Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(назва факультету (відділення))
Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«МЕТОДИ ОПТИМІЗАЦІЇ ГРАФІЧНИХ РУШІЇВ 2D ІГОР»

Виконав: студент ІІ-го курсу, групи 2ПІ-22м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Колісниченко Г. М.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Рейда О. М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«15» грудня 2023 р.

Опонент: к.т.н., доц. каф. ОТ Тарновський М. Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«15» грудня 2023 р.

Допущено до захисту
Завідувач кафедрою ПЗ
д.т.н., проф., Романюк О. Н.
(прізвище та ініціали)
«15» грудня 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет Інформаційних технологій та комп'ютерної інженерії
Кафедра Програмного забезпечення
Рівень вищої освіти II-й (магістерський)
Галузь знань – 12 Інформаційні системи
Спеціальність – 121 Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
Романюк О. Н.
«19» вересня 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Колісниченку Георгію Миколайовичу

1. Тема роботи – «Методи оптимізації графічних рушіїв 2D ігор».
Керівник роботи: Рейда Олександр Миколайович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від «18» вересня 2023 р. № 247.
2. Строк подання студентом роботи
9 грудня 2023 року
3. Вихідні дані до роботи: алгоритм аналізу заповненості буфера за допомогою проміжної буферизації з паралельними обчисленнями графічних даних.
4. Зміст текстової частини записки: Модифікація методу подвійної буферизації, для підвищення якості відображення графічних зображень в процесі динаміного оновлення.
5. Перелік графічного матеріалу: використання програмнної компоненти для підвищення якості відоображення в 2D грі «Magical adventure».
6. Консультанти розділів магістерської кваліфікаційної роботи.

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–4	Рейда О. М. доцент кафедри ПЗ, к.т.н.	19.09.2023	05.12.2023
5	Кавецький В.В. доц. каф. ЕПВМ, к.е.н.	17.11.2023	25.11.2023

7. Дата видачі завдання 19 вересня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістреської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного стану питання та обгрунтування задачі	20.09.23 – 01.10.23	вик
2	Порівняльний аналіз методів оптимізації	02.10.23 – 14.10.23	вик
3	Розробка моделі роботи ігрової системи	15.10.23 – 22.10.23	вик
4	Розробка програмної компоненти	23.10.23 – 14.11.23	вик
5	Тестування програми	15.11.23 – 23.11.23	вик
6	Економічна частина	24.11.23 – 26.11.23	вик
7	Оформлення матеріалів до захисту МКР	27.11.23 – 29.12.23	вик

Студент _____ (підпис) **Колісниченко Г.М.** (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ (підпис) **Рейда О. М.** (прізвище та ініціали)

АНОТАЦІЯ

Колісниченко Г. М. Методи оптимізації графічних рушіїв 2D ігор. Магістерська кваліфікаційно робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – 121 Інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 130 с.

На укр. мові. Бібліогр.: 37 назв; рис.: 40; табл.7.

У магістерській кваліфікаційній роботі проведено дослідження методів оптимізації графічних рушіїв 2D ігор для підвищення якості відображення графічних зображень в процесі динаміного оновлення.

Подальшого розвитку отримав метод подвійної буферизації, у якому, на відміно від існуючого, використано аналіз наповненості буферу, що дозволило підвищити швидкість оновлення графічних даних для відображення на екрані і як наслідок підвищення продуктивності.

Подальшого розвитку набув метод проміжної буферизації даних у якому, на відміно від існуючого, використано паралельні обчислення даних графічного буферу для підвищення швидкодії формування даних.

У роботі проведено аналіз існуючих методів і засобів для визначення напрямків підвищення продуктивності виведення графічних даних, розроблено програмні компоненти основі запропонованих методів, проведено експериментальні дослідження розроблених засобів оптимізації графічних рушіїв 2D ігор для підвищення якості відображення графічних зображень в процесі динаміного оновлення.

Ключові слова: оптимізація, подвійний буфер, проміжна буферизація.

ANNOTATION

Kolisnychenko G. M. Optimization methods of graphic engines of 2D games. Master's thesis on the specialty 121 - Software engineering, educational program - 121 Software engineering. Vinnytsia: VNTU, 2023. 130 p.

In Ukrainian speech Bibliography: 37 titles; Fig.: 40; table 7.

In the master's qualification work, a study of optimization methods of graphic engines of 2D games to improve the quality of displaying graphic images in the dynamic updating process was carried out.

The method of double buffering received further development, in which, unlike the existing one, the analysis of buffer fullness was used, which made it possible to increase the speed of updating graphic data for display on the screen and, as a result, increase productivity.

The method of intermediate data buffering gained further development, in which, unlike the existing one, parallel data calculations of the graphic buffer were used to increase the speed of data formation.

In the work, an analysis of existing methods and means for determining directions for increasing the productivity of graphic data output was carried out, software components were developed based on the proposed methods, experimental research was carried out on the developed means of optimizing the graphic engines of 2D games to improve the quality of the display of graphic images in the process of dynamic updating.

Keywords: optimization, double buffer, intermediate buffering.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Графічне відображення інформації на пристроях є важливим аспектом взаємодії користувача з ПК. Графіка дозволяє передавати складні дані, ідеї та емоції у зрозумілій та ефективній формі. Графіка також сприяє залученню, зацікавленості та сприйняттю користувачів, якщо вона добре розроблена та адаптована до контексту. Графічне відображення інформації на пристроях потребує врахування ряду факторів, таких як розмір екрану, роздільна здатність, кольорова гамма, типографіка, композиція, анімація тощо. Відображення інформації на пристроях також повинно бути сумісним з різними форматами файлів, платформами та стандартами. Без графічного відображення інформації на екранах пристроїв неможлива успішна реалізації програм і мобільних додатків. За відображення графіки відповідають графічні рушії. Графічний рушій – це програмне забезпечення, яке відповідає за створення і відображення графіки на екрані.

Сучасна 2D графіка може бути статичною або анімованою, векторною або растровою, піксельною або згладженою. Графіка комп'ютерних ігор має багато переваг таких, як: низькі вимоги до обчислювальної потужності, широкий спектр художніх стилів та естетик, легкість розробки та портування на різні платформи. Проте, 2D графіка комп'ютерних ігор також має свої недоліки такі, як: обмежена реалістичність, складність створення перспективи та глибини, втрата популярності серед частини гравців. Однак вона залишається актуальною та привабливою для багатьох розробників та гравців, які цінують її креативність, виразність та ностальгію.

Графічні рушії є основою сучасних комп'ютерних ігор, які є популярною та перспективною галуззю інформаційних технологій. Оптимізація графічних рушіїв спрямована на покращення якості графіки, швидкості рендерингу, ефективності використання ресурсів та зменшення витрат на розробку ігор. 2D ігри мають свої особливості та вимоги до графічних рушіїв, які вимагають

спеціальних методів оптимізації, таких як атласи текстур, спрайтове пакування, паралакс-скролинг, кешування тайлових карт тощо.

Підвищення плавності виведення графічних даних можна досягти при збереженні другого сформованого зображення в другому буфері, з алгоритмом аналізу його наповненості та методом паралельних обчислень даних графічного буферу, що дасть можливість уникнути мерехтінню і розривів зображення, тому важливими є питання модифікації методу подвійної буферизації.

При виведені графічних даних реалізованих подвійною буферизацією, використовується два буфери: передній та задній. Переключення між буферами вимагає додаткової обробки з боку процесора та відеокарти, що може стати проблемою у випадку великої кількості графічних операцій або при використанні менш потужних пристроїв. Іноді може виникнути несинхронізоване відображення між переднім і заднім буферами, що може призвести до артефактів на екрані, таких як розриви, "розриви" чи інші неочікувані візуальні ефекти. Тому важливими є питання модифікації методу подвійної буферизації.

Формування плавної графічної сцени у реальному часі та в інтерактивному режимі висуває великі вимоги до продуктивності графічного рушія. При цьому використання простих методів оптимізації є неприйнятним для формування плавного виведення зображення.

Тому актуальними є питання підвищення продуктивності та швидкості виведення графічних сцен, оскільки існуючі методи оптимізації не задовольняють потреби ігрової галузі застосування двовимірної комп'ютерної графіки. Це передбачає модифікацію існуючого методу подвійної буферизації.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення якості відображення графічних зображень в процесі динаміного оновлення шляхом

дослідження методів оптимізації графічних рушіїв 2D ігор.

Основними задачами дослідження є:

– провести аналіз існуючих методів і засобів для визначення напрямків підвищення продуктивності виведення графічних даних;

– модифікувати метод подвійної буферизації виводу графічних даних на екран;

– модифікувати метод проміжної буферизації даних графічного буферу;

– розробити програмні компоненти та систему візуалізації на основі запропонованих методів;

– провести експериментальні дослідження розроблених засобів.

Об’єкт дослідження – процес оптимізації графічних рушіїв 2D ігор.

Предмет дослідження – методи та засоби оптимізації графічних рушіїв 2D ігор.

Методи дослідження. У процесі дослідження використовувались: теорія чисел та чисельних методів, теорія диференціально-інтегрального числення, лінійна алгебра, методи аналітичної геометрії для розробки моделей двомірних об’єктів; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Подальшого розвитку набув метод подвійної буферизації, у якому на відміно від існуючого використано аналіз наповненості буферу, що дозволило підвищити швидкість оновлення графічних даних для відображення на екрані і як наслідок підвищення продуктивності.

2. Подальшого розвитку набув метод проміжної буферизації даних у якому на відміно від існуючого використано паралельні обчислень даних графічного буферу для підвищення швидкодії формування даних.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано методи та

розроблено програмні засоби буферизації для комп’ютерних систем високореалістичної візуалізації двовимірних зображень.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: комп'ютерна 2D гра «Magical adventure» жанру action-adventure; аналіз методу подвійної буфернізації.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародних та всеукраїнських конференціях: Всеукраїнській науково-технічній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2022). Секція Інформаційні технології та комп‘ютерна інженерія» [1], Міжнародна Науково-Практична Інтернет-Конференція «Електронні Інформаційні Ресурси: Створення, Використання, Доступ». [2]

Публікації. Основні результати досліджень опубліковано в 2 наукових працях, у тому числі 2 – у матеріалах конференцій.

1. АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАДАЧІ

1.1 Аналіз стану питання оптимізації графічних рушіїв 2D ігор

Оптимізація програмного забезпечення – це процес покращення продуктивності, швидкості, використання ресурсів та ефективності програмного продукту. Метою оптимізації є зменшення витрат ресурсів (таких як час виконання, пам'ять, обчислювальна потужність) та покращення функціональності або робочої продуктивності програми без зміни її функціональності або якості [3].

Основні види оптимізації програмного забезпечення:

1. Оптимізація швидкості – це покращення часу виконання програми або її частини. Це може бути досягнуто за допомогою оптимізації компілятора, використання ефективних алгоритмів і структур даних, уникнення непотрібних обчислень, паралелізації тощо.

2. Оптимізація пам'яті – це зменшення споживання пам'яті програмою або її частинами. Це може бути досягнуто за допомогою видалення недосяжних об'єктів, уникнення витоку пам'яті, використання розумних вказівників, реалокації пам'яті тощо.

3. Оптимізація енергоефективності – це зниження споживання енергії програмою або її частинами. Це може бути досягнуто за допомогою використання низькоенергетичних пристроїв, адаптації частоти процесора, управління живленням тощо.

4. Оптимізація безпеки – це покращення захисту програми або її частин від зовнішнього втручання або вразливостей. Це може бути досягнуто за допомогою шифрування даних, аутентифікації користувачів, перевірки входів, оновлення патчів тощо.

5. Оптимізація якості – це покращення функціональності, надійності та корисності програми або її частин для кінцевих користувачів. Це може бути

досягнуто за допомогою тестування програми, виявлення та усунення помилок, рефакторингу коду, документування тощо [4].

Графічний рушій (Graphics Engines або Rendering Engines) – це набір інструментів, бібліотек та фреймворків, який дозволяє розробникам створювати та відображати графіку у своїх програмах [5].

Графічні рушії призначені для:

1. Рендерингу графіки: Вони відповідають за створення візуальних ефектів, відображення об'єктів, анімацій, освітлення та тіней в програмах.

2. Оптимізації графічного відтворення: Графічні рушії забезпечують оптимізацію та управління ресурсами для максимально ефективного відображення графіки на різних пристроях.

3. Підтримки графічних ефектів: Вони надають інструменти для створення різноманітних ефектів, таких як частинки, освітлення, текстури, анімації тощо.

Графічні рушії використовуються в багатьох сферах:

1. Розробка відеоігор: Вони є основою для створення візуально привабливих та захоплюючих графічно ігрових світів.

2. Віртуальна реальність (VR) та доповнена реальність (AR): Графічні рушії використовуються для створення і відображення віртуальних об'єктів та ефектів у просторі реального світу.

3. Архітектурний дизайн та візуалізація: Вони застосовуються для створення візуалізацій будівель, інтер'єрів та інших архітектурних об'єктів.

4. Графічні рушії працюють, використовуючи алгоритми рендерингу та використання графічних API (Application Programming Interface), які дозволяють взаємодіяти з апаратними графічними пристроями (наприклад, GPU). Вони оптимізовані для виконання обчислень, пов'язаних із створенням графіки, та забезпечують інтерфейс для розробників для керування та відображенням об'єктів, ефектів та анімацій у своїх програмах чи додатках [6].

Оптимізація графічних рушіїв залишається актуальною темою в індустрії розробки відеоігор та різних програм, де використовуються графічні ефекти. Ось кілька причин, чому ця тема є важливою і актуальною:

1. Обмеженість ресурсів пристроїв: Ігрові консолі, мобільні пристрої, старі ПК та ноутбуки мають обмежені обчислювальні та графічні можливості. Оптимізація графіки допомагає забезпечити стабільну роботу гри на цих пристроях і за рахунок цього збільшити кількість потенційних користувачів.

2. Відсутність плавності геймплею: Оптимізовані графічні рушії дозволяють збільшити кадрову частоту (FPS) та знизити затримки у відображенні графіки, що робить геймплей більш плавним та приємним для користувача.

3. Високі вимоги до графіки: З кожним роком вимоги до графіки в іграх та програмах зростають через велику кількість однотипної графіки в сучасних проєктах. Щоб створити графіку яка приємно здивує користувача необхідно не лише новий творчий підхід, а і технічна можливість відобразити його на моніторі ненових ПК та смартфонів, якими володіє велика кількість користувачів, які не планують його оновлювати, через економічну ситуацію в світі.

4. Оптимізована графіка, краще продається: Оптимізована графіка дозволяє розробникам вирізнятися на ринку завдяки високій продуктивності та якості графіки в їх іграх та програмах.

2D ігри знову набирають популярність. Навіть у контексті появи багатоігрових тривимірних світів, 2D ігри залишаються дуже популярними через ряд причин:

1. Ретро-стиль і ностальгія: Багато гравців відчувають ностальгію за класичними 2D іграми, які нагадують їм про часи, коли грали в такі гри у своєму дитинстві. Це стимулює популярність ретро-стилю ігор [7].

2. Простота та доступність: 2D ігри, зазвичай, простіше у розумінні та геймплеї, що робить їх привабливими для різних гравців, включаючи тих, хто тільки починає відкривати світ відеоігор.

3. Широкий спектр жанрів: Від платформерів до інді-рольових ігор, 2D формат дозволяє створювати різноманітні геймплейні ідеї та концепції, що привертає увагу широкого кола гравців.

4. Спрощені вимоги до ресурсів: 2D ігри часто вимагають менше обчислювальних ресурсів та часу на розробку порівняно з тривимірними проектами, що робить їх популярними серед невеликих студій та інді-розробників.

5. Естетика та художній стиль: Унікальна графіка 2D ігор може вражати своєю мистецькою цінністю та стилем, що робить їх привабливими для гравців, які цінують візуальну естетику [8].

Тому такі ігри знову займають своє місце у відеоігровій індустрії, пропонуючи не лише класичний геймплей, але й унікальність та характер, що відмінюється від традиційних тривимірних проектів.

У зв'язку з постійними змінами технологій, великої кількості користувачів що користуються малопотужними гаджетами та зростаючими очікуваннями щодо графіки у комп’ютерних 2D іграх, оптимізація графічних рушіїв 2D ігор залишається є актуальним для досягнення комерційного успіху та задоволення потреб користувачів.

1.2 Порівняльний аналіз методів оптимізації

У якості аналогів для порівняльного аналізу були обрані такі методи: «Подвійна буферизація» (Подвійний буфер), «Керування оновленням екрану» та «Використання спрайтів та груп».

1. Метод «Подвійна буферизація» (Подвійний буфер). Подвійна буферизація – це тимчасова область зберігання в оперативній пам'яті, яка дозволяє зберігати дані під час їх передачі. Менеджер буфера відповідає за виділення місця в буфері для зберігання даних. Всі дії диспетчера буфера

виконуються внутрішньо і невидимі в програмі.

Наприклад, під час читання програми А можна писати програму В, а під час читання програми В можна писати програму А. Це відбувається у двох місцях одночасно, що зображено на рисунку 1.1. [9]

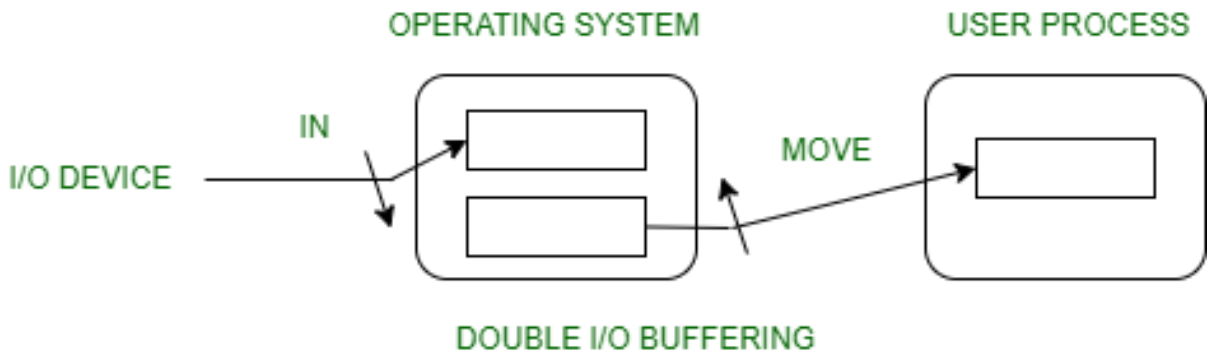

Рисунок 1.1 – Схема роботи подвійного буфера

Подвійний буфер – це метод оптимізації виводу графіки на екран, який полягає в тому, що зображення формується не безпосередньо в відеопам'яті, а в окремому буфері пам'яті.

Він використовує два буфери пам'яті для зображень: передній (front buffer) та задній (back buffer).

Передній буфер – це та область пам'яті, яка відображається на екрані. Задній буфер – це область, де обробляється новий вміст, який має бути відображений. Коли зображення у задньому буфері готове, воно переміщується у передній буфер, що призводить до зміни відображення на екрані.

Використання двох буферів дозволяє уникнути мерехтіння чи спотворень при відображенні зображення на екрані. Коли новий вміст готовий у задньому буфері, відбувається швидкий обмін місцями буферів, що приховує цю зміну від користувача. Результат – плавне та безперервне оновлення зображення [10].

Цей метод забезпечує відображення зображень без спотворень, що особливо важливо у випадках, коли потрібно забезпечити плавну анімацію чи

інтерактивний геймплей без помітних переривань для користувача.

Метод «Подвійна буферизація» використовується в різних областях, де необхідне плавне та безперервне відображення графічного вмісту на екрані:

1. Комп'ютерні ігри: У розробці ігор, цей метод дозволяє створювати плавну анімацію, уникнути мерехтіння та спотворень під час швидкого перемальовування екрану в реальному часі. Метод «Подвійна буферизація» використовують в таких ігрових рушіях як Unity та Unreal Engine.

2. Графічні програми та редактори: В графічних програмах, таких як фото- та відео-редактори наприклад, Adobe Photoshop, After Effects, і багато інших. «Подвійна буферизація» допомагає уникнути мерехтіння при зміні вмісту на екрані під час маніпуляцій з графікою.

3. Відеопрогравачі та мультимедіа: У відеопрогравачах, цей метод допомагає гладко відтворювати відео, уникнути мерехтіння чи спотворень під час швидкого перемальовування кадрів.

4. Операційні системи та GUI: У операційних системах і графічних інтерфейсах користувача (GUI), «Подвійна буферизація» використовується для забезпечення плавного оновлення вікон та вмісту екрану, уникнення мерехтіння під час переміщення та оновлення вікон.

5. Графічні бібліотеки та фреймворки: Більшість бібліотек для графічного програмування мають можливість реалізації «Подвійна буферизація». Наприклад, JavaFX, бібліотека для створення графічних інтерфейсів у Java, може використовувати цей метод для плавного оновлення зображення [11].

Метод «Подвійна буферизація» можна реалізувати за допомогою багатьох мов програмування, оскільки він стосується операцій з графікою та роботи з буферами пам'яті. Ось кілька мов, які часто використовуються для реалізації цього методу:

– C++: Часто використовується для графічного програмування завдяки швидкості та можливостям маніпулювання пам'яттю. Бібліотеки, такі як SDL (Simple DirectMedia Layer) чи SFML (Simple and Fast Multimedia

Library), дозволяють реалізувати подвійну буферизацію.

– Java: Ця мова програмування також має можливості для графічного програмування. Використання JavaFX або Swing у поєднанні з подвійною буферизацією дозволяє створювати графічні програми з плавним оновленням.

– Python: Мова Python за допомогою бібліотеки Pygame чи Tkinter також може бути використана для створення графічних програм з використанням «Подвійна буферизація».

– JavaScript: Для веб-розробки можна використовувати HTML5 Canvas разом із JavaScript для створення графічних програм з підтримкою подвійної буферизації [12].

2. Метод «Керування оновленням екрану». Оновлення екрану – це процес, за допомогою якого зображення на екрані монітора змінюється відповідно до даних, що надходять від процесора або відеокарти. Оновлення екрану може бути синхронним або асинхронним (рисунок 1.2). Синхронне оновлення екрану означає, що зображення на екрані змінюється тоді, коли відеокарта надсилає новий кадр. Асинхронне оновлення екрану означає, що зображення на екрані змінюється тоді, коли відеокарта готова надіслати новий кадр, незалежно від частоти оновлення монітора. Оновлення екрану в комп'ютерній графіці має важливе значення для плавності зображення, особливо у випадках, коли зображення має динамічний характер, наприклад у відеоіграх або анімаціях.

Цей метод може включати в себе різні стратегії та алгоритми:

– оновлення за запитом (on-demand updating), екран оновлюється тільки в тих випадках, коли система отримує команду або сигнал на оновлення, зменшуючи навантаження на процесор та ресурси.

– адаптивне оновлення (adaptive updating), система виявляє зміни в вмісті та оновлює екран лише тоді, коли це необхідно, залежно від змін в даних чи взаємодії користувача.

– оновлення за графіком (scheduled updating), оновлення екрану

відбувається за певним графіком або планом, щоб уникнути перевантаження системи в процесі оновлення. [13]

Приклади програмного забезпечення, яке використовує цей метод:

1. Графічні редактори та програми для дизайну: Програми, такі як Adobe Photoshop, Illustrator, Sketch, використовують метод «Керування оновленням екрану» для плавного відображення змін, що дозволяє користувачам працювати з графікою без спотворень та мерехтіння.

2. Відеоредактори та програми для відеомонтажу: Програми, наприклад Adobe Premiere Pro, Final Cut Pro або DaVinci Resolve, використовують цей метод для плавного відображення та редагування відео, забезпечуючи безперервне оновлення вмісту без спотворень.

3. Анімаційні програми: Програми для створення анімацій, такі як Adobe After Effects, Toon Boom Harmony, Moho (Anime Studio), використовують метод керування оновленням екрану для плавного відображення анімацій та мультимедійного вмісту.

4. Розробка ігор: У світі геймінгу цей метод використовується для створення плавного геймплею та анімацій в комп'ютерних іграх, таких як Unity, Unreal Engine, або інших геймдвигунах.

5. Операційні системи та графічні інтерфейси користувача (GUI): Графічні інтерфейси операційних систем, такі як Windows, macOS, Linux, використовують метод «Керування оновленням екрану» для ефективного відображення вікон, кнопок та елементів інтерфейсу.

