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

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ КОМП'ЮТЕРНОЇ ГРИ «Phoenix»

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

Керівник: д. т. н., професор 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 2024р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Нагорному Михайлу Руслановичу

1. Тема роботи: Програмний модуль комп'ютерної гри «Phoenix»
керівник роботи: Іванчук Я. В., д.т.н., 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 – не менше 1 чол.; частота кадрів – 60 кадр./сек; кількість рівнів – 1 од.; кількість типів ворогів – 2 од.; розширення екрану – 1920/1080 пікселів.
4. Зміст текстової частини (перелік питань, які потрібно розробити):
аналіз жанрів комп'ютерних ігор; аналіз рушіїв комп'ютерних ігор; аналіз комп'ютерних ігор жанру 2D шутер; розробка архітектури комп'ютерної гри; розробка взаємодії функціональних елементів програмного модуля комп'ютерної гри; розробка алгоритму роботи програмного модуля комп'ютерної гри; обґрунтування вибору мови програмування та середовища розробки; програмна реалізація комп'ютерної гри; тестування програмного модуля комп'ютерної гри.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
типова монолітної архітектури комп'ютерної гри; типова схема компонентно-орієнтована архітектури комп'ютерної гри; типова схема архітектури Модель-Вид-Контролер комп'ютерної гри; типова схема архітектури комп'ютерної гри; типова схема взаємодії між функціональними елементами; загальна схема алгоритму роботи циклу програмного модуля; типова схема функціональної моделі персонажу; конструктивна схема користувацького інтерфейсу; конструктивна схема зображення головного меню; загальний вигляд інтерфейсного вікна генерування ігрових сцен типу «карти»; загальний вигляд інтерфейсного вікна типу «Головного меню»; загальний вигляд комп'ютерної гри Phoenix; загальний вигляд закінчення гри при виграші; Загальний вигляд закінчення гри при поразці.

6. Консультанти розділів проекту (роботи)

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

7. Дата видачі завдання 07.03.2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Приміт-ка
		Початок	Закінчення	
1	Аналіз жанрів комп'ютерних ігор	06.05.24	9.05.24	
2	Аналіз рушіїв комп'ютерних ігор	10.05.24	13.05.24	
3	Розробка архітектури комп'ютерної гри	14.05.24	18.05.24	
4	Розробка алгоритму роботи програмного модуля комп'ютерної гри	19.05.24	23.05.24	
5	Обґрунтування вибору мови програмування та середовища розробки	24.05.24	27.05.24	
6	Програмна реалізація комп'ютерної гри	28.05.24	01.06.24	
7	Тестування розробленого модуля	02.06.24	03.06.24	
8	Оформлення матеріалів до захисту БДР	04.06.24	06.06.24	

Студент _____ **Нагорний М. Р.**
(підпис) *(прізвище та ініціали)*

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

АНОТАЦІЯ

Бакалаврська дипломна робота складається зі 78 сторінок формату А4, на яких є 19 рисунків, 2 таблиці, список використаних джерел містить 18 найменувань.

У бакалаврській дипломній роботі запропоновано програмний модуль для комп'ютерної гри «Phoenix», орієнтований на покращення ігрового процесу та взаємодії користувачів з ігровим світом.

Розроблено елементи ігрової механіки та додаткові функції, які реалізовані в демонстраційній версії гри «Phoenix». Програмний продукт було створено з використанням мови програмування C# та ігрового рушія Unity. Для розробки графічних елементів використовувалися інструменти Adobe Photoshop. В якості середовища розробки програмного забезпечення було обрано Visual Studio.

Розроблений модуль гри може бути використаний при створенні сучасних ігрових систем.

Ключові слова: комп'ютерна гра, програмний модуль, Unity, C#.

ABSTRACT

The bachelor thesis consists of 78 pages of A4 format, on which there are 19 figures, 2 tables, the list of used sources contains 18 names.

In the bachelor thesis, we offer a software module for the computer game "Phoenix", aimed at improving the gameplay and user interaction with the game world.

Elements of game mechanics and additional functions implemented in the demo version of the game "Phoenix" have been developed. The software product was created using the C# programming language and the Unity game engine. Adobe Photoshop tools were used to develop graphic elements. Visual Studio was chosen as the software development environment.

The developed game module can be used in the creation of modern game systems.

Keywords: computer game, software module, Unity, C#.

ЗМІСТ

ВСТУП

Актуальність досліджень. Ігри починали як прості аркади та ігрові консолі, але з прогресом технологій стали більш складними та різноманітними. Сьогодні ігрова індустрія охоплює різні платформи, включаючи персональні комп'ютери, консолі, мобільні пристрої, і пропонує широкий спектр жанрів, від екшн-ігор та стратегій до ігор для розв'язання складних завдань або творчості [1]. У сучасному світі індустрія комп'ютерних ігор стрімко розвивається, стаючи однією з найбільш прибуткових галузей розваг. Їх аудиторія налічує мільйони людей різного віку та з різними вподобаннями. За даними Statista, у 2024 році очікується що світовий ринок комп’ютерних ігор досягне доходу в 282,30 мільярдів доларів США, і щорічно буде зростати на 8,76% [2]. Процес створення комп'ютерних ігор включає численні етапи, кожен з яких вимагає спеціалізованих знань і навичок. Від початкової ідеї та концептуального дизайну до програмування, тестування і випуску готового продукту – кожен крок потребує ретельного планування і високого рівня професіоналізму. Важливим аспектом є також інтеграція сучасних технологій, таких як віртуальна та доповнена реальність, штучний інтелект, високопродуктивні графічні процесори, що дозволяє створювати ще більш захоплюючі та реалістичні ігрові світи. 2D шутери, незважаючи на свою відносну простоту, залишаються популярними серед гравців завдяки своїй динамічності, доступності та можливості створювати різноманітні ігрові світи та механіки. Розробка нових ігор у цьому жанрі дозволяє залучити широку аудиторію гравців та сприяє розвитку ігрової індустрії в цілому.

Одним з ключових аспектів успіху комп'ютерної гри є її програмний модуль, який відповідає за реалізацію ігрової логіки, взаємодію з користувачем, графіку, звук та інші важливі елементи. Розробка та оптимізація програмного модуля є складним та багатогранним завданням, що вимагає глибокого розуміння принципів програмування, знання ігрових рушіїв та вміння працювати з різними технологіями.

У цьому контексті, актуальність даної роботи полягає у дослідженні та розробці програмного модуля для 2D шутера "Phoenix", що є популярним жанром комп'ютерних ігор. Розробка такого модуля дозволить не тільки отримати практичний досвід у створенні ігор, але й вивчити сучасні підходи до розробки програмного забезпечення, оптимізації коду та ресурсів, а також реалізації ігрових механік. Крім того, створення власної гри є важливим кроком у розвитку професійних навичок та компетенцій, що може стати основою для подальшої кар'єри у сфері розробки ігор.

Метою дослідження є підвищенню ефективності тренування тактичного мислення користувача за допомогою розробки та використання програмного модуля 2D комп’ютерної гри жанру «шутер» з компонентами розвитку зорової уваги та швидкості прийняття тактичних рішень, яка сприятиме тренуванню швидкості прийняття рішень гравцем в екстремальних умовах.

Об’єктом дослідження є процес розробки програмного модуля комп’ютерної гри «Phoenix».

Предметом дослідження є алгоритми та програмні засоби програмного модуля комп’ютерної гри «Phoenix».

Задачі дослідження:

1. Провести аналіз жанрів комп’ютерних ігор.
2. Провести аналіз рушіїв комп’ютерних ігор.
3. Розробити архітектури комп’ютерної гри.
4. Розробити алгоритм роботи програмного модуля комп’ютерної гри.
5. Розробити програмну реалізацію програмного модуля комп’ютерної гри.
6. Провести тестування та оптимізацію програмного модуля комп’ютерної гри.
7. Розробити інструкцію користувача.

Апробація була проведена на Всеукраїнській науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації (2024) [3].

Публікації: електронні тези науково-технічної конференції «LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)» [3].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз жанрів комп’ютерних ігор

Жанри комп’ютерних ігор відіграють важливу роль у структурі ігрової індустрії, визначаючи характер геймплею, цільову аудиторію та комерційний потенціал кожної гри. Кожен жанр має свої унікальні особливості, механіки та стиль, що приваблює певні групи гравців. Розглянемо основні жанри комп'ютерних ігор, їхні характеристики та приклади популярних ігор у кожному жанрі [4].

Жанр "Екшн" у відеоіграх є одним з найпопулярніших жанрів. Він включає в себе ігри, де гравці мають активно брати участь у подіях, часто за допомогою бойових дій, стрибків, бігу та інших швидких фізичних дій. Основні характеристики екшн-ігор включають інтенсивний ігровий процес, що вимагає від гравців високої концентрації, швидкої реакції та точної координації. Гравці часто повинні приймати рішення на ходу і швидко реагувати на змінні обставини. Бойові механіки є важливим елементом екшн-ігор, де гравці беруть участь у перестрілках, рукопашних боях або використовують різноманітну зброю та бойові техніки. Успіх у таких іграх часто залежить від майстерного володіння бойовими системами гри. Керування персонажем у реальному часі також є ключовим аспектом, що включає рух, атаки, стрибки, ухилення та інші активні дії.

Екшн-ігри часто використовують реалістичні графічні ефекти та звукове оформлення, щоб створити атмосферу занурення. Гравці повинні відчувати себе частиною ігрового світу, що підсилює їх емоційний досвід. Піджанри екшн-ігор включають шутери від першої особи (FPS), де гравці бачать світ через очі персонажа (наприклад, "Call of Duty", "Counter-Strike", "DOOM" (рис. 1.1)), та шутери від третьої особи (TPS), де гравці бачать персонажа ззовні (наприклад, "Gears of War", "Uncharted", "The Division"). Платформери, такі як "Super Mario", "Celeste" та "Sonic the Hedgehog", зосереджуються на стрибках між платформами та подоланні перешкод, вимагаючи точних стрибків і швидких рефлексів. Бітем-апи, як "Streets of Rage", "Double Dragon" та "Final Fight", зосереджені на боротьбі з численними

ворогами на різних рівнях. Hack and Slash ігри, такі як "Devil May Cry", "God of War" та "Bayonetta", пропонують інтенсивні бої з використанням зброї ближнього бою і часто містять елементи RPG [4].

Рисунок 1.1 – Загальний вигляд комп’ютерної гри Doom

Елементи геймплею екшн-ігор включають виконання різноманітних місій та завдань, які можуть включати рятувальні операції, знищення ворогів або дослідження територій. У багатьох екшн-іграх гравці збирають різноманітні ресурси або предмети, які можуть бути використані для покращення зброї, отримання нових навичок або відновлення здоров'я персонажа. Деякі екшн-ігри включають елементи RPG, де гравці можуть покращувати навички та здібності персонажа, що додає глибини геймплею і стимулює гравців до подальшого прогресу. Також важливими є бос-бої, де гравці стикаються з великими і складними ворогами, які вимагають особливих стратегій для перемоги. Ці бої зазвичай є ключовими моментами гри.

Серед відомих екшн-ігор варто виділити "The Legend of Zelda: Breath of the Wild", яка відома своїм відкритим світом та свободою дій, дозволяючи гравцям досліджувати величезний світ, виконувати завдання та брати участь у бойових діях. "Grand Theft Auto V" поєднує елементи стрілянини, гонок та відкритого світу, де

гравці можуть виконувати місії, взаємодіяти з різними персонажами та досліджувати великий міський світ. "Dark Souls" відомий своїм високим рівнем складності та глибокою механікою бою, де гравці повинні стратегічно підходити до боїв та досліджувати небезпечні локації.

Жанр екшн-ігор пропонує широкий спектр можливостей для гравців, які шукають динамічний і захоплюючий ігровий процес. Завдяки своїй різноманітності, екшн-ігри можуть задовольнити різні уподобання, від інтенсивних шутерів до захоплюючих платформерів та складних боїв у hack and slash.

Жанр Рольові ігри пропонує гравцям можливість зануритися в детально опрацьовані світи, брати участь у захоплюючих історіях та розвивати своїх персонажів, впливаючи на їхні здібності, характеристики та долю.

Рольові ігри характеризуються кількома ключовими елементами, які визначають їхній геймплей та привабливість. Одним з головних аспектів є сюжет і наратив, що дозволяє гравцям впливати на розвиток подій через свої рішення. Ці рішення можуть мати короткострокові або довгострокові наслідки, змінюючи хід історії. Розвиток персонажа також є важливою складовою RPG. Гравці можуть покращувати навички та здібності, збирати досвід, підвищувати рівні, а також вибирати класи, професії або спеціалізації для своїх персонажів [5].

Інтерактивний світ у RPG зазвичай відкритий або напіввідкритий, що дозволяє гравцям досліджувати його. Ці світи населені різними NPC (неігровими персонажами), з якими можна взаємодіяти, виконувати квести, торгувати та інше. Бойова система в багатьох RPG є складною, вона може бути покроковою або реального часу. Гравці використовують різні стратегії та тактики, щоб перемогти ворогів, часто комбінуючи атаки, магію та спеціальні здібності. Збирання предметів (луту) також є важливою частиною геймплею. Гравці можуть знаходити або купувати зброю, броню, магічні предмети та інше обладнання, що покращує їхні характеристики і надає додаткові можливості.

Елементи геймплею рольових ігор включають виконання різноманітних квестів і завдань, які просувають сюжет, розкривають світ гри і надають різні нагороди. Квести можуть включати бойові завдання, дослідження, вирішення

головоломок і багато іншого. В RPG часто присутня система вибору, де рішення гравця впливають на сюжет, стосунки з NPC і кінцівку гри. Це додає глибини та реіграбельності, оскільки різні вибори можуть призвести до різних результатів. Гравці можуть спеціалізувати персонажа, обираючи різні навички, таланти або здібності, що дозволяє створювати унікальні стратегії та стилі гри. Багато RPG мають внутрішню економіку, де гравці можуть торгувати з NPC або іншими гравцями, купувати і продавати предмети, а також керувати ресурсами. Кастомізація персонажа дозволяє гравцям змінювати зовнішній вигляд і екіпіровку, що додає персоналізації і дозволяє виразити свою індивідуальність у грі.

Серед відомих рольових ігор варто виділити "The Elder Scrolls V: Skyrim" (рис. 1.2), який відомий своїм великим відкритим світом, де гравці можуть досліджувати, виконувати квести і розвивати персонажа у різних напрямах. "The Witcher 3: Wild Hunt" пропонує глибокий сюжет, інтерактивний світ і складну бойову систему, що робить його однією з найпопулярніших RPG. "Final Fantasy VII" є класичною японською RPG, відомою своїм епічним сюжетом і багатими можливостями розвитку персонажів. "Mass Effect" є науково-фантастичною RPG, де гравці роблять вибори, які впливають на сюжет і взаємодію з іншими персонажами, створюючи унікальний досвід для кожного гравця.

Рисунок 1.2 – Загальний вигляд комп’ютерна гра The Elder Scrolls V: Skyrim

Жанр рольових ігор пропонує гравцям захоплюючий і глибокий досвід, що поєднує багатий сюжет, розвиток персонажа, інтерактивні світи та безліч можливостей для кастомізації та персоналізації. Завдяки своїй різноманітності RPG можуть задовольнити різні уподобання гравців, від епічних фентезі-пригод до науково-фантастичних епосів та інтенсивних тактичних битв.

Жанр Симулятори у відеоіграх є одним із найбільш реалістичних жанрів, спрямованих на імітацію реальних або вигаданих процесів, ситуацій та дій. Він охоплює широкий спектр тем, від керування транспортними засобами до симуляції соціальних взаємин і управління бізнесом. Основна мета симуляторів полягає в тому, щоб створити максимально реалістичний і детальний досвід, дозволяючи гравцям відчути себе в ролі професіонала або учасника певної діяльності [6].

Основні характеристики симуляторів включають високу ступінь реалістичності та деталізації, що дозволяє гравцям поринути в середовище, яке вони симулюють. Симулятори часто вимагають від гравців глибокого розуміння механік та систем, на яких базується гра. Наприклад, у симуляторах керування транспортними засобами гравці повинні враховувати фізику руху, правила дорожнього руху та технічне обслуговування машин. У симуляторах управління бізнесом гравці можуть зосередитися на економічних аспектах, маркетингу, фінансах та управлінні персоналом.

Елементи геймплею в симуляторах часто включають керування різноманітними аспектами симульованої діяльності. У авіаційних симуляторах гравці можуть керувати літаками, слідкувати за приладами, виконувати маневри та посадки. У сільськогосподарських симуляторах гравці можуть садити та збирати врожай, доглядати за тваринами та керувати фінансами ферми. У симуляторах управління містами гравці планують забудову, керують бюджетом міста, забезпечують громадські послуги та вирішують кризи.

Симулятори також часто включають елементи навчання та тренувань, дозволяючи гравцям здобувати нові знання та навички, які можуть бути корисними в реальному житті. Наприклад, авіаційні симулятори можуть використовуватися для

тренувань майбутніх пілотів, а медичні симулятори - для навчання лікарів і медичних працівників.

