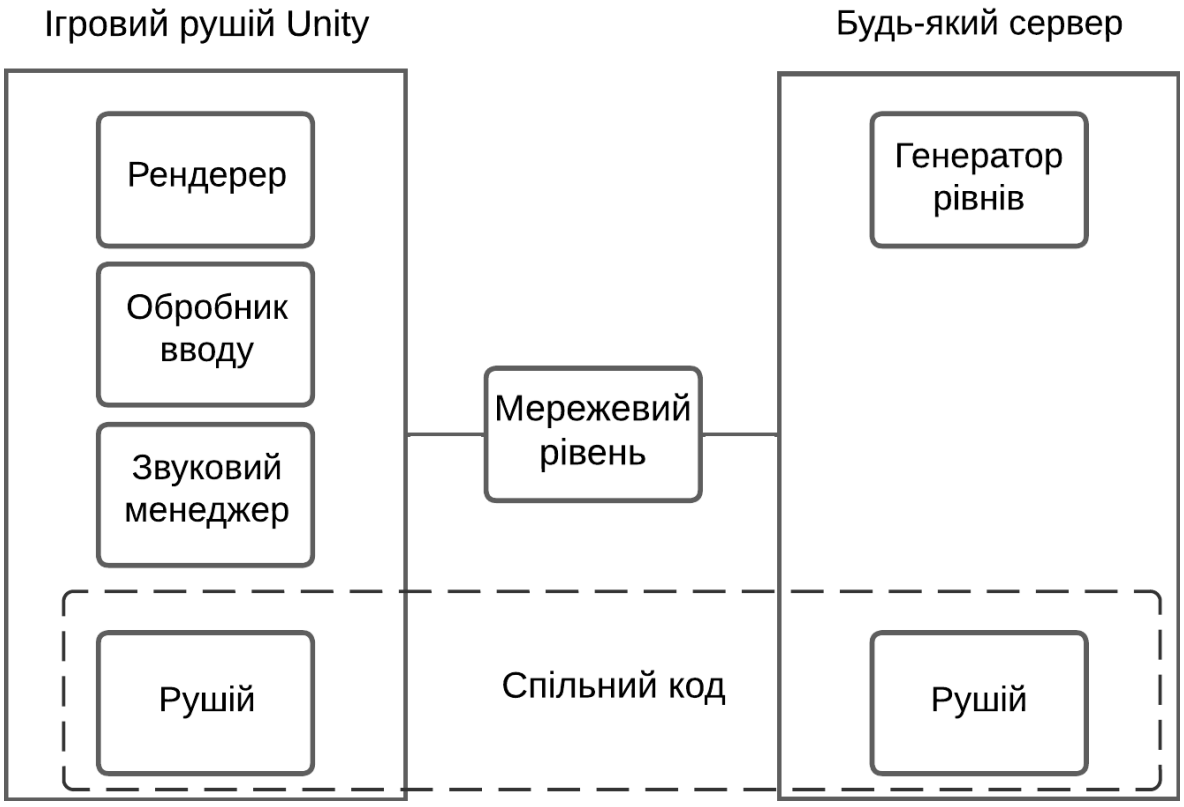

Рисунок Г.1 – Діаграма основних компонентів архітектури програмного модулю відеогри «The Cave of Evil»