Метод «Використання спрайтів та груп». Спрайт – це невеликі графічні об'єкти чи зображення, які можуть бути рухомими, статичними чи анімованими. Зазвичай це різноманітні персонажі, предмети, ефекти, які використовуються у візуальних додатках, особливо в іграх. Спрайти часто використовуються в відеоіграх, анімаційних фільмах та інтерактивних медіа. Спрайти можуть мати різну форму, розмір, кольори та стиль.

Група – це набір спрайтів, які об'єднуються за певною ознакою або функцією. Групи дозволяють керувати спрайтами як одним цілим, наприклад,

переміщати, обертати, змінювати розмір або взаємодіяти з іншими об'єктами на сцені. [14]

Спрайти та групи є важливими інструментами в комп'ютерній графіці, оскільки вони дозволяють створювати складні та динамічні сцени з простих елементів. Спрайти та групи також сприяють оптимізації ресурсів та продуктивності, оскільки вони зменшують кількість обчислень, необхідних для рендерингу зображень [15].

Основні концепції методу «Спрайти та групи»:

1. Групи (Groups): Це колекції спрайтів, які об'єднують об'єкти з однаковими характеристиками, поведінкою чи функціоналом. Наприклад, всі спрайти персонажів можуть бути згруповані разом.

2. Керування спрайтами (Sprite Management): Метод включає в себе методи для керування розміщенням, переміщенням, масштабуванням, обертанням та зміною властивостей спрайтів у межах групи.

3. Колізії та взаємодія (Collisions and Interactions): Використання груп дозволяє легше виявляти колізії між спрайтами, взаємодіяти між ними та керувати реакцією на взаємодію.

4. Оптимізація роботи з пам'яттю: Групи дозволяють ефективно керувати пам'яттю, оскільки вони дозволяють об'єднувати та керувати багатьма спрайтами одночасно, зменшуючи затрати ресурсів на обробку.

Основні сфери застосування цього методу наведені нижче метод використовується:

1. Розробка ігор: У створенні відеоігор цей метод є важливим. Більшість геймдвигунів, таких як Unity, Unreal Engine, Godot, використовують спрайти та групи для управління та організації графічними об'єктами, такими як персонажі, об'єкти у грі, ефекти тощо. [16]

2. Графічні редактори: Програми для створення графіки та анімацій, такі як Adobe Photoshop, Adobe Animate, Blender, використовують спрайти та групи для організації шарів, об'єктів та елементів сцени.

3. Відеоредактори та анімаційні програми: Програми для

відеомонтажу, такі як Adobe After Effects, анімаційні програми, наприклад, Toon Boom Harmony, також використовують спрайти та групи для організації об'єктів у процесі створення анімацій та відео.

4. Веб-розробка: У веб-розробці спрайти та групи використовуються для управління графічними об'єктами у веб-дизайні та інтерактивних веб-додатках за допомогою фреймворків та бібліотек, наприклад, у JavaScript або CSS.

5. Програми для мультимедіа: Програми для створення презентацій, відео-арту або інших мультимедійних проєктів також можуть використовувати цей метод для організації та управління графічним вмістом.

Приклад створення спрайтів в програмі Sprite Editor наведений на рисунку 1.2.

Рисунок 1.2 – Створення спрайтів в Sprite Editor

Результати порівняння цих трьох методів оптимізації наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння методів оптимізації

Критерії	Метод «Подвійна буферизація»	Метод «Керування оновленням екрану»	Метод «Спрайти та групи»
Плавність	1	0,5	0,5
Мінімізація мерехтіння	1	0,5	1
Затримка	1	1	0,5
Стабільність відображення	0,5	0,5	0,5
Синхронізація	1	0	1
Низьке навантаження ЦП	0,5	0	0,5
Універсальність	1	1	0
Всього	6	3,5	4

Отже, проаналізувавши метод «Подвійна буферизації», «Керування оновленням екрану» і «Сипрайти та груп , було прийнято рішення, обрати метод «Подвійної буферизації» для модифікування.

1.3 Постановка задачі

Після аналізу стану питання оптимізації графічних рушіїв 2D ігор та порівняльного аналізу методів оптимізації було визначено такі завдання дослідження:

– аналіз існуючих методів і засобів для визначення напрямків підвищення продуктивності виведення графічних даних;

– модифікування методу подвійної буферизації виводу графічних даних на екран.

– модифікування методу проміжної буферизації даних графічного

буферу

– розробка програмних компонент та систем візуалізації на основі запропонованих методів;

– проведення експериментальних досліджень розроблених засобів виведення графічних даних.

Технічне завдання на розробку наведено у Додатку А.

1.4 Висновки

В розділі проведено аналіз методів оптимізації графічних рушіїв 2D ігор, результатом чого є визначені основні види оптимізації програмного забезпечення та окреслено призначення графічного рушія.

Проведено порівняльний аналіз таких методів оптимізації: «Подвійна буферизація», «Керування оновленням екрану» і «Спрайти та групи».

В результаті порівняльного аналізу було обрано метод «Подвійна буферизація» для подальшої модифікації.

Визначено такі задачі:

– аналіз існуючих методів і засобів для визначення напрямків підвищення продуктивності виведення графічних даних;

– модифікування методу подвійної буферизації виводу графічних даних на екран;

– модифікування методу проміжної буферизації даних графічного буферу;

– розробка програмних компонент та систем візуалізації на основі запропонованих методів;

– проведення експериментальних досліджень розроблених засобів виведення графічних даних.

2 МЕТОДИ ОПТИМІЗАЦІЇ

2.1 Аналіз методів оптимізації графічних рушіїв 2D ігор

Оптимізація – це процес вдосконалення якості, продуктивності або ефективності програмного забезпечення за допомогою зміни його параметрів, алгоритмів, структури або дизайну. Метою оптимізації є досягнення кращих результатів за менший час, з меншими витратами ресурсів або з більшою задоволеністю користувачів. Оптимізація може бути застосована на різних етапах розробки програмного забезпечення, таких як аналіз, проектування, кодування, тестування або супроводження. [17] Оптимізація може використовувати різні методи, такі як математичне моделювання, емпіричне порівняння, евристичні алгоритми, машинне навчання або інженерні практики. Оптимізація вимагає визначення критеріїв оцінки якості програмного забезпечення, таких як швидкодія, надійність, масштабованість, безпека, зручність використання або сумісність. Оптимізація також потребує врахування обмежень та компромісів, пов'язаних з ресурсами, бюджетом, термінами або вимогами замовника [18].

Види оптимізації програмного забезпечення:

1. Оптимізація часу виконання – це спрямована на зменшення часу, необхідного для виконання програми або її окремих частин. Це може включати використання ефективніших алгоритмів, структур даних, компіляторів, паралелізму тощо.

2. Оптимізація пам'яті – це спрямована на зменшення обсягу пам'яті, який використовує програма або її окремі частини. Це може включати використання менш об'ємних типів даних, компресії, кешування, рециклінгу тощо [19].

3. Оптимізація енергоспоживання – це спрямована на зменшення кількості енергії, яку споживає програма або її окремі частини. Це може включати використання менш потужних процесорів, сенсорів, пристроїв введення-виведення тощо.

4. Оптимізація безпеки – це спрямована на збільшення рівня захисту програми або її окремих частин від несанкціонованого доступу, злому, витоку інформації тощо. Це може включати використання шифрування, аутентифікації, цифрових підписів тощо.

5. Оптимізація користувацького інтерфейсу – це спрямована на покращення зручності, привабливості, інтуїтивності або інших характеристик інтерфейсу програми для користувача. Це може включати використання графіки, анімації, звуку, голосового управління тощо. [20]

При необхідності покращення графічних рушіїв 2D ігор необхідна графічна оптимізація може включати такі аспекти:

– Налаштування параметрів рендерингу, таких як роздільна здатність, частота кадрів, освітлення, тіні, відбивання, пост-ефекти тощо.

– Використання технологій, що зменшують навантаження на графічний процесор, наприклад, обмеження видимості, імпостери, LOD (рівень деталей), бейкінг (попереднє обчислення) тощо [21].

– Оптимізація ресурсів графіки, таких як текстури, меші, шейдери, анімації тощо. Це може включати компресію, атласи, інстансинг (множинне використання одного об'єкта) тощо.

– Профайлінг (аналіз) графічної продуктивності за допомогою спеціальних інструментів, таких як GPUView, RenderDoc, PIX тощо. Це дозволяє виявити та усунути «вузькі місця» (bottlenecks) у графічному конвеєрі.

Графічна оптимізація є важливою складовою розробки комп'ютерної гри, оскільки вона впливає на загальне враження від гри та її доступність для різних платформ та пристроїв. Графічна оптимізація потребує балансу між естетикою та ефективністю, а також постійного тестування та налагодження. [22].

Оптимізація програмного забезпечення є важливим аспектом розробки та підтримки програм. Вона може допомогти покращити продуктивність, якість та задоволення користувача програмами.

2.2 Розробка моделі роботи ігрової системи

Так як 2D гра «Magical adventure» є грою жанру екшн-пригода необхідно було розробити модель ігрової системи, для розуміння гемплею (рисунок 2.1).

Ця модель передбачає що:

1. Зустріч з ворогом. Гравець при зустрічі з ворогом може його знищити за допомогою зброї, магії чи втекти.

2. Здоров'я і енергія. Гравець має слідкувати за шкалою здоров'я та енергії. Гравцю необхідно правильно підібрати час і позицію для нанесення удару, аби не втрачати здоров'я від ударів ворога. Відновити втрачене здоров'я гравець може лише за допомогою заклинання, на застосування якого втрачається енергія, яка повільно відновлюється.

3. Зброя і магія. Знищити ворога можна як зброєю різного виду так і бойовим заклинанням.

4. Бали досвіду. За знищення ворога гравцю будуть нараховуватися бали досвіду, кількість яких залежить від виду знищеного ворога.

5. Підвищення характеристик. Бали досвіду гравець зможе витратити на покращення 5 базових характеристик персонажа. Чим більші характеристики персонажа, тим легшим завданням в майбутньому буде знищення більш небезпечних ворогів, що зменшує ймовірність програшу.

6. Необмеженість підвищення. Гравець може підвищити кожну характеристику п'ять разів, підвищення однієї характеристики не блокує, можливість підвищити іншу. Гравець може мати різні комбінації характеристик, що урізноманітнює ігровий процес при кожному проходженні гри.

Також модель дозволить урізноманітнити тактику ведення бою з ворогами, тим самим тренуючи тактичне мислення користувача. У користувача буде можливість грати в гру з великою кількістю комбінацій значень характеристик, що, у свою чергу добре, збільшить реіграбельність гри.

Рисунок 2.1 – Модель роботи ігрової системи

Розробка алгоритмів роботи 2D гри «Magical adventure»

Алгоритм роботи комп'ютерної 2D гри «Magical adventure» починається із завантаження персонажа гравця в ігрову локацію.

1. Початок гри:

– гравець вибирає локацію для дослідження і починає дослідження локації.

2. Зустріч з ворогами:

– Гравець має можливість обрати зброю для бою.

– Гравець може змінювати зброю під час бою.

– Можливість використання магії для знищення ворогів.

3. Бій з ворогами:

– якщо гравець перемагає ворога

– нараховуються бали досвіду.

– гравець має можливість вибору

– можливість збільшувати одну або декілька характеристик одночасно.

– перевірка наявності достатньо балів досвіду для збільшення.

– продовження дослідження локації зі зіткненням з більш важкими ворогами.

4. Програш у бою:

– якщо гравець програє гра завершується.

Блок-схема алгоритму роботи гри зображена на рисунку 2.2.

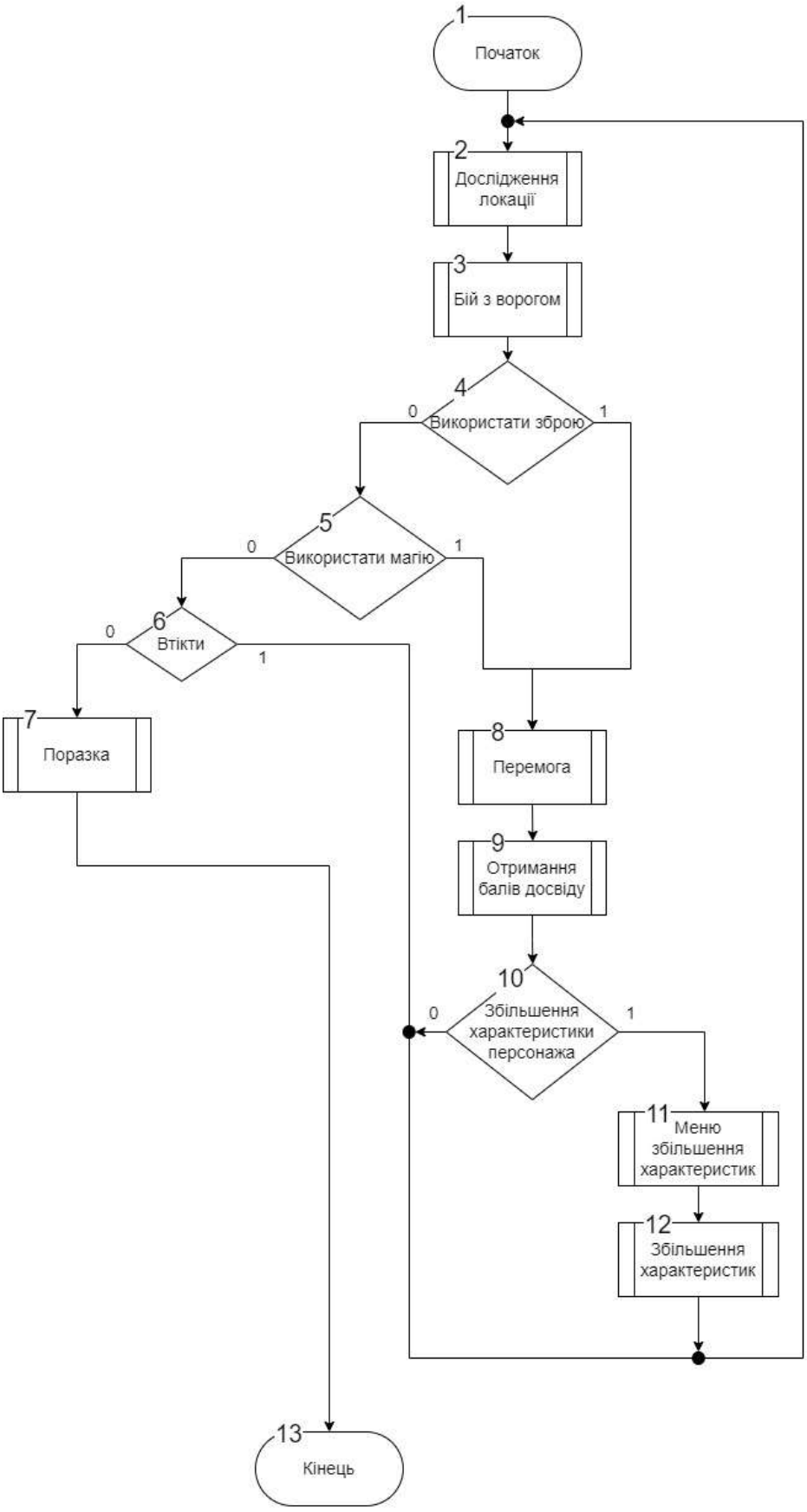

Рисунок 2.2 – Блок-схема загального алгоритму програми

Функціональна модель персонажу зображена на рисунку 2.3.

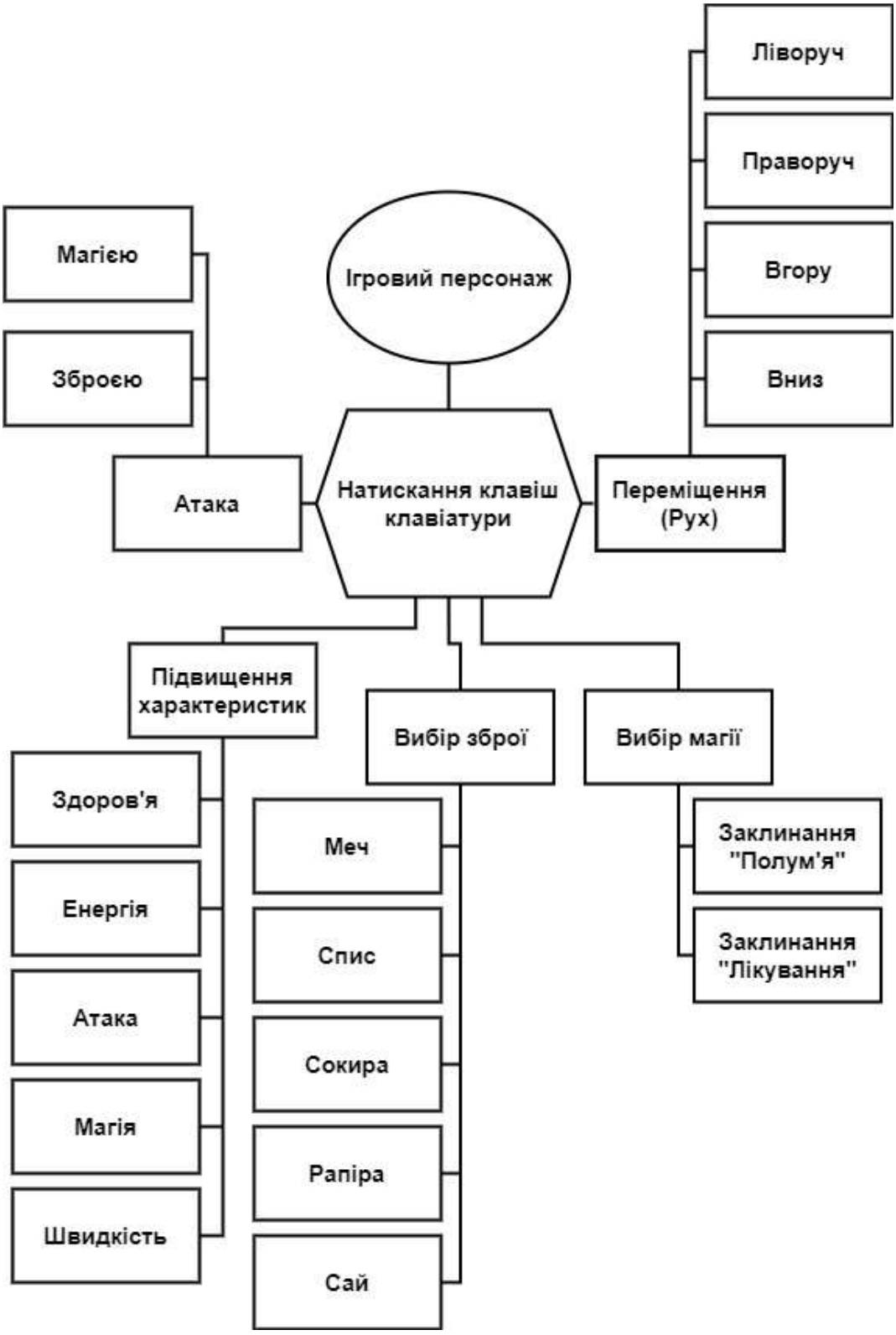

Рисунок 2.3 – Функціональна модель персонажу

На моделі персонажу продемонстровано всі можливості які гра надає

гравцю, а саме:

1. Переміщення в 4 боки (ліворуч, праворуч, вгору, вниз)
2. Можливість перемкнутися на іншу зброю:

– меч;

– спис;

– сокира;

– рапіра;

– сай.

3. Можливість використовувати заклинання:

– Полум’я;

– Лікування.

4. Атакувати за рахунок як зброї так і магією.
5. Підвищувати характеристики персонажу за рахунок балів досвіду, а саме:

– Здоров’я;

– Енергія;

– Атака;

– Магія;

– Швидкість.

2.3 Метод подвійної буферизації

З розвитком комп'ютерної технології важливість аналізу даних в буферах графічних систем значно зросла. Розвиток програмного забезпечення, особливо у галузі візуалізації, вимагає оптимізації процесу відображення графічного вмісту на екрані. [23]

Аналіз даних в буферах дозволяє не лише ефективно відслідковувати зміни в графічному вмісті, але і використовувати ресурси системи більш ефективно. Шляхом ідентифікації лише змінених областей екрану можна підвищити продуктивність відображення, зменшити навантаження на процесор та пам'ять, а також покращити якість візуалізації.

Ключові переваги проведення аналізу даних в буфері такі:

1. Оптимізація рендерингу: Аналіз буферів дозволяє ідентифікувати лише змінені частини графічного вмісту. Замість повного оновлення всього екрану перерисовуються лише ті області, де відбулися зміни, що значно зменшує навантаження на процесор і пам'ять.

2. Підвищення продуктивності: Зменшення обсягу даних для оновлення на екрані дозволяє прискорити частоту кадрів у графічних додатках, забезпечуючи плавніше та швидше відображення.

3. Мінімізація використання ресурсів: Аналіз даних у буферах зменшує потребу в великому обсязі обчислень та передачі даних між пам'яттю та графічним процесором, що дозволяє ефективніше використовувати ресурси системи.

4. Зменшення мерехтіння та артефактів: Перемальовування тільки змінених областей екрану допомагає уникнути мерехтіння та артефактів, які можуть виникнути при повному оновленні зображення.

5. Підтримка анімацій та ігрових процесів: Для графічних додатків, де потрібна плавна анімація чи високочастотний оновлюваний вміст, аналіз даних в буферах допомагає забезпечити плавність та швидкість процесу відображення.

6. Ефективне використання пам'яті: Аналіз даних в буферах допомагає мінімізувати об'єм використовуваної пам'яті, оскільки лише змінені області перерисовуються, що зменшує вимоги до обсягу доступної пам'яті.

Усе це сприяє оптимізації роботи графічних систем, покращенню продуктивності та забезпечує користувачам більш якісне та плавне відображення графічного вмісту на екрані.

Алгоритм аналізу різниці між переднім і заднім буферами. Він працює з різницею між попереднім і новим кадрами для визначення змінених областей, які потребують оновлення на екрані.

Основна ідея полягає в тому, щоб оновлювати лише ті частини екрану, які дійсно зазнали змін. Це забезпечує ефективне використання обчислювальних ресурсів та прискорює процес відображення графіки.

Алгоритм може використовувати передній та задній буфери. Він порівнює попередній (або базовий) кадр, який може бути збережений у задньому (графічному) буфері, з новим кадром, що створюється в передньому буфері.

Коли графічний об'єкт отримує оновлення чи зміни, його нове зображення відображається у передньому буфері. Далі алгоритм порівнює ці два кадри для визначення змінених областей. Ці змінені області можуть бути у формі прямокутників або інших форм, які охоплюють мінімальні області змін.

Після визначення змінених областей вони оновлюються у задньому (графічному) буфері. Це дозволяє уникнути оновлення частин екрану, які залишилися без змін.

Використання алгоритму дозволяє ефективно оновлювати тільки змінені області екрану, зберігаючи ресурси і покращуючи продуктивність відображення.

Блок-схема роботи алгоритму зображена на рисунку 2.4.

Процес роботи алгоритму включає такі кроки:

1. Ініціалізація буферів: Початковий стан переднього буфера відображається на екрані, а задній буфер використовується для внесення змін.

2. Відстеження змін: Після кожної зміни графічного вмісту на задньому буфері, алгоритм визначає прямокутники (або інші форми областей), де відбулися зміни. Це може включати зміну кольору, текстури, положення об'єктів тощо.

3. Створення областей змін: Операція визначення змін формує області змін, які потребують оновлення. Їх необхідно оновити на передньому буфері для відображення на екрані.

4. Оновлення переднього буфера: Лише області змін оновлюються на передньому буфері. Цей процес полягає у внесенні змін відповідно до виявлених змін на задньому буфері.

5. Активація переднього буфера: Після оновлення областей змін передній буфер стає активним і відображається на екрані.

6. Повторення процесу: Цей процес повторюється при кожній зміні графічного вмісту, що дозволяє ефективно оновлювати лише частини екрану, де відбулися зміни.