Жанр "Стратегії" у відеоіграх зосереджується на плануванні, управлінні ресурсами і тактичних діях для досягнення перемоги над супротивниками. У цих іграх гравці керують арміями, будують бази або розвивають цивілізації, приймаючи стратегічні рішення для досягнення своїх цілей.

Стратегії поділяються на дві основні категорії: стратегії в реальному часі (RTS) і покрокові стратегії (TBS). У RTS, таких як "StarCraft" та "Age of Empires", дії відбуваються в реальному часі, вимагаючи швидкого прийняття рішень і одночасного управління багатьма аспектами. У TBS, як-от "Civilization" (рис. 1.3) та "XCOM", гравці по черзі планують і виконують свої дії, що дозволяє більш ретельно продумувати стратегії.

Ці ігри часто включають елементи економічного управління, військової тактики і дипломатії. Стратегії приваблюють гравців, які насолоджуються інтелектуальними викликами, складними рішеннями і довгостроковим плануванням для досягнення успіху.

Рисунок 1.3 – Загальний вигляд комп’ютерна гра Civilization

Жанр Спортивних ігр імітує різні види спорту, надаючи гравцям можливість відчути себе професійними спортсменами або тренерами. Ці ігри включають командні види спорту, як футбол і баскетбол, та індивідуальні змагання, як теніс і гольф. Основна мета спортивних ігор — передати реалістичність та емоції справжніх змагань.

Футбольні симулятори, такі як серії "FIFA" та "Pro Evolution Soccer" (PES), дозволяють керувати командами, проводити матчі та брати участь у турнірах. Баскетбольні симулятори, як "NBA 2K", дозволяють грати за команди НБА, брати участь у сезонних змаганнях та розвивати гравців. Симулятори гонок, такі як "Gran Turismo" і "Forza Motorsport", зосереджуються на реалістичному відтворенні автомобільних гонок. Симулятори бойових мистецтв, як серії "UFC" та "Fight Night", дозволяють брати участь у змішаних бойових мистецтвах.

Геймплей у спортивних іграх зазвичай включає управління командами та гравцями, розробку стратегій, участь у змаганнях та турнірах. Часто присутні режими кар'єри, де гравці можуть розвивати своїх персонажів або команди протягом кількох сезонів. Також доступні мультиплеєрні режими для змагань з іншими гравцями онлайн.

Серед відомих спортивних ігор — серія "FIFA" (рис 1.4) зображено на рисунку 1.4 зі своїм реалістичним відтворенням футболу, "NBA 2K" з можливістю грати за команди НБА, та "Gran Turismo" і "Forza Motorsport" з реалістичними гонками. Жанр спортивних ігор пропонує гравцям занурення у світ спорту, надаючи реалістичний досвід та задовольняючи уподобання фанатів різних видів спорту.

Рисунок 1.4 – Загальний вигляд комп’ютерна гра FIFA

Жанр "Головоломки" у відеоіграх зосереджується на вирішенні завдань, які вимагають логічного мислення, стратегічного планування і творчості. Ці ігри пропонують гравцям різноманітні виклики, що стимулюють розумову діяльність і вимагають вирішення різних типів головоломок, таких як логічні, просторові, математичні та лінгвістичні завдання.

Приклади відомих головоломок включають серії "Tetris" (рис. 1.5) зображено на рисунок 1.5 та "Portal". "Tetris" вимагає від гравців стратегічно розміщувати блоки, щоб створювати повні рядки та уникати заповнення екрана. "Portal" пропонує унікальні фізичні головоломки, де гравці використовують портал-пушку для переміщення через рівні, вирішуючи завдання за допомогою просторового мислення та фізики.

Головоломки можуть бути як окремими іграми, так і частиною більш комплексних жанрів, додаючи інтелектуальні виклики до пригодницьких або екшн-ігор. Цей жанр приваблює гравців, які люблять розв'язувати завдання і отримують задоволення від інтелектуальних викликів.

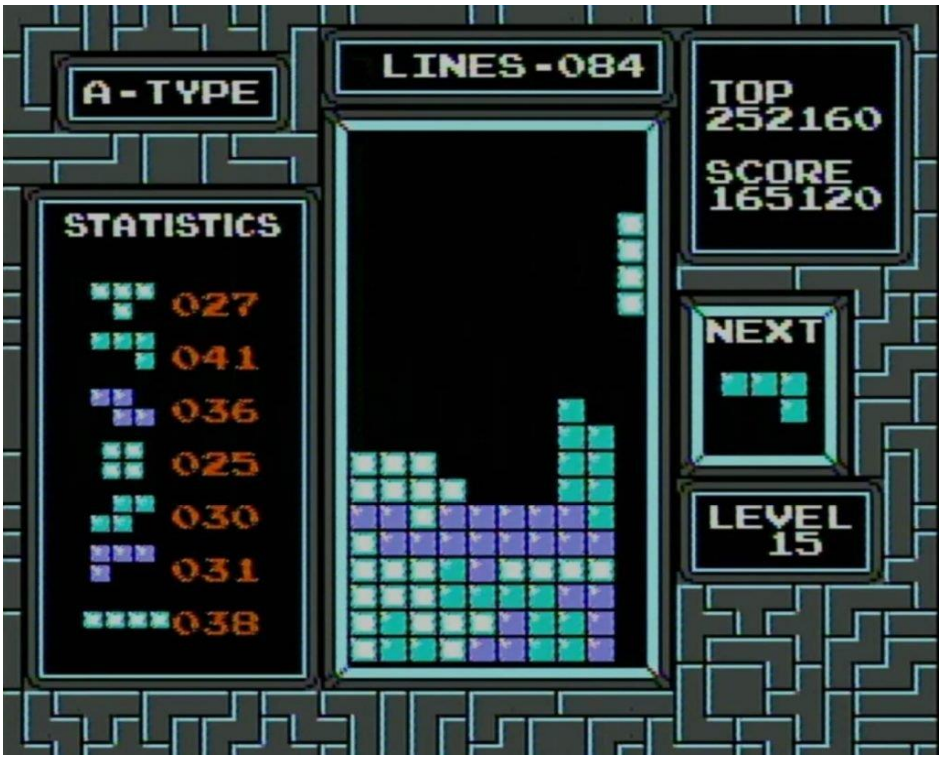

Рисунок 1.5 – Загальний вигляд комп’ютерна гра Tetris

Статистика популярності жанрів компютерних ігор за віковими групами наведена на рисунку 1.6 [7].

Characteristic	16-24 years old	25-34 years old	35-44 years old	45-54 years old	55-64 years old
Shooter	62.4%	58.6%	50.5%	40.8%	29.3%
Action adventure	56.8%	53.5%	49.2%	39.7%	28.8%
Simulation	39.3%	36.5%	31.5%	24.4%	20.5%
Sports	37.8%	40.2%	36.3%	30%	22%
MOBA	36.8%	36.9%	30.3%	22.5%	16.6%
Racing	36.2%	37.8%	34.6%	28.3%	20.4%
Strategy	34%	35%	33.2%	27.1%	21.8%
Puzzle platform	32.9%	35.6%	34.7%	30.6%	28.9%
Action platform	30.7%	32.3%	30.2%	24%	17.6%
Fighting	-	32.8%	29.4%	22.1%	-
Battle royale	35.5%	-	-	-	-
Online board games	-	-	-	-	18.5%

Рисунок 1.6 – Рейтинг популярності жанрів комп’ютерних ігор

1.2 Аналіз рушіїв комп’ютерних ігор

Ігрові рушії (або ігрові движки) – це програмне забезпечення, яке забезпечує розробку, створення та управління комп'ютерними іграми. Вони надають розробникам інструменти для створення ігор, включаючи графіку, фізику, звукові ефекти, скрипти, анімацію і штучний інтелект. Використання ігрових рушіїв значно полегшує процес розробки, оскільки розробникам не потрібно писати все з нуля [8].

Unity – це кросплатформенний ігровий рушій, розроблений Unity Technologies, що надає розробникам широкий набір інструментів для створення 2D та 3D ігор, інтерактивних додатків і віртуальної реальності. Unity дозволяє експортувати проекти на безліч платформ, включаючи ПК, мобільні пристрої, консолі та VR/AR гарнітури [9].

Unity використовує концепцію сцен (Scenes) як основні будівельні блоки гри.

Кожна сцена містить різноманітні об’єкти (GameObjects), такі як персонажі, камери, джерела світла та елементи оточення. GameObjects є базовими об'єктами, які можна налаштовувати за допомогою додавання компонентів. Кожен GameObject може мати декілька компонентів, що визначають його властивості та поведінку.

Компоненти (Components) в Unity додають функціональність GameObject. Наприклад, компоненти Transform визначають позицію, обертання і масштаб об'єкта, Rigidbody додає фізичні властивості, а Collider визначає області зіткнень. Скрипти, написані на мові програмування C#, використовуються для додаткової кастомізації та створення поведінки об'єктів. Більшість скриптів наслідують від класу MonoBehaviour, що дозволяє використовувати спеціальні методи життєвого циклу, такі як Start() та Update().

Unity оснащений потужним рендеринг-рушієм, що підтримує сучасні графічні технології. Це включає шейдери, освітлення, тіні та постобробку. Два основних скриптованих рендеринг-канали (Scriptable Render Pipelines, SRP) — Universal Render Pipeline (URP) і High Definition Render Pipeline (HDRP) — дозволяють розробникам обирати оптимальний інструмент для їхніх потреб, забезпечуючи баланс між продуктивністю та якістю зображення.

Матеріали в Unity визначають, як поверхня об'єкта виглядатиме під час рендерингу, тоді як шейдери — це спеціалізовані програми, що виконуються на графічному процесорі для рендерингу пікселів, вершин та інших графічних даних. Це забезпечує високий рівень деталізації та реалістичності у візуальних ефектах.

Unity включає в себе потужний фізичний рушій, що дозволяє симулювати реалістичну фізику. Основними фізичними компонентами є Rigidbody (додає об'єкту фізичні властивості, такі як маса і сила тяжіння) і Collider (відповідає за зіткнення). Ці компоненти дозволяють створювати складні фізичні взаємодії між об'єктами, включаючи зіткнення, рух, сили і моменти.

Unity Editor — це інтегроване середовище розробки (IDE), яке надає всі необхідні інструменти для створення та редагування ігор. Це включає роботу зі сценами, анімаціями, звуками, матеріалами та іншими аспектами гри. Інтерфейс редактора інтуїтивно зрозумілий, що полегшує навчання та швидкий старт у

розробці.

Unity Asset Store — це онлайн-магазин, де розробники можуть купувати або безкоштовно завантажувати ресурси, такі як моделі, текстури, звуки, скрипти та інші корисні інструменти. Це значно скорочує час розробки, дозволяючи зосередитися на створенні унікальних елементів гри [10].

Однією з ключових переваг Unity є його здатність підтримувати розробку для різних платформ з однієї кодової бази. Це включає ПК (Windows, macOS, Linux), мобільні пристрої (iOS, Android), консолі (PlayStation, Xbox, Nintendo Switch), веб (WebGL) і VR/AR пристрої (Oculus, HTC Vive, Microsoft HoloLens). Це дозволяє розробникам охопити широку аудиторію та максимізувати потенціал свого проекту.
Переваги ігрового рушія Unity:

1) Кросплатформеність: Unity підтримує розробку ігор для різних платформ, включаючи ПК, консолі, мобільні пристрої, веб-браузери та віртуальну реальність. Це дозволяє розробникам створювати одну гру та легко портувати її на різні платформи.
2) Простота освоєння: Unity має інтуїтивно зрозумілий інтерфейс та безліч навчальних матеріалів, включаючи документацію, відеоуроки та форуми спільноти. Це робить його доступним навіть для новачків у розробці ігор.
3) Велика спільнота: Unity має велику та активну спільноту розробників, що забезпечує доступ до великої кількості безкоштовних та платних ресурсів, таких як скрипти, графіка та плагіни.
4) Багатофункціональний інструментарій: Unity пропонує потужні інструменти для розробки ігор, включаючи редактор сцен, систему анімації, фізичний рушій та інструменти для роботи з аудіо. Інтеграція з Visual Studio дозволяє зручно писати та відлагоджувати код.
5) Підтримка 2D та 3D: Unity однаково добре підходить для розробки як 2D, так і 3D ігор, надаючи розробникам широкі можливості для створення різноманітних проектів.
6) Asset Store: Unity Asset Store пропонує тисячі готових ресурсів, таких як моделі, текстури, скрипти та інструменти, що значно спрощує та прискорює

процес розробки.

Недоліки Unity:

1) Вимоги до продуктивності: Unity може вимагати високопродуктивне обладнання для створення та тестування великих та складних проектів. Деякі проекти можуть страждати від проблем з продуктивністю, особливо на старих або менш потужних пристроях.
2) Висока ціна ліцензії: Хоча базова версія Unity безкоштовна, для доступу до більш розширених функцій та можливостей (наприклад, для командної роботи та преміум-підтримки) необхідно придбати платну ліцензію, яка може бути дорогою для невеликих студій та інді-розробників.
3) Обмеження безкоштовної версії: Безкоштовна версія Unity має деякі обмеження, такі як відсутність доступу до певних функцій та обов'язковий сплеш-екран Unity при запуску гри, що може виглядати непрофесійно.
4) Менеджмент пам'яті: Unity може мати проблеми з менеджментом пам'яті, що може призводити до витоків пам'яті та нестабільності гри, особливо у великих проектах.
5) Обмежена графічна якість: Хоча Unity може створювати вражаючі візуальні ефекти, він може поступатися іншим рушіям, таким як Unreal Engine, у плані графічної якості та реалістичності.
6) Складність оптимізації: Оптимізація ігор на Unity може бути складною, особливо для новачків. Це може вимагати значних зусиль для забезпечення стабільної роботи гри на різних платформах та пристроях.

Unity є універсальним і потужним інструментом для розробки ігор, що забезпечує розробникам широкі можливості для створення якісних проектів. Завдяки своїй гнучкості, великій спільноті та підтримці багатьох платформ, Unity є одним з найпопулярніших ігрових рушіїв у світі. Його використання дозволяє значно скоротити час розробки та підвищити якість кінцевого продукту, роблячи його відмінним вибором як для новачків, так і для досвідчених розробників.

Godot – це потужний і безкоштовний рушій для розробки ігор, який надає можливість створювати 2D та 3D ігри. Він був випущений під ліцензією MIT, що

означає, що він є відкритим та безкоштовним для використання, навіть для комерційних проєктів. Рушій Godot розробляється і підтримується спільнотою розробників з усього світу, що забезпечує його постійний розвиток і оновлення [11].

Однією з ключових особливостей Godot є його інтуїтивний інтерфейс користувача та зрозуміла архітектура сцени. У Godot все будується на основі сцен і вузлів. Кожна сцена може містити інші сцени та вузли, що дозволяє створювати складні структури гри з високим рівнем організації. Це робить розробку гри модульною і зручною для управління.

Godot підтримує два основні типи сценаріїв: візуальні та текстові. Візуальні сценарії дозволяють створювати логіку гри за допомогою блок-схем, що робить їх зручними для початківців або дизайнерів, які не знайомі з програмуванням. Текстові сценарії зазвичай пишуться мовою GDScript, яка є спеціальною мовою програмування, створеною для Godot. GDScript дуже схожий на Python, що робить його легким для вивчення і використання. Крім того, Godot підтримує мови C#, C++ і навіть візуальні мови, що робить його дуже гнучким для різних потреб розробників.

Для розробки 2D-ігор Godot пропонує багатий набір інструментів, включаючи редактори анімацій, фізику, світло та тіні. Рушій підтримує векторну графіку та піксельну точність, що дозволяє створювати високоякісні 2D-ігри з чудовою продуктивністю. У 3D-режимі Godot також надає широкі можливості, включаючи підтримку PBR-рендерінгу, систем частинок, анімаційні системи та багато іншого.

Однією з важливих переваг Godot є його кросплатформеність. Рушій дозволяє експортувати ігри на різні платформи, включаючи Windows, macOS, Linux, Android, iOS, HTML5 і навіть консолі. Це дає можливість розробникам охоплювати широку аудиторію гравців без необхідності витрачати час і ресурси на адаптацію гри до кожної окремої платформи.

Ігровий рушій Godot також відомий своєю активною і підтримуючою спільнотою. Існує багато навчальних матеріалів, відеоуроків, документації та форумів, де розробники можуть знайти допомогу і підтримку. Це робить процес вивчення і роботи з рушієм значно легшим і приємнішим.

Переваги ігрового рушія Godot:

1) Безкоштовний і відкритий код: Godot є повністю безкоштовним і відкритим рушієм, ліцензованим за MIT, що дозволяє розробникам використовувати його без жодних обмежень для комерційних і некомерційних проєктів.
2) Інтуїтивний інтерфейс: Godot має зрозумілий і добре організований інтерфейс, що робить його зручним як для початківців, так і для досвідчених розробників. Архітектура сцен і вузлів спрощує організацію проєктів.
3) Підтримка 2D і 3D: Godot надає потужні інструменти для розробки як 2D, так і 3D ігор, з підтримкою різних графічних і фізичних систем, що дозволяє створювати візуально привабливі та технічно складні ігри.
4) GDScript: Спеціальна мова програмування GDScript, яка схожа на Python, легка у вивченні та використанні. Крім того, підтримуються інші мови програмування, такі як C#, C++.
5) Кросплатформеність: Godot дозволяє експортувати ігри на різні платформи, включаючи Windows, macOS, Linux, Android, iOS, HTML5 та консолі, що дозволяє охоплювати широку аудиторію.
6) Активна спільнота і документація: Існує велика кількість навчальних матеріалів, відеоуроків, документації та активних форумів, що допомагає швидко знайти відповіді на питання та отримати підтримку від спільноти.