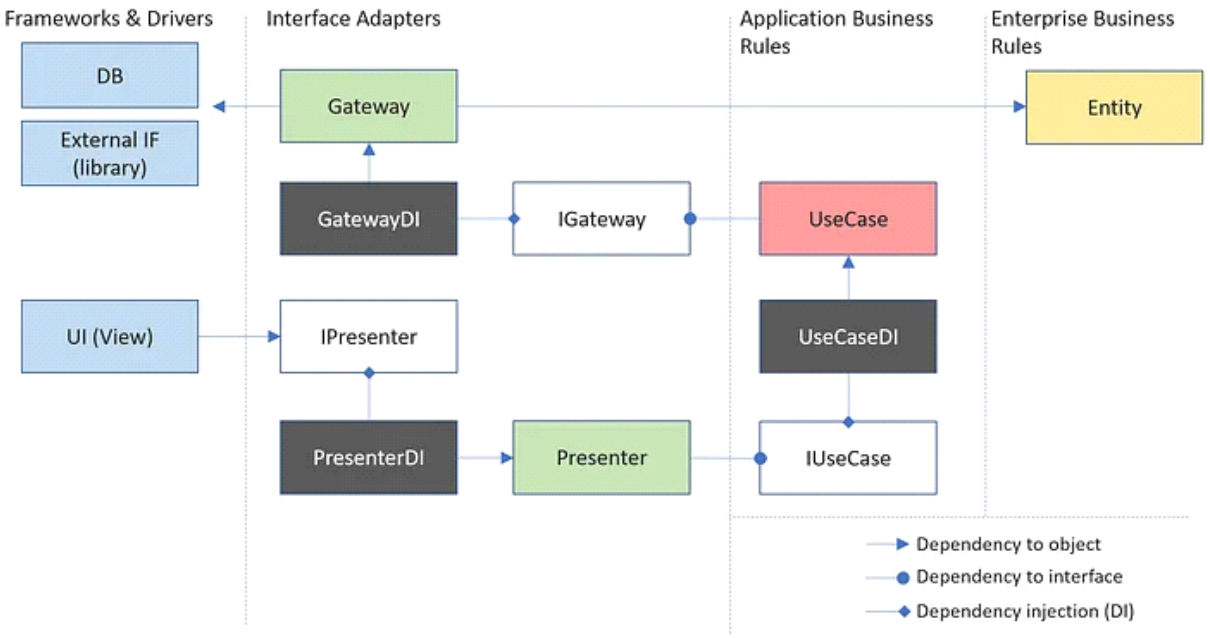

Рисунок Г.2– Діаграма принципу взаємодії різних шарів архітектури

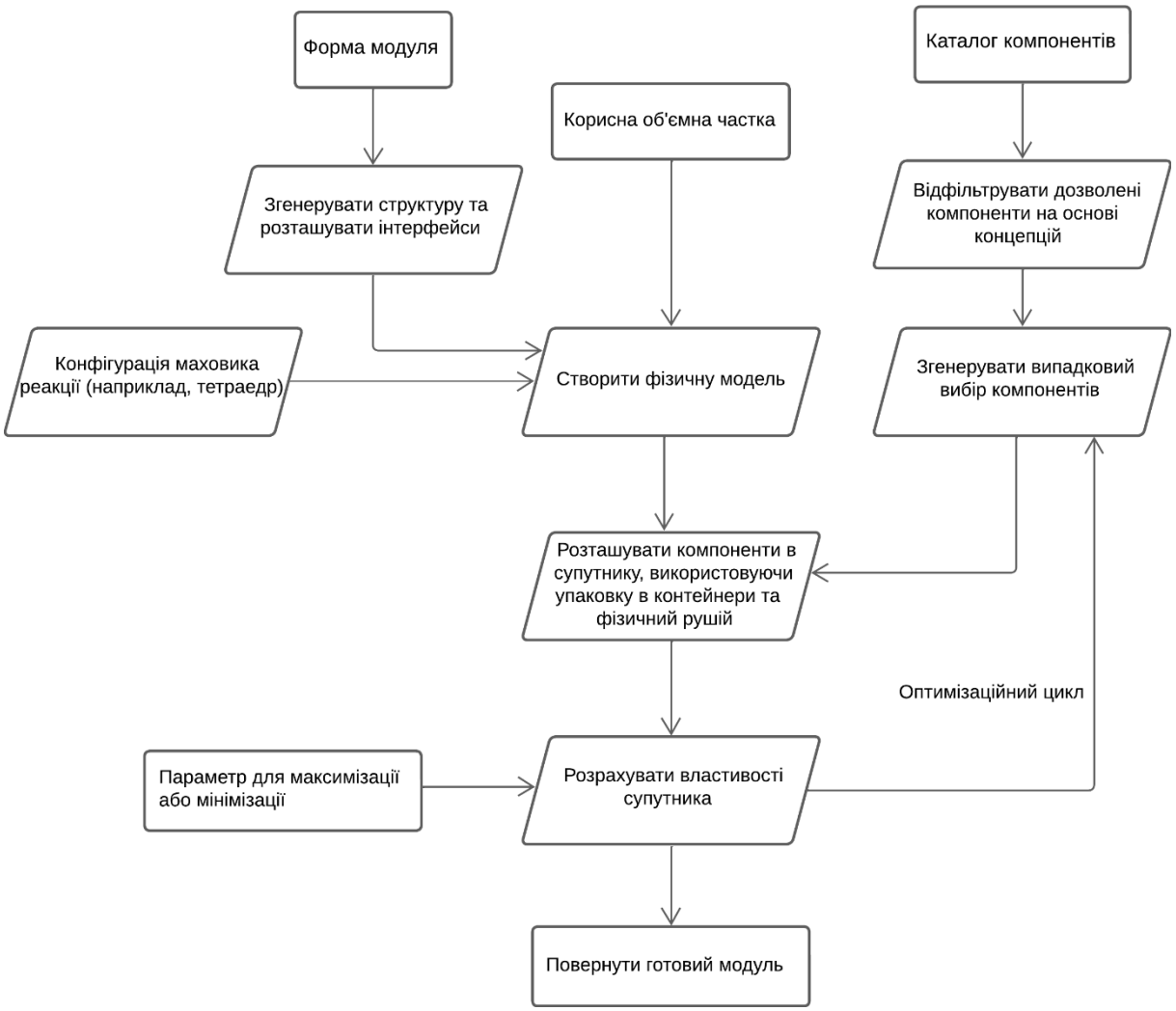

Рисунок Г.3 – Діаграма процедури генерації для створення ландшафту

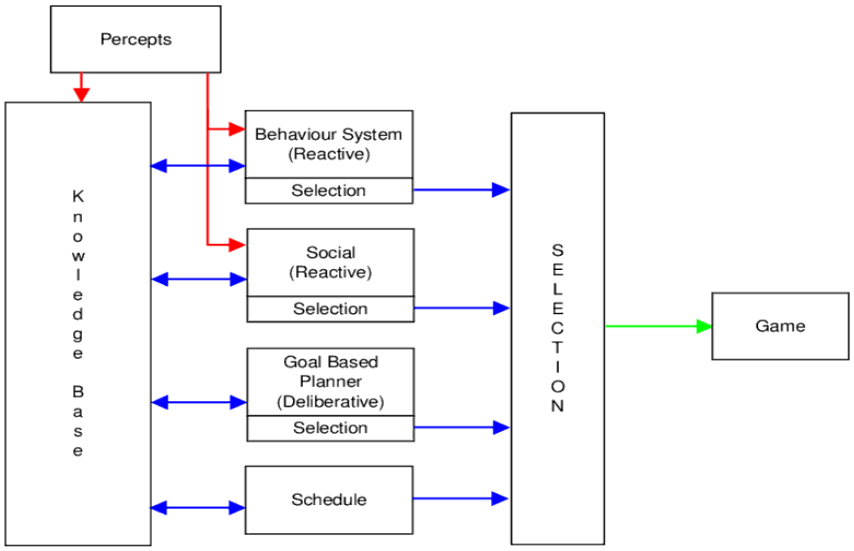

Рисунок Г.4– Діаграма модульного дизайну у розробці відеоігор