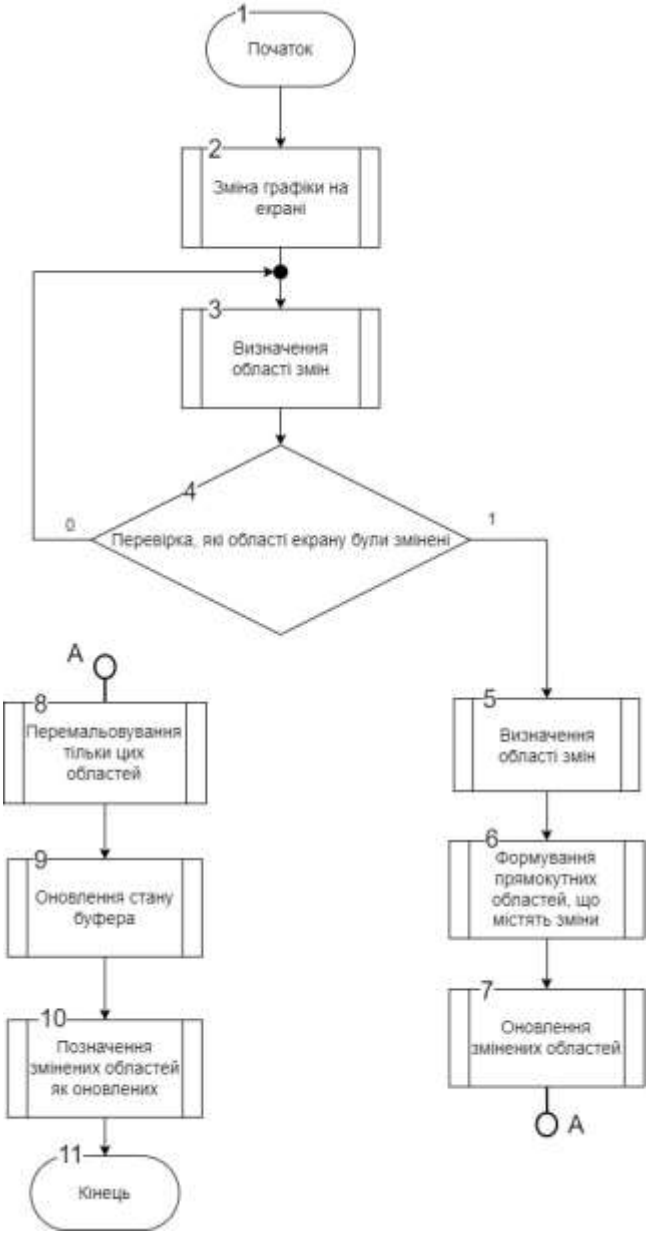

Рисунок 2.4 – Блок-схема роботи алгоритму аналізу заповненості буфера

2.4 Метод проміжної буферизації даних графічного буферу

Алгоритм аналізу даних за допомогою паралельними обчислення (команди XMM і MMX).

Так як алгоритм аналізу різниці між переднім і заднім буферами потребує великої кількості обчислень, було прийнято використати паралельні обчислення для пришвидшення аналізу.

Паралельні обчислення – одночасне використання кількох ресурсів ЕОМ для розв'язування задач:

1. Задача розбивається на підзадачі, які можуть виконуватися у один і той самий момент часу.
2. Кожна підзадача в свою чергу розбивається на послідовність інструкцій.
3. Інструкції кожної підзадачі виконуються одночасно на різних процесорах.
4. У процесі обчислень використовується загальний механізм контролю-координації. [24]

Схема паралельних обчислень наведена на рисунку 2.5.

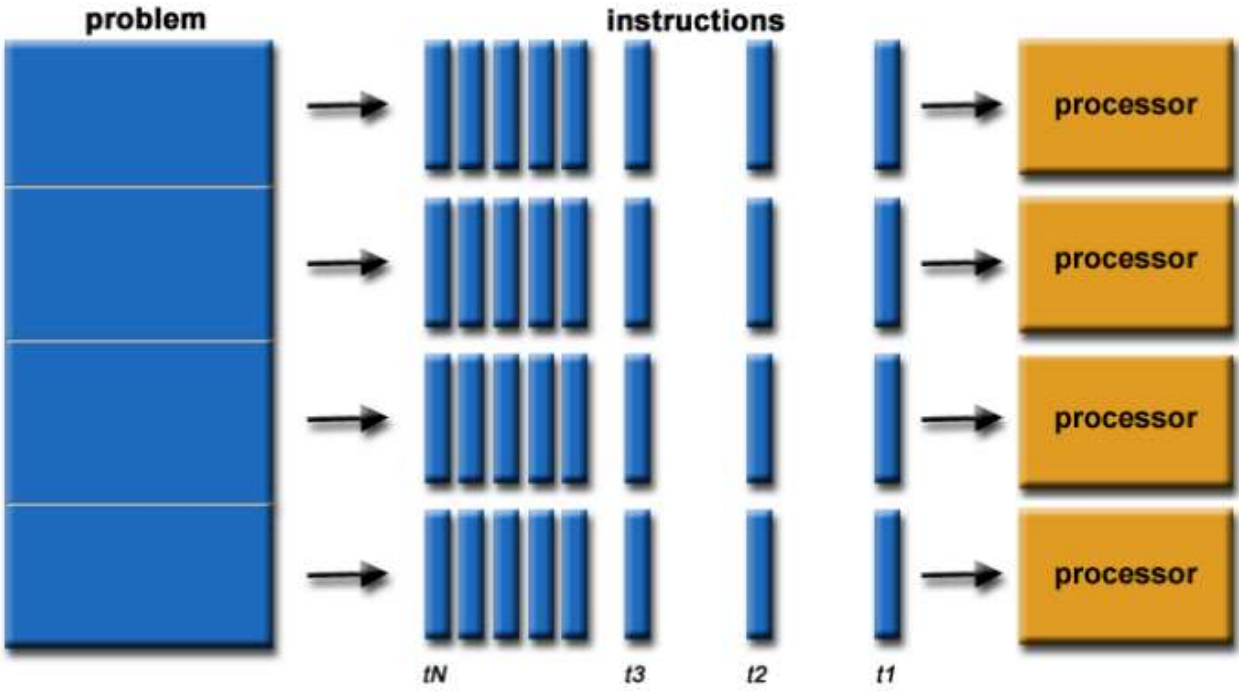

Рисунок 2.5 – Схема паралельних обчислень

Засоби, які дають змогу втілити парадигму паралелізму, можна класифікувати наступним чином:

1. Апаратні:
   – засоби для проведення обчислень (обчислювальна техніка);
   – обчислювальна техніка, зібрана з стандартних комплектуючих;
   – обчислювальна техніка, зібрана з спеціальних комплектуючих;
   – засоби візуалізації;
   – засоби для зберігання і обробки даних.
2. Програмні:
   – програмні засоби загального призначення (операційні системи, стандартні бібліотеки, мови;
   – програмування, компілятори, профайлери, дебагери і т.п.);
   – спеціальні програмні засоби: бібліотеки (PVM, MPI); засоби об’єднання ресурсів (Dynamite, Globus та інші).

Паралельні обчислення мають велике значення для обробки графіки через кілька критичних аспектів:

1. Швидкодія: Обробка графіки вимагає великої кількості обчислень, особливо при рендерингу складних сцен або відображенні великих обсягів даних, наприклад, при відтворенні відео. Паралельні обчислення дозволяють виконувати ці обчислення одночасно, розподіляючи їх на кілька обчислювальних одиниць.
2. Візуальна якість: Реалістичність графічного відображення може бути покращена завдяки паралельним обчисленням. Обчислення світла, тіней, текстур та інших деталей сцени в паралельних потоках може забезпечити більш точне й деталізоване відтворення.
3. Відтворення в реальному часі: У відеоіграх та віртуальній реальності (VR) важливо мати здатність відтворювати графіку в реальному часі без затримок. Паралельні обчислення допомагають оптимізувати цей процес, щоб забезпечити плавне та швидке відображення сцен.

4. Ефекти та анімація: Використання паралельних обчислень дозволяє створювати складні спеціальні ефекти (наприклад, ефекти частинок, води, диму) та анімацію з високою деталізацією, що додає реалізму та привабливості графічним сценам.

5. Обробка великих обсягів даних: Паралельність дозволяє ефективно обробляти великі обсяги графічних даних, таких як великі текстури, великі об'єкти або великі масиви даних, що полегшує роботу з ними та швидше їх відображення.

6. Візуалізація наукових даних: У наукових областях паралельні обчислення допомагають відобразити складні дані у вигляді графіків, діаграм або моделей, що сприяє кращому розумінню та аналізу цих даних.

Для реалізації паралельних обчислень на платформах з процесорами можна використати MMX/XMM команди.

MMX (MultiMedia eXtension) та XMM (eXtended MultiMedia) команди в наборі SIMD (Single Instruction, Multiple Data) інструкцій, спрямованих на оптимізацію обробки мультимедійних даних на процесорах від Intel. Вони дозволяють виконувати однакові операції над декількома даними одночасно, що особливо корисно для обробки графіки. [25]

Ці команди використовуються у графічних операціях для роботи з пікселями, обробки текстур, операцій над зображеннями та векторної графіки. Вони дозволяють виконувати операції швидше та більш ефективно за рахунок виконання однієї операції над кількома елементами даних одночасно.

MMX/XMM можуть використовуватись для:

1. Обробки зображень: Виконання операцій, таких як фільтрація, згладжування, розмиття, підсилення контрасту та інших ефектів обробки зображень на кількох пікселях одночасно.

2. Текстурна обробка: Застосування текстур та їх модифікація на основі MMX/XMM команд для підвищення деталізації та реалістичності текстур у графіці.

3. Операції з векторною графікою: Векторна графіка також може бути оптимізована за допомогою MMX/XMM, використовуючи їх для операцій над векторами та швидшого відображення складних форм із застосуванням оптимізованих векторних операцій.

4. Графічні ефекти: Реалізація спеціальних ефектів, таких як плавне рухоме розмиття, псевдо-3D ефекти, освітлення та тіні, за допомогою ефективних мультимедійних операцій.

Команди MMX/XMM можна використовувати деякими мовами програмування:

1. Assembler: Найбільш прямий спосіб використовувати MMX/XMM команди – це через мову асемблера, оскільки вони вбудовані безпосередньо в архітектуру процесора. Асемблер дозволяє прямий доступ до низькорівневих інструкцій процесора, що дозволяє максимально ефективно використовувати MMX/XMM.

2. C/C++: Ці мови програмування також підтримують використання MMX/XMM команд через спеціальні функції та бібліотеки. Наприклад, у C/C++ можна використовувати SIMD (Single Instruction, Multiple Data) розширення, такі як SSE, SSE2, SSE3, які використовують MMX/XMM.

3. Fortran: Також існують можливості для використання MMX/XMM через Fortran, використовуючи спеціальні бібліотеки та функції.

4. Python: Існують бібліотеки, які надають доступ до оптимізованих функцій на основі MMX/XMM для Python. Наприклад, бібліотека Numba може використовувати SIMD інструкції для оптимізації обчислень.

Регістри SSE (рисунок 2.6) дозволяють обчислювати декілька наборів цілих даних і даних з плаваючою комою одночасно. SSE працює з усіма стандартними типами даних, включаючи числа з плаваючою комою подвійної точності та будь-які цілі числа від восьми до 128 біт. [26]

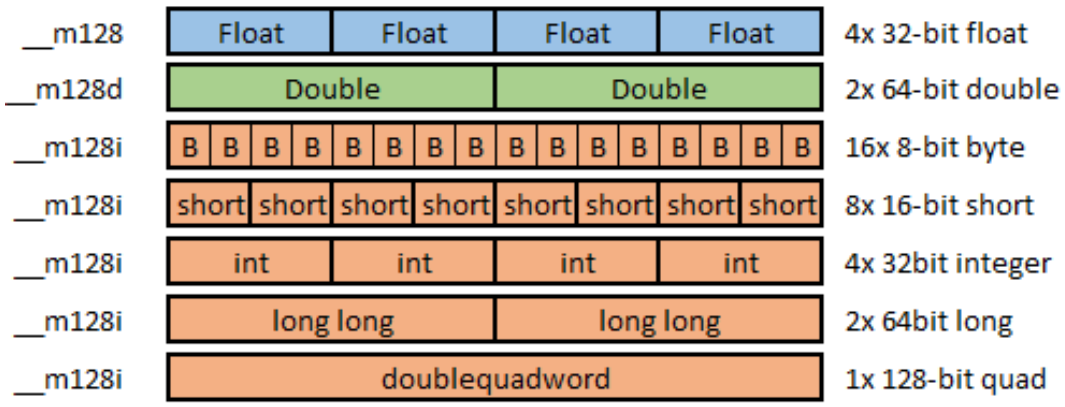

Рисунок 2.6 – Тип даних регістрів SSE

Тут ми надали трохи інформації про комп'ютер, на якому цей код буде виконуватися. Зокрема, ми сказали компілятору, що цільовий процесор підтримує розширення набору інструкцій x86 під назвою AVX2. AVX2 є одним з багатьох так званих «SIMD розширень» для x86. Ці розширення включають інструкції, які працюють зі спеціальними регістрами, здатними зберігати 128, 256 або навіть 512 біт даних, використовуючи підхід «одна інструкція, декілька даних» (SIMD). Замість того, щоб працювати з одним скалярним значенням, інструкції SIMD розбивають дані в регістрах на блоки по 8, 16, 32 або 64 біти і виконують одну і ту ж операцію над ними паралельно, що призводить до пропорційного збільшення продуктивності.

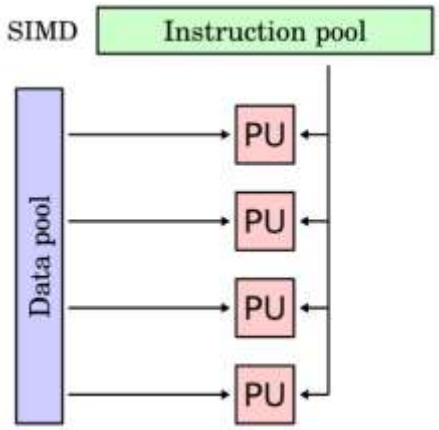

Рисунок 2.7 – Підхід SIMD «Одна інструкція, декілька даних»

Ці розширення є відносно новими, і їх підтримка в процесорах впроваджувалася поступово, зберігаючи при цьому зворотну сумісність. Окрім додавання більш спеціалізованих інструкцій, найважливіша відмінність між ними полягає у впровадженні все ширших регістрів. Еволюція SIMD наведена на рисунку 2.8.

Зокрема, AVX2 має інструкції для роботи з 256-бітними регістрами, тоді як за замовчуванням GCC припускає, що нічого довшого за 128-біт SSE2 не включено. Отже, після вказівки оптимізатору, що він може використовувати інструкції, які додають одразу 8 цілих чисел замість 4, продуктивність була збільшена вдвічі.

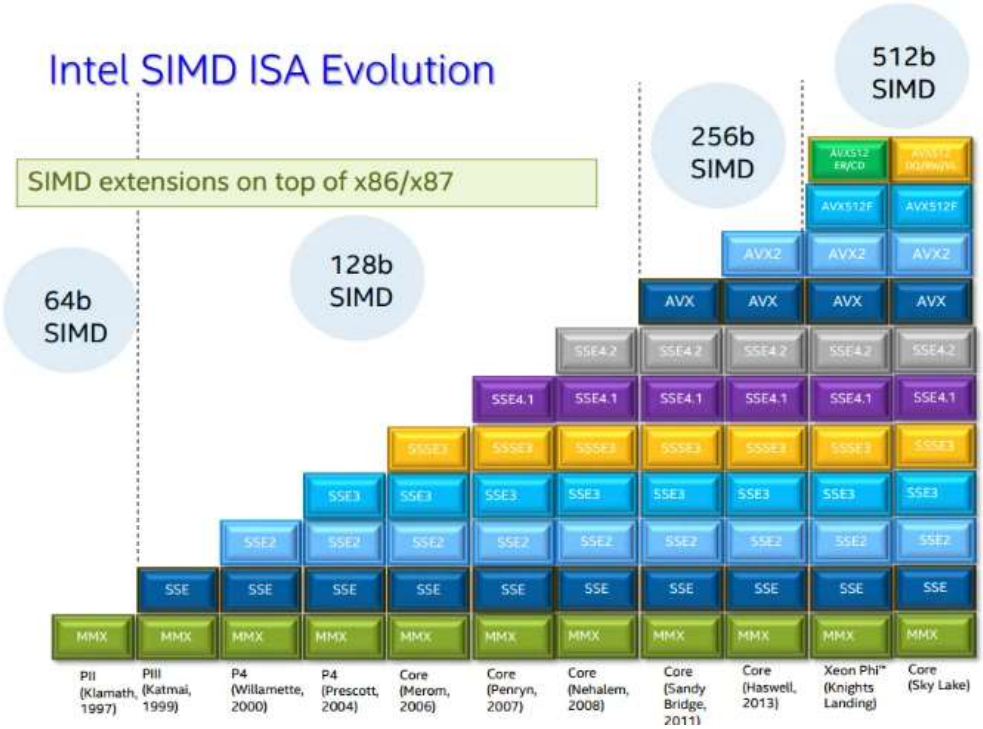

Рисунок 2.8 – Еволюція SIMD

Процесори MMX мали бути переведені в режим MMX, який перетворював перші 64 біти восьми 80-бітних регістрів з плаваючою комою процесора в регістри MMX. Коли 8, 16 або 32-розрядні цілі числа завантажувалися в регістри, їх можна було додавати і множити одночасно (рисунок 2.9). Починаючи з Pentium III, MMX було замінено на потокові

розширення SIMD від Intel (SSE), і операції з плаваючою комою та мультимедійні операції більше не були взаємовиключними.

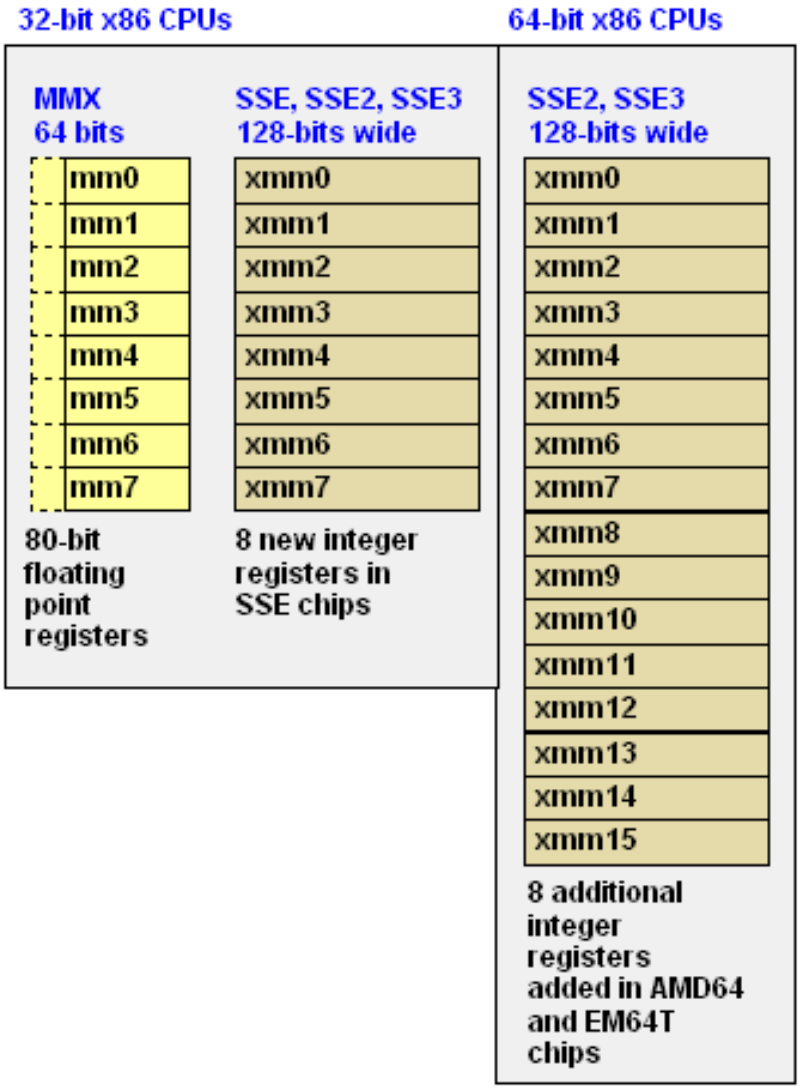

Рисунок 2.9 – Структура 32 і 64 бітних процесорів з MMX/SSE

Блок-схема роботи алгоритму аналізу змін даних в буферах з використанням паралельних обчислень наведена на рисунку 2.10

Алгоритм аналізу різниці між переднім та заднім буфером використанням паралельних обчислень буде виглядати таким чином:

1. Розбиття буферів на сегменти: Передній і задній буфери можна розділити на менші сегменти для паралельної обробки. Кожен сегмент може бути призначений окремому потоку або задачі для аналізу.

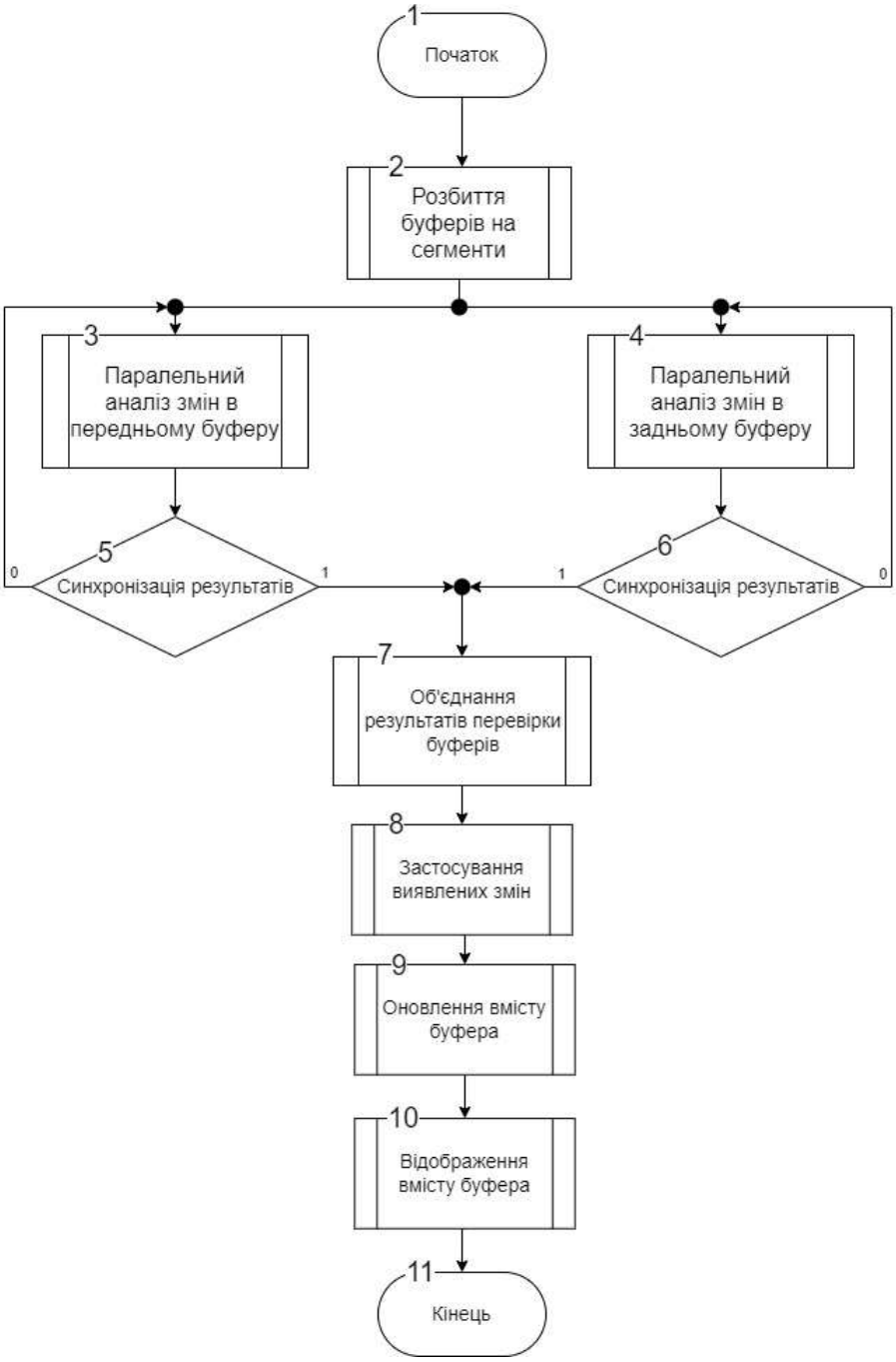

Рисунок 2.10 – Блок-схема роботи алгоритму аналізу змін даних в буферах з використанням паралельних обчислень

2. Паралельний аналіз змін: Кожен сегмент переднього та заднього буфера обробляється паралельно. Потоки або задачі аналізують різницю між відповідними сегментами для виявлення змін.