Недоліки Godot:

1) Менша популярність: Порівняно з іншими популярними рушіями, такими як Unity чи Unreal Engine, Godot має меншу базу користувачів, що може вплинути на кількість доступних ресурсів і плагінів.
2) Обмежені ресурси для 3D: Хоча Godot підтримує 3D-розробку, його можливості в цій сфері можуть бути обмежені порівняно з провідними рушіями. Це може стати проблемою для розробників, які працюють над високотехнологічними 3D-проєктами.
3) Менше комерційних плагінів і інструментів: Через меншу популярність рушія, кількість доступних комерційних плагінів та інструментів є обмеженою, що може потребувати від розробників більше часу на створення власних рішень.

4) Молодість платформи: Godot є відносно молодим рушієм, тому деякі функції та інструменти можуть бути менш зрілими або розвинутими порівняно з тими, що пропонуються більш відомими рушіями.
5) Відсутність офіційної підтримки: На відміну від деяких комерційних рушіїв, Godot не має офіційної підтримки з боку корпорацій, що може ускладнити вирішення певних технічних проблем.

CryEngine – це потужний і популярний ігровий рушій, розроблений німецькою компанією Crytek. Вперше він був представлений у 2004 році з випуском гри Far Cry, і відтоді зазнав численних оновлень та поліпшень. CryEngine славиться своїми передовими графічними можливостями, включаючи підтримку високоякісної графіки, реалістичного освітлення, тіней та відображень, що дозволяє створювати вражаючі візуальні ефекти. Він підтримує DirectX 12 і Vulkan API, що забезпечує кращу продуктивність та оптимізацію для різних платформ [12].

Рушій має інтегроване середовище розробки CryEngine Sandbox, яке надає зручні інструменти для створення ігор. Це WYSIWYG редактор, що дозволяє розробникам бачити зміни в реальному часі. Також доступна система візуального програмування Flow Graph, яка дозволяє створювати ігрову логіку без написання коду, що особливо корисно для дизайнерів та художників. CryEngine має вбудовану систему фізики, яка підтримує реалістичну симуляцію твердої механіки, рідин та тканин. Анімаційна система рушія дозволяє створювати реалістичні рухи персонажів завдяки технології риггінгу та інверсної кінематики.

CryEngine підтримує об'ємний звук, просторовий звуковий дизайн та інтеграцію з популярними аудіо-системами, такими як Wwise і FMOD. Рушій також має розширену підтримку багатокористувацьких ігор з функціями синхронізації даних, управління сесіями та мережевої безпеки. Завдяки високоякісній графіці та можливостям фізики, CryEngine дозволяє створювати реалістичні ігрові світи. Інтегроване середовище розробки та системи візуального програмування спрощують процес створення ігор. Рушій підтримує розробку для різних платформ, включаючи ПК, консолі та мобільні пристрої.

Ігровий рушій CryEngine використовувався для створення багатьох відомих

ігор, таких як Far Cry (2004), серія Crysis (2007-2013), Ryse: Son of Rome (2013), Kingdom Come: Deliverance (2018).

Переваги ігрового рушія CryEngine:

1) Реалістичність та деталізація: Високоякісна графіка та можливості фізики забезпечують створення реалістичних ігрових світів.
2) Потужні інструменти: Інтегроване середовище розробки та системи візуального програмування спрощують процес створення ігор.
3) Мультиплатформність: Рушій підтримує розробку для різних платформ, включаючи ПК, консолі та мобільні пристрої.

Недоліки:

1) Високі вимоги до апаратного забезпечення: Для повного використання можливостей рушія необхідне потужне обладнання, що може бути проблемою для деяких розробників.
2) Складність освоєння: Незважаючи на наявність інструментів візуального програмування, CryEngine може бути складним для новачків через велику кількість функцій та налаштувань.

1.3 Аналіз комп’ютерних ігор жанру 2D шутер

Жанр 2D шутерів є однією з класичних категорій комп’ютерних ігор, що зосереджуються на стрільбі та бойових діях у двовимірному просторі. В таких іграх гравець керує персонажем, зазвичай озброєним різними видами зброї, з метою знищення ворогів і ухилення від їхніх атак. Більшість 2D шутерів використовують схеми горизонтального або вертикального скролінгу, де персонаж рухається по екрану зліва направо, зверху вниз або навпаки [13].

Графіка та візуальний стиль в 2D шутерах використовують двовимірні спрайти та фони, а стиль може варіюватися від реалістичного до карикатурного чи піксель-арту. Зброя та боєприпаси у таких іграх представлені широким асортиментом, від пістолетів до фантастичних бластерів. Часто гравець може знаходити або купувати нову зброю і поліпшення до неї, що додає різноманітності в

ігровий процес.

Вороги в 2D шутерах мають різноманітні поведінкові патерни, що робить кожну сутичку унікальною. Крім того, у грі зазвичай присутні боси з унікальними атаками та слабкими місцями, що додає додаткового виклику. Рівні та середовища в таких іграх різноманітні з унікальними ландшафтами, пастками та перешкодами, що робить кожен рівень неповторним. Дизайн рівнів може бути як лінійним, так і нелінійним, що впливає на стратегію проходження гри.

Відомими представниками жанру 2D шутерів є "Broforce" (рис. 1.7), "My Friend Pedro" (рис. 1.8) та "Hotline Miami" (рис. 1.9). У "Broforce" гравці керують командою "бро", кожен з яких є пародією на відомих екшн-героїв з фільмів 80-х та 90-х років. Кожен персонаж має свої унікальні здібності та зброю, що робить гру різноманітною та захоплюючою. Гра пропонує інтенсивний, динамічний геймплей з великим рівнем руйнування оточення, що додає глибини та стратегії.

Рисунок 1.7 – Загальний вигляд комп’ютерна гра Broforce

"My Friend Pedro" відзначається своєю унікальною механікою, що поєднує в собі платформер та шутер. Гравець контролює персонажа, який може виконувати акробатичні трюки, стрибаючи та стріляючи одночасно. Завдяки цьому, гра має плавний, кінематографічний геймплей, який виглядає як сцени з бойовиків. Основною особливістю гри є можливість сповільнення часу, що дозволяє гравцеві виконувати неймовірні трюки та знищувати ворогів у найефектніший спосіб.

Рисунок 1.8 – Загальний вигляд комп’ютерна гра My Friend Pedro

"Hotline Miami" пропонує гравцям інтенсивний геймплей з видом зверху вниз, де кожен рівень вимагає швидкої реакції та точного планування. Гравець бере на себе роль таємничого вбивці, який виконує завдання, знищуючи ворогів у будівлях, повних небезпек. Гра відома своїм високим рівнем складності та потребує від гравця ретельного аналізу кожної ситуації. Крім того, "Hotline Miami" відзначається своїм унікальним стилем, який поєднує в собі неонову естетику 80-х років і динамічний саундтрек, що створює неповторну атмосферу.

Рисунок 1.9 – Загальний вигляд комп’ютерна гра Hotline Miami

Ігровий процес в 2D шутерах зазвичай включає просте, але інтуїтивно зрозуміле управління: рухи вліво-вправо, стрибки та стрілянину. Часто є можливість стріляти в різні напрямки, включаючи діагоналі, що додає тактичної глибини. Складність у цих іграх зазвичай висока, з великою кількістю ворогів на екрані, що вимагає від гравця швидкої реакції та точного прицілювання. Часто в іграх є система "життів" та континью, яка дозволяє продовжити гру після поразки.

Бонуси та поліпшення, які гравець може збирати, дають тимчасові або постійні покращення, що допомагає успішно завершити рівень. Сучасні 2D шутери включають більш складні графічні ефекти, анімацію та інтерактивні середовища. Вони можуть мати елементи RPG, такі як прокачка персонажа та збирання ресурсів. Багато незалежних розробників створюють 2D шутери з унікальними стилями та ігровими механіками, експериментуючи з жанром та розширюючи можливості гравця. Жанр 2D шутерів залишається популярним завдяки своїй простоті та можливості для розробників створювати інтенсивний і захоплюючий геймплей.

1.4 Висновок до розділу 1

Було проведено аналіз предметної області, включаючи дослідження ігрових механік, жанрових особливостей та сучасних тенденцій у розробці 2D шутерів. На основі аналізу було обрано ігровий рушій Unity, який забезпечив необхідні інструменти та функціональність для створення гри.

2 РОЗРОБКА ТА ПРОЄКТУВАННЯ КОМП’ЮТЕРНОЇ ГРИ

2.1 Розробка архітектури комп’ютерної гри

Архітектурні підходи для розробки 2D комп’ютерні ігри можуть варіюватися залежно від вимог проекту, обсягу гри та наявного досвіду розробників. Основні підходи включають:

1. Монолітна архітектура.

У цьому підході всі компоненти гри інтегровані в один великий додаток. Це простіший варіант, але може призвести до проблем з масштабуванням та підтримкою гри (рис. 2.1) [14].

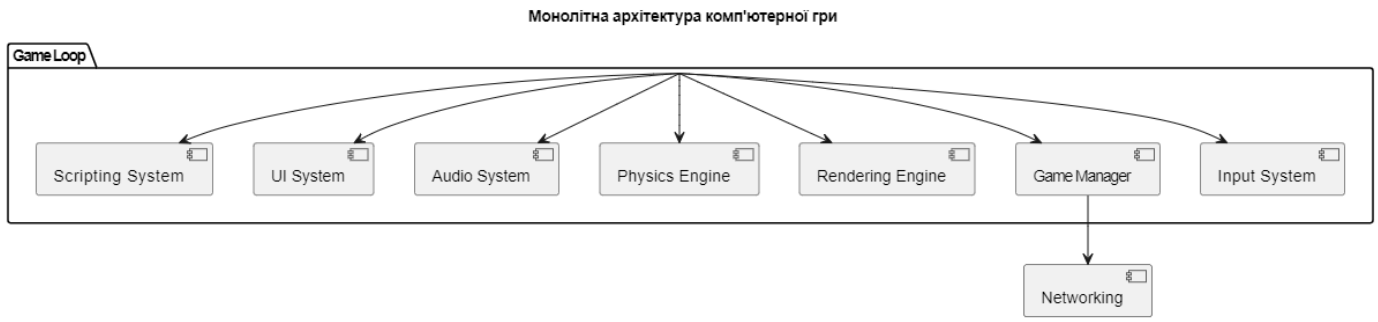

Рисунок 2.1 – Принципова схема монолітної архітектури комп’ютерної гри

Переваги: простота реалізації для невеликих проектів, швидке створення прототипів.

Недоліки: складність у підтримці та розширенні великого коду, обмежена модульність.

2. Компонентно-орієнтована архітектура (Component-Based Architecture).

У цьому підході ігрові об'єкти створюються як комбінації незалежних компонентів. Кожен компонент виконує певну функцію (рух, відображення, взаємодія і т.д.) (рис. 2.2) [15].

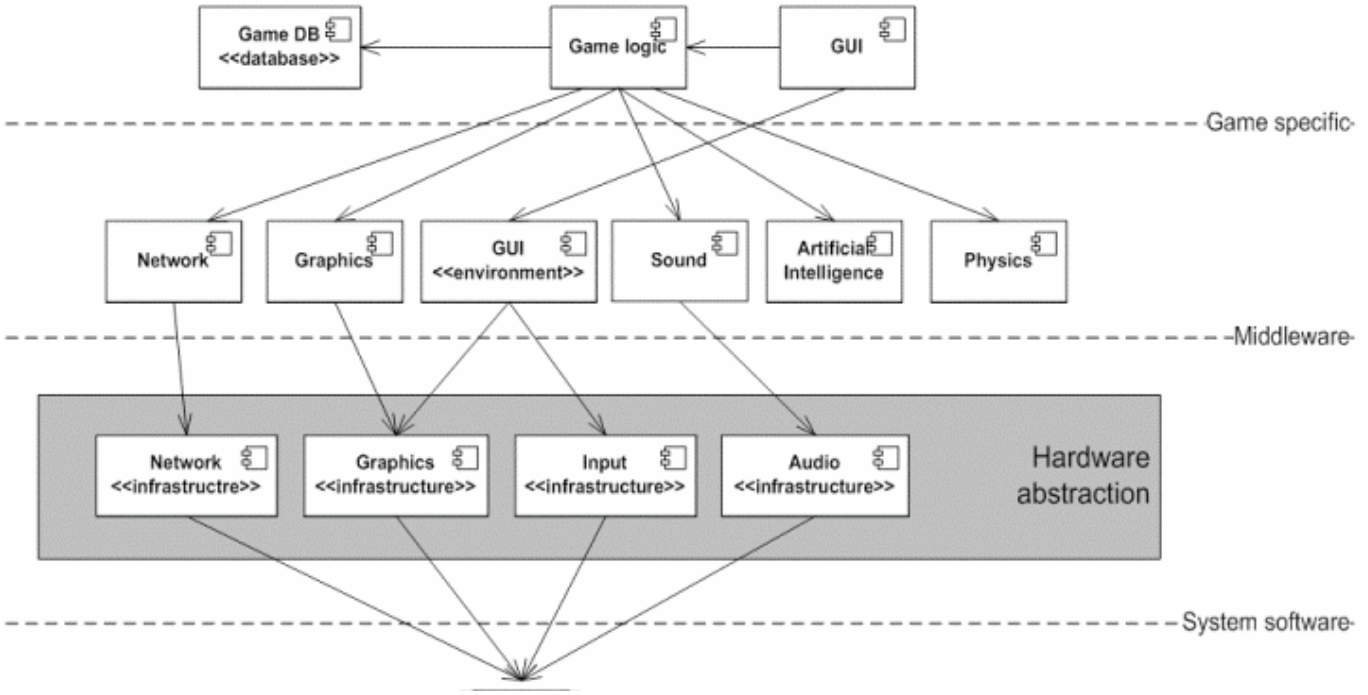

Рисунок 2.2 – Принципова схема компонентно-орієнтована архітектури комп’ютерної гри

Переваги: висока модульність і повторне використання коду, легкість в розширенні функціоналу.

Недоліки: можлива складність у менеджменті залежностей між компонентами, початкова складність в організації системи.

3. Архітектура на основі систем (System-Based Architecture).

Цей підхід схожий на компонентно-орієнтовану архітектуру, але акцент робиться на системах, які обробляють певні аспекти гри (фізика, рендеринг, аудіо) окремо.

Переваги: висока модульність, легкість в тестуванні і обслуговуванні.

Недоліки: початкова складність у проектуванні, можлива проблема з продуктивністю через численні системи.

4. Модель-Вид-Контролер (MVC).

Цей підхід розділяє гру на три основні компоненти: модель (логіка гри та дані), вид (інтерфейс користувача) і контролер (обробка вводу) (рис 2.3.)[16].

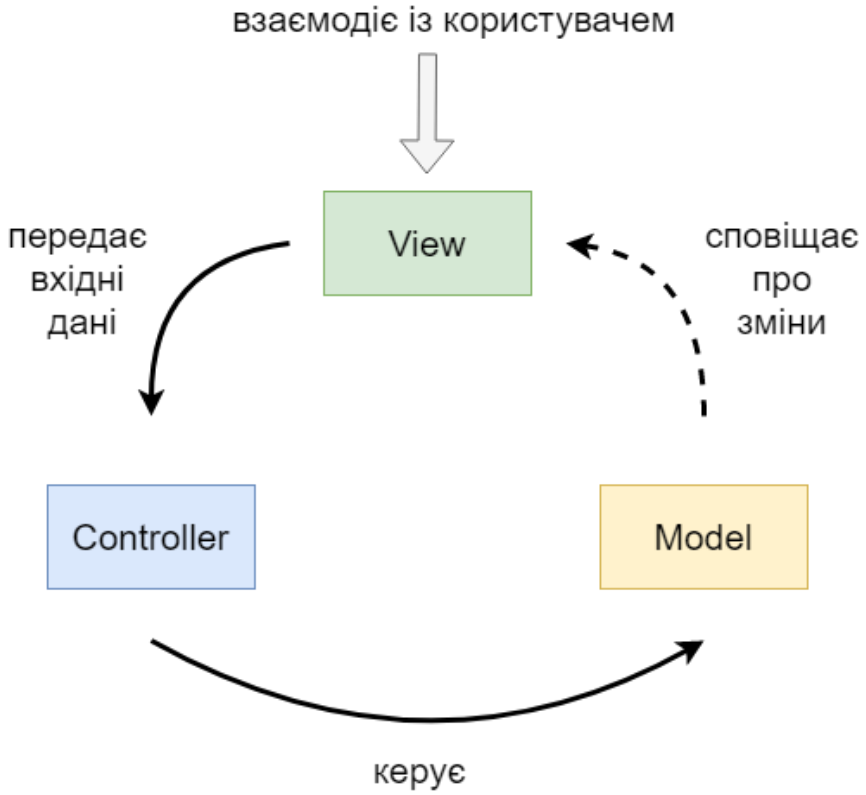

Рисунок 2.3 – Принципова схема архітектури Модель-Вид-Контролер комп’ютерної гри

Переваги: чітке розділення відповідальностей, полегшення тестування та налагодження.