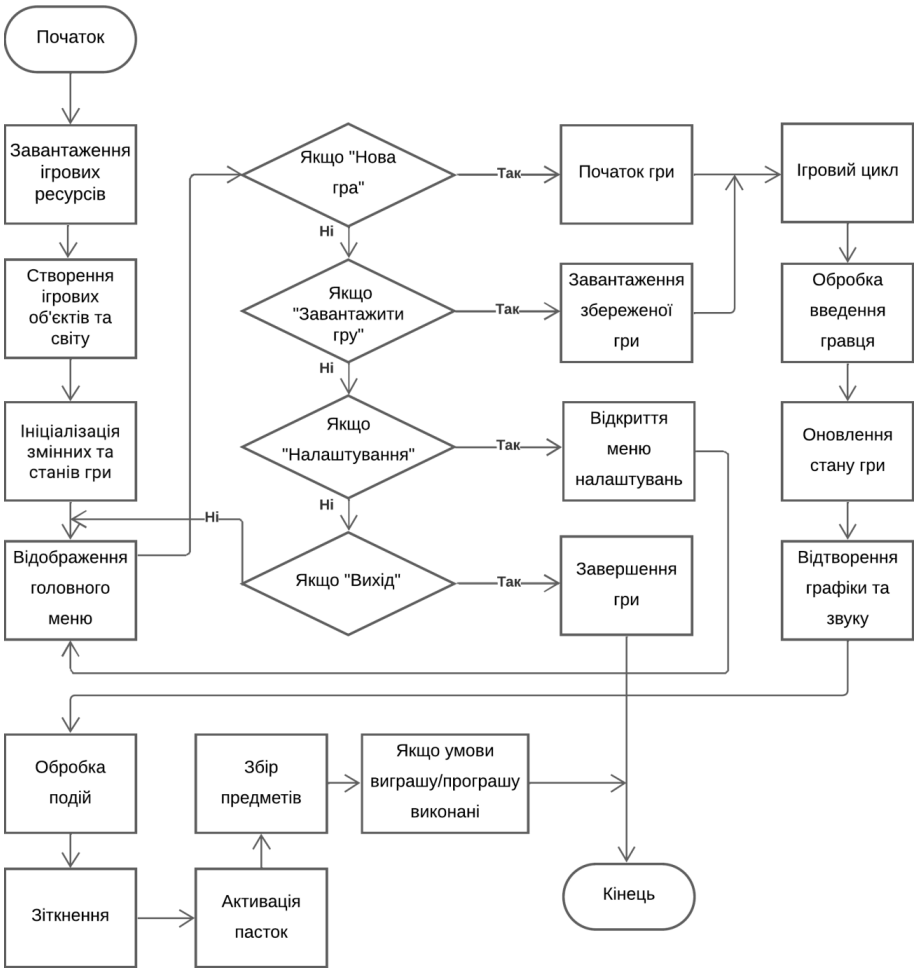

Рисунок Г.5 – Схема алгоритму функціонування програмного модулю комп’ютерної гри «The Cave of Evil»

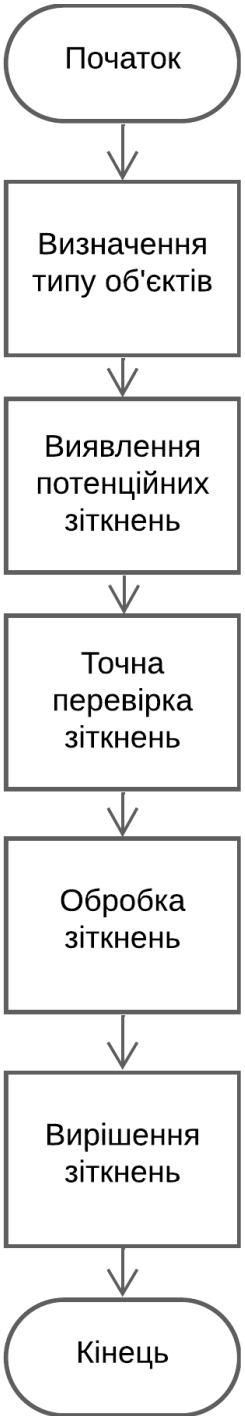

Рисунок Г.6 – Схема алгоритму зіткнень об’єктів в комп’ютерні грі "The Cave of Evil"

Рисунок Г.7 – Загальний вигляд інтерфейсного вікна типу «Головне меню» комп’ютерної гри «The cave of evil»

Рисунок Г.8 – Загальний вигляд інтерфейсного вікна типу «Локація» комп’ютерної гри «The cave of evil»

Рисунок Г.9 – Загальний вигляд інтерфейсного вікна типу «Зображення персонажа, що перепригує перешкоду»

Рисунок Г.10 – Загальний вигляд інтерфейсного вікна типу «Зображення персонажа, що завдає шкоду ворогу»