3. Синхронізація та об'єднання результатів: Після аналізу кожного сегмента, потрібно синхронізувати результати та об'єднати їх, щоб отримати повну картину змін між переднім і заднім буферами.

4. Застосування виявлених змін: Виявлені різниці вмісту між буферами можуть бути використані для оновлення тільки змінених областей на екрані, що дозволить ефективно відображати лише зміни, а не весь вміст.

5. Оновлення та відображення: Після виявлення змін та їх застосування, вміст переднього буфера оновлюється та відображається на екрані.

Алгоритм аналізу зміни даних буферів з використанням паралельних обчислень сприяє швидкому та ефективному аналізу змін між переднім та заднім буферами, що є важливим для забезпечення високої продуктивності при відображенні графічного вмісту на екрані.

2.5 Висновки

В розділі проведено аналіз інформаційного забезпечення методу оптимізації, результаном якого є визначення оптимізації програмного забзпечення і графічної оптимізації.

Було розроблено модель роботи ігрової системи, вона передбачає основну ігрового процесу, в який буде занурений гравець. Також було розроблено алгоритм роботи 2D гри «Magical adventure», який зображений в блок схемі і функціональну модель персонажу.

Розглянуто метод подвійної буферизації, у якому на відміно від існуючого використано аналіз наповненості буферу, що дозволило підвищити швидкість оновлення графічних даних для відображення на екрані і як наслідок підвищення продуктивності. Розглянуто метод проміжної буферизації даних у якому на відміно від існуючого використано паралельні обчислень даних графічного буферу для підвищення швидкодії формування даних.

Розроблено два алгоритми оптимізації, алгоритм аналізу заповненості буфера та алгоритм проміжної буферизації з паралельними обчисленнями.

3 РОЗРОБКА ПРОГРАМНОЇ КОМПОНЕНТИ

3.1 Варіантний аналіз і обґрунтування вибору мови програмування

Об'єктно-орієнтована мова програмування це мова, яка базується на понятті об'єктів, які містять дані та поведінку. Об'єкти можуть взаємодіяти один з одним за допомогою повідомлень, які викликають методи об'єктів. Об'єктно-орієнтоване програмування має багато переваг, таких як абстракція, інкапсуляція, поліморфізм та наслідування, які сприяють модульності, реюзабельності, зрозумілості та підтримці коду. Об'єктно-орієнтоване програмування використовується у багатьох сучасних мовах програмування, таких як Python, C++, C# та інших.

Java – це сучасна, об'єктно-орієнтована мова, яка дозволяє створювати надійні, швидкі та багатофункціональні програми для різних платформ. Java використовує віртуальну машину, яка забезпечує незалежність від апаратного забезпечення та операційної системи. Java має багатий набір бібліотек, які надають можливості для роботи з графікою, мережами, базами даних, безпекою та іншими аспектами програмування. Java також підтримує парадигми функціонального та паралельного програмування, що збільшує продуктивність та гнучкість розробки. Використовується для розробки веб-додатків, мобільних додатків, великих корпоративних систем та багатьох інших застосувань. Має середню продуктивність, що робить його менш ефективним за C++, але швидшим за Python [27].

Перевагами Java є:

1. Багатопоточність. Java підтримує багатопоточне програмування. Це означає, що код на Java може виконувати декілька завдань одночасно за допомогою потоків. Такий підхід покращує продуктивність та використання ресурсів системи, особливо у випадках, коли програма має виконувати складні або тривалі операції.

2. Безпека. Java надає високий рівень безпеки для своїх застосунків.

Це досягається за рахунок ряду механізмів, таких як перевірка типів даних, управління пам'яттю, перевантаження операторів, захист виключень та безпечне завантаження класів. Крім того, Java використовує концепцію пісочниці (sandbox), яка ізолює код від небезпечних або неавторизованих джерел.

3. Стандартизація. Java має багато стандартних бібліотек та фреймворків, які надають готові розв'язки для поширених проблем програмування.

4. Бібліотеки. Java має широкий набір вбудованих бібліотек для різних завдань, що полегшує розробку програм та дозволяє використовувати готові рішення.

На Java розробляють:

1. Веб-додатки: Java використовується для створення веб-додатків за допомогою фреймворків, таких як Spring та Java EE.

2. Мобільні додатки: Для розробки Android-додатків використовується Java, а також Kotlin (яка працює на JVM).

3. Вбудовані системи: Java використовується для розробки програмного забезпечення вбудованих систем, IoT-пристроїв та багатьох інших областей.

4. Бізнес-додатки: Java використовується для розробки великих корпоративних систем, ERP та CRM систем.

5. Ігри: Хоча не так популярна, як C++ у галузі ігор, Java використовується для розробки деяких варіантів ігор.

Java залишається однією з найпопулярніших мов програмування завдяки своїй переносимості, надійності та широкому спектру застосувань. Її велика спільнота розробників та набір інструментів роблять її потужним інструментом для багатьох завдань у світі програмування.

Мова програмування C++ є однією з найпопулярніших та потужних мов програмування, яка підтримує об'єктно-орієнтоване, процедурне та узагальнене програмування. C++ є розширенням мови C, яка додає підтримку

класів, наслідування, поліморфізму, шаблонів та інших функцій. C++ використовується для розробки різноманітних застосунків, таких як операційні системи, графічні інтерфейси користувача, ігри, бази даних, мережеве програмування тощо. C++ є стандартизованою мовою, яка має багато реалізацій для різних платформ. Для написання програм на C++ потрібно мати компілятор C++, який перетворює вихідний код на виконуваний файл. [28]

Перевагами мови програмування C++:

1. Швидкодія: Висока продуктивність та швидкість виконання завдяки можливості оптимізації та контролю над апаратними ресурсами.
2. Близькість до машинного коду: Можливість працювати на низькому рівні дає контроль над пам'яттю та ресурсами комп'ютера.
3. Широкий вибір бібліотек та фреймворків: Велика кількість готових рішень полегшує розробку та розширення функціоналу програм.
4. Безпека та стабільність: Достатньо висока рівні надійності та стабільності виконання коду.
5. Широкий спектр інструментів та підтримки: Велика спільнота розробників, багато ресурсів, документації та інструментів для розробки.
6. Розширюваність: Можливість використовувати низькорівневі механізми для оптимізації та розширення функціоналу програм.

Сфери застосування мови програмування C++:

1. Системне програмування: C++ використовується для розробки операційних систем, драйверів, компіляторів та іншого системного програмного забезпечення.
2. Вбудовані системи: Мова дозволяє розробляти програмне забезпечення для електроніки, IoT-пристроїв, мікроконтролерів та інших вбудованих систем.
3. Ігрова індустрія: Часто використовується для розробки ігор, так як C++ надає близькість до апаратних ресурсів та високу продуктивність.
4. Фінансові програми: Великі фінансові установи використовують

C++ для розробки торгових систем, програм для аналізу даних та інших фінансових програм.

5. Графічні програми та візуалізація: Використовується для розробки графічних програм, комп'ютерної графіки, CAD/CAM-систем, обробки зображень та відео.

6. Мобільні додатки: Хоча Java та Kotlin популярніші для Android-розробки, C++ використовується для розробки певних частин Android-додатків та геймдеву.

7. Наукові дослідження: Використовується для моделювання, обчислень у наукових та інженерних дослідженнях через високу швидкодію та можливості оптимізації.

8. Великі системи та корпоративні програми: C++ використовується для розробки великих корпоративних систем, серверних додатків та високонавантажених систем.

Це лише найбільші сфери, де використовується мова програмування C++, завдяки її ефективності, швидкості та можливостям контролю над апаратними ресурсами.

Python – це високорівнева, інтерпретована, загальнопризначена мова програмування, яка була розроблена Гвідо ван Россумом і випущена у 1991 році. Вона має простий синтаксис, що робить її дуже доступною для початківців і в той же час потужною для професіоналів. Python підтримує кілька парадигм програмування, зокрема процедурне, об'єктно-орієнтоване, аспектно-орієнтоване та функціональне програмування.

Основні переваги мови програмування Python:

1. Простий синтаксис: Python має простий та легкий для вивчення синтаксис, що робить його доступним для початківців у програмуванні.

2. Кросплатформеність: Код Python може запускатися на різних операційних системах без змін, що робить його кросплатформеним.

3. Великий набір бібліотек: Python має велику кількість стандартних бібліотек та сторонніх модулів, які дозволяють виконувати різноманітні

завдання без початкової реалізації.

4. Швидка розробка: Завдяки простому синтаксису та великому набору інструментів, розробка в Python може бути швидшою порівняно з іншими мовами програмування.

5. Комуніті та підтримка: Python має велику та активну спільноту розробників, яка надає безліч ресурсів, бібліотек, фреймворків та допомогу у вирішенні проблем.

6. Розширюваність: Можливість інтеграції коду з іншими мовами, такими як C/C++, робить Python потужним інструментом для оптимізації та розширення функціоналу програм. [29]

Основні сфери застосування мови програмування Python:

1. Розробка веб-додатків. Python має багато фреймворків, таких як Django, Flask, Pyramid, які дозволяють створювати динамічні та інтерактивні веб-сайти та API.

2. Аналіз даних та наука про дані. Python має потужні бібліотеки для обробки, візуалізації та моделювання даних, такі як pandas, Numba, scipy, matplotlib, seaborn, scikit-learn, tensorflow та інші.

3. Машинне навчання та штучний інтелект. Python є однією з найпопулярніших мов для розробки алгоритмів машинного навчання та штучного інтелекту, завдяки своїй гнучкості, простоті та великій кількості доступних ресурсів.

4. Автоматизація та скриптинг. Python дозволяє автоматизувати рутинні та складні задачі, такі як парсинг веб-сторінок, робота з файлами та папками, надсилання електронних листів, робота з базами даних тощо.

5. Розробка настільних додатків. Python може використовуватися для створення графічних користувацьких інтерфейсів (GUI) за допомогою бібліотек, таких як Tkinter, PyQt, wxPython тощо.

Порівння мов програмування Java, C++ і Python наведено в таблиці 3.1.

Таблиця 3.1 – Порівняльння мов програмування

Характеристика	Java	C++	Python
Синтаксис	1	0	1
Продуктивність	0	1	0-
Кросплатформеність	1	0	1
Документація	1	1	1
Підтримка бібліотек	0	1	1
Аналіз даних	1	0	1
Всього	4	3	5

Отже, проаналізувавши відмінні ознаки Python, Java та C++, було прийнято рішення, обрати мову програмування Python.

В якості графічного рушія було обрано Pygame, який підтримує мову програмування Python.

Pygame (рисунок 3.1) – це набір модулів Python, призначених для написання відеоігор. Pygame додає функціональність на додаток до бібліотеки SDL, це дозволяє створювати повнофункціональні ігри та мультимедійні програми на мові Python. [30]

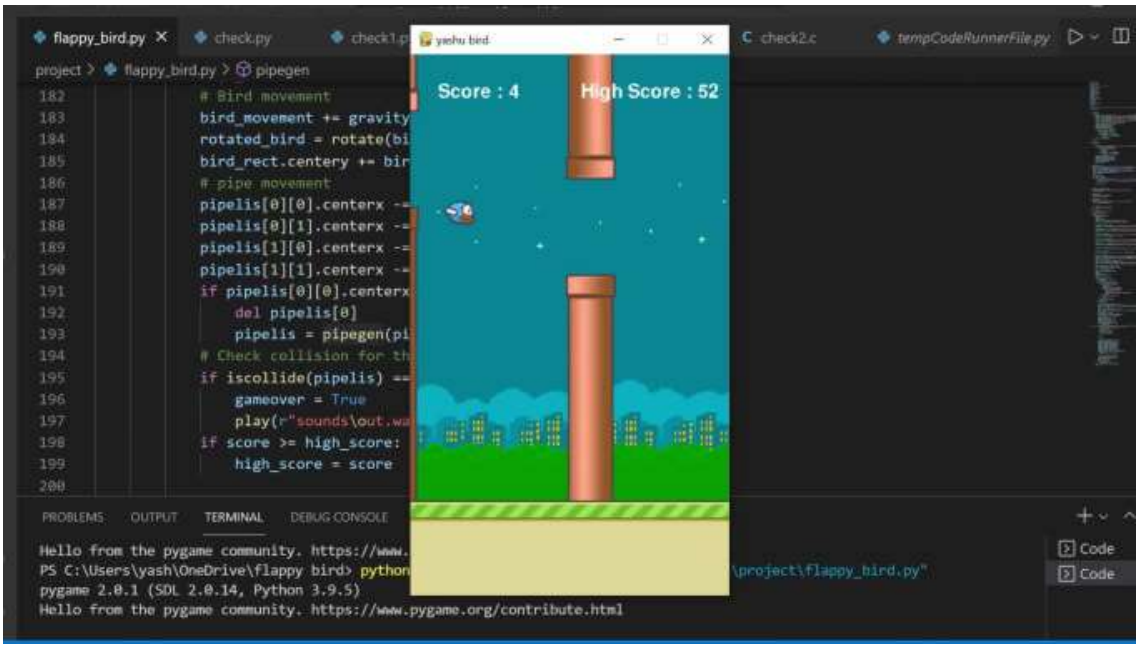

Рисунок 3.1 – Запуск гри у Pygame

Pygame він має такі переваги:

1. LGPL-ліцензія – абсолютно безкоштовний доступ як для приватних осіб, так і для комерційних компаній.

2. Портативний. Підтримує Linux (Pygame постачається з більшістю основних дистрибутивів Linux), Windows (95, 98, ME, 2000, XP, Vista, 64-розрядна Windows тощо), Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX і QNX.

3. Простий і легкий у використанні. Pygame використовується в проекті OLPC і викладався на курсах есе для маленьких дітей і студентів.

4. Велика база релізів ігор. У тому числі фіналісти Indie Game Festival, фіналісти австралійського фестивалю ігор, популярне умовно-безкоштовне програмне забезпечення, мультимедійні проекти та ігри з відкритим кодом. На веб-сайтах Pygame опубліковано понад 660 проектів, таких як: список необхідних. Багато інших ігор було випущено з SDL (на якому базується pygame), тому можна бути впевненим, що багато з них добре перевірено мільйонами користувачів.

5. Невелика кількість коду. У ньому немає сотень тисяч рядків коду для речей, які не будуть використовувані. Ядро залишається простим, а додаткові речі, такі як бібліотеки графічного інтерфейсу та ефекти, розробляються окремо за межами Pygame.

6. Детальна документація;

7. Багатоядерні процесори можна легко використовувати. Завдяки звичайним двоядерним ЦП і 8-ядерним ЦП, доступним дешево в настільних системах, використання багатоядерних ЦП дозволяє робити більше у своїх ігрових додатках.

8. Модульний. Є можливість використовувати частини Pygame окремо. Наприклад для використовування інших звукових бібліотек. Багато основних модулів можна ініціалізувати та використовувати окремо.

Pygame має достатню кількість інструментів для реалізації поставлених задач.

3.2 Вибір середовища розробки

Для вибору середовища розробки було розглянуто наступні середовища:

1. PyCharm.
2. Thonny.
3. Spyder.
4. Visual Studio Code.

PyCharm (рисунок 3.2) – це потужне та універсальне інтегроване середовище розробки (IDE) для програмістів на Python. Воно пропонує ряд функцій, які допоможуть вам легко та ефективно писати, налагоджувати, тестувати та рефакторити ваш код. Ось деякі з переваг використання PyCharm:

1. Підтримує декілька версій Python та інтерпретаторів, включаючи віртуальні середовища та середовища conda.
2. Інтелектуальне завершення коду, підсвічування синтаксису, аналіз коду та інструменти навігації по коду.
3. Інтегрується з популярними фреймворками та бібліотеками, такими як Django, Flask, Numba, Pandas, TensorFlow тощо.
4. Має вбудований налагоджувач, профайлер, запуск тестів та інструмент покриття коду.
5. Він підтримує веб-розробку, науку про дані, машинне навчання та інші сфери за допомогою спеціалізованих плагінів та інструментів. Деякі з плагінів, які ви можете встановити в PyCharm: PyLint, PyTest, Django Template Debugger, Markdown Support, Database Tools та інші. Ви можете переглядати та керувати плагінами за допомогою діалогового вікна Налаштування/Параметри.
6. Він має настроюваний і зручний інтерфейс, який адаптується до ваших уподобань і робочого процесу.
7. PyCharm доступний у двох редакціях: Professional і Community. Professional версія має більше можливостей і підтримує проекти з веб-розробки та науки про дані. Редакція Community є безкоштовною і має відкритий вихідний код, але має менше можливостей і підтримує лише

розробку на Python. [31]

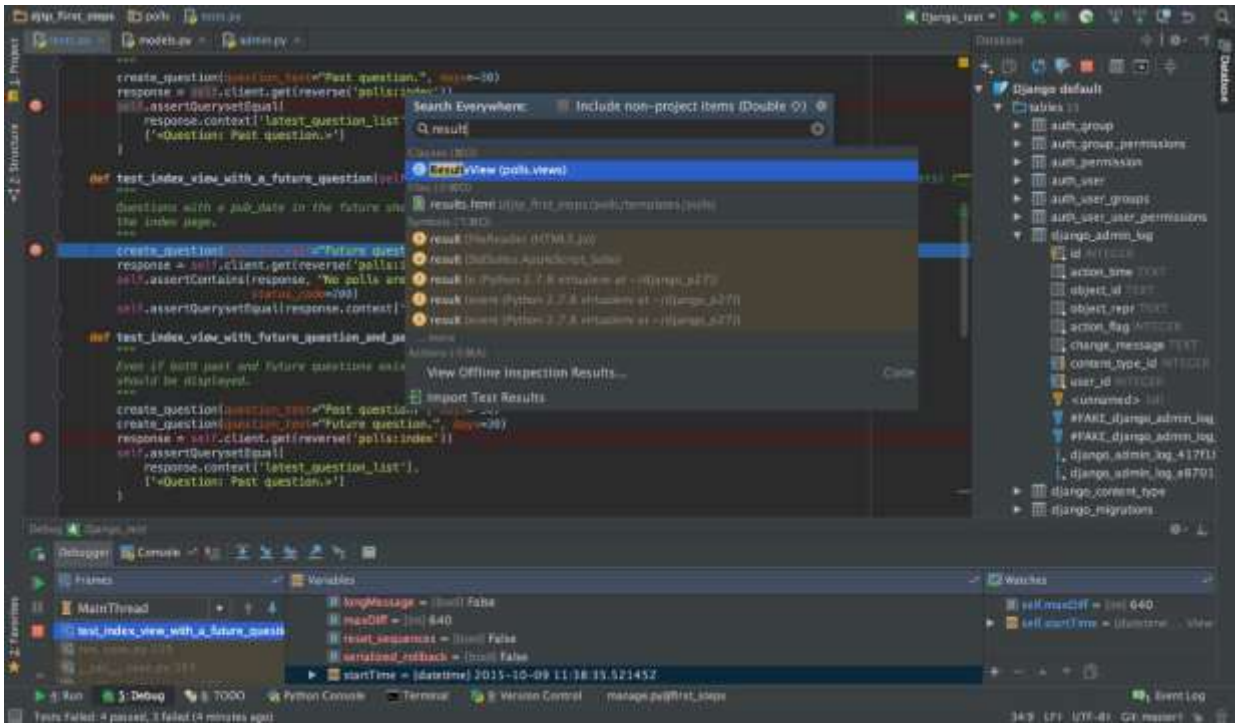

Рисунок 3.2 – Вікно програми PyCharm

Thonny (рисунок 3.3) – це інтегроване середовище розробки (IDE) для мови програмування Python, яке спрямоване на початківців та любителів. Thonny надає простий та інтуїтивний інтерфейс, який дозволяє створювати, редагувати, запускати та налагоджувати Python-програми без зайвих налаштувань або складних команд. Thonny також має вбудований інтерпретатор Python, який дозволяє виконувати код крок за кроком та спостерігати за змінами змінних, стеку викликів та пам'яті. Thonny підтримує роботу з різними версіями Python, включаючи Python 3.10, та дозволяє встановлювати додаткові пакети за допомогою менеджера пакетів pip. Thonny є безкоштовним та відкритим програмним забезпеченням, яке доступне для Windows, Mac OS X та Linux. [32]

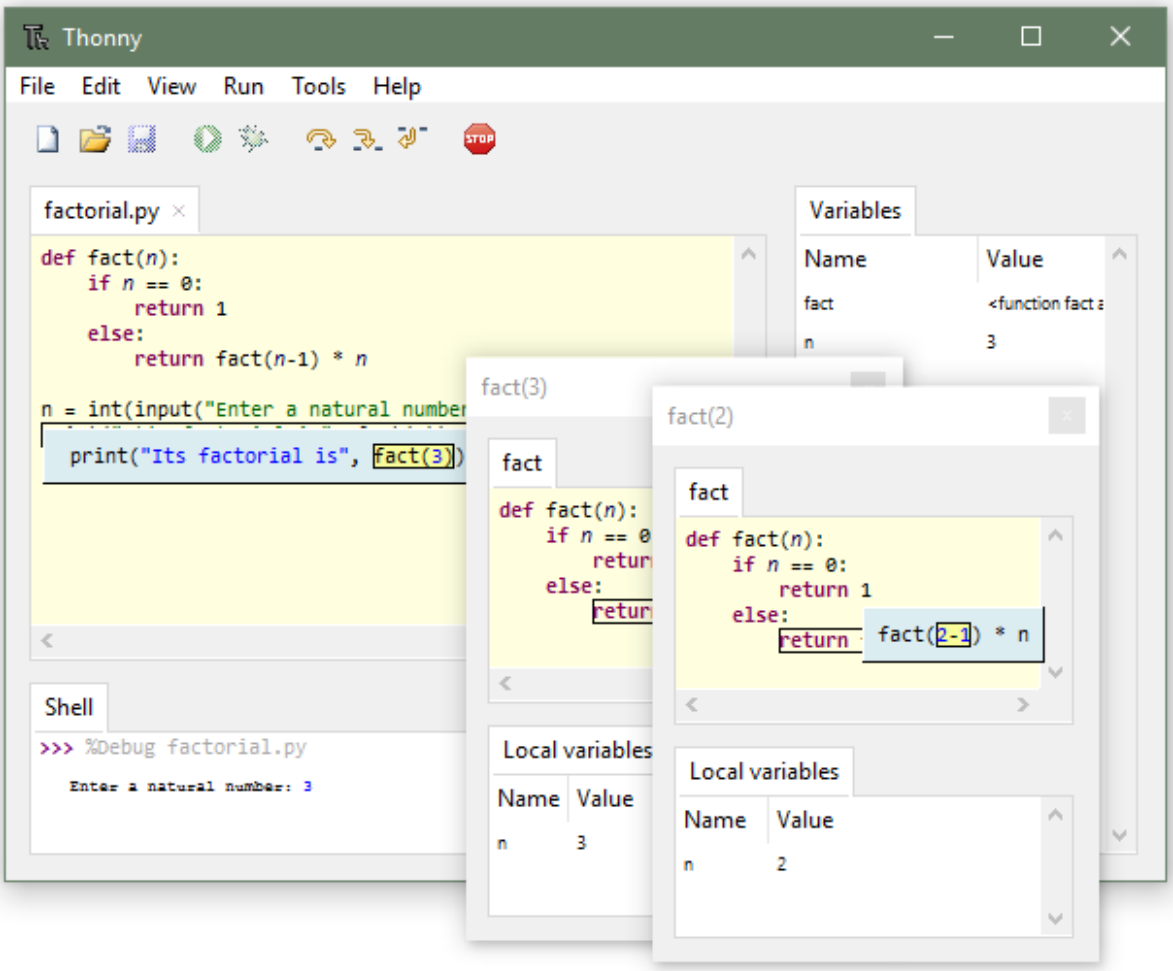

Рисунок 3.3 – Інтерфейс програми «Thonny»

Spyder (рисунок. 3.4) – це наукове середовище розробки (IDE), спеціально спроектоване для роботи з мовою програмування Python, зокрема орієнтоване на наукові обчислення та аналіз даних. Воно пропонує багато корисних інструментів та функцій, спрямованих на полегшення роботи з кодом, особливо у сфері аналітики даних та наукових досліджень.