Недоліки: може бути занадто складним для простих ігор, початкове проектування може бути більш трудомістким.

5. Архітектура на основі об'єктів (Object-Oriented Architecture).

Цей підхід використовує об'єктно-орієнтоване програмування (ООП) для організації гри. Ігрові об'єкти представляються як класи з властивостями та методами.

Переваги: знайомість для багатьох розробників, легкість у розширенні та підтримці.

Недоліки: може призвести до жорсткої структури, можливі проблеми з продуктивністю через складні ієрархії класів.

6. Архітектура з використанням шаблонів проектування (Design Patterns)

Використання різних шаблонів проектування, таких як фабрика (Factory), синглтон (Singleton), спостерігач (Observer), може допомогти організувати код та зменшити залежності.

Переваги: підвищення гнучкості та повторного використання коду, полегшення підтримки та розширення.

Недоліки: може ускладнити проектування для новачків, превантаження проекту надмірним використанням шаблонів.

7. Ентіті-Компонент-Система (ECS).

Цей підхід відокремлює дані (ентіт) від логіки (система), використовуючи компоненти як властивості. ECS забезпечує високу продуктивність та гнучкість.

Переваги: висока продуктивність, гнучкість у розширенні та підтримці.

Недоліки: складність у початковому проектуванні, вимагає глибокого розуміння концепцій ECS.

Unity природно підтримує компонентно-орієнтовану архітектуру, що робить її ідеальним вибором для проекту. Кожен об'єкт у Unity є GameObject, який може мати різні компоненти, що додають функціональність.

Основні компоненти архітектури для 2D шутера:

1. Ігровий цикл (Game Loop): Unity автоматично управляє ігровим циклом через функції Update та FixedUpdate в MonoBehaviour.
2. Менеджер ресурсів (Resource Manager): Використовуйте вбудовану систему ресурсів Unity (папки Resources, Asset Bundles) для завантаження текстур, звуків та інших ресурсів.
3. Сцени та рівні (Scenes and Levels): Розділяйте гру на сцени (рівні, головне меню, налаштування). Використовуйте SceneManager для перемикання між сценами.
4. Менеджер об'єктів (Entity Manager): Використовуйте скрипти для керування об'єктами гри, такими як вороги, герої, перешкоди.
5. Фізичний движок (Physics Engine): Використовуйте вбудований фізичний движок Unity 2D для обробки зіткнень та фізичних взаємодій.
6. Користувацький інтерфейс (UI): Використовуйте Canvas і UI компоненти Unity для створення інтерфейсу (меню, кнопки, індикатори здоров'я).
7. Введення (Input): Використовуйте Input Manager для обробки вводу з клавіатури та миші.

8. Звук (Audio): Використовуйте AudioSource та AudioClip для відтворення звуків та музики.
9. Система подій (Event System): Використовуйте систему подій Unity для обробки взаємодій між об'єктами (наприклад, колізії).
10. Система збереження (Save System): Використовуйте PlayerPrefs або створіть власну систему для збереження прогресу гри.

Архітектура комп’ютерної гри є ключовим елементом, який визначає загальну структуру програмного забезпечення, принципи взаємодії між різними компонентами та їх функціональну відповідальність. Розробка архітектури включає визначення основних модулів, їх інтерфейсів, а також способів їх взаємодії. У випадку 2D шутера, важливо забезпечити ефективність, масштабованість та гнучкість архітектури.

Основними компонентами архітектури є ігровий рушій, менеджер ресурсів, система управління подіями, ігровий цикл та скрипти ігрової логіки. Ігровий рушій відповідає за рендеринг графіки, фізику, анімацію та обробку введення. Рендеринг здійснюється за допомогою бібліотек, таких як OpenGL або DirectX, що дозволяють відображати спрайти, анімації та фон. Фізика включає обробку зіткнень, гравітації та інших фізичних взаємодій, які можуть бути реалізовані через фізичні рушії, такі як Box2D, або власні алгоритми. Анімація керується через спрайт-листи, що дозволяють створювати плавні переходи між різними станами персонажів.

Менеджер ресурсів займається завантаженням та управлінням ресурсами гри, такими як текстури, звуки та анімації. Він забезпечує оптимізацію доступу до ресурсів шляхом кешування та багатопоточності. Ресурси можуть бути описані у форматі JSON або XML, що спрощує їхнє завантаження та використання в грі. Для зниження витрат пам'яті менеджер ресурсів використовує пул ресурсів, що дозволяє повторно використовувати завантажені об'єкти.

Система управління подіями обробляє введення з клавіатури, миші або геймпада, а також управляє подіями в грі, такими як зіткнення об'єктів та зміни станів. Введення мапується на відповідні дії гравця, що дозволяє миттєво реагувати на дії користувача. Для обробки подій використовується паттерн "спостерігач", що

дозволяє викликати відповідні обробники подій на основі їхнього типу.

Ігровий цикл є основним процесом, який виконується безперервно протягом гри. Він включає етапи оновлення стану гри, рендерингу та обробки подій. На етапі оновлення перераховуються позиції об'єктів, перевіряються умови зіткнення та оновлюються стани об'єктів. Рендеринг відбувається за допомогою подвійного буферу, що знижує мерехтіння та забезпечує плавний ігровий процес. Для управління таймінгом використовуються фіксовані інтервали часу, що забезпечує стабільність гри та синхронізацію оновлень і рендерингу.

Скрипти ігрової логіки реалізують правила гри, поведінку ворогів та управління рівнями. Використання скриптових мов, таких як Lua або Python, дозволяє гнучко змінювати логіку гри без потреби компілювати код. Поведінка ворогів реалізується через стани, що моделюють різні дії, такі як атака, переслідування або втеча. Для руху ворогів у просторі гри використовуються алгоритми пошуку шляху, що забезпечують інтелектуальну навігацію.

Ця структура забезпечує гнучкість та модульність гри, дозволяючи легко додавати нові функції, змінювати існуючі компоненти та підтримувати високу продуктивність. Взаємодія між компонентами архітектури дозволяє створити ефективну та надійну систему, що забезпечує захоплюючий ігровий процес для користувачів.

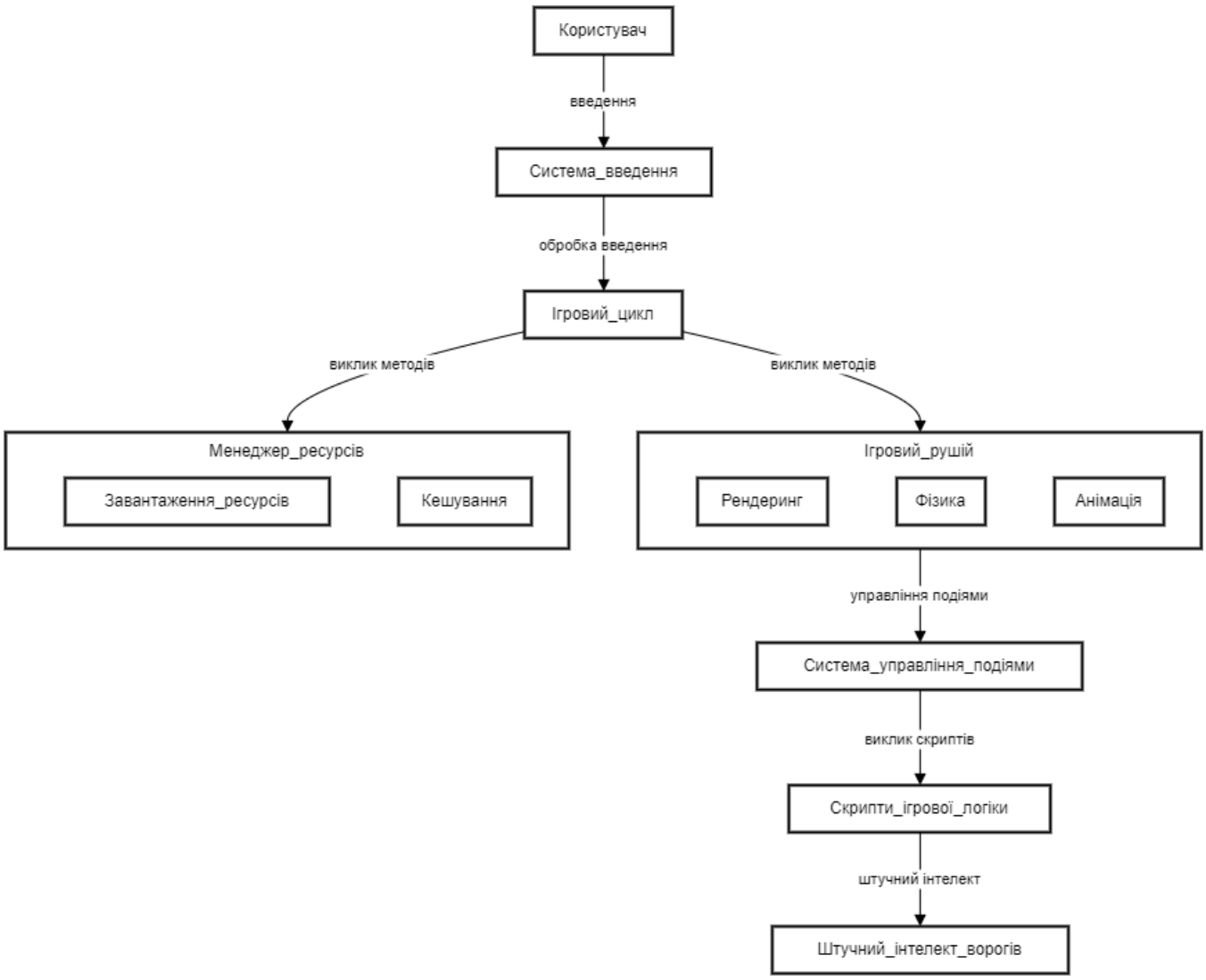

Рисунок 2.4 – Типова схема архітектури комп’ютерної гри

2.2 Розробка взаємодії функціональних елементів програмного модуля комп’ютерної гри

У цьому розділі ми розглянемо, як основні функціональні елементи програмного модуля 2D шутера взаємодіють між собою. Взаємодія компонентів є важливим аспектом архітектури гри, що забезпечує її безперебійну роботу та високу продуктивність.

Опис основних елементів:

PlayerController - включає опис основних функцій, які він виконує, а також приклад коду, що демонструє, як цей контролер може бути реалізований у Unity. Основні функції PlayerController.cs включають: рух гравця (Movement), який

контролює рухи гравця за допомогою клавіатури або геймпада, а також обробляє фізичні взаємодії, такі як зіткнення; стрільбу (Shooting), яка відповідає за створення снарядів, їх запуск та обробку попадань; здоров'я (Health), що управляє здоров'ям гравця, включаючи отримання шкоди та відображення стану здоров'я на екрані; анімації (Animations), які керують анімаціями гравця, такими як біг, стрибки, стрільба.

Код містить поля та змінні, які визначають параметри руху, стрибка, стрільби та здоров'я, а також компоненти Rigidbody2D, Animator та стан на землі. Методи включають Start(), який ініціалізує компоненти та здоров'я; Update(), який викликає методи для руху, стрибка та стрільби кожен кадр; Move(), який обробляє горизонтальний рух та анімації; Jump(), який обробляє стрибок та анімації; Shoot(), який створює та запускає снаряди; Flip(), який змінює напрямок гравця; OnCollisionEnter2D(), який визначає колізії для обробки стану на землі; TakeDamage(int damage), який обробляє отримання шкоди та перевірку смерті; Die(), який обробляє смерть гравця, включаючи анімації та перезавантаження рівня. Цей контролер охоплює основні аспекти керування гравцем і може бути розширений для додавання нових функцій або поліпшення існуючих.

EnemyController - включає основні функції, які він виконує, а також приклад коду, що демонструє, як цей контролер може бути реалізований у Unity. Основні функції EnemyController.cs включають рух ворога, який визначає поведінку руху ворога, наприклад, патрулювання, переслідування гравця або випадковий рух; атаку, яка обробляє атаки ворога, включаючи наближення до гравця та нанесення шкоди; здоров'я, яке управляє здоров'ям ворога, включаючи отримання шкоди та смерть; анімації, які керують анімаціями ворога, такими як рух, атака та смерть.

Код містить поля та змінні, які визначають параметри руху, стрибка, стрільби та здоров'я, а також компоненти Rigidbody2D, Animator та стан на землі. Методи включають Start(), який ініціалізує компоненти, пошук гравця за тегом та встановлення початкового здоров'я; Update(), який викликає методи для руху та перевірки можливості атаки кожен кадр; Move(), який обробляє рух ворога, включаючи переслідування гравця, якщо він виявлений у межах заданого радіусу;

CheckForAttack(), який перевіряє відстань до гравця та кулдаун атаки для визначення можливості атаки; Attack(), який викликає анімацію атаки та наносить шкоду гравцю; TakeDamage(int damage), який обробляє отримання шкоди та перевірку смерті ворога; Die(), який обробляє смерть ворога, включаючи виклик анімації та видалення об'єкта з сцени. Цей контролер охоплює основні аспекти керування ворогами і може бути розширений для додавання нових функцій або поліпшення існуючих.

GameManager - виконує центральну роль у координації та управлінні різними аспектами гри, такими як управління рівнями, збереження та завантаження стану гри, відстеження стану гравця та інших об'єктів, а також взаємодія з користувачем через інтерфейс. GameManager зазвичай є Singleton класом, що гарантує наявність лише одного екземпляру протягом усього ігрового процесу.

instance: GameManager – це є статична змінна, яка зберігає єдиний екземпляр GameManager. Використання статичної змінної дозволяє забезпечити доступ до GameManager з будь-якої частини коду, що важливо для централізованого управління грою. Цей атрибут зазвичай ініціалізується у методі Awake() або Start().
currentLevel: int - Змінна, яка зберігає номер поточного рівня гри. Вона може використовуватися для відображення прогресу гравця або для завантаження відповідного контенту рівня. Значення цієї змінної може встановлюватися на початку гри та змінюватися при переході на новий рівень.

Метод startGame() викликається для початку гри або нового рівня. Він може виконувати різні дії, такі як ініціалізація рівня, налаштування параметрів гри та виклик методів інших компонентів для підготовки до гри. Типові дії включають завантаження даних рівня, ініціалізацію PlayerController, ініціалізацію ворогів через EnemyController, відтворення музики або звуків через AudioManager, та оновлення інтерфейсу користувача через UIManager.

Метод endGame() викликається для завершення гри. Він може виконувати різні дії, такі як зупинка всіх процесів гри, збереження результатів та відображення екрану завершення гри. Типові дії включають зупинку всіх рухів і дій персонажів, виклик методу showGameOverScreen() з UIManager, та відтворення відповідного

звуку або музики через AudioManager.

Основні функції GameManager включають управління станом гри, яке забезпечує перехід між різними станами гри, такими як головне меню, гра, пауза, кінцевий екран. Це включає ініціалізацію необхідних об'єктів та їх дезактивацію залежно від поточного стану. Управління рівнями, що включає завантаження, перезавантаження та перехід між рівнями. Це передбачає створення ігрових об'єктів, ініціалізацію позицій гравця та ворогів, а також завантаження необхідних ресурсів для кожного рівня.

Збереження та завантаження стану гри є важливою функцією, що дозволяє гравцям зберігати свій прогрес і повертатися до гри з того ж місця. Це може включати збереження поточного рівня, позиції гравця, стану ворогів та інших об'єктів, а також різних ігрових параметрів. Взаємодія з користувачем через інтерфейс є ще однією важливою функцією, що забезпечує відображення інформації на екрані, такої як очки, здоров'я, інструкції та меню. GameManager часто взаємодіє з UI елементами для оновлення цієї інформації в реальному часі.

AudioManager - виконує роль центрального компонента, відповідального за управління усіма звуковими ефектами та музикою в грі. Він дозволяє централізовано контролювати відтворення, зупинку та налаштування звуків, що допомагає зберігати порядок та організованість у проекті. AudioManager зазвичай реалізується як Singleton, що забезпечує його доступність з будь-якої частини коду та гарантує, що в грі існує лише один екземпляр цього менеджера. Метод Awake() перевіряє, чи існує вже екземпляр AudioManager, і якщо так, знищує зайвий екземпляр. DontDestroyOnLoad(gameObject) гарантує, що AudioManager не буде знищено при завантаженні нових сцен, що дозволяє зберігати його стан між рівнями.

Методи PlayMusic(int index) та PlaySFX(int index) відповідають за відтворення музики та звукових ефектів відповідно. Вони отримують індекс аудіо кліпу у масиві та відтворюють відповідний кліп за допомогою AudioSource. Методи StopMusic() та StopSFX() зупиняють відтворення музики та звукових ефектів. Методи SetMusicVolume(float volume) та SetSFXVolume(float volume) дозволяють змінювати

гучність музики та звукових ефектів окремо.

UIManager - виконує роль центрального компонента, відповідального за управління усіма елементами користувацького інтерфейсу (UI) у грі. Це включає відображення та оновлення інформації на екрані, такої як очки, здоров'я, інструкції, меню, повідомлення та інші UI елементи, що взаємодіють з гравцем. UIManager часто реалізується як Singleton, щоб забезпечити легкий доступ до нього з будь-якої частини коду та гарантувати, що існує лише один екземпляр цього менеджера.

Основні функції UIManager включають відображення та оновлення інформації, управління меню, відображення повідомлень та інструкцій, взаємодію з гравцем, а також анімації та візуальні ефекти. Відображення та оновлення інформації охоплює параметри, такі як очки, здоров'я, рівень енергії, кількість зібраних предметів та інших ігрових показників. UIManager взаємодіє з текстовими елементами, прогрес-барами та іншими UI компонентами, щоб своєчасно оновлювати ці показники відповідно до змін у грі. Управління меню включає відображення та приховування різних меню, таких як головне меню, меню паузи, настройки та кінцеві екрани гри, обробляючи перехід між цими меню для забезпечення плавного та зручного інтерфейсу для гравця. Відображення повідомлень та інструкцій може включати короткі повідомлення, що з'являються на екрані під час гри, такі як підказки, попередження або інструкції для гравця.

UIManager відповідає за своєчасне відображення та зникнення таких повідомлень. Взаємодія з гравцем охоплює обробку вводу користувача через UI елементи, такі як кнопки, повзунки та поля вводу, включаючи обробку кліків, зміни значень та інших дій, які гравець виконує через інтерфейс. UIManager також може відповідати за анімації та візуальні ефекти, пов'язані з UI елементами, такі як плавне зникнення та з'явлення елементів, зміна кольорів, масштабування та інші ефекти, що покращують взаємодію з гравцем.

SaveSystem - виконує важливу роль у збереженні та відновленні стану гри, дозволяючи гравцям зберігати свій прогрес і відновлювати гру з попереднього моменту. Це включає збереження інформації про рівень, здоров'я гравця, кількість зібраних предметів та інші важливі параметри, які визначають поточний стан гри.

Збережені дані можуть бути розміщені на локальному пристрої або в хмарному сховищі, що забезпечує гнучкість у доступі до них.

SaveSystem також відповідає за відновлення гри з збережених даних, що дозволяє гравцям продовжити гру з того місця, де вони зупинилися. Це особливо важливо для ігор з великим обсягом контенту або високою складністю, де гравці можуть витрачати багато часу на досягнення певних цілей.

Окрім основних функцій збереження і відновлення, SaveSystem може підтримувати множинні слоти для збереження, що дозволяє гравцям зберігати кілька різних ігор одночасно. Це особливо корисно для гравців, які хочуть експериментувати з різними стратегіями або мати кілька паралельних проходжень гри.

Ще однією важливою функцією SaveSystem є збереження налаштувань гри, таких як налаштування графіки, звуку та управління. Це дозволяє гравцям зберігати свої індивідуальні налаштування і автоматично застосовувати їх при кожному запуску гри, що забезпечує комфортний і послідовний ігровий досвід.

.

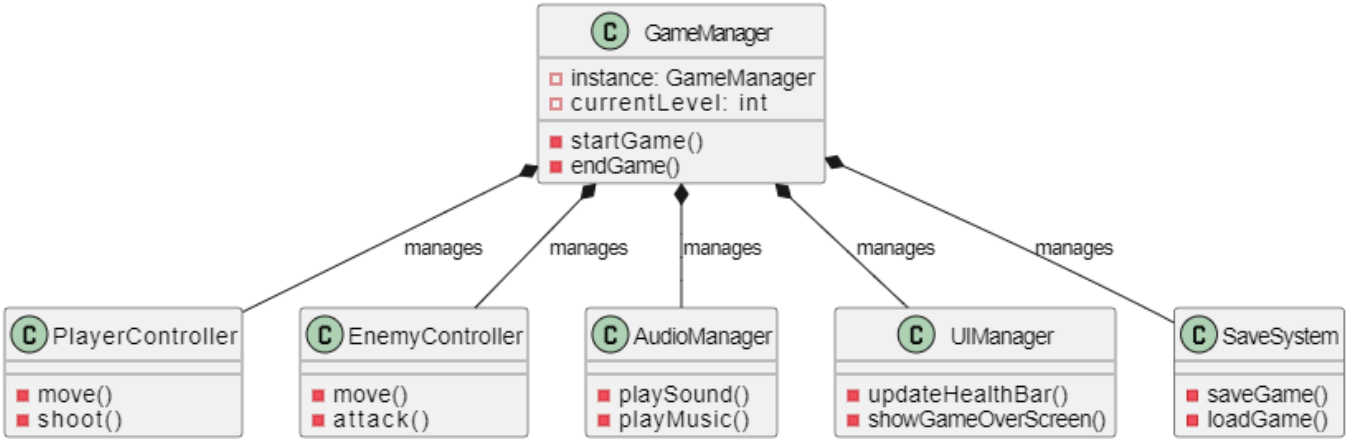

Рисунок 2.5 – Типова схема взаємодії між функціональними елементами комп’ютерної гри

2.3 Розробка алгоритму роботи програмного модуля комп’ютерної гри

Розглянем модель взаємодії гравця і ворога. Гра починається з того, що створюється ворог, який одразу ж починає рухатись у напрямку до гравця. Це

створює загрозу, на яку гравець має реагувати. Коли ворог наближається на певну відстань, гравець може розпочати атаку. Для цього у гравця є кілька варіантів: він може вибрати між мечем для ближнього бою або автоматом для дальнього бою. Вибір зброї залежить від ситуації та стратегії гравця.

Якщо гравець успішно знищує ворога, йому додаються очки до рахунку. Ці очки можуть використовуватися для визначення прогресу в грі або для відкриття нових можливостей та зброї. Гра триває до тих пір, поки у гравця залишаються очки здоров'я. Кожного разу, коли ворог атакує, гравець втрачає частину своїх очок здоров'я. Коли гравець втрачає останнє очко здоров'я, гра закінчується, і гравець або програє, або має починати гру знову, або завантажити збережену гру. Блок-схема яка продемонстроваа далі допомагає зрозуміти основну логіку ігрового процесу, від створення ворога до кінцевого результату гри. Вона показує ключові моменти взаємодії між гравцем і ворогом, що є важливою частиною проектування гри, оскільки дозволяє розробникам чітко уявити послідовність подій та можливі дії гравця у різних ситуаціях.

Додаткові можливості для розширення моделі:

1) Різні типи ворогів: Можна додати різних ворогів з унікальними характеристиками (швидкість, сила, здоров'я).
2) Рівні складності: Впровадження рівнів складності, де кількість ворогів, їх сила та швидкість збільшуються.
3) Бонуси та перешкоди: Додавання бонусів (збільшення здоров'я, додаткова зброя) та перешкод (пастки, лабіринти).
4) Кілька життів: Надання гравцеві кількох життів, щоб продовжити гру після втрати здоров’я.

Алгоритм роботи програмного модуля комп'ютерної гри "Phoenix", який зображено на рисунку 2.6 можна описати наступним чином:

Ініціалізація:

1) Ігровий менеджер створює та ініціалізує всі необхідні компоненти гри.
2) Завантажується початкова сцена гри.

Основний цикл гри:

1) Обробка введення користувача: Менеджер введення зчитує дані з клавіатури, миші та інших пристроїв введення.
2) Оновлення стану гри: Ігровий менеджер оновлює стан всіх ігрових об'єктів та компонентів.
3) Обробка фізики: Фізичний рушій обчислює фізичні взаємодії між об'єктами.
4) Рендеринг: Графічний рушій відображає ігрову сцену на екрані.
5) Обробка звуку: Аудіосистема відтворює звукові ефекти та музику.

Завершення гри:

1) Зберігається стан гри (за потреби).
2) Звільняються всі ресурси, що використовуються грою.

Рисунок 2.6 – Загальна схема алгоритму роботи циклу програмного модуля

Загальна концепція функціонування комп'ютерної гри "Phoenix", включаючи порядок етапів, взаємозв'язки між компонентами та можливі напрямки розвитку ігрового процесу, детально зображена на рисунку 2.7. Ця схема допомагає зрозуміти основні логічні етапи та їх взаємодію, що важливо для правильного проектування та програмування гри "Phoenix".

1. Початок (Start):
    1) Початковий блок (овал), який позначає старт програми. Це точка входу, де починається виконання програмного коду.
    2) З цього блоку стрілка веде до блоку "Меню".
2. Меню (Menu):
    1) Ромб, який позначає пункт прийняття рішення.
    2) Користувач має три можливості:
        1. Налаштування (Settings): Прямокутник, який позначає перехід до налаштувань програми. Це місце, де користувач може змінювати параметри гри, такі як графіка, звук, управління тощо. Після внесення змін користувач повертається до меню.
        2. Вихід (Exit): Прямокутник, який позначає завершення роботи програми. При виборі цього пункту програма завершує свою роботу, і стрілка веде до блоку "Кінець".
        3. Початок гри (Start Game): Прямокутник, який позначає перехід до початку гри. Це включає ініціалізацію ігрових параметрів та підготовку до старту.
3. Налаштування (Settings):
    1) Прямокутник, до якого веде стрілка з блоку "Меню".
    2) Користувач може змінити налаштування гри. Після цього він повертається назад до меню (стрілка з налаштувань веде назад до меню).
4. Вихід (Exit):
    1) Прямокутник, до якого веде стрілка з блоку "Меню".
    2) При виборі цього пункту програма завершує свою роботу, і стрілка веде до блоку "Кінець".

5. Початок гри (Start Game):
    1) Прямокутник, до якого веде стрілка з блоку "Меню".
    2) Цей блок позначає початок ігрового процесу. Після його вибору, гра переходить до наступного блоку "Створення карти".
6. Створення карти (Map Creation):
    1) Прямокутник, до якого веде стрілка з блоку "Початок гри".
    2) На цьому етапі відбувається створення ігрової карти. Це може включати генерацію ігрового середовища, розміщення об'єктів, налаштування початкових умов тощо.
7. Процес гри (Game Process):
    1) Прямокутник, до якого веде стрілка з блоку "Створення карти".
    2) Основний ігровий процес. На цьому етапі користувач безпосередньо взаємодіє з грою, виконуючи дії, досягаючи цілей, взаємодіючи з іншими персонажами або об'єктами гри.
8. Кінець гри (End Game):
    1) Прямокутник, до якого веде стрілка з блоку "Процес гри".
    2) Після завершення ігрового процесу (програш або виграш у грі), користувач повертається до меню.
9. Кінець (End):
    1) Овал, до якого веде стрілка з блоку "Вихід".
    2) Це кінцева точка програми. Вона позначає завершення виконання коду і вихід з програми.

Ця блок-схема представляє основний цикл роботи програмного забезпечення з трьома основними напрямками в меню (налаштування, початок гри, вихід) і деталізацією ігрового процесу від створення карти до закінчення гри.

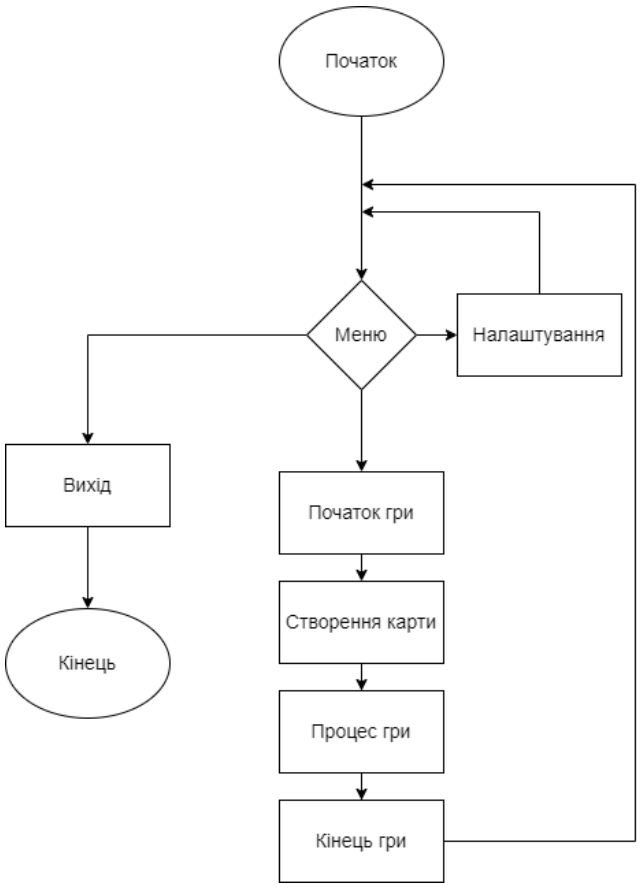

Рисунок 2.7 – Схема алгоритму роботи програмного модуля комп'ютерної гри.

Блок-схема функціональної моделі персонажу (рис. 2.8) описує керування ігровим персонажем в комп'ютерній грі. Вона деталізує, як гравець може контролювати персонажа за допомогою клавіатури.

Рисунок 2.8 – Типова схема функціональної моделі керування ігровим персонажем

Ігровий персонаж керується натисканням клавіш клавіатури. Ці натискання можуть виконувати різні дії:

1) Переміщення персонажа: Гравець може переміщувати персонажа в різних напрямках: праворуч, ліворуч і вгору. Це досягається натисканням відповідних клавіш для переміщення.
2) Атака: Гравець може атакувати ворогів. Атака здійснюється через натискання клавіші, яка відповідає за атаку.

Опис елементів схеми які зображено на рисунку 2.9:

1 – Кількість одиниць здоров’я

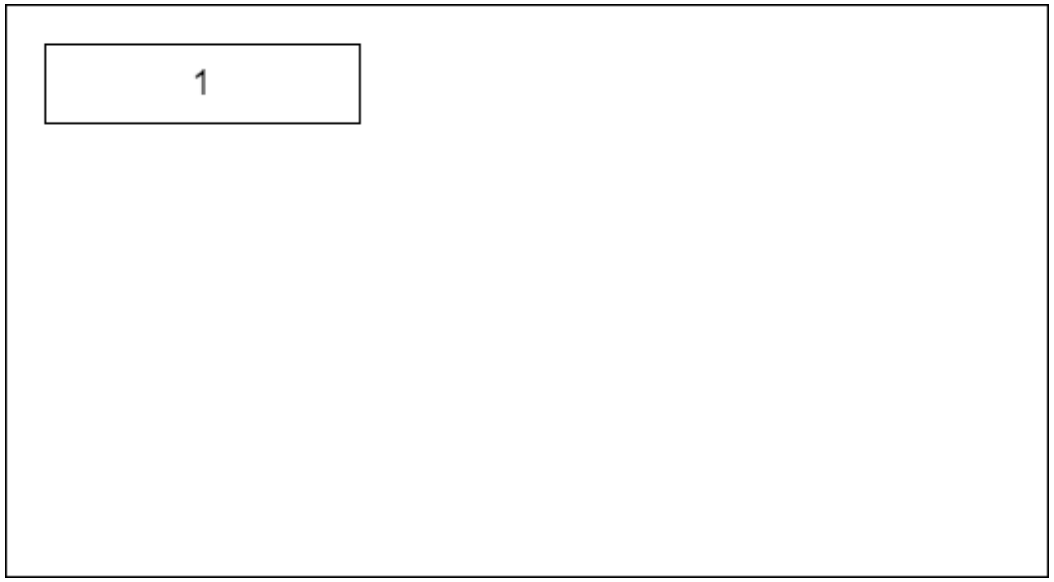

Рисунок 2.9 – Конструктивна схема користувацького інтерфейсу

Опис елементів схеми які зображено на рисунку 2.10:

1 – Назва гри.

2 – Кнопка Play.

3 – Кнопка Options.

4 – Кнопка Exit.