Основні можливості Spyder:

1. Робота з науковими бібліотеками: Spyder має вбудовану підтримку наукових бібліотек, таких як NumPy, SciPy, Pandas та Matplotlib, що дозволяє легко виконувати аналіз даних, обробку числових даних та візуалізацію результатів.

2. Відлагоджування: IDE має вбудований відлагоджувач (debugger), який допомагає виявляти та виправляти помилки в коді, а також аналізувати

значення змінних під час виконання програми.

3. Редактор з розширеними можливостями: Spyder має потужний текстовий редактор з можливістю автоматичного доповнення коду, підсвічування синтаксису, можливістю робити відступи та багато іншого.

4. Робота з консолью IPython: Вбудована консоль IPython дозволяє виконувати код по частинах та одержувати миттєві результати, що дуже зручно під час експериментів з даними.

5. Підтримка розширень: Spyder може бути розширений за допомогою плагінів та додаткових інструментів, що розширює його функціональність.

6. Управління проектами: Це середовище дозволяє організувати ваші проекти, створювати файли, взаємодіяти з ресурсами та структурувати вашу роботу.

Spyder є відмінним вибором для науковців, дослідників та аналітиків даних, оскільки воно надає широкий спектр інструментів для обробки даних, виконання наукових обчислень та візуалізації результатів, усе це з комфортом та зручністю роботи. [33]

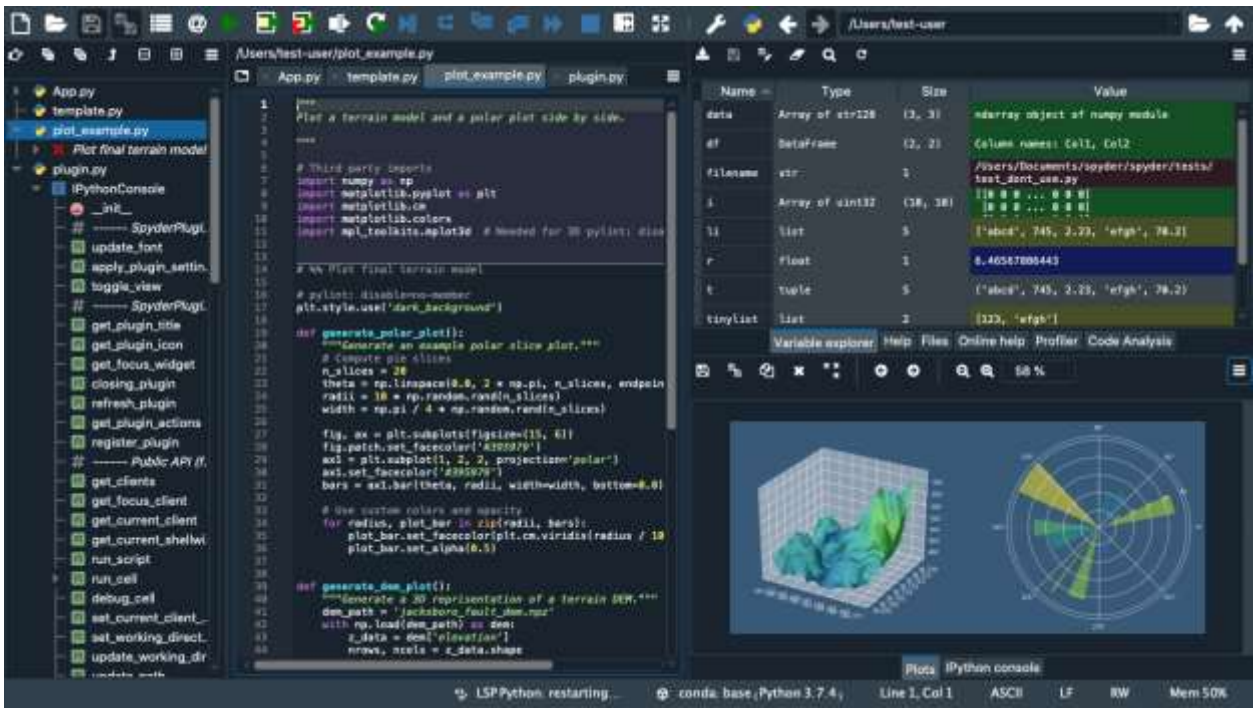

Рисунок 3.4 – Вікно IDE Spyder

Visual Studio Code (рисунок 3.5) – це легке, потужне та відкрите середовище розробки, розроблене компанією Microsoft. Воно є одним з найпопулярніших інструментів серед розробників для написання коду на різних мовах програмування, включаючи Python. [36] Основні можливості Visual Studio Code:

1. Розширені можливості редагування: VS Code має потужний редактор з підсвічуванням синтаксису, автоматичним доповненням коду, рефакторингом та багатьма іншими функціями, що полегшують написання коду.

2. Робота з мовами програмування: Хоча спочатку був створений для роботи з JavaScript та web-технологіями, VS Code підтримує багато мов програмування, включаючи Python, Java, C++, PHP, Ruby, і багато інших.

3. Розширюваність та плагіни: VS Code має широкий вибір розширень, які можна встановити для розширення функціональності редактора. Це дозволяє налаштувати його під свої потреби та отримати доступ до різноманітних інструментів.

4. Інтеграція з Git: Вбудована підтримка системи контролю версій Git дозволяє легко відслідковувати та керувати змінами в коді безпосередньо з редактора.

5. Відлагоджувач (Debugger): Інтегрований відлагоджувач спрощує виявлення та виправлення помилок у вашому коді.

6. Робота з розширеними проектами: VS Code може легко працювати з великими та складними проектами, надаючи можливості для організації файлів та папок.

5. Спільне використання: Зручність використання та можливості спільної роботи над проектами зробили VS Code популярним серед розробників, що працюють в команді. [34]

Це лише деякі з можливостей Visual Studio Code.

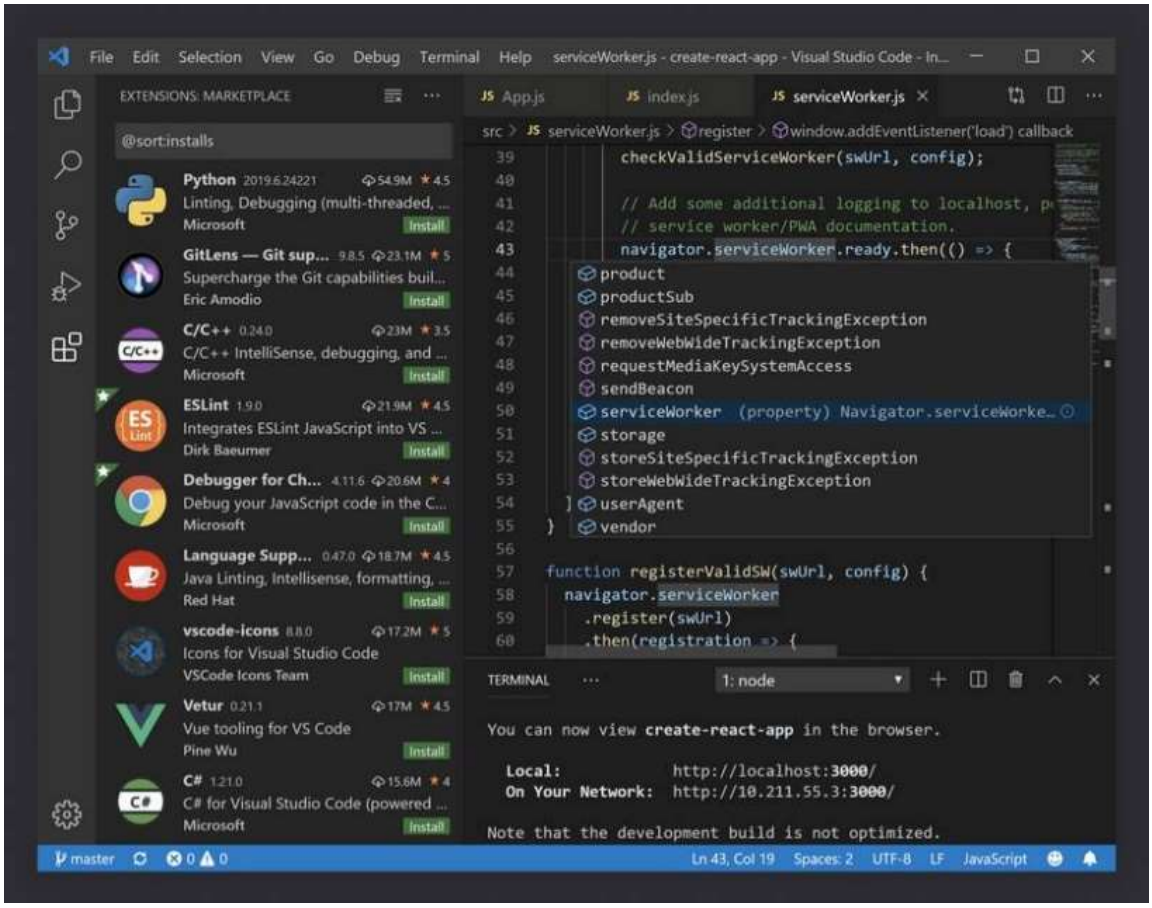

Рисунок 3.5 – Інтерфейс програми «Visual Studio Code»

Visual Studio Code гнучкість, легкість використання та багатий функціонал роблять його привабливим вибором для розробників у будь-якій сфері програмування

Проаналізувавши різні середовища розробки, було складено порівняльну таблицю 3.2.

Таблиця 3.2 – Порівняння середовищ розробки

Критерії	PyCharm	Thonny	Spyder	Visual Studio Code
Легкість використання	0	1	1	1
Підтримка віртуальних середовищ	1	0	0	1

Продовження таблиці 3.2

Критерії	PyCharm	Thonny	Spyder	Visual Studio Code
Підтримка розширень та плагінів	1	1	0	1
Вбудований відлагоджувач	1	1	1	1
Інтеграція з системами контролю версій	1	0	0	1
Розширені можливості редактора	1	1	1	0
Наявність вбудованих інструментів для наукових обчислень	0	0	1	1
Підтримка Pygame	1	0	0	1
Всього	6	4	4	7

Аналізуючи вищенаведені дані, Visual Studio Code відповідає більшій кількості критерій ніж конкуренти в обличчі Spyder, Thonny і PyCharm. Тому її було обрано для розробки програмні компоненти.

3.3 Розробка програмної реалізації модифікованого методу

Python мова програмування має широкий спектр можливостей у сфері обробки графіки. Вона пропонує багато інструментів і бібліотек, які дозволяють створювати, обробляти, візуалізувати та аналізувати графічні дані та зображення.

У Python робота з графічними буферами може бути виконана за допомогою різноманітних бібліотек, які надають інструменти для отримання, маніпулювання та відображення графічних даних безпосередньо на екрані. Інструменти для роботи з графічними буферами у Python:

1. Pillow (PIL Fork): Це бібліотека для обробки зображень, яка дозволяє завантажувати, зберігати, маніпулювати та відображати зображення на екрані. Вона надає інструменти для роботи з пікселями, фільтрації, зміни розміру та інших операцій.

2. OpenCV: Це потужна бібліотека для комп'ютерного зору та обробки зображень. OpenCV може завантажувати, обробляти та відображати зображення, а також виконувати аналіз та розпізнавання об'єктів на зображеннях.

3. Matplotlib: Ця бібліотека використовується для візуалізації даних, включаючи створення графіків та діаграм. Вона також може використовуватися для створення зображень і відображення їх у вікнах чи ноутбуках.

4. Pygame: Ця бібліотека призначена для створення ігор та інших графічних додатків. Вона надає інструменти для малювання графічних об'єктів, роботи з мишею та клавіатурою, а також відображення графіки на екрані.

5. Tkinter, PyQt, wxPython: Ці бібліотеки дозволяють створювати графічні інтерфейси користувача (GUI) та відображати на них графіку.

6. OpenGL: Це стандарт для роботи з графікою, який дозволяє створювати 3D графіку та відображати її на екрані. [35]

Програмна реалізація модифікованого методу подвійної буферизації буде розроблена за допомогою Pygame.

Pygame підтримує подвійний буфер за допомогою функцій pygame.Surface (створення буферів) і pygame.display.flip(), яка оновлює весь екран за один раз. Це забезпечує плавну анімацію та високу швидкість рендерингу.

Для того, щоб використовувати подвійний буфер в Pygame, потрібно створити об'єкт pygame.display, який представляє екран. Потім потрібно створити об'єкт pygame.Surface, який представляє буфер для малювання. На цьому буфері можна виконувати різні операції з графікою, такі як рисування

фігур, тексту, зображень тощо. Після того, як буфер заповнений, потрібно скопіювати його на екран за допомогою функції pygame.display.blit(). Нарешті, потрібно викликати функцію pygame.display.flip(), щоб оновити екран і показати новий кадр.

Для збільшення швидкості аналізу графічних буферів, необхідно використати бібліотеку Numba. Бібліотека Numba може використовувати SIMD інструкції для оптимізації обчислень.

Аналіз даних заповненості буфера у Pygame за допомогою бібліотеки Numba можна здійснити таким чином:

1. Створення вікна Pygame та буферів:

– Спочатку необхідно створити вікно Pygame, де буде відображатися графіка.

– Створити два буфери: передній та задній за допомогою pygame.Surface.

2. Малювання в задньому буфері:

– Використати функції Pygame для малювання (наприклад, pygame.draw), для створення графічних об'єктів в задньому буфері.

– Операції малювання із змінними координатами, кольорами, текстурами та формами будуть відображені в задньому буфері.

3. Копіювання заднього буферу в передній:

– Після малювання в задньому буфері скопіюйте його вміст в передній буфер.

– Використати функції Numba для копіювання даних між буферами.

4. Оновлення екрану:

– Після копіювання заднього буферу в передній, використати Pygame для відображення змін на екрані.

– Функція pygame.display.flip() або pygame.display.update() дозволяє оновити екран та показати зміни, зроблені в передньому буфері.

5. Аналіз даних у буферах за допомогою команд MMX/XMM (SIMD) використовуючи бібліотеку Numba:

– За допомогою Numba отримати доступ до значень пікселів у буферах.

– Виконати векторизовані операції Numba для отримати значення пікселів та їх порівняння для виявлення змін в графічних даних переднього і заднього буферу.

6. Формую область змінених даних dirty_rectangles.

7. Копіюю передній буфер на вікно pygame.surfarray.blit_array. Оновлюю лише області зі змінами pygame.display.update.

8. Очищення списку областей зі змінами після оновлення dirty_rectangles = []

Результат реалізації цього алгоритму в 2D грі «Magical adventure» створеній в Pygame буде комп’ютерна гра з оптимізованим управлінням пам'яттю та ресурсами. Що в свою чергу забезпечить плавність анімації та відображення графічних об'єктів завдяки оптимізації оновлення лише змінених областей екрану. Гра матиме більшу швидкодію завдяки використанню Numba для аналізу даних та оптимізації роботи з буферами графіки.

3.4 Висновки

В розділі проведено аналіз і обґрунтування мов програмувань Java, C++, Python і визначено переваги цих мов і сфери застосування. В результат чого для реалізації програмних компонент та систем візуалізації було обрано мову програмування Python.

Було обґрунтовано вибір середовища розробки Visual Studio Code. Тому що вона має ряд переваг над аналогами PyCharm, Thonny та Spyder, а саме підтримка бібліотеки Pygame; вбудований відлагоджувач; інтеграція з системами контролю версій.

Розроблена програмна реалізація модифікованого методу «Подвійна буферизація» алгоритм аналізу заповненості буферу та алгоритм проміжної буферизації з паралельними обчисленнями.

4 ТЕСТУВАННЯ ПРОГРАМНОЇ КОМПОНЕНТИ

4.1 Вибір методу тестування програмного забезпечення

Тестування комп'ютерних ігор – це важливий процес, спрямований на перевірку функціональності, якості та відповідності гри вимогам. Це починається з аналізу вимог до гри та розробки тестових сценаріїв. Тестери працюють на виконання цих сценаріїв, випробовуючи різні аспекти гри – від ігрового процесу та інтерфейсу до мультиплеєра.

Під час тестування фіксуються будь-які виявлені проблеми чи помилки – баги. Після виявлення багів їх документують, описуючи способи їх відтворення та вплив на гру. Після цього розробники виправляють помилки, і тестування повторюється для перевірки виправлень, а також для впевненості, що виправлені баги не вплинули на інші аспекти гри.

Крім тестування функціональності, важливим є перевірка продуктивності гри на різних пристроях і умовах, щоб впевнитися, що гра працює стабільно та швидко. Також тестують інші аспекти, такі як стабільність (чи гра не видає помилку під час тривалого ігрового процесу) та враження від користувача.

Усі ці етапи спрямовані на те, щоб забезпечити високу якість гри та створити задоволення для гравців, уникнувши вказівки на конкретні переваги чи недоліки

Тестування комп'ютерних ігор – це надзвичайно важливий етап у процесі їх розробки. Ось кілька причин, проводити тестування:
Якість геймплею: Тестування дозволяє переконатися, що гра цікава, викликає емоції та має правильно збалансований геймплей. Тестери відіграють роль звичайних гравців, щоб виявити будь-які недоліки чи проблеми, які можуть зіпсувати геймплей.

1. Виявлення помилок та багів: Тестування допомагає виявити та виправити будь-які технічні помилки, баги або недоліки у грі, такі як збої, неправильна робота ігрових механік чи інтерфейсу.

2. Оптимізація продуктивності: Тестування допомагає знайти обмеження продуктивності гри на різних пристроях та конфігураціях комп'ютерів. Це дозволяє розробникам оптимізувати гру для максимальної продуктивності.

3. Взаємодія користувача з ігрою: Тестери допомагають виявити труднощі або проблеми, які гравці можуть зустріти під час гри, та пропонують шляхи їх вирішення, покращення інтерфейсу користувача та узагальнення геймплею.

4. Забезпечення стабільності та надійності: Важливо, щоб гра працювала стабільно без збоїв або зависань, тому тестування допомагає забезпечити стабільну та надійну роботу гри.

5. Відповідність вимогам та очікуванням користувачів: Тестування допомагає переконатися, що гра відповідає очікуванням цільової аудиторії та вимогам, встановленим розробниками.

Загалом класифікацію тестування програмного виглядає так:

1. Рівні тестування програмного забезпечення:
– модульне тестування;
– інтеграційне тестування;
– тестування системи;
– приймальне тестування.
2. Типи тестування програмного забезпечення:
– випробування димом;
– тестування продуктивності;
– перевірка відповідності;
– функціональне тестування;
– юзабіліті-тестування;
– регресійне тестування;
– тестування безпеки.
3. Методи тестування програмного забезпечення:
– спеціальне тестування;

– гнучке тестування;
– тестування чорної скриньки;
– тестування білої скриньки;
– тестування сірої скриньки.

Методика «Чорна скринька» в тестуванні програмного забезпечення орієнтована на перевірку функціональності системи, не вдаючись у деталі її внутрішньої реалізації. Тестувальники взаємодіють з програмою, спираючись на зовнішнє спостереження. Вони подають вхідні дані та аналізують вихідні результати, переконуючись у відповідності реакцій програми очікуваним.

Цей підхід не передбачає знання внутрішніх механізмів роботи програми, фокусуючись на тому, як система поводиться ззовні. Він дозволяє оцінити реальні можливості системи і перевірити, чи вона відповідає вимогам та очікуванням користувачів.

Тестувальники створюють тестові сценарії на основі вимог до програми. Вони перевіряють різні аспекти системи, включаючи функціональність, інтерфейс користувача та можливість виконання основних операцій. Виявлені проблеми документуються та передаються розробникам для виправлення.

Одним з головних переваг цього методу є його спрямованість на оцінку зовнішніх характеристик системи, але при цьому він може пропускати внутрішні помилки чи проблеми реалізації, що потребують детального технічного аналізу для виявлення. У таблиці 4.1 перераховані переваги та недоліки тестування методики «Чорна скринька».

Методика тестування «Біла скринька» – це підхід до тестування програмного забезпечення, при якому тестувальник має доступ до внутрішніх деталей системи, включаючи код, структуру даних та алгоритми. Основна ідея полягає в тому, щоб перевірити логіку, архітектуру та реалізацію програми для забезпечення його коректності та відповідності специфікаціям.

У методиці «Біла скринька» тестувальники аналізують внутрішню структуру програми, використовуючи ці знання для розробки тестових

сценаріїв. Вони виконують різні типи тестів, такі як юніт-тести, інтеграційні тести, системні тести та тести прийняття.

Таблиця 4.1 – Переваги і недоліки методики тестування «Чорна скринька»

Переваги	Недоліки
Орієнтована на користувацький досвід та функціональність	Може пропустити внутрішні помилки та деталі реалізації
Швидше розробка тестів	Обмежене покриття внутрішніх аспектів програми
Незалежність від внутрішньої структури програми	Менше можливостей для виявлення складних проблем
Може бути ефективною для зовнішнього оцінювання функціональності	Обмежена можливість точного виявлення внутрішніх проблем
Простота в створенні тестів	Менше можливостей для виявлення внутрішніх помилок
Фокус на зовнішніх аспектах програми	Важкість у виявленні і відлагодженні внутрішніх проблем
Ефективна для функціонального тестування	Може бути менш ефективною для виявлення внутрішніх проблем логіки програми

Юніт-тести перевіряють окремі частини програми, зазвичай функції чи методи. Це дозволяє перевірити, чи працюють вони правильно відповідно до специфікацій. Інтеграційні тести перевіряють взаємодію між різними частинами програми, а системні тести – роботу системи як цілісної одиниці.

Під час тестування "Біла скринька" тестувальники створюють власні тестові дані для перевірки різних сценаріїв та відображення різних шляхів виконання програми. Вони також використовують методи аналізу покриття коду для визначення, які частини програми були протестовані, а які ні.

Ця методика тестування дозволяє виявити внутрішні помилки програми, а також забезпечує можливість тестування різноманітних сценаріїв взаємодії та поведінки програми. Однак вона вимагає глибокого технічного розуміння програмування та системи, що тестується, та може бути часо- та ресурсомісткою.

У таблиці 4.2 наведено переваги та недоліки методу тестування «Біла скринька».

Таблиця 4.2 – Переваги і недоліки методу тестування «Біла скринька»

Переваги	Недоліки
Можливість детального виявлення внутрішніх помилок	Вимагає глибокого розуміння внутрішньої структури програми
Ефективне виявлення складних внутрішніх проблем	Вимагає більшого обсягу ресурсів та часу для розробки тестів
Здатність до точного покриття внутрішніх аспектів	Залежність від конкретної реалізації програми
Глибше розуміння логіки та структури програми	Вимагає постійного оновлення тестів при змінах у коді
Можливість ефективної перевірки внутрішніх алгоритмів	Може бути менш ефективним для зовнішнього оцінювання функціональності
Ширший охоплення аспектів безпеки та стабільності	Більша складність у виявленні проблем, пов'язаних з зовнішніми факторами
Можливість виявлення масштабних проблем реалізації	Обмежена можливість тестування всіх можливих сценаріїв використання

Враховуючи переваги та недоліки обох методик тестування, а також направленість програмної компоненти, було обрано методику "Чорна скринька" для проведення тестування.

4.2 Тестування програмної реалізації модифікованого методу подвійної буферизації

Для проведення тестування плавності роботи гри в клас main було додано метод get_fps() об'єкта Clock, щоб отримати поточне значення FPS, він оновлює заголовок вікна з відображенням цього значення. Значення FPS до застосування програмної реалізації модифікованого методу «Подвійна буферизація» зображено на рисунку 4.1.

Рисунок 4.1 – Кількість FPS до застосування програмної реалізації модифікованого методу.

Значення FPS не стабільне, воно варіюється від 15 до 27. При переміщенні персонажа виникають розриви та підторморужування. При руху ворогів до гравця з’являється мерехтіння.

Застосування програмної реалзіації алгоритму аналізу заповненості буферу і алгоритму проміжної буферизації. Полягає в створенні двох буферів за допомогою pygame.Surface в класі level. Далі використати метод pygame.PixelArray() в тому ж класі Levsel для аналізу змінених пікселів в буферах, з подальшим виділенням їх в області змін, які будуть оновленні. Скопіювати ці області змін в передній буфер, а потім скопіювати його на екран за допомогою функції pygame.display.blit(). Нарешті, потрібно

викликати функцію pygame.display.flip(). Для пришвидшення аналізу пікселів використовується бібліотека Numba.