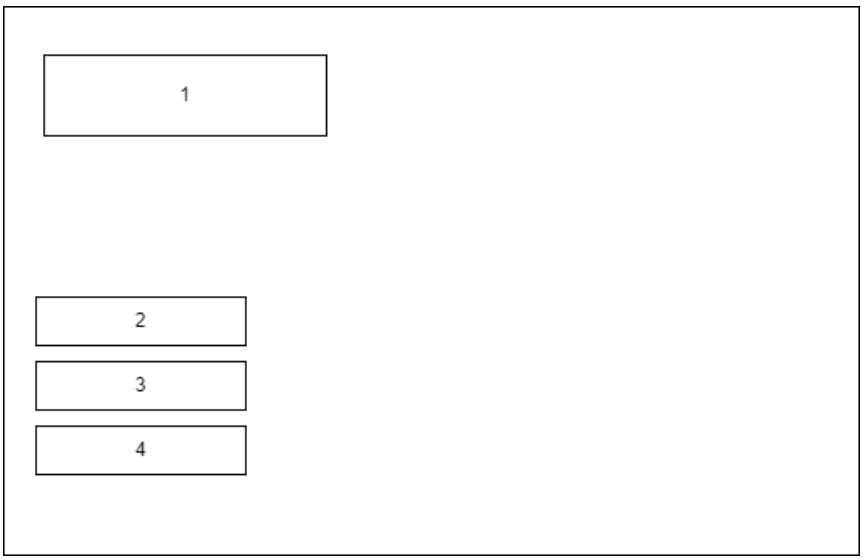

Рисунок 2.10 - Конструктивна схема зображення головного меню

2.4 Висновок до розділу 2

У цьому розділі була розроблена чітка архітектура програмного забезпечення, де кожен компонент мав свою відповідальність, що забезпечило модульність, масштабованість та ефективність гри. Далі було спроектовано взаємодію між елементами гри, встановивши зв'язки та механізми обміну даними, що гарантувало злагоджену роботу всіх частин та реалізацію ігрової логіки. Був розроблений алгоритм роботи, який покроково описував основні процеси гри, включаючи ініціалізацію, основний цикл та завершення. Це дозволило реалізувати ігровий процес, обробку введення, оновлення стану, фізику, рендеринг та інші важливі аспекти.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО МОДУЛЯ КОМП'ЮТЕРНОЇ ГРИ

3.1 Обґрунтування вибору мови програмування та середовища розробки

Visual Studio є потужним інтегрованим середовищем розробки (IDE), яке надає розробникам широкий набір інструментів для створення програмного забезпечення, включаючи ігри [17]. Ось детальний огляд, як Visual Studio може бути корисним для розробки ігор та його основні переваги:

Переваг використання Visual Studio для розробки ігор:

- Потужне середовище розробки: Visual Studio пропонує розширені інструменти для написання коду, включаючи автозаповнення, рефакторинг коду, інтеграцію з системами контролю версій (наприклад, Git), і відлагодження. Це робить процес розробки більш ефективним та зручним.
- Інтеграція з Unity та Unreal Engine: Visual Studio повністю інтегрований з популярними ігровими двигунами, такими як Unity та Unreal Engine. Це дозволяє розробникам писати та налагоджувати код безпосередньо в Visual Studio, що спрощує процес розробки ігор.
- Підтримка кількох мов програмування: Visual Studio підтримує різні мови програмування, включаючи C#, C++, Python та інші. Це дозволяє розробникам вибирати мову, яка найкраще підходить для їхнього проекту.
- Розширюваність та налаштовуваність: Visual Studio має величезну кількість плагінів і розширень, які можна використовувати для додавання нових функцій або налаштування середовища розробки під власні потреби.
- Потужні засоби відлагодження та тестування: Visual Studio пропонує потужні інструменти для відлагодження та тестування, що дозволяє знаходити та виправляти помилки на ранніх етапах розробки.
- Підтримка Azure DevOps та інших інструментів DevOps: Інтеграція з Azure DevOps дозволяє налаштовувати CI/CD процеси, автоматизувати тестування та розгортання, що є важливим для великих ігрових проектів.

- Документація та спільнота: Visual Studio має велику базу документації та активну спільноту розробників, що може бути корисним для отримання допомоги та навчання.

C++ є однією з найбільш популярних мов програмування для розробки ігор завдяки своїм високим показникам продуктивності та контролю над апаратним забезпеченням. Висока продуктивність C++ дозволяє розробникам максимально ефективно використовувати апаратне забезпечення, що особливо важливо для ігор, які потребують великої кількості обчислень і швидкого рендерингу графіки. Контроль над пам'яттю в C++ надає можливість детального управління пам'яттю, що дозволяє оптимізувати використання ресурсів і уникати зайвих витрат пам'яті.

Об'єктно-орієнтоване програмування (ООП), яке підтримує C++, дозволяє розробникам створювати більш структурований і модульний код, спрощуючи розробку складних ігор, які складаються з багатьох взаємодіючих компонентів. Існує багато ігрових рушіїв і бібліотек, написаних на C++ або підтримуючих C++, таких як Unreal Engine, CryEngine, Ogre3D та інші, що надає розробникам доступ до потужних інструментів і ресурсів, які можуть значно спростити процес розробки. C++ є кросплатформною мовою, що дозволяє розробникам створювати ігри для різних операційних систем і платформ, таких як Windows, macOS, Linux, ігрові консолі та мобільні пристрої.

Завдяки цим характеристикам C++ залишається одним із найкращих виборів для професійних розробників ігор, особливо тих, хто прагне досягти високого рівня продуктивності та якості у своїх проектах.

C# є одним з основних мов програмування, що використовується для розробки ігор на платформі Unity [18]. Ось деякі з його переваг:

- Сила та гнучкість: C# є сильно типізованою мовою програмування, що допомагає зменшити кількість помилок на етапі компіляції. Це дозволяє розробникам писати чистий і структурований код.
- Об'єктно-орієнтоване програмування (ООП): C# підтримує всі принципи ООП, такі як наслідування, поліморфізм, інкапсуляція та абстракція, що

полегшує моделювання реальних об'єктів у грі та сприяє повторному використанню коду.

- Розширюваність: Завдяки використанню скриптів на C#, Unity дозволяє легко додавати нові функції та модифікувати поведінку гри без потреби змінювати основний код двигуна.
- Потужні інструменти відлагодження: Visual Studio та інші IDE для C# надають розширені можливості відлагодження, такі як брекпойнти, перевірка змінних у реальному часі, що значно полегшує процес розробки та виправлення помилок.
- Широка спільнота: Завдяки популярності C# і Unity, існує велика спільнота розробників, яка може надати підтримку, поділитися знаннями та прикладами коду, що особливо корисно для початківців.
- Підтримка сторонніх бібліотек та фреймворків: C# має широкий вибір бібліотек і фреймворків, що дозволяє розширювати функціонал гри, інтегрувати сторонні сервіси та прискорювати процес розробки.
- Кросплатформенність: За допомогою Unity на C# можна створювати ігри для різних платформ, включаючи Windows, macOS, iOS, Android, та консолі, що робить її ідеальним вибором для розробки кросплатформних проектів.
- Підтримка асинхронного програмування: C# дозволяє легко реалізувати асинхронне програмування, що є корисним для обробки задач, які можуть тривати довгий час, таких як завантаження даних або комунікація з сервером.
- Автоматичне керування пам'яттю**:** Збирач сміття (garbage collector) звільняє розробника від необхідності вручну керувати пам'яттю, що знижує ризик витоків пам'яті та спрощує розробку.
- LINQ: інноваційний компонент платформи .NET, який надає розробникам потужний та зручний інструмент для роботи з даними безпосередньо в коді C#.

Порівняльна таблиця двох мов наведена у таблиці 3.1.

Таблиця 3.1 – Порівняльна таблиця мов програмування

Характеристика	C++		C#
Продуктивність	Висока продуктивність завдяки низькорівневому доступу до апаратного забезпечення.		Досить висока продуктивність, але часто повільніша за C++ через управління пам'яттю
Контроль над пам'яттю	Повний контроль над управлінням пам'яттю.		Автоматичне управління пам'яттю через зборку сміття (Garbage Collection).
Легкість у вивченні	Складніша в порівнянні з C# через низькорівневі концепції та синтаксис.		Легша у вивченні завдяки більш високорівневому синтаксису та більшій кількості абстракцій.
Об'єктно-орієнтованість	Підтримка ООП, проте синтаксис може бути складнішим		Потужна підтримка ООП з простішим синтаксисом.
Ігрові рушії	Підтримка рушіїв, таких як Unreal Engine, CryEngine.		Основний рушій - Unity, також підтримує інші рушії (наприклад, Godot).
Кросплатформність	Кросплатформна мова, може використовуватися на різних платформах без значних змін		Також кросплатформна, але може потребувати специфічних бібліотек та середовищ.
Спільнота та підтримка	Велика та активна спільнота, багато ресурсів та документації.		Велика спільнота Unity, багато ресурсів для початківців та професіоналів.
Розширюваність	Дуже висока, можна інтегрувати з різними системами та інструментами.		Висока, але залежить від специфіки платформи та інструментів, які використовуються.
Модульність	Висока, але потребує більше зусиль для підтримки модульності		Висока, завдяки сучасним можливостям мови та інструментів.

3.2 Програмна реалізація комп’ютерної гри

BossControl - Скрипт керує поведінкою босів у грі. Він відстежує кількість живих босів, зменшує цю кількість при вбивстві боса і активує екран перемоги, коли

всі боси знищені:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class BossControl : MonoBehaviour {
	int boss = 1;
	int bossesAlive = 1;
	public Image win;
	void Start(){
	}
	public void killABoss() {
		bossesAlive--;
	}
	public int getBossesLeft () {
		return boss;
	}
	public int getBosseAlive () {
		return bossesAlive;
	}
	public void UpdateBossesLeft(){
		boss--;
	}
	public bool isBoss(){
		int i = getBossesLeft();
		if (i > 0){
			return true;
		}
		else{
			return false;
```

```
            }
        }
        public void isGame(){
            killABoss();
            int i = getBosseAlive();
            if (i <= 0){
                win.gameObject.SetActive (true);
            }
        }
}.
```

ArrowDOWN, ArrowLEFT, ArrowRIGHT, ArrowUP - використовуються для переміщення. Коли гравець стикається з активною стрілкою, гравець і камера переміщуються в напрямку стрілки на задану відстань:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class ArrowDOWN : MonoBehaviour {
    public bool ACTIVE = false;
    public GameObject player;
    public GameObject camera;
    void Start (){
        player = GameObject.Find("Player");
        camera = GameObject.Find("Main Camera");
    }
    void OnTriggerEnter2D(Collider2D other) {
        if (ACTIVE && other.gameObject.name == "Player"){
            player.SetActive(false);
            player.transform.position = new Vector2 (player.transform.position.x ,
player.transform.position.y - 22.214f);
            player.SetActive(true);
```

```
            camera.transform.position = new Vector3 (camera.transform.position.x ,
camera.transform.position.y - 32.52f, -10f);
        }
    }
    void changeState(){
        ACTIVE = true;
    }
}.
```

ControlCamera - Скрипт контролює положення камери відносно гравця. Він зберігає початкове зміщення камери від гравця і постійно оновлює положення камери, щоб вона слідувала за гравцем:

```
using UnityEngine;
using System.Collections;
public class ControlCamera : MonoBehaviour {
    public GameObject player;
    private Vector3 offset
    void Start ()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        offset = transform.position - player.transform.position;
    }
    void LateUpdate ()
    {
        transform.position = player.transform.position + offset;
    }
}.
```

MapAssigner - Цей клас відповідає за розміщення кімнат на карті, приклад генерування наведено на рисунку 3.1. Він приймає двовимірний масив кімнат і створює екземпляри об’єктів Room на основі розташування та дверей кімнати:

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;
public class MapAssigner : MonoBehaviour {
	public GameObject RoomObj;
	public Vector2 roomDimensions = new Vector2(17,17);
	public Vector2 gutterSize = new Vector2(17,17);
	public void Assign(Roomy[,] rooms){
		foreach (Roomy room in rooms){
			if (room == null){
				continue;
			}
			//pick a random index for the array
			Vector3 pos = new Vector3(room.gridPos.x * (roomDimensions.x + gutterSize.x), room.gridPos.y * (roomDimensions.y + gutterSize.y), 0);
			Rooms myRoom = Instantiate(RoomObj, pos, Quaternion.identity).GetComponent<Rooms>();
			myRoom.Setup(room.gridPos, room.doorTop, room.doorBot, room.doorLeft, room.doorRight);
		}
	}
}.
```

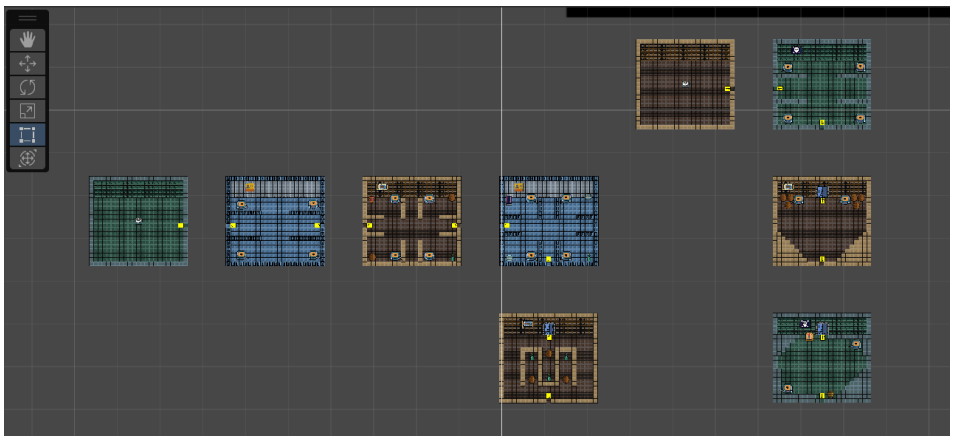

Рисунок 3.1 – Загальний вигляд інтерфейсного вікна генерування ігрових сцен типу «карти»

3.3 Тестування програмного модуля комп’ютерної гри

Для запуску гри потрібно на головному меню натиснути на кнопку Play (рис. 3.2).

Рисунок 3.2 – Загальний вигляд інтерфейсного вікна типу “Головного меню”

Після натиснення починається гра, щоб виграти у грі потрібно знищити головного боса, але щоб до нього дійти потрібно ходити по різним кімнатам і знищувати слабших ворогів (рис. 3.3). Також в різних кімнатах є бонуси, наприклад новий вид зброї або додаткова одиниця здоров’я. Одиниці здоров’я зменшуються при попаданні по гравцю кулею, якщо здоров’я повністю закінчаться гра буде програною (рис. 3.5).

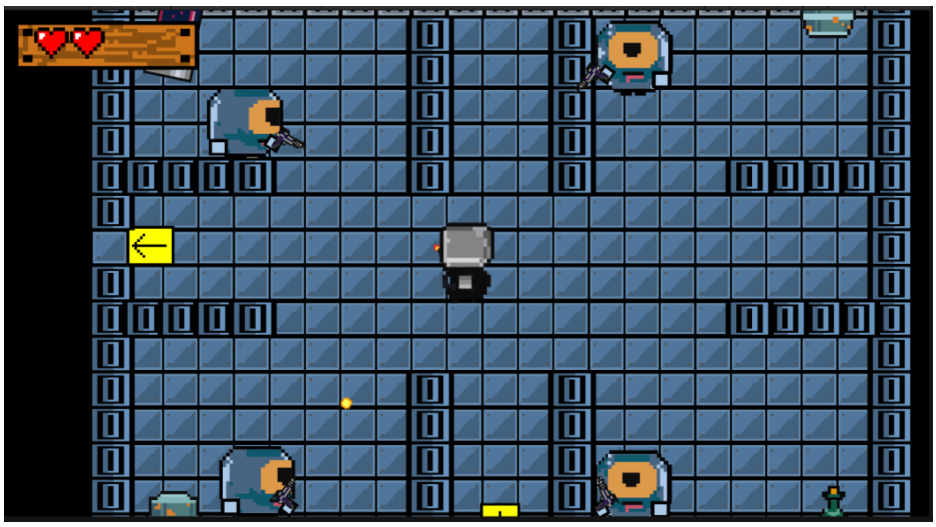

Рисунок 3.3 – Загальний вигляд комп’ютерної гри Phoenix

Рисунок 3.4 – Загальний вигляд закінчення гри при виграші

Рисунок 3.5 – Загальний вигляд закінчення гри при поразці

Порівнявши розроблений модуль гри а також модуль гри аналога Enter the Gungeon можна зробити висновок що модуль гри Phoenix використовує менше ресурсів оперативної пам’яті та центрального процесора тобто є оптимізованішим ніж аналог (табл. 3.1).

Таблиця 3.1 - Порівняння програмних модулів ігор

	Enter the gungeon	Phoenix
Середній показник використання оперативної пам’яті	1123,4 Мб	321,8 Мб
Середній показник використання процесора	76%	40%

3.4 Висновок до розділу 3

У третьому розділі було успішно створено та налаштовано проект Unity для розробки 2D шутера "Phoenix". Було розроблено декілька сцен, включаючи головне меню, ігрові рівні та екран завершення гри. Графіка гри була створена з використанням спрайтів та анімацій, що забезпечують візуальну привабливість та динаміку ігрового процесу.

Важливим етапом було створення скриптів на мові C#, які реалізують основну логіку гри, включаючи рух персонажів, стрільбу, взаємодію з об'єктами та управління ігровим світом. Був реалізований програмний модуль комп’ютерної гри "Phoenix". Було проведено тестування програмного модуля, яке підтвердило його коректну роботу та відповідність поставленим вимогам.

ВИСНОВОК

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

- було досліджено різноманіття жанрів комп’ютерних ігор, що дозволило визначити їх основні характеристики та популярність серед гравців;
- проведено огляд сучасних ігрових рушіїв. Визначено їхні сильні та слабкі сторони, а також особливості, що роблять їх придатними для різних типів ігор;
- розроблено архітектурний план гри, що включав основні компоненти та їх взаємодію;
- створено алгоритм для одного з ключових модулів гри, який визначає логіку його роботи та взаємодію з іншими модулями;
- здійснено програмну реалізацію розробленого алгоритму;
- проведено тестування програмного модуля для виявлення можливих помилок.

У ході проведеного дослідження було комплексно проаналізовано різноманіття аспектів розробки комп'ютерних ігор. Вивчення жанрів комп'ютерних ігор дозволило виявити основні тенденції та вподобання гравців. Огляд сучасних ігрових рушіїв допоміг визначити їх сильні та слабкі сторони, що є критично важливим для вибору оптимального інструментарію для розробки конкретної гри. Створення архітектурного плану гри забезпечило основу для подальшої розробки, визначивши основні компоненти та їх взаємодію. Розробка алгоритму ключового модуля та його програмна реалізація стали важливим кроком у створенні функціональної гри. Проведене тестування програмного модуля дозволило виявити та виправити потенційні помилки, забезпечуючи стабільність та якість гри.

Мета роботи – підвищенню ефективності тренування тактичного мислення користувача за допомогою розробки та використання програмного модуля 2D комп’ютерної гри жанру «шутер» з компонентами розвитку зорової уваги та швидкості прийняття тактичних рішень, яка сприятиме тренуванню швидкості прийняття рішень гравцем в екстремальних умовах. Це було досягнути за рахунок оптимізації та удосконалення програмного модуля комп’ютерної гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What’s the Most Popular Gaming Platform [Електронний ресурс]. - Режим доступу: https://pinglestudio.com/blog/industry-news/whats-the-most-popular-gaming-platform-in-2023.
2. Video game statistics [Електронний ресурс]. - Режим доступу: https://www.statista.com/outlook/dmo/digital-media/videogames/worldwide.
3. Нагорний М. Р., Іванчук Я. В. «Функціональний програмний модуль комп’ютерної гри»: матеріали конференції LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024). URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20913.
4. Жанри ігор [Електронний ресурс]. - Режим доступу: https://avada-media.ua/ua/services/vibor-zhanra-igry/.
5. What Is A Role-Playing Game? RPG Types Explained [Електронний ресурс]. - Режим доступу: https://www.forbes.com/sites/technology/article/what-are-rpg-games/?sh=60bc672e3c7d.
6. Simulation Games [Електронний ресурс]. - Режим доступу: https://blog.acer.com/en/discussion/254/simulation-games-and-simulators.
7. Most popular video game genres [Електронний ресурс]. - Режим доступу: https://www.statista.com/statistics/1263585/top-video-game-genres-worldwide-by-age/.
8. Ігровий рушій [Електронний ресурс]. - Режим доступу: https://www.wikidata.uk-ua.nina.az/%D0%A0%D1%83%D1%88%D1%96%D0%B9_%D0%B3%D1%80%D0%B8.html.
9. Unity Game Engine [Електронний ресурс]. - Режим доступу: https://stackinterface.com/unity-game-engine/.
10. Unity Asset Server [Електронний ресурс]. - Режим доступу:

https://docs.unity3d.com/2019.4/Documentation/Manual/AssetServer.html.

11. Godot [Електронний ресурс]. - Режим доступу: https://docs.godotengine.org/uk/4.x/getting_started/introduction/introduction_to_godot.html.
12. CryEngine [Електронний ресурс]. - Режим доступу: https://la.by/game-engines/cryengine.
13. Shooter game [Електронний ресурс]. - Режим доступу: https://en.wikipedia.org/wiki/Shooter_game.
14. Монолітна архітектура [Електронний ресурс]. - Режим доступу: https://camunda.com/blog/2023/08/monolith-vs-microservice-architecture-comparison/.
15. Компонентно-орієнтована архітектура [Електронний ресурс]. - Режим доступу: https://www.researchgate.net/figure/A-reference-architecture-for-the-games-domain_fig1_221406953.
16. Модель-вид-контролер [Електронний ресурс]. - Режим доступу: https://www.wikiwand.com/uk/Модель-вид-контролер
17. What is Visual Studio? [Електронний ресурс]. - Режим доступу: https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022.
18. C# [Електронний ресурс]. - Режим доступу: https://www.tutorialspoint.com/csharp/index.html.

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний модуль комп'ютерної гри «Phoenix»

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

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

Показники звіту подібності Unicheck

Оригінальність 96,27% Схожість 3,73%

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 Нагорний М.Р.

Керівник роботи Іванчук Я.В.

ДОДАТОК Б

Інструкція користувача

Щоб запустити гру потрібно виконати декілька пунктів:

1. Встановити Unity з офіційного сайту
2. Додати проект гри до Unity hub рисунок Б.1.

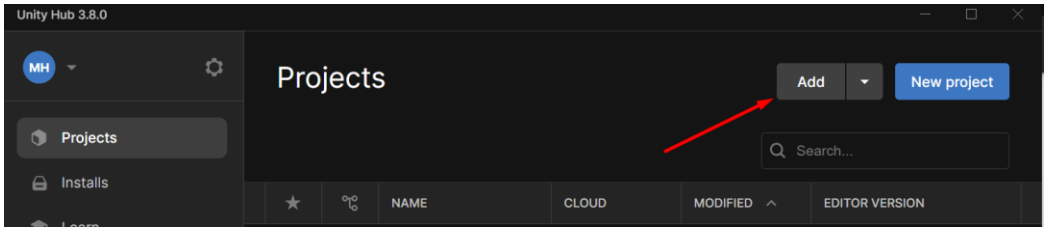

Рисунок Б.1 – Додання проекту до Unity hub

3. Після додання проекту користувач відкриває його в Unity і може розпочати гру натиснувши кнопку play рисунок Б.2.

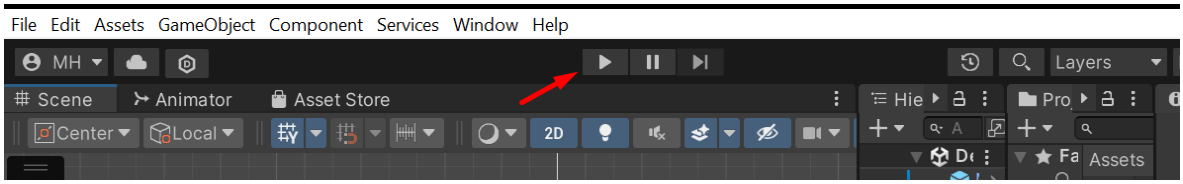

Рисунок Б.2 – Інтерфейс Unity

4. Гра завершується коли користувач програє або виграє у грі

ДОДАТОК В

Фрагмент лістингу програмного коду

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

public class BossControl : MonoBehaviour {
	int boss = 1;
	int bossesAlive = 1;

	public Image win;

	void Start(){
	}
	public void killABoss() {
		bossesAlive--;
	}

	public int getBossesLeft () {
		return boss;
	}
	public int getBosseAlive () {
		return bossesAlive;
	}

	public void UpdateBossesLeft(){
		boss--;
	}
	public bool isBoss(){
		int i = getBossesLeft();
		if (i > 0){
			return true;
		}
		else{
			return false;
		}
	}
	public void isGame(){
		killABoss();
		int i = getBosseAlive();
		if (i <= 0){
			win.gameObject.SetActive (true);
		}
	}

}


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

public class ArrowDOWN : MonoBehaviour {

    public bool ACTIVE = false;
    public GameObject player;
    public GameObject camera;
```

```
    void Start (){
        player = GameObject.Find("Player");
        camera = GameObject.Find("Main Camera");
    }
    void OnTriggerEnter2D(Collider2D other) {
        if (ACTIVE && other.gameObject.name == "Player"){
            player.SetActive(false);
            player.transform.position = new Vector2 (player.transform.position.x ,
player.transform.position.y - 22.214f);
            player.SetActive(true);
            camera.transform.position = new Vector3 (camera.transform.position.x ,
camera.transform.position.y - 32.52f, -10f);
        }
    }

    void changeState(){
        ACTIVE = true;
    }
}


using UnityEngine;
using System.Collections;

public class ControlCamera : MonoBehaviour {

    public GameObject player;


    private Vector3 offset

    void Start ()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        .
        offset = transform.position - player.transform.position;
    }


    void LateUpdate ()
    {
        transform.position = player.transform.position + offset;
    }
}




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

public class Enemies : MonoBehaviour {


       public float speed;


       public float scale_X;
       public float scale_Y;
```

```
	public Sprite frontSprite;
	public Sprite backSprite;


	public bool right = true;
	public bool front = true;


	Animator anim;
	const float timeout = 4.0f;


	public GameObject player;


	int life = 2;
	int  destroyTime = 3;


	bool disparo;
	bool dead = false;



	void Start () {

		player = GameObject.FindGameObjectWithTag ("Player");
		anim = gameObject.GetComponent<Animator> ();

	}


	void Update () {

		Vector3 mov = new Vector3 (
			0.01f,
			0.01f,
			0
		);


		if (!dead){
			if (Vector2.Distance (transform.localPosition, player.transform.position)
< 5){
				disparo = true;
				speed = 0;
				anim.SetBool ("stand", false);
			}


			else if (Vector2.Distance (transform.localPosition,
player.transform.position) > 5 && Vector2.Distance (transform.localPosition,
player.transform.position) < 15) {
				disparo = true;
				speed = 2.0f;
				transform.position = Vector3.MoveTowards (transform.localPosition,
player.transform.localPosition + mov, speed * Time.deltaTime);
				anim.SetBool ("stand", false);

			}
```

```
			else{
				disparo = false;
				anim.SetBool ("stand", true);
			}
		}else{
			speed = 0;
		}
		anim.SetFloat ("Axis_X", mov.x);
		anim.SetFloat ("Axis_Y", mov.y);




		if (mov.y < 0) {
			front = true;
			right = false;
			GetComponent<SpriteRenderer>().sprite = frontSprite;
			anim.SetBool("front", front);
			anim.SetBool("right", right);

		} else if (mov.y > 0) {
			front = false;
			right = false;
			GetComponent<SpriteRenderer>().sprite = backSprite;
			anim.SetBool("front", front);
			anim.SetBool("right", right);
		}
		if (mov.x < 0) {
			if (player.transform.position.x < transform.position.x) {
				transform.localScale = new Vector3 (-scale_X, scale_Y, 1);
				right = false;
				front = false;
			} else {
				transform.localScale = new Vector3 (scale_X, scale_Y, 1);
				right = true;
				front = false;
			}
			anim.SetBool("right", right);
			anim.SetBool("front", front);

		}
		else if(mov.x > 0){
			if (player.transform.position.x > transform.position.x) {
				transform.localScale = new Vector3 (scale_X, scale_Y,1);
				right = true;
				front = false;
			} else {
				transform.localScale = new Vector3 (-scale_X, scale_Y, 1);
				right = false;
				front = false;
			}
			anim.SetBool("right", right);
			anim.SetBool("front", front);
		}



}

void OnTriggerEnter2D(Collider2D other){
		if (other.gameObject.name == "Bullet2(Clone)") {
			Destroy (other.gameObject);
```

```
                if (life == 0) {
                        dead = true;
                        disparo = false;
                        anim.Play("jdead");
                        Destroy (gameObject, destroyTime);
                } else {
                        life--;
                }
        }
    }

    public bool GetSide(){
        return right;
    }
    public bool GetFront(){
        return front;
    }
    public bool GetDisparo(){
        return disparo;
    }


}

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

public class PlayerMovement : MonoBehaviour {



    public float scale_x;
    public float scale_y;

    public Sprite frontSprite;
    public Sprite backSprite;

    public bool right = true;
    public bool front = true;

    public float topspeed = 4f;
    Animator anim;
    const float timeout = 4.0f;


    void Start () {
        anim = GetComponent<Animator> ();
    }


    void Update () {

        Vector3 mov = new Vector3 (
            Input.GetAxisRaw("Horizontal"),
            Input.GetAxisRaw("Vertical"),
            0
        );
        transform.position = Vector3.MoveTowards (

            transform.position,
            transform.position+ mov,
```

```
                topspeed * Time.deltaTime

            );

            anim.SetFloat ("Axis_x", mov.x);
            anim.SetFloat("Axis_y", mov.y);
            if (mov.y < 0) {
                front = true;
                GetComponent<SpriteRenderer>().sprite = frontSprite;
            } else if (mov.y > 0) {
                front = false;
                GetComponent<SpriteRenderer>().sprite = backSprite;
            }
            if (mov.x < 0) {
                transform.localScale = new Vector3 (-scale_x, scale_y,1);
                right = true;
                front = true;

            }
            else if(mov.x > 0){
                transform.localScale = new Vector3 (scale_x, scale_y,1);
                right = false;
                front = true;
            }


            anim.SetBool("Front", front);

            anim.SetBool("Right", right);

        }

        public float GetX(){
            float x = Input.GetAxisRaw("Horizontal");
            return x;
        }
        public float GetY(){
            float y = Input.GetAxisRaw("Vertical");
            return y;
        }

        public bool GetSide(){
            return right;
        }
        public bool GetFront(){
            return front;
        }
        public void transformScales(){
            transform.localScale = new Vector3 (-scale_x, scale_y,1);
        }
        public void transformScalesReverse(){
            transform.localScale = new Vector3 (scale_x, scale_y,1);
        }
        public void SetFront(bool front){
            this.front = front;
        }
        public void SetSide(bool right){
            this.right = right;
        }
}.

using System.Collections;
```

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

public class MapAssigner : MonoBehaviour {
	public GameObject RoomObj;
	public Vector2 roomDimensions = new Vector2(17,17);
	public Vector2 gutterSize = new Vector2(17,17);
	public void Assign(Roomy[,] rooms){
		foreach (Roomy room in rooms){