В результаті максимальне значення FPS виросло до 53. Також виросла стабільність значення FPS, змінюється лише в діапазоні 50-53 кадрів (див. рисунок 4.2). Стабільність FPS забезпечу плавність гри, зниження лагів та консистентність досвіду.

Рисунок 4.2 – Значення FPS після застосування програмної реалізації модифікованого методу

Першим необхідно протестувати запуск гри. При на запуску файла main.py запускається гра.

Очікуваний результат. Відкриття вікна гри, в якому буде відображатися персонаж гравця, ділянка карти зі усіма оточуючими об’єктами і почне відтворюватися звукова тема гри. Результат запуску програми відображений на рисунку 4.3.

Рисунок 4.3 – Початкова сцена гри після запуска

Після вдалого запуску програми, необхідно протестувати можливість переміщення гравця по карті. Тестування переміщення гравця за допомогою визначених клавіш «Left», «Right», «Up» та «Down»(клавіші з символом ←, →,↑ і ↓).

Очікуваний результат:

1. Персонаж з анімацією рухається в чотири сторони.
2. Персонаж не проходить через тверді об’єкти.
3. Персонаж повертається обличчя до напрямку свого руху.
4. Швидкість персонажа регулюється одноіменною характеристикою.
5. При натисканні клавіши «Left» персонаж рухається вліво.
6. При натисканні клавіши «Right» персонаж рухається праворуч.
7. При натисканні клавіши «Up» персонаж рухається вгору.
8. При натисканні клавіши «Down» персонаж рухається до низу.

Результат переміщення відображений на рисунку 4.4.

Рисунок 4.4 – Персонаж перемістився після натискання необхідних клавіш

Після вдалого тесту переміщення гравця. Необхідно протестувати переміщення ворогів до гравця, якщо гравець знаходиться в полі зору ворога.

У ворога кожного виду параметр «Швидкість» має своє значення:

– «Демонічний кальмар» 3;
– «Гіганський тхір» 2;
– «Привид» 4;
– «Дерево» 3.

Очікуваний результат:

7. Ворог має рухатися до гравця, якщо останній опинився в його полі зору.
8. Рух ворога має бути анімований.
9. Ворог не повинен проходити через тверді об'єкти.

Результат переміщення ворогів наведений на рисунку 4.5.

Рисунок 4.5 – Ворог перемістився до гравця

Для ліквідації ворогів гравець може використати зброю натиснувши клавішу «Space».

Очікуваний результат.:

1. Атака гравця має бути анімована, рух руки в якій зброя.
2. При потрапляні атакою по ворогу, останній має відступати трохи назад.
3. При потраплянні атаки по ворогу має 1 секунду блимати, що буде світчити нанесення йому шкоди.
4. Зброя гравця немає вбивати ворога за один удар.
5. Здоров’я ворога немає бути нескінченим.
6. Смерть ворога супроводжується анімацією смерті. Після якої зникає з карти.

Результат ліквідації ворога зброєю наведений на рисунку 4.6.

Рисунок 4.6 – Результат гравець атакує і знищує ворога

Тест пройдено вдало. Тест атаки ворога. Ворог при атаці має наносити пошкодження, що в свою чергу має зменшувати шкалу здоров’я персонажа. Кожний з ворогів має своє значення параметру «Пошкодження»:

– «Демонічний кальмар» 20;
– «Гіганський тхір» 40;
– «Привид» 8;
– «Дерево» 6.

Очікуваний результат:

1. Атака ворога має супроводжуватися анімацією.
2. При отриманні пошкодження гравець 1 секунду блимає, що означає що йому нанесено пошкодження.
3. Ворог немає вбивати гравця за одну атаку.
4. При отриманні пошкодження шкала здоров’я гравця має зменшуватися на значення рівне параметру «Пошкодження» ворога.

Результат переміщення ворогів наведений на рисунку 4.7.

Рисунок 4.7 – Результат ворог наносить атаку по гравцю шкала здоров’я зменшується

Для відновлення запасу здоров’я гравцю необхідно використати заклинання «Лікування» натискаючи на клавішу «Ctrl». Параметрами заклинання «Лікування» є «Сила» зі значенням 20 і «Вартість» зі значенням 10.

Очікуваний результат при використанні заклинання «Лікування»:

1. Шкала енергії має зменшуватися рівно до моменту повного заповнення, шкали здоров’я.
2. При використанні заклинання має відображатися унікальна анімація.
3. Гравець не може використати заклинання «Лікування», якщо його шкала здоров’я повна.

Результат відновлення здоров’я за допмогою заклинання «Лікування» наведений на рис. 4.8.

Рисунок 4.8 – Результат відновлення здоров’я, за допомогою заклинання «Лікування»

Для знищення ворога гравець може застосувати заклинання «Полум’я» за допомогою клавішу «Ctrl», воно наносить пошкодження і має анімацію застосування, за використання заклинання витрачається енергія, шкала енергії зменшується на певну частину. Параметрами заклинання «Полум’я» є «Сила» зі значенням 5 і «Вартість» зі значенням 20.

Очікуваний результат при застосуванні заклинання «Полум’я»:

1. При атаці віддображається анімація вогню направленого лінією, в бік куди дивиться персонаж.
2. При контакті з ворогом, наноситься пошкодження ворогу якщо він знаходитья на лінії дії заклинання.
3. За використання заклинання має зменшуватись шкала енергії персонажа.
4. Ворог при контакті з заклинанням має 1 секунду блимати, що має свідчити про нанесення йому шкоди.

Результат атаки заклинанням «Полум’я» наведений на рисунку 4.9.

Рисунок 4.9 – Результат застосування «Полум’я» для знищення ворога

Гравець за ліквідацію ворога отримує бали досвіду, які може використати для підвищення характеристик свого персонажа. За ліквідацію різних ворогів гравець отримує різну кількість балів досвіду, а саме:

– «Дерево» 120 балів досвіду;

– «Демонічний кальмар» 100 балів досвіду;

– «Привид» 110 балів досвіду;

– «Гіганський тхір» 250 балів досвіду.

Очікуваний результат:

1. За ліквідацію ворога «Дерево» гравцю нарахується 120 балів досвіду.

2. Лічильник балів досвіду має збільшитися на 120 одиниць.

3. Лічильник балів досвіду немає бути обмежений.

4. Зарахування балів досвіду відбувається під час запуску анімації смерті ворога.

Результат відновлення здоров’я наведений на рисунку 4.10.

Рисунок 4.10 – Результат нарахування балів досвіду за ліквідацію ворога «Дерево»

Під підвищення характеристик персонажа виділено спеціальне меню яке викликається клавішою «M» (англійського алфавіту).

Очікуваний результат:

1. При натисканні клавіши «M» коректно відображається меню підвищення характеристик.

2. Меню підвищення характеристик має складатися з 5 блоків, кожний зі свої заголовком, який відповідає одній із характеристик «HEALTH», «ENERGY», «ATTACK», «MAGIC» і «SPEED».

3. Значення всіх характеристик за замовчуванням має бути 100. В блоках характеристик мають відображатися повзунки, які допомагають гравцю відслідковувати підвищення харктеристик.

Результат відкриття меню підвищення характеристик наведений на рисунку 4.11.

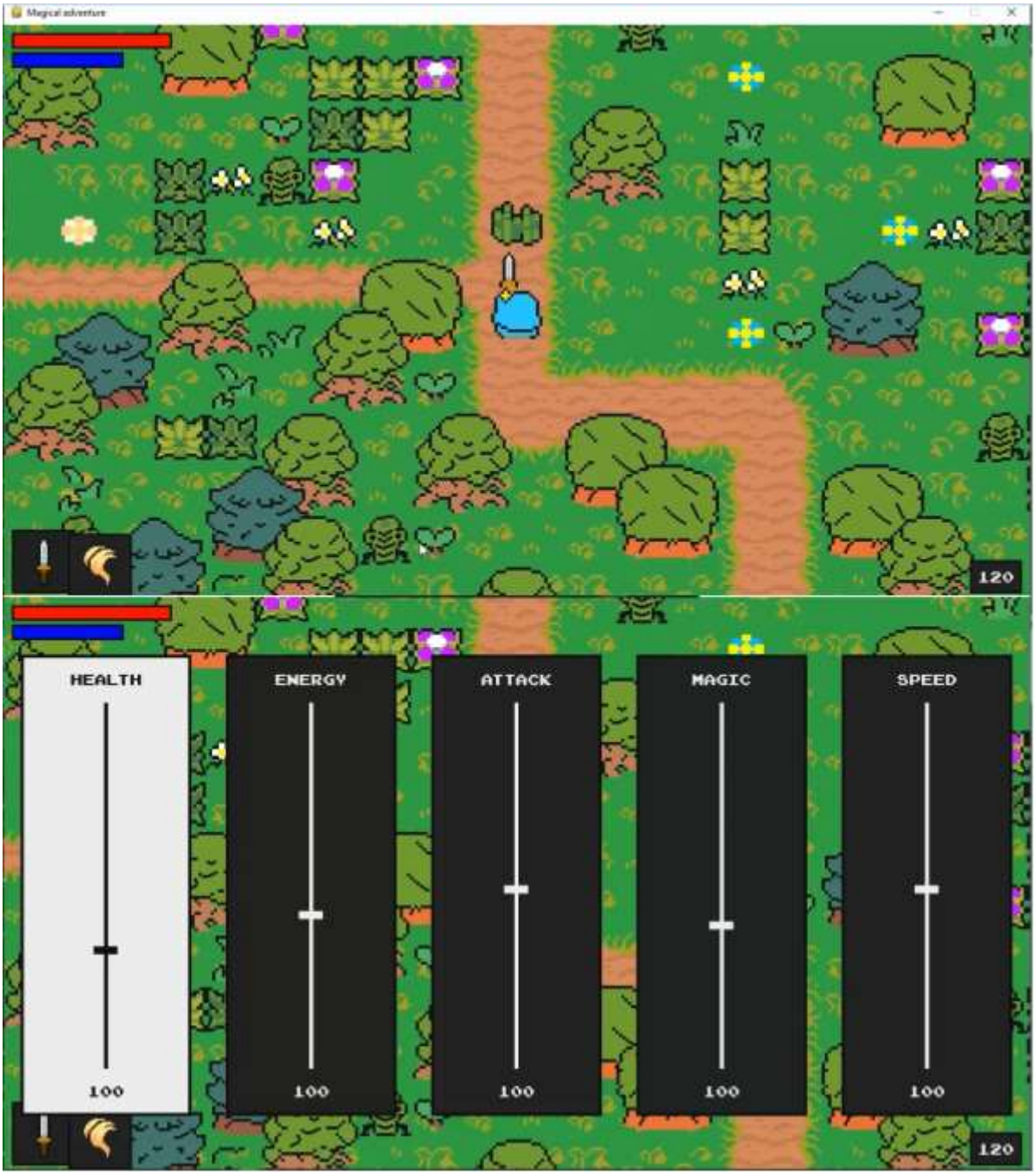

Рисунок 4.11 – Результат отримання балів досвіду за ліквідацію ворога «Дерево»

Тест пройдено успішно. За допомогою клавіш «Left» (←) і «Right» (→), обрати необхідну характеристику і збільшити її натискаючи клавішу «Space» за рахунок 240 балів досвіду (отриманих за ліквідацію двох ворогів «Дерево»), і за бажанням збільшити іншу.

Очікуваний результат. Підвищення характеристики і відображення зміни значення ідентифікатора під повзунком обраної характеристики.

Значення лічильника балів досвіду має бути 40 (100 балів після двох підвищень характеристик. Результат тесту збільшення характеристик в меню підвищення характеристик наведений на рисунку 4.12.

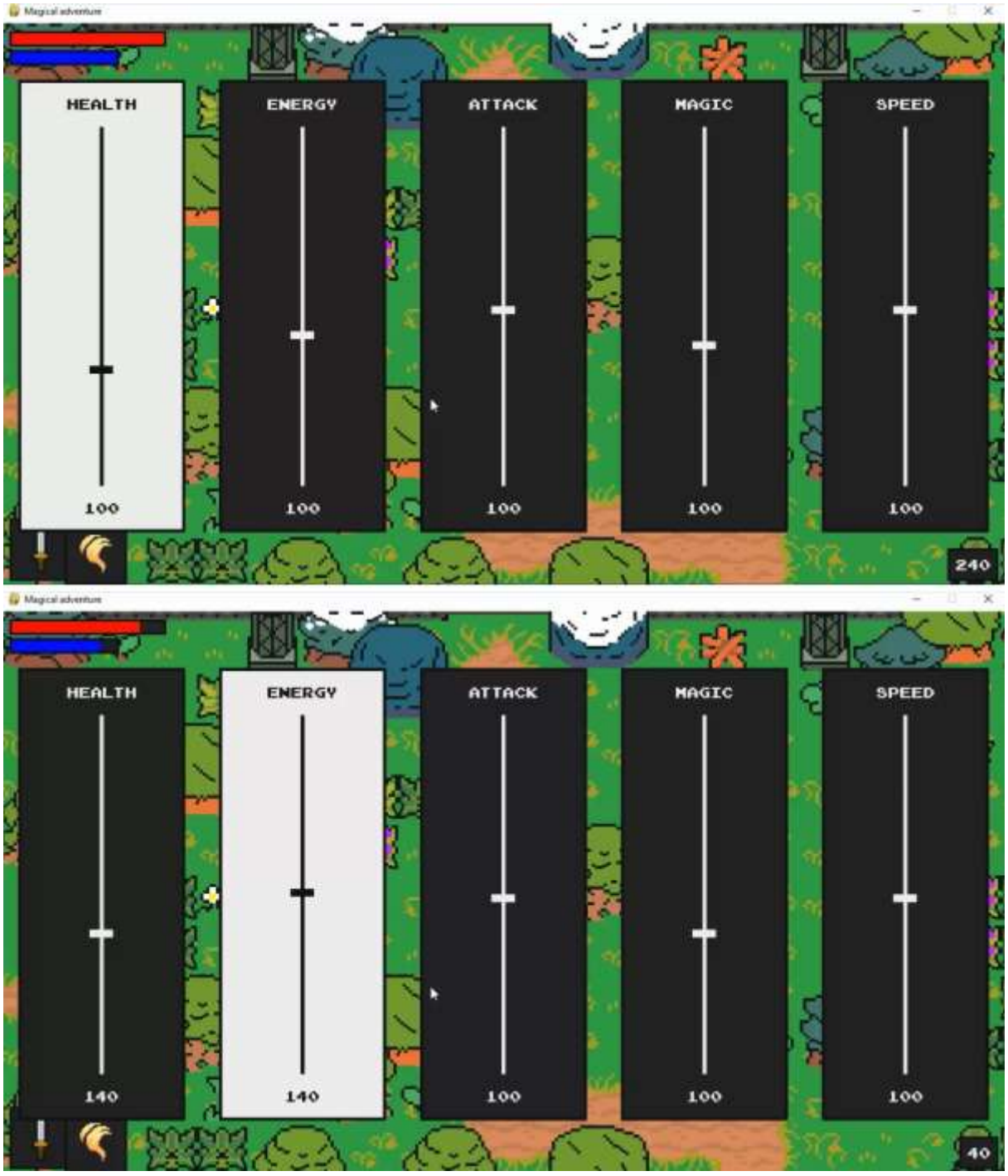

Рисунок 4.12 – Результат збільшення характеристик

Крім ворога «Дерево» є ще декілька, найскладніший з них це «Гіганський тхір». Він має багато здоров’я і наносить багато пошкоджень

гравцю. У «Гіганський тхір» є анімація атаки і смерті.

Очікуваний результат. Ворог «Гіганський тхір» має слідкувати за гравцем якщо той знаходиться в його полі зору, його атака має значно зменшувати здоров'я персонажа. Атаки має супроводжуватися анімацією. Також у «Гіганський тхір» має бути анімація смерті. Результат тесту переміщення, анімованої атаки «Гіганський тхір» і анімації смерті наведений на рисунку 4.13.

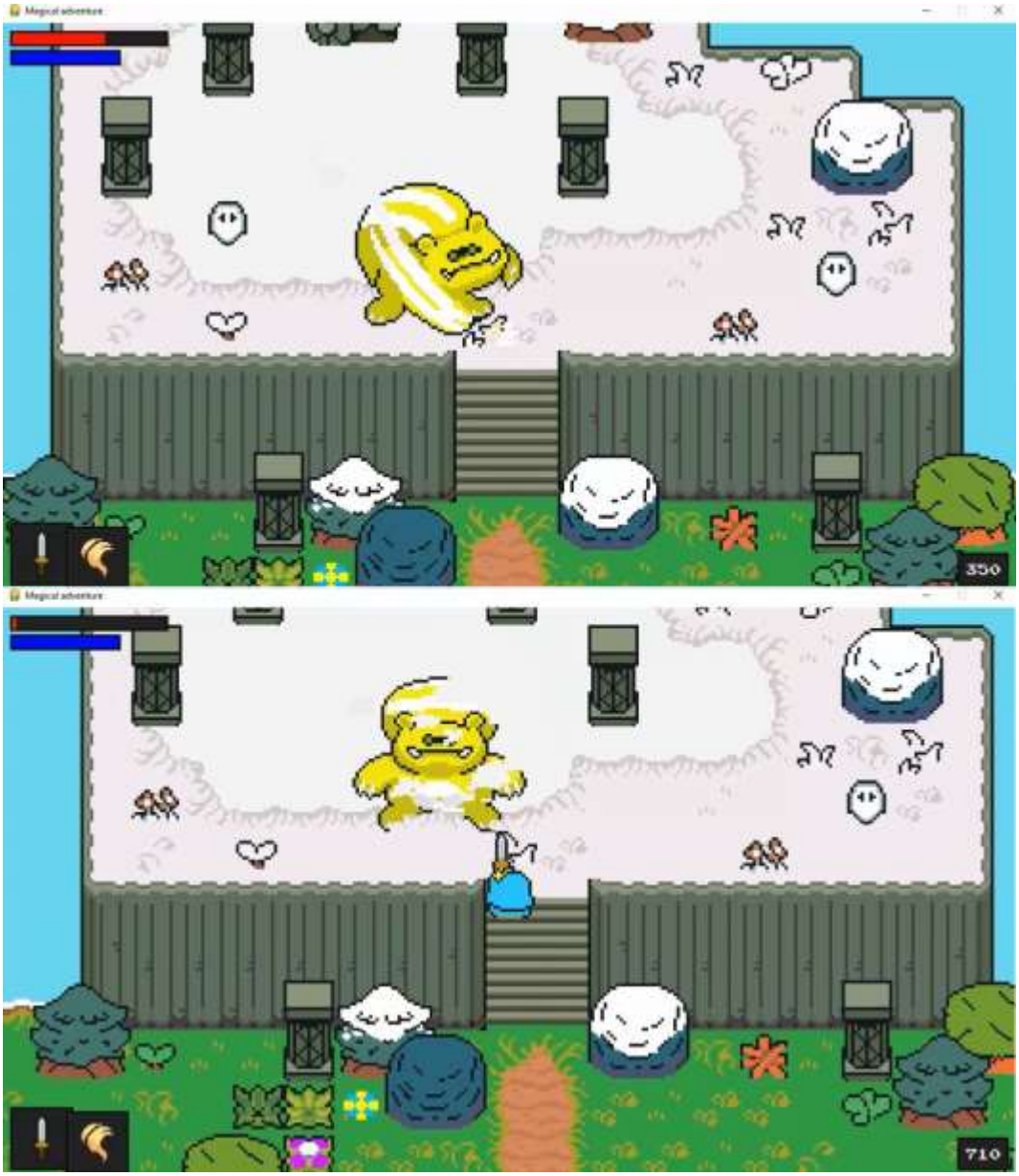

Рисунок 4.13 – Результат переміщення, атаки «Гіганський тхір» і анімації атаки та смерті

Ще одним ворогом є «Привид». Він має велику швидкість переміщення, що робить втечу від нього важкою справою. Його атака супроводжується анімацією блискавки. Смерть «Привида» теж має анімацію.

Очікуваний результат. «Привид» рухається до гравця, якщо той знаходиться в полі його зору. Його атака має зменшувати здоров'я гравця та бути анімованою. Смерть «Привида» теж має бути анімованою. Результат тесту ворога «Привид» наведений на рисунку 4.14.

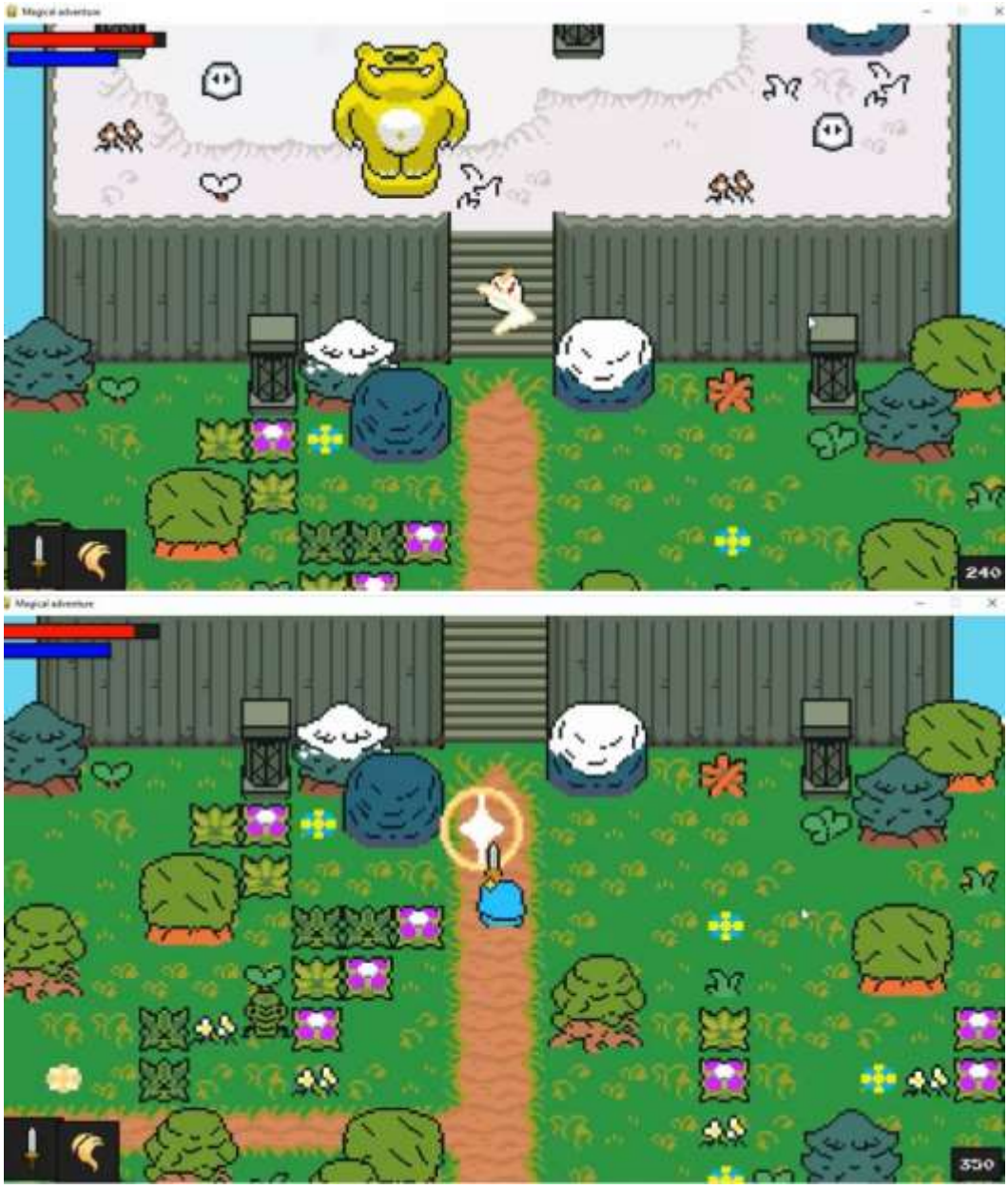

Рисунок 4.14 – Результат «Привид» наносить удар відбувається анімація атаки, зменшення здоров'я гравця і відображення анімації смерті

Тест пройдено успішно. Останнім ворогом гравця є «Демонічний кальмар», так як інші вороги може переміщуватись і атакувати персонажа гравця, має анімацію атаки та анімацію смерті. Він може нанести більше шкоди ніж «Дерево» і «Привид», але менше ніж «Гіганський тхір».