			if (room == null){
				continue;
			}
			//pick a random index for the array

			Vector3 pos = new Vector3(room.gridPos.x * (roomDimensions.x +
gutterSize.x), room.gridPos.y * (roomDimensions.y + gutterSize.y), 0);
			Rooms myRoom = Instantiate(RoomObj, pos,
Quaternion.identity).GetComponent<Rooms>();
			myRoom.Setup(room.gridPos, room.doorTop, room.doorBot, room.doorLeft,
room.doorRight);
		}
	}
}
```

ДОДАТОК Г

Графічна частина

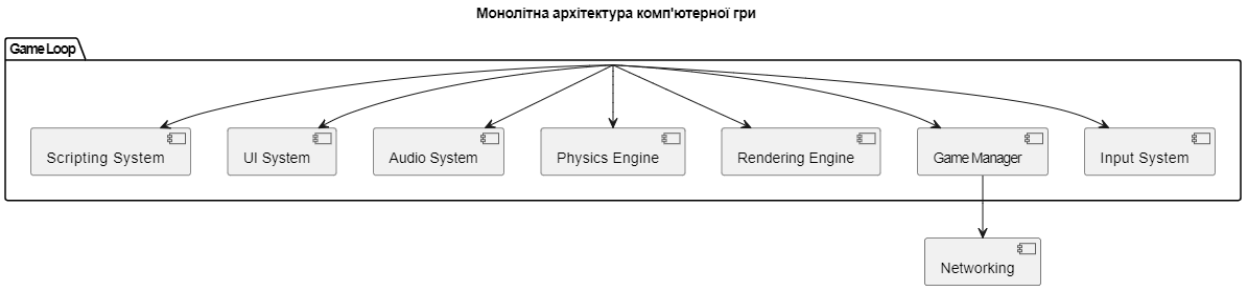

Рисунок Г.1 – Принципова схема монолітної архітектури комп’ютерної гри

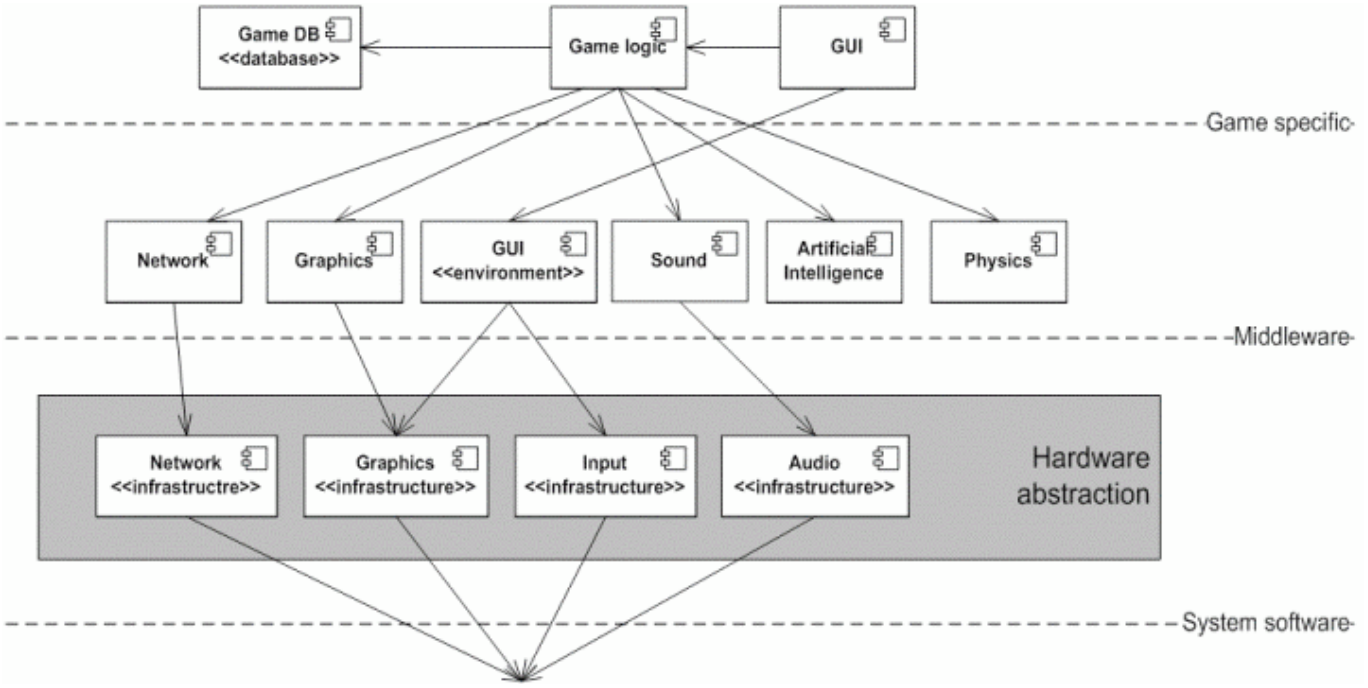

Рисунок Г.2 – Принципова схема компонентно-орієнтована архітектури комп’ютерної гри

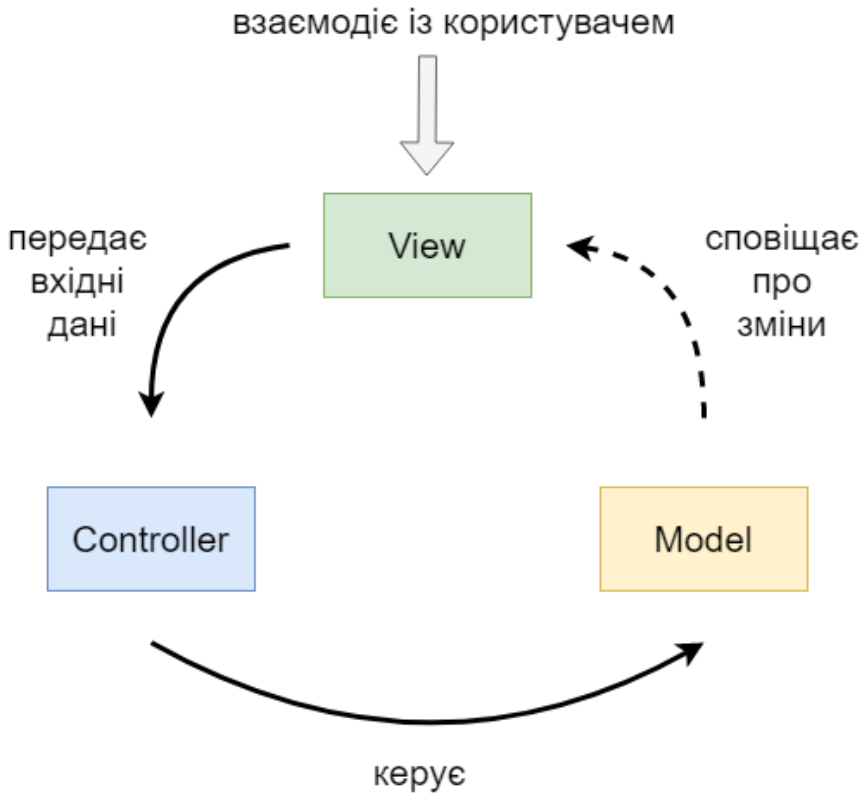

Рисунок Г.3 – Принципова схема архітектури Модель-Вид-Контролер комп’ютерної гри

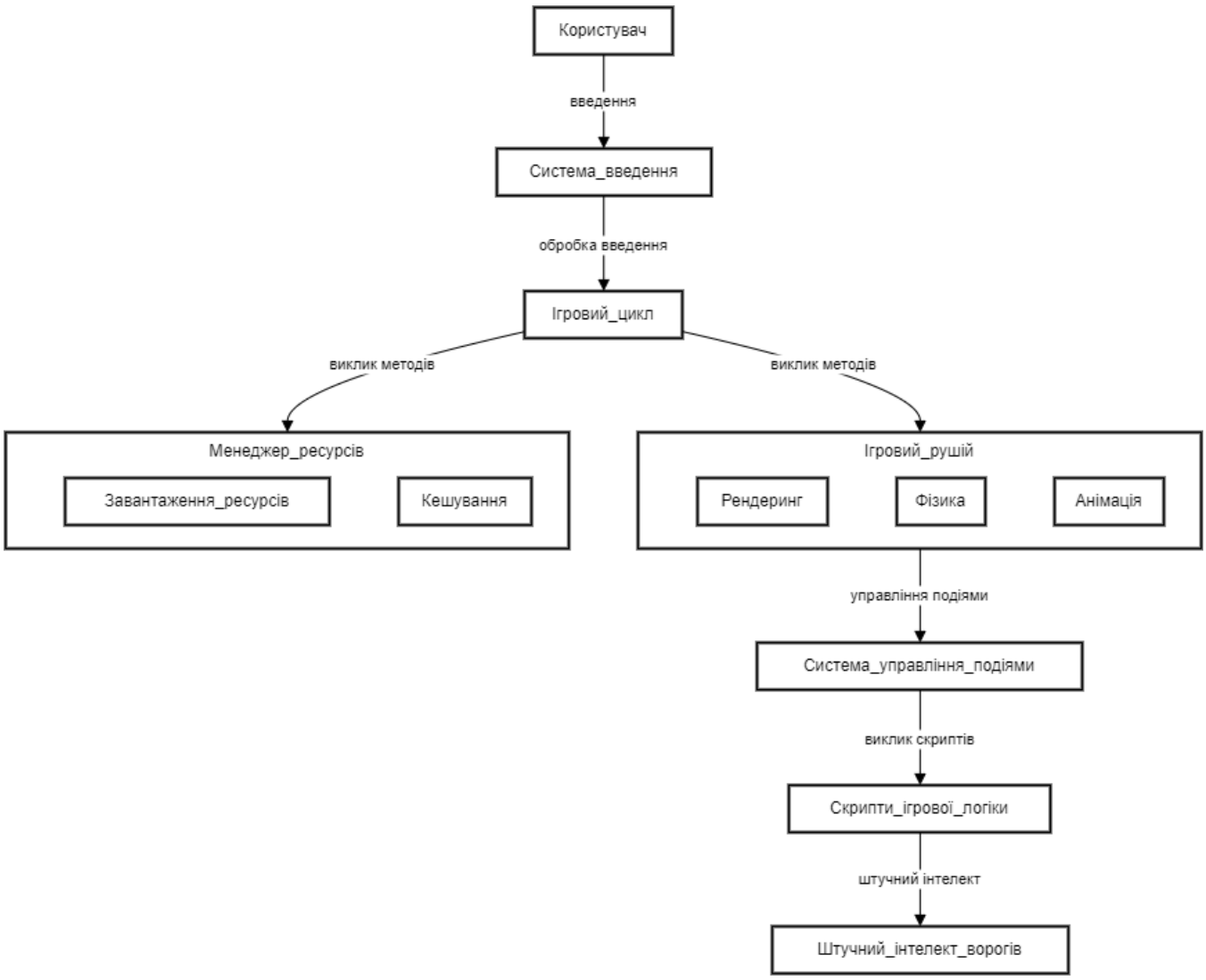

Рисунок Г.4 – Типова схема архітектури комп’ютерної гри

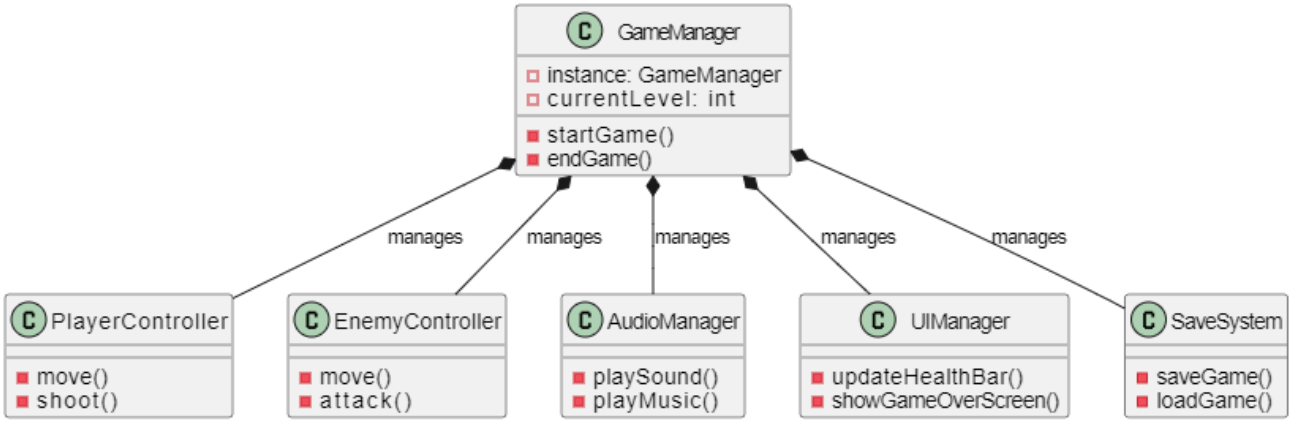

Рисунок Г.5 – Типова схема взаємодії між функціональними елементами

Рисунок Г.6 – Загальна схема алгоритму роботи циклу програмного модуля

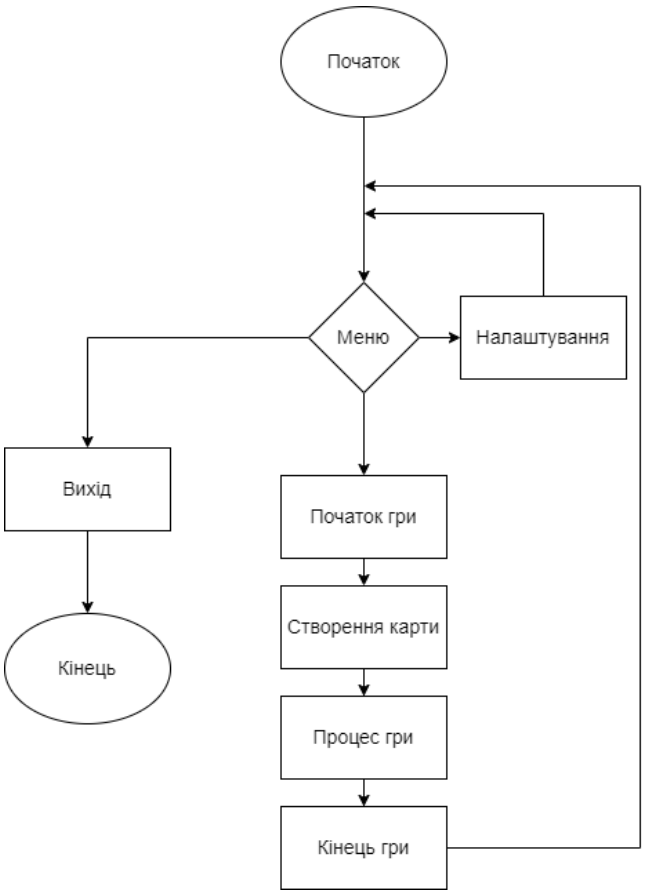

Рисунок Г.7 – Схема алгоритму роботи програмного модуля комп'ютерної гри.

Рисунок Г.8 – Типова схема функціональної моделі персонажу

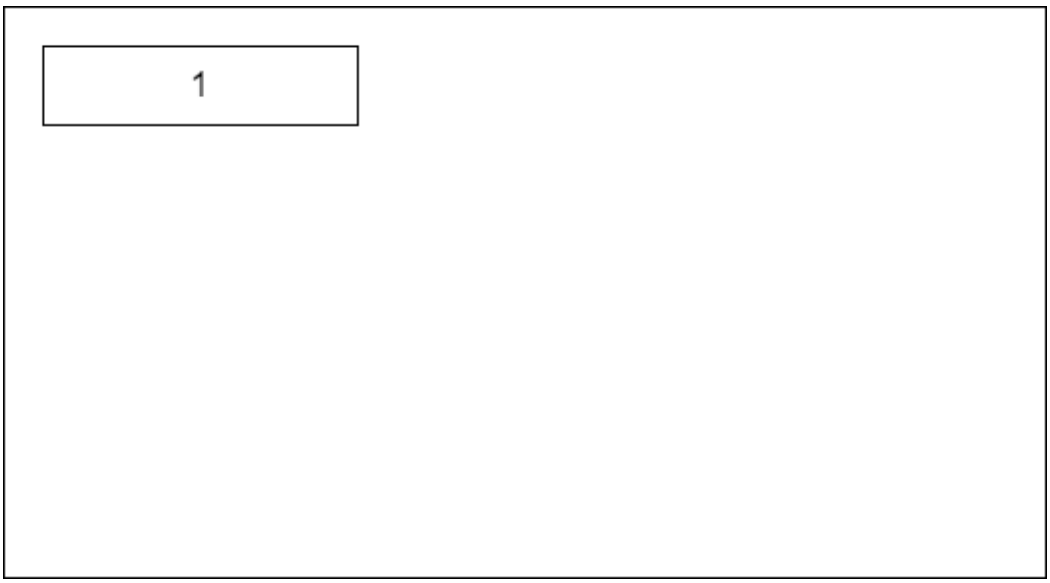

Рисунок Г.9 – Конструктивна схема користувацького інтерфейсу

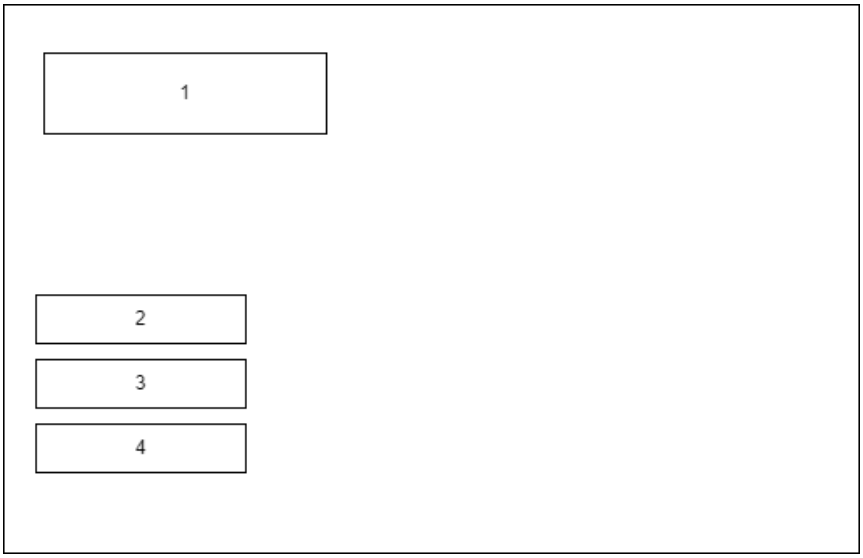

Рисунок Г.10 - Конструктивна схема зображення головного меню

Рисунок Г.11 – Загальний вигляд інтерфейсного вікна генерування ігрових сцен типу «карти»

Рисунок Г.12 – Загальний вигляд інтерфейсного вікна типу «Головного меню»

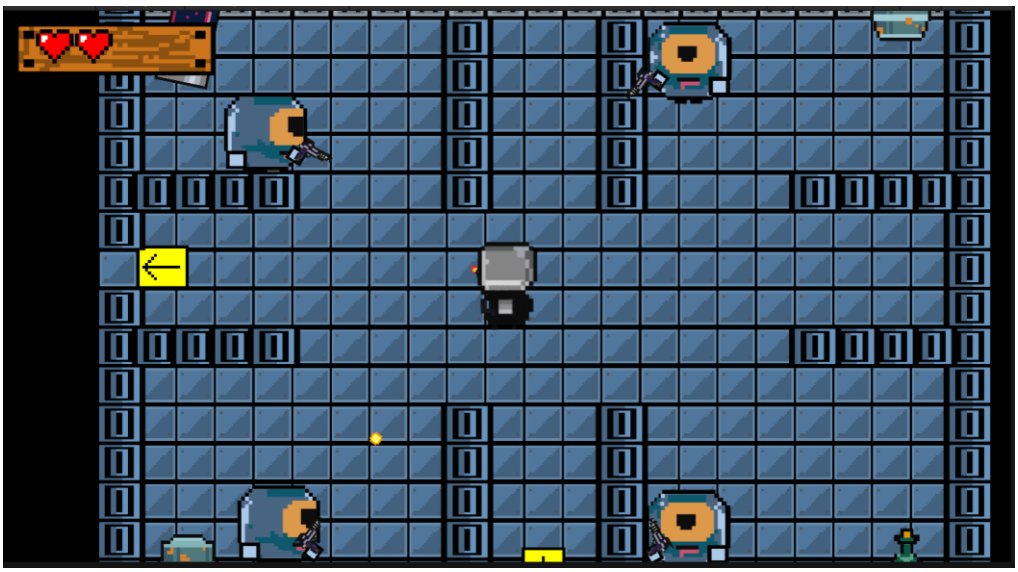

Рисунок Г.13 – Загальний вигляд комп’ютерної гри Phoenix

Рисунок Г.14 – Загальний вигляд закінчення гри при виграші

Рисунок Г.15 – Загальний вигляд закінчення гри при поразці