Характеристики ворога «Демонічний кальмар»:

1. Здоров’я – 120 .
2. Пошкодження – 20.
3. Бали досвіду – 100.
4. Швидкість – 3.
5. Стійкість – 3.
6. Радіус поля зору – 360.

Очікуваний результат:

1. Ворог «Демонічний кальмар» має рухатись в бік гравця якщо той в полі його зору.
2. Рухатися «Демонічний кальмар» зі своїм значення параметру «Швидкість».
3. Атака «Демонічний кальмар» має бути анімована.
4. Атака «Демонічний кальмар» повинена зменшувати здоров’я гравця на значення параметру «Пошкодження».
5. При отриманні атаки ворога «Демонічний кальмар», гравець має 1 секунду блимнути, що означає отримання пошкодження від атаки.
6. Ворог «Демонічний кальмар» не може вбити гравця однією атакою.
7. Смерть ворога «Демонічний кальмар» має бути анімована.
8. Після смерті «Демонічний кальмар» зникає з карти.

Результат тесту ворога «Демонічний кальмар» наведений на рисунку 4.15.

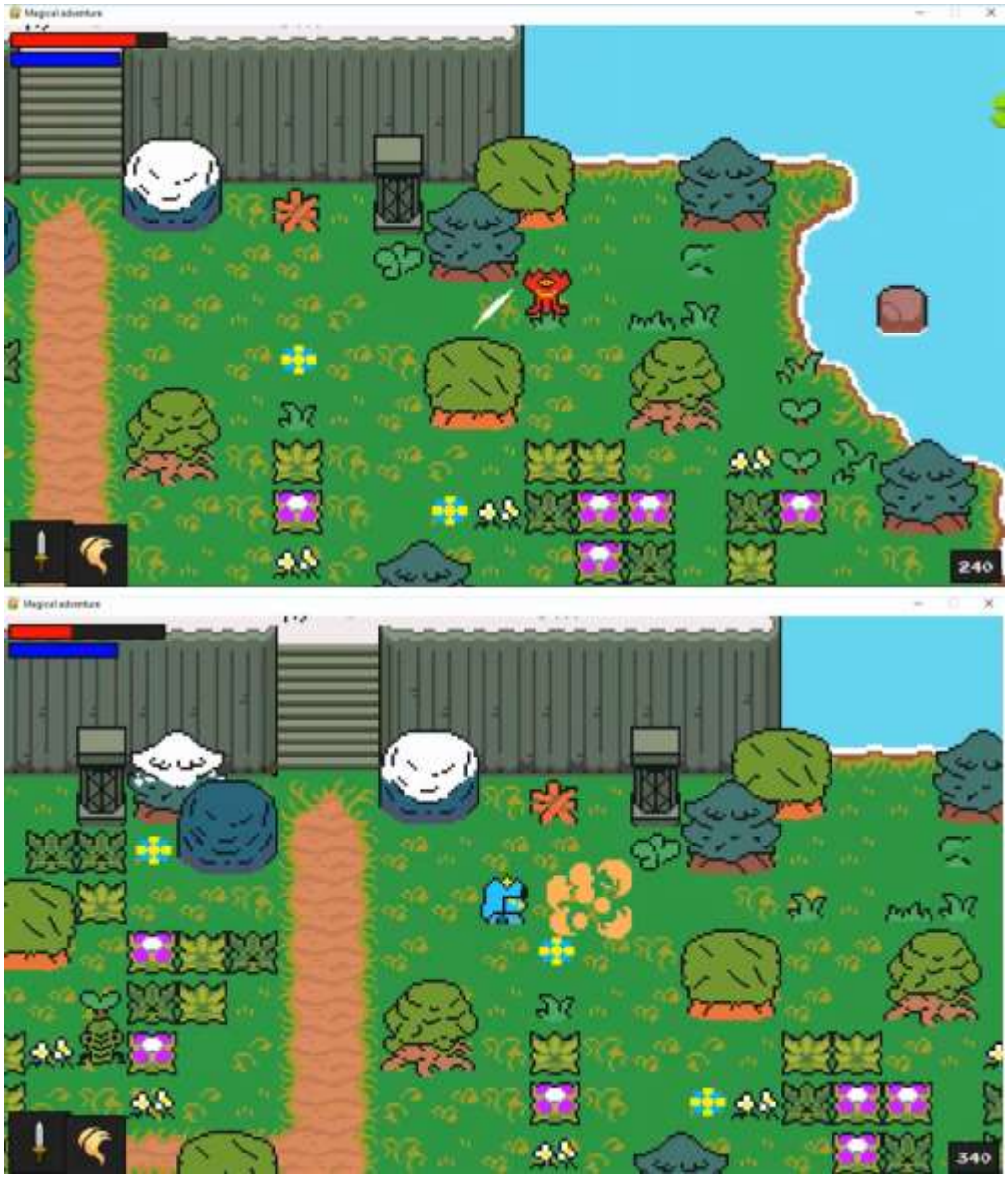

Рисунок 4.15 – Результат «Демонічний кальмар» наносить удар відбувається анімація атаки і зменшення здоров’я гравця. Відображення анімації його смерті

Гравець має декілька видів зброї і заклинань. У нього є можливість змінити зброю на іншу використовуючи клавішу «Q» і заклинання клавішу «E».

Очікуваний результат. Натискаючи клавішу «Q» відбувається зміна іконки зброї, іконка по контуру підсвічується жовтим кольором. Гравець може використовувати інше зброєю. Натискаючи клавішу «E» відбувається зміна іконки магії, іконка по контуру підсвічується жовтим кольором, може

використовувати інше закликання.

Результат зміни зброї і заклинання гравця наведено на рисунку 4.16.

Рисунок 4.16 – Результат зміни зброї та заклинання гравця

Збільшення характеристик персонажа впливає на його живучість, а саме здоров’я дозволяє гравцю витримувати більшу кількість атак ворогів.

Очікуваний результат. При збільшенні характеристики «HEALTH» на максимум, атака ворога «Гіганський тхір» буде відбирати меншу частину шкали здоров’я гравця.

Результат збільшення характеристики «HEALTH» наведено на рисунку 4.17.

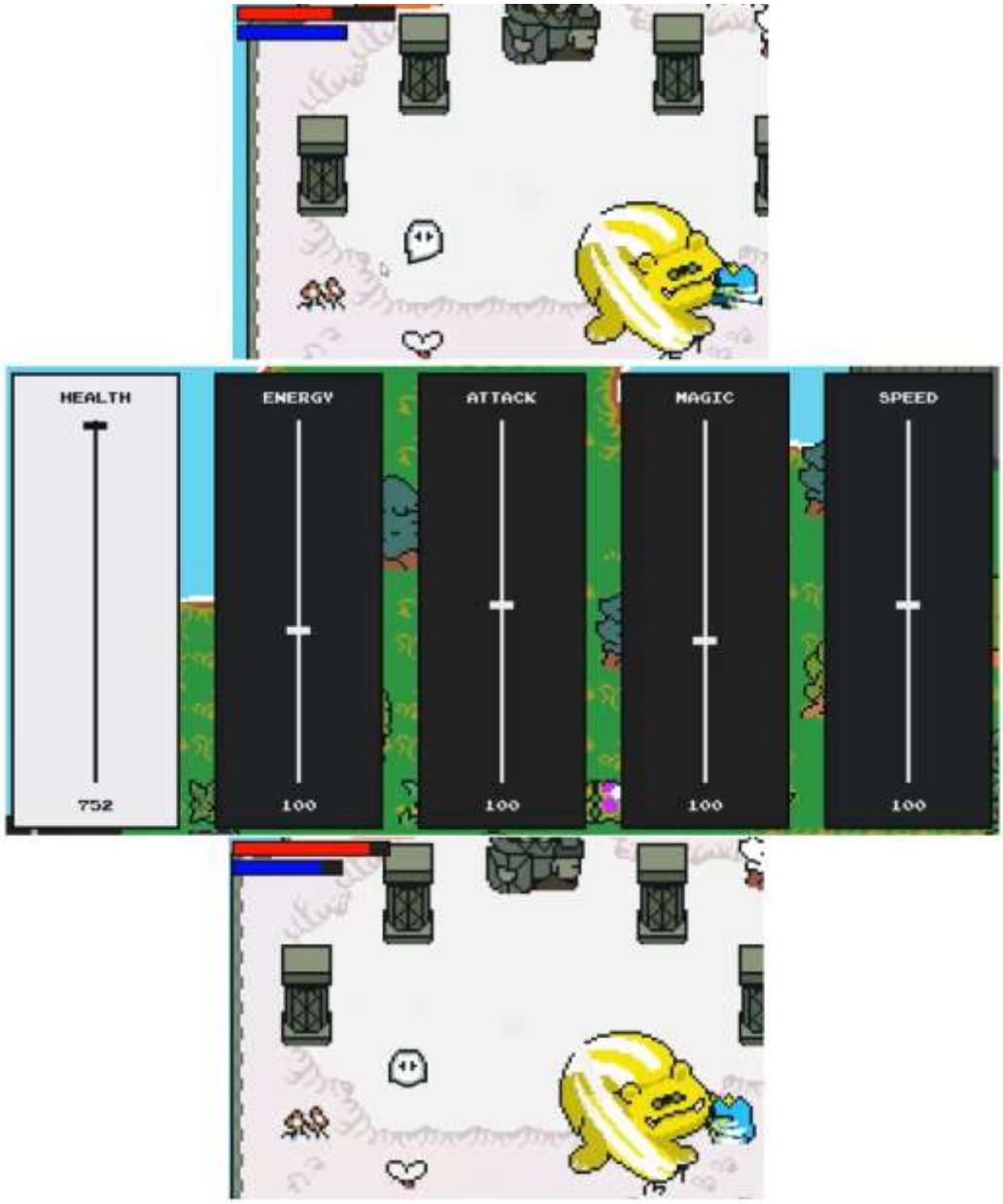

Рисунок 4.17 – Результат збільшення характеристики «HEALTH» гравця

Тест пройдено. Тестування демонструє коректну роботу гри.

4.3 Висновки

У четвертому розділі було проаналізовано методики тестування і проведено порівняння їх переваг, і недоліків в результаті було обрано метод тестування «Чорна скринька». Відповідно до обраної методики були проведені тести до розроблюваної комп'ютерної гри та підтверджено повну працездатність.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення наукового аудиту науково-дослідної роботи

Для наукових і пошукових науково-дослідних робіт зазвичай здійснюють оцінювання наукового ефекту. Основними ознаками наукового ефекту науково-дослідної роботи є новизна роботи, рівень її теоретичного опрацювання, перспективність, рівень розповсюдження результатів, можливість реалізації. Науковий ефект НДР можна охарактеризувати двома показниками: ступенем наукової новизни та рівнем теоретичного опрацювання. Значення показників ступеня новизни і рівня теоретичного опрацювання науково-дослідної роботи в балах наведено в табл. 5.1 та 5.2. [37]

Таблиця 5.1 – Показники ступеня новизни науково-дослідної роботи

Ступінь новизни	Характеристика ступеня новизни	Значення показника ступеня новизни, бали
Принципово нова	ота якісно нова за постановкою задачі і ґрунтується на застосуванні оригінальних методів дослідження. Результати дослідження відкривають новий напрям в цій галузі науки і техніки. Отримано принципово нові факти, закономірності; розроблено нову теорію. Створено принципово новийпристрій, спосіб, метод	60…100
Нова	Отримано нову інформацію, яка суттєво зменшує невизначеність наявних значень (по-новому або вперше пояснено відомі факти, закономірності, впроваджено нові поняття, розкрито структуру змісту). Проведено суттєве вдосконалення, доповнення і уточнення раніше досягнутих результатів	40…60

Продовження таблиці 5.1

Ступінь новизни	Характеристика ступеня новизни	Значення показника ступеня новизни, бали
Відносно нова	Робота має елементи новизни в постановці задачі і методах дослідження. Результати дослідження систематизують і узагальнюють наявну інформацію, визначають шляхи подальших досліджень; вперше знайдено зв'язок (або знайдено новий зв'язок) між явищами. В принципі, відомі положення поширено на велику кількість об'єктів, в результаті чого знайдено ефективне рішення.	10…40
Ступінь новизни	Характеристика ступеня новизни	Значення показника ступеня новизни, бали
Традиційна	Робота виконана за традиційною методикою. Результати дослідження мають інформаційний характер. Підтверджено або поставлено під сумнів відомі факти та твердження, які потребують перевірки. Знайдено новий варіант рішення, який не дає суттєвих переваг порівняно з існуючим	2…10
Не нова	Отримано результат, який раніше зафіксований в інформаційному полі та не був відомий авторам	1…2

Таблиця 5.2 – Показники рівня теоретичного опрацювання

Характеристика рівня теоретичного опрацювання	Значення показника рівня теоретичного опрацювання, бали
Відкриття закону, розробка теорії	80…100

Продовження таблиці 5.2

Характеристика рівня теоретичного опрацювання	Значення показника рівня теоретичного опрацювання, бали
Глибоке опрацювання проблеми: багатоаспектний аналіз зв'язків, взаємозалежності між фактами з наявністю пояснень, наукової систематизації з побудовою евристичної моделі або комплексного прогнозу	60…80
Розробка способу (алгоритму, програми), пристрою, отримання нової речовини	20…60
Елементарний аналіз зв'язків між фактами та наявною гіпотезою, класифікація, практичні рекомендації для окремого випадку тощо	6…20
Опис окремих елементарних фактів, викладення досвіду, результатів спостережень, вимірювань тощо	1…5

Показник, який характреризує науковий ефект, визначається за формулою:

$$\mathrm{E}_{\text{нау}} = 0.6 * k_{\text{нов}} + 0.4 * k_{\text{теор}} \quad (5.1)$$

$$k_{\text{нов}} = 9;\ k_{\text{теор}} = 25;$$

$$\mathrm{E}_{\text{нау}} = 0.6 * 25 + 0.4 * 9 = 18.6$$

Розробка має елементи новизни в постановці задачі. Результати дослідження систематизують і узагальнюють наявну інформацію, визначають шляхи подальших досліджень методів оптимізації графічних рушіїв 2D ігор.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

1. Витрати пов'язані на проведення науково-дослідних робіт можливо скласти в наступні статті:

– витрати на оплату праці;

– витрати на матеріальні засоби (канцтовари, обладнання);

– витрати на інструментальне та інфраструктурне програмне забезпечення;

Основна заробітна плата кожного із залучених осіб визначається за формулою (5.2):

$$З_0 = \sum_{i=1}^{K} \frac{M_{ni} * t_i}{T_p} \text{ (грн)} \quad (5.2)$$

де M – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$Т_р$– число робочих днів в місяці; приблизно $Т_р = 22$;

t – кількість робочих днів роботи розробника.

Для проведення дослідження методів оптимізації графічних рушіїв 2D ігор, який є достатнім необхідно залучити – Технічний розробник ігрового рушія та Інженер-програміст. Посадові оклади, число днів роботи та витрати на компенсацію додаємо в таблицю 5.3.

5.3 Оцінювання важливості та наукової значимості науково-дослідної роботи фундаментального чи пошукового характеру

Для обґрунтування доцільності виконання науково-дослідної роботи використовується спеціальний комплексний показник, що враховує важливість, результативність роботи, можливість впровадження її результатів у виробництво, величину витрат на роботу Комплексний показник рівня науково-дослідної роботи може бути розрахований за формулою:

$$К_р = \frac{I^n * T_C * R}{B * t}, \quad (5.15)$$

де I – коефіцієнт важливості роботи, I = 2...5;

n – коефіцієнт використання результатів роботи;

n = 0, коли результати роботи не будуть використовуватись;

n =1, коли результати роботи будуть використовуватись частково;

n = 2, коли результати роботи будуть використовуватись в дослідно-конструкторських розробках;

n = 3 , коли результати можуть використовуватись навіть без проведення

дослідно-конструкторських розробок.

T_c – коефіцієнт складності роботи, $T_c = 1 \dots 3$;

R – коефіцієнт результативності роботи; якщо результати роботи плануються вище відомих, то R = 4 ; якщо результати роботи відповідають відомому рівню, то R = 3; якщо нижче відомих результатів, то R =1;

B – вартість науково-дослідної роботи, тис. грн;

t – час проведення дослідження, років.

Визначення показників I, n, T_c, R, B, t здійснюється експертним шляхом або на основі нормативів.

Якщо $К_р > 1$, то науково-дослідну роботу можна вважати ефективною з високим науковим, технічним і економічним рівнями.

$$К_р = \frac{4^2 * 3 * 4}{617 * 0.25} = \frac{192}{154.25} = 1.2$$

Науково-дослідна робота є достатньо коштовною з економічної точки зору.

5.4 Висновки

Було визначено рівень наукового ефекту науково-дослідної роботи дослідження методів оптимізації графічних рушіїв 2D ігор, який є достатнім.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 308 733 грн. Прогнозування загальних втрат ЗВ на виконання науково-дослідної роботи та оформлення її результатів складе 3 087 330 грн.

Оцінено важливість та наукову значимість науково-дослідної роботи за допомогою спеціального комплексного показника $К_р$, який дорівнює 1,2.

ВИСНОВКИ

У магістерської кваліфікаційної роботі проведено дослідження методів оптимізації графічних рушіїв 2D ігор. Розроблено алгоритм аналізу заповненості даних з використанням проміжної буферизації з паралельними обчисленнями для модифікації методу подвійної буферизації. Такий модифікований метод подвійної буферизації рекомендовано для використання при розробці 2D ігор, в графічних рушіях яких необхідно вирішити проблему повільного оновлення графічних даних на екрані, через яке з'являється мерехтіння і розриви зображення. В основі розробки лежить аналіз заповненості буфера за допомогою паралельних обчислень.

В першому розділі було проведено аналіз питання оптимізації графічних рушіїв 2D ігор, результатом якого стали визначені основні види оптимізації програмного забезпечення та окреслено призначення графічного рушія. Проведено порівняльний аналіз таких методів оптимізації «Подвійна буферизація», «Керування оновленням екрану» і «Спрайти та групи». В результаті порівняльного аналізу було обрано метод «Подвійна буферизація» для подальшої модифікації. Також в розділі були визначено задачі роботи.

В другому розділі проведено аналіз і обґрунтування мов програмувань Java, C++, Python і визначено переваги цих мов і сфери застосування. Результатом якого стало обрання мови програмування Python для реалізації програмних компонент та систем візуалізації. Обґрунтовано вибір середовища розробки Visual Studio Code серед конкурентів PyCharm, Thonny та Spyder. Так як вона має ряд ключових переваг, а саме: підтримка бібліотеки Pygame; вбудований відлагоджувач; інтеграція з системами контролю версій. Розроблено моделі роботи ігрової системи і алгоритм роботи 2D «Magical adventure». Також розроблено програмна реалізація модифікованого методу «Подвійна буферизація» алгоритм аналізу заповненості буферу та алгоритм проміжної буферизації з паралельними обчисленнями. Ідеєю алгоритму аналізу заповненості буфера є оновлення лише тих частин зображення, які

були змінені, для швидкого визначення таких областей змін використовується алгоритм проміжної буферизації, який використовує бібліотеку Numba для проведення паралельних обчислень.

У роботі розв’язано такі задачі, як: аналіз існуючих методів і засобів для визначення напрямків підвищення продуктивності виведення графічних даних; модифікування методу подвійної буферизації виводу графічних даних на екран; модифікування методу проміжної буферизації даних графічного буферу; розробка програмних компонент та систем візуалізації на основі запропонованих методів; проведення експериментальних досліджень розроблених засобів виведення графічних даних.

В четвертому розділі було обґрунтовано вибір методу тестування «Чорна скринька» програмної реалізіції модифікованого методу «Подвійної буферизації» та роботи 2D гри «Magical adventure»

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Войтко В.В. Особливості розробки комп'ютерної 2D гри «MAGICAL ADVENTURE» жанру Action-Adventure / В.В. Войтко, А. В. Денисюк, О. В. Гаврилюк, Н. Є. Барчук, Г. М. Колісниченко // Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи -2022», Секція - Інформаційні технології та комп‘ютерна інженерія. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/viewFile/16197/13633

2. Г. М. Колісниченко Дослідження методів оптимізації графічних рушіїв 2-d ігор / Г. М. Колісниченко, О. М. Рейда // Міжнародна Науково-Практична Інтернет-Конференція «Електронні Інформаційні Ресурси: Створення, Використання, Доступ» [Електронний ресурс] – Режим доступу до ресурсу: https://drive.google.com/file/d/1oVmxS3W_sEQPjes9S9AzWDaJxDxi6I1X/view

3. Optimization: Overview and Examples in Technical Analysis [Електронний ресурс] – Режим доступу: https://www.investopedia.com/terms/o/optimization.asp.

4. Performance Optimization in Software Development [Електронний ресурс] – Режим доступу: https://medium.com/the-andela-way/performance-optimization-in-software-development-ae7952ab885e.

5. Graphics Engine [Електронний ресурс] – Режим доступу: https://support.touchgfx.com/docs/basic-concepts/graphics-engine

6. What is the difference between graphics engine and game engine? [Електронний ресурс] – Режим доступу: https://www.vintageisthenewold.com/game-pedia/what-is-the-difference-between-graphics-engine-and-game-engine.

7. 2D Graphics Programming for Games (Paperback) Підручник / John

Pile Jr: Видавець Taylor & Francis Inc, 2019 – 240с.

8. Are 2D Games Still Popular? [Електронний ресурс] – Режим доступу: https://retrostylegames.com/blog/are-2d-games-still-popular/.

9. 2D GAME ART STYLES: THE ULTIMATE GUIDE [Електронний ресурс] – Режим доступу: https://3d-ace.com/blog/2d-game-art-styles-the-ultimate-guide/.

10. Double Buffering [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/double-buffering/.

11. Double Buffer [Електронний ресурс] – Режим доступу: https://gameprogrammingpatterns.com/double-buffer.html

12. Eliminating Flashing: Implementing Double Buffering [Електронний ресурс] – Режим доступу: http://www.cs.fsu.edu/~jtbauer/cis3931/tutorial/ui/drawing/doubleBuffer.html

13. Double Buffering [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/double-buffering.

14. Improving Performance in Pygame – Speed Up Your Game [Електронний ресурс] – Режим доступу: https://www.codeproject.com/Articles/5298051/Improving-Performance-in-Pygame-Speed-Up-Your-Game.

15. What Is Refresh Rate and Why Is It Important? [Електронний ресурс] – Режим доступу: https://www.intel.com/content/www/us/en/gaming/resources/highest-refresh-rate-gaming.html.

16. What is a sprite? [Електронний ресурс] – Режим доступу: https://www.codeandweb.com/knowledgebase/what-is-a-sprite.

17. What are Sprites [Електронний ресурс] – Режим доступу: https://docs.cocos.com/cocos2d-x/manual/en/sprites/creating.html.

18. 2D SPRITES IN GAME DESIGN: WHY GAME ARTISTS STILL USE THEM [Електронний ресурс] – Режим доступу: https://retrostylegames.com/blog/2d-sprites-in-game-design/.

19. Gradient Descent, Stochastic Optimization, and Other Tales: Підручник / Jun Lu: Видавець Eliva Press, 2022 – 94с.

20. An Introduction to Optimization on Smooth Manifolds New Edition Підручник / Nicolas Boumal: Видавець Cambridge University Press, 2023 – 400с.

21. Algorithms for Convex Optimization 1st Edition Підручник / Nisheeth K. Vishnoi: Видавець Cambridge University Press, 2021 – 200с

22. Evolutionary Optimization Algorithms 1st Edition: Підручник / Altaf Q. H. Badar: Видавець CRC Press, 2021 – 274с.

23. Math for Programmers 3D graphics, machine learning, and simulations with Python: Підручник / Paul Orland: Видавець Manning Publications, 2022 – 688с.

24. Mathematical Optimization and Evolutionary Algorithms with Applications. Підручник / Antonin Ponsich: Видавець Mdpi AG, 2023 – 384с.

25. Analysis of a Memory Architecture for Fast Packet Buffers [Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/2945127_Analysis_of_a_Memory_Architecture_for_Fast_Packet_Buffers.

26. Parallel Computing [Електронний ресурс] – Режим доступу: https://www.heavy.ai/technical-glossary/parallel-computing.

27. Distinction Amidst MMX and XMM Registers? [Електронний ресурс] – Режим доступу: https://www.deepl.com/uk/translator#en/uk/Distinction%20Amidst%20MMX%20and%20XMM%20Registers%3F.

28. SSE Instructions [Електронний ресурс] – Режим доступу: https://docs.oracle.com/cd/E26502_01/html/E28388/eojde.html

29. What Can You Do with Java and What is it Used for? [Електронний ресурс] – Режим доступу: https://exoft.net/what-can-you-do-with-java/

30. C++ programming language [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/c-plus-plus/.

31. Learn Python Programming [Електронний ресурс] – Режим доступу:

https://www.programiz.com/python-programming.

32. Python Pygame (Game Development Library) [Електронний ресурс] – Режим доступу: https://www.javatpoint.com/pygame.

33. What is PyCharm? [Електронний ресурс] – Режим доступу: https://intellipaat.com/blog/what-is-pycharm/.

34. What is Thonny? [Електронний ресурс] – Режим доступу: https://www.educative.io/answers/what-is-thonny.

35. What is Python Spyder IDE and How to use it? [Електронний ресурс] – Режим доступу: https://medium.com/edureka/spyder-ide-2a91caac4e46.

36. What is Visual Studio Code? Practical Guide(2023) [Електронний ресурс] – Режим доступу: https://www.educba.com/what-is-visual-studio-code/.

37. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

ДОДАТОК А
(обов'язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
«20» вересня 2023 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Методи оптимізації графічних рушіїв 2D ігор» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
к.т.н., доцент кафедри ПЗ, Рейда О.М
«20» вересня 2023 р.
Виконав:
_____ студент гр. 2ПІ-22м Г. М. Колісниченко
«20» вересня 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Методи оптимізації графічних рушіїв 2D ігор».

Галузь застосування – оптимізація процесу роботи пристроїв, менше навантаження на технічні пристрої.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 247 від «18» вересня 2023 р. ректора ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є дослідження методів оптимізації графічних рушіїв 2D ігор для підвищення якості відображення графічних зображень в процесі динаміного онлвлення.

Основними задачами дослідження є:

– провести аналіз існуючих методів і засобів для визначення напрямків підвищення продуктивності виведення графічних даних;

– модифікувати метод подвійної буферизації виводу графічних даних на екран;

– модифікувати метод проміжної буферизації даних графічного буферу;

– розробити програмні компоненти та систему візуалізації на основі запропонованих методів;

– провести експериментальні дослідження розроблених засобів текстурування.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР:

1. Optimization: Overview and Examples in Technical Analysis [Електронний ресурс] – Режим доступу: https://www.investopedia.com/terms/o/optimization.asp.

2. Game Development Using Python Підручник / James R. Parker: Видавець Mercury Learning & Information, 2019. – 500с.

3. Evolutionary Optimization Algorithms 1st Edition: Підручник / Altaf Q. H. Badar: Видавець CRC Press, 2021 – 274с.

4. Developing Graphics Frameworks with Python and OpenGL: Підручник / Lee Stemkoski, Michael Pascale: Видавець: CRC Press, 2021 – 157с.

5. Python Pygame (Game Development Library) [Електронний ресурс] – Режим доступу: https://www.javatpoint.com/pygame.

6. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

5. Технічні вимоги

– середовище ігрового рушія: Pygame;

– середовище розробки – Microsoft Visual Studio Code;

– мова програмування – Python;

– метод подвійної буферизації – використання аналізу заповненості буферу за допомогою проміжної буферизації з паралельними обчисленнями;

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до МКР.

2. Технічне завдання.

3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз сучасного стану питання та обґрунтування задачі	20.09.23 – 01.10.23
2	Порівняльний аналіз методів оптимізації	02.10.23 – 14.10.23
3	Розробка моделі роботи ігрової системи	15.10.23 – 22.10.23
4	Розробка програмної компоненти	23.10.23 – 14.11.23
5	Тестування програми	15.11.23 – 23.11.23
6	Економічна частина	24.11.23 –26.11.23
7	Оформлення матеріалів до захисту МКР	27.11.23 – 29.12.23

10. Порядок контролю та прийняття

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

ДОДАТОК Б

(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Методи оптимізації графічних рушіїв 2D ігор.

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Рейда О. М.

Оригінальність	91.6%
Схожість	8.4%

Аналіз звіту подібності

■ **Запозичення, виявлені 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 автора. Роботу направити на доопрацювання.

□ 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 _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Колісниченко Г. М.

Керівник роботи _____ Рейда О. М.

ДОДАТОК В

(довідковий)

Лістинг коду програмної реалізації модифікованого методу подвійної буферизації

debug.py

```
import pygame
pygame.init()
font = pygame.font.Font(None,30)

def debug(info,y = 10, x = 10):
        display_surface = pygame.display.get_surface()
        debug_surf = font.render(str(info),True,'White')
        debug_rect = debug_surf.get_rect(topleft = (x,y))
        pygame.draw.rect(display_surface,'Black',debug_rect)
        display_surface.blit(debug_surf,debug_rect)
```

enemy.py

```
import pygame
from settings import *
from entity import Entity
from support import *

class Enemy(Entity):
        def
__init__(self,monster_name,pos,groups,obstacle_sprites,damage_player,trigger_death_particles,
add_exp):
                # загальне налаштування
                super().__init__(groups)
                self.sprite_type = 'enemy'

                # налаштування графіки
                self.import_graphics(monster_name)
                self.status = 'idle'
                self.image = self.animations[self.status][self.frame_index]

                #рух
                self.rect = self.image.get_rect(topleft = pos)
                self.hitbox = self.rect.inflate(0,-10)
                self.obstacle_sprites = obstacle_sprites

                # характеристики
                self.monster_name = monster_name
                monster_info = monster_data[self.monster_name]
                self.health = monster_info['health']
                self.exp = monster_info['exp']
                self.speed = monster_info['speed']
                self.attack_damage = monster_info['damage']
                self.resistance = monster_info['resistance']
```

```
        self.attack_radius = monster_info['attack_radius']
        self.notice_radius = monster_info['notice_radius']
        self.attack_type = monster_info['attack_type']

        # взаємодія гравця
        self.can_attack = True
        self.attack_time = None
        self.attack_cooldown = 400
        self.damage_player = damage_player
        self.trigger_death_particles = trigger_death_particles
        self.add_exp = add_exp

        # таймер непереможності
        self.vulnerable = True
        self.hit_time = None
        self.invincibility_duration = 300

        # звуки
        self.death_sound = pygame.mixer.Sound('../audio/death.wav')
        self.hit_sound = pygame.mixer.Sound('../audio/hit.wav')
        self.attack_sound = pygame.mixer.Sound(monster_info['attack_sound'])
        self.death_sound.set_volume(0.6)
        self.hit_sound.set_volume(0.6)
        self.attack_sound.set_volume(0.6)

    def import_graphics(self,name):
        self.animations = {'idle':[],'move':[],'attack':[]}
        main_path = f'../graphics/monsters/{name}/'
        for animation in self.animations.keys():
            self.animations[animation] = import_folder(main_path + animation)

    def get_player_distance_direction(self,player):
        enemy_vec = pygame.math.Vector2(self.rect.center)
        player_vec = pygame.math.Vector2(player.rect.center)
        distance = (player_vec - enemy_vec).magnitude()

        if distance > 0:
            direction = (player_vec - enemy_vec).normalize()
        else:
            direction = pygame.math.Vector2()

        return (distance,direction)

    def get_status(self, player):
        distance = self.get_player_distance_direction(player)[0]

        if distance <= self.attack_radius and self.can_attack:
            if self.status != 'attack':
                self.frame_index = 0
            self.status = 'attack'
        elif distance <= self.notice_radius:
            self.status = 'move'
```

```
            else:
                self.status = 'idle'

    def actions(self,player):
        if self.status == 'attack':
            self.attack_time = pygame.time.get_ticks()
            self.damage_player(self.attack_damage,self.attack_type)
            self.attack_sound.play()
        elif self.status == 'move':
            self.direction = self.get_player_distance_direction(player)[1]
        else:
            self.direction = pygame.math.Vector2()

    def animate(self):
        animation = self.animations[self.status]

        self.frame_index += self.animation_speed
        if self.frame_index >= len(animation):
            if self.status == 'attack':
                self.can_attack = False
            self.frame_index = 0

        self.image = animation[int(self.frame_index)]
        self.rect = self.image.get_rect(center = self.hitbox.center)

        if not self.vulnerable:
            alpha = self.wave_value()
            self.image.set_alpha(alpha)
        else:
            self.image.set_alpha(255)

    def cooldowns(self):
        current_time = pygame.time.get_ticks()
        if not self.can_attack:
            if current_time - self.attack_time >= self.attack_cooldown:
                self.can_attack = True

        if not self.vulnerable:
            if current_time - self.hit_time >= self.invincibility_duration:
                self.vulnerable = True

    def get_damage(self,player,attack_type):
        if self.vulnerable:
            self.hit_sound.play()
            self.direction = self.get_player_distance_direction(player)[1]
            if attack_type == 'weapon':
                self.health -= player.get_full_weapon_damage()
            else:
                self.health -= player.get_full_magic_damage()
            self.hit_time = pygame.time.get_ticks()
            self.vulnerable = False
```

```
        def check_death(self):
                if self.health <= 0:
                        self.kill()
                        self.trigger_death_particles(self.rect.center,self.monster_name)
                        self.add_exp(self.exp)
                        self.death_sound.play()

        def hit_reaction(self):
                if not self.vulnerable:
                        self.direction *= -self.resistance

        def update(self):
                self.hit_reaction()
                self.move(self.speed)
                self.animate()
                self.cooldowns()
                self.check_death()

        def enemy_update(self,player):
                self.get_status(player)
                self.actions(player)
```

entity.py

```
import pygame
from math import sin

class Entity(pygame.sprite.Sprite):
        def __init__(self,groups):
                super().__init__(groups)
                self.frame_index = 0
                self.animation_speed = 0.15
                self.direction = pygame.math.Vector2()

        def move(self,speed):
                if self.direction.magnitude() != 0:
                        self.direction = self.direction.normalize()

                self.hitbox.x += self.direction.x * speed
                self.collision('horizontal')
                self.hitbox.y += self.direction.y * speed
                self.collision('vertical')
                self.rect.center = self.hitbox.center

        def collision(self,direction):
                if direction == 'horizontal':
                        for sprite in self.obstacle_sprites:
                                if sprite.hitbox.colliderect(self.hitbox):
                                        if self.direction.x > 0: # рухаючись праворуч
                                                self.hitbox.right = sprite.hitbox.left
                                        if self.direction.x < 0: # рухаючись ліворуч
                                                self.hitbox.left = sprite.hitbox.right
```

```
                if direction == 'vertical':
                        for sprite in self.obstacle_sprites:
                                if sprite.hitbox.colliderect(self.hitbox):
                                        if self.direction.y > 0: # рухаючись вниз
                                                self.hitbox.bottom = sprite.hitbox.top
                                        if self.direction.y < 0: # рухаючись вгору
                                                self.hitbox.top = sprite.hitbox.bottom

        def wave_value(self):
                value = sin(pygame.time.get_ticks())
                if value >= 0:
                        return 255
                else:
                        return 0
```

level.py

```
import pygame
from settings import *
from tile import Tile
from player import Player
from debug import debug
from support import *
from random import choice, randint
from weapon import Weapon
from ui import UI
from enemy import Enemy
from particles import AnimationPlayer
from magic import MagicPlayer
from upgrade import Upgrade
import numba

class Level:
        def __init__(self):

                # get the display surface
                self.display_surface = pygame.display.get_surface()
                self.game_paused = False

                # sprite group setup
                self.visible_sprites = YSortCameraGroup()
                self.obstacle_sprites = pygame.sprite.Group()

                # attack sprites
                self.current_attack = None
                self.attack_sprites = pygame.sprite.Group()
                self.attackable_sprites = pygame.sprite.Group()

                # sprite setup
                self.create_map()

                # user interface
                self.ui = UI()
```

```
        self.upgrade = Upgrade(self.player)

        # particles
        self.animation_player = AnimationPlayer()
        self.magic_player = MagicPlayer(self.animation_player)

    def create_map(self):
        layouts = {
            'boundary': import_csv_layout('../map/map_FloorBlocks.csv'),
            'grass': import_csv_layout('../map/map_Grass.csv'),
            'object': import_csv_layout('../map/map_Objects.csv'),
            'entities': import_csv_layout('../map/map_Entities.csv')
        }
        graphics = {
            'grass': import_folder('../graphics/Grass'),
            'objects': import_folder('../graphics/objects')
        }

        for style,layout in layouts.items():
            for row_index,row in enumerate(layout):
                for col_index, col in enumerate(row):
                    if col != '-1':
                        x = col_index * TILESIZE
                        y = row_index * TILESIZE
                        if style == 'boundary':
                            Tile((x,y),[self.obstacle_sprites],'invisible')
                        if style == 'grass':
                            random_grass_image =
choice(graphics['grass'])
                            Tile(
                                (x,y),

    [self.visible_sprites,self.obstacle_sprites,self.attackable_sprites],
                                'grass',
                                random_grass_image)

                        if style == 'object':
                            surf = graphics['objects'][int(col)]

    Tile((x,y),[self.visible_sprites,self.obstacle_sprites],'object',surf)

                        if style == 'entities':
                            if col == '394':
                                self.player = Player(
                                    (x,y),
                                    [self.visible_sprites],
                                    self.obstacle_sprites,
                                    self.create_attack,
                                    self.destroy_attack,
                                    self.create_magic)
                            else:
                                if col == '390': monster_name =
```

```
'bamboo'
                                                    elif col == '391': monster_name =
'spirit'
                                                    elif col == '392': monster_name
='raccoon'
                                                    else: monster_name = 'squid'
                                                    Enemy(
                                                            monster_name,
                                                            (x,y),

        [self.visible_sprites,self.attackable_sprites],
                                                            self.obstacle_sprites,
                                                            self.damage_player,
                                                            self.trigger_death_particles,
                                                            self.add_exp)

        def create_attack(self):

                self.current_attack = Weapon(self.player,[self.visible_sprites,self.attack_sprites])

        def create_magic(self,style,strength,cost):
                if style == 'heal':
                        self.magic_player.heal(self.player,strength,cost,[self.visible_sprites])

                if style == 'flame':

        self.magic_player.flame(self.player,cost,[self.visible_sprites,self.attack_sprites])

        def destroy_attack(self):
                if self.current_attack:
                        self.current_attack.kill()
                self.current_attack = None

        def player_attack_logic(self):
                if self.attack_sprites:
                        for attack_sprite in self.attack_sprites:
                                collision_sprites =
pygame.sprite.spritecollide(attack_sprite,self.attackable_sprites,False)
                                if collision_sprites:
                                        for target_sprite in collision_sprites:
                                                if target_sprite.sprite_type == 'grass':
                                                        pos = target_sprite.rect.center
                                                        offset = pygame.math.Vector2(0,75)
                                                        for leaf in range(randint(3,6)):

        self.animation_player.create_grass_particles(pos  - offset,[self.visible_sprites])
                                                        target_sprite.kill()
                                                else:

        target_sprite.get_damage(self.player,attack_sprite.sprite_type)

        def damage_player(self,amount,attack_type):
```

```
                if self.player.vulnerable:
                        self.player.health -= amount
                        self.player.vulnerable = False
                        self.player.hurt_time = pygame.time.get_ticks()

        self.animation_player.create_particles(attack_type,self.player.rect.center,[self.visible_spr
ites])

        def trigger_death_particles(self,pos,particle_type):

                self.animation_player.create_particles(particle_type,pos,self.visible_sprites)

        def add_exp(self,amount):

                self.player.exp += amount

        def toggle_menu(self):

                self.game_paused = not self.game_paused

        def run(self):
                self.visible_sprites.custom_draw(self.player)
                self.ui.display(self.player)

                if self.game_paused:
                        self.upgrade.display()
                else:
                        self.visible_sprites.update()
                        self.visible_sprites.enemy_update(self.player)
                        self.player_attack_logic()


class YSortCameraGroup(pygame.sprite.Group):
        def __init__(self):

                # general setup
                super().__init__()
                self.display_surface = pygame.display.get_surface()
                self.half_width = self.display_surface.get_size()[0] // 2
                self.half_height = self.display_surface.get_size()[1] // 2
                self.offset = pygame.math.Vector2()

                # creating the floor
                self.floor_surf = pygame.image.load('../graphics/tilemap/ground.png').convert()
                self.floor_rect = self.floor_surf.get_rect(topleft = (0,0))

        def custom_draw(self,player):

                # getting the offset
                self.offset.x = player.rect.centerx - self.half_width
                self.offset.y = player.rect.centery - self.half_height
```

```
                # drawing the floor
                floor_offset_pos = self.floor_rect.topleft - self.offset
                self.display_surface.blit(self.floor_surf,floor_offset_pos)

                # for sprite in self.sprites():
                for sprite in sorted(self.sprites(),key = lambda sprite: sprite.rect.centery):
                        offset_pos = sprite.rect.topleft - self.offset
                        self.display_surface.blit(sprite.image,offset_pos)

        def enemy_update(self,player):
                enemy_sprites = [sprite for sprite in self.sprites() if hasattr(sprite,'sprite_type') and
sprite.sprite_type == 'enemy']
                for enemy in enemy_sprites:
                        enemy.enemy_update(player)
```

magic.py

```
import pygame
from settings import *
from random import randint

class MagicPlayer:
        def __init__(self,animation_player):
                self.animation_player = animation_player
                self.sounds = {
                'heal': pygame.mixer.Sound('../audio/heal.wav'),
                'flame':pygame.mixer.Sound('../audio/Fire.wav')
                }

        def heal(self,player,strength,cost,groups):
                if player.energy >= cost:
                        self.sounds['heal'].play()
                        player.health += strength
                        player.energy -= cost
                        if player.health >= player.stats['health']:
                                player.health = player.stats['health']
                        self.animation_player.create_particles('aura',player.rect.center,groups)
                        self.animation_player.create_particles('heal',player.rect.center,groups)

        def flame(self,player,cost,groups):
                if player.energy >= cost:
                        player.energy -= cost
                        self.sounds['flame'].play()

                        if player.status.split('_')[0] == 'right': direction =
pygame.math.Vector2(1,0)
                        elif player.status.split('_')[0] == 'left': direction = pygame.math.Vector2(-
1,0)
                        elif player.status.split('_')[0] == 'up': direction = pygame.math.Vector2(0,-
1)
                        else: direction = pygame.math.Vector2(0,1)

                        for i in range(1,6):
```

```
                        if direction.x: #horizontal
                                offset_x = (direction.x * i) * TILESIZE
                                x = player.rect.centerx + offset_x + randint(-TILESIZE //
3, TILESIZE // 3)
                                y = player.rect.centery + randint(-TILESIZE // 3,
TILESIZE // 3)
                                self.animation_player.create_particles('flame',(x,y),groups)
                        else: # vertical
                                offset_y = (direction.y * i) * TILESIZE
                                x = player.rect.centerx + randint(-TILESIZE // 3,
TILESIZE // 3)
                                y = player.rect.centery + offset_y + randint(-TILESIZE //
3, TILESIZE // 3)
                                self.animation_player.create_particles('flame',(x,y),groups)
```

main.py

```
import pygame, sys
from settings import *
from level import Level

class Game:
        def __init__(self):

                # загальне налаштування
                pygame.init()
                self.screen = pygame.display.set_mode((WIDTH,HEIGTH))
                pygame.display.set_caption('Magical adventure')
                self.clock = pygame.time.Clock()

                self.level = Level()

                # звук
                main_sound = pygame.mixer.Sound('../audio/main.ogg')
                main_sound.set_volume(0.5)
                main_sound.play(loops = -1)

        def run(self):
                while True:
                        for event in pygame.event.get():
                                if event.type == pygame.QUIT:
                                        pygame.quit()
                                        sys.exit()
                                if event.type == pygame.KEYDOWN:
                                        if event.key == pygame.K_m:
                                                self.level.toggle_menu()

                        self.screen.fill(WATER_COLOR)
                        self.level.run()
                        pygame.display.update()
                        fps = self.clock.get_fps()
                        pygame.display.set_caption(f'Magical adventure - FPS: {int(fps)}')
                        self.clock.tick(FPS)
```

```
if __name__ == '__main__':
	game = Game()
	game.run()
```

particles.py

```
import pygame
from support import import_folder
from random import choice

class AnimationPlayer:
	def __init__(self):
		self.frames = {
			#магія
			'flame': import_folder('../graphics/particles/flame/frames'),
			'aura': import_folder('../graphics/particles/aura'),
			'heal': import_folder('../graphics/particles/heal/frames'),

			# атаки
			'claw': import_folder('../graphics/particles/claw'),
			'slash': import_folder('../graphics/particles/slash'),
			'sparkle': import_folder('../graphics/particles/sparkle'),
			'leaf_attack': import_folder('../graphics/particles/leaf_attack'),
			'thunder': import_folder('../graphics/particles/thunder'),

			# смерті монстра
			'squid': import_folder('../graphics/particles/smoke_orange'),
			'raccoon': import_folder('../graphics/particles/raccoon'),
			'spirit': import_folder('../graphics/particles/nova'),
			'bamboo': import_folder('../graphics/particles/bamboo'),

			# листочок
			'leaf': (
				import_folder('../graphics/particles/leaf1'),
				import_folder('../graphics/particles/leaf2'),
				import_folder('../graphics/particles/leaf3'),
				import_folder('../graphics/particles/leaf4'),
				import_folder('../graphics/particles/leaf5'),
				import_folder('../graphics/particles/leaf6'),
				self.reflect_images(import_folder('../graphics/particles/leaf1')),
				self.reflect_images(import_folder('../graphics/particles/leaf2')),
				self.reflect_images(import_folder('../graphics/particles/leaf3')),
				self.reflect_images(import_folder('../graphics/particles/leaf4')),
				self.reflect_images(import_folder('../graphics/particles/leaf5')),
				self.reflect_images(import_folder('../graphics/particles/leaf6'))
				)
			}

	def reflect_images(self,frames):
		new_frames = []

		for frame in frames:
```

```
                flipped_frame = pygame.transform.flip(frame,True,False)
                new_frames.append(flipped_frame)
        return new_frames

    def create_grass_particles(self,pos,groups):
        animation_frames = choice(self.frames['leaf'])
        ParticleEffect(pos,animation_frames,groups)

    def create_particles(self,animation_type,pos,groups):
        animation_frames = self.frames[animation_type]
        ParticleEffect(pos,animation_frames,groups)


class ParticleEffect(pygame.sprite.Sprite):
    def __init__(self,pos,animation_frames,groups):
        super().__init__(groups)
        self.sprite_type = 'magic'
        self.frame_index = 0
        self.animation_speed = 0.15
        self.frames = animation_frames
        self.image = self.frames[self.frame_index]
        self.rect = self.image.get_rect(center = pos)

    def animate(self):
        self.frame_index += self.animation_speed
        if self.frame_index >= len(self.frames):
            self.kill()
        else:
            self.image = self.frames[int(self.frame_index)]

    def update(self):
        self.animate()
```

player.py

```
import pygame
from settings import *
from support import import_folder
from entity import Entity

class Player(Entity):
    def __init__(self,pos,groups,obstacle_sprites,create_attack,destroy_attack,create_magic):
        super().__init__(groups)
        self.image = pygame.image.load('../graphics/test/player.png').convert_alpha()
        self.rect = self.image.get_rect(topleft = pos)
        self.hitbox = self.rect.inflate(-6,HITBOX_OFFSET['player'])

        # налаштування графіки
        self.import_player_assets()
        self.status = 'down'

        #рух
        self.attacking = False
```

```
                self.attack_cooldown = 400
                self.attack_time = None
                self.obstacle_sprites = obstacle_sprites

                # зброя
                self.create_attack = create_attack
                self.destroy_attack = destroy_attack
                self.weapon_index = 0
                self.weapon = list(weapon_data.keys())[self.weapon_index]
                self.can_switch_weapon = True
                self.weapon_switch_time = None
                self.switch_duration_cooldown = 200

                #магія
                self.create_magic = create_magic
                self.magic_index = 0
                self.magic = list(magic_data.keys())[self.magic_index]
                self.can_switch_magic = True
                self.magic_switch_time = None

                # характеристика
                self.stats = {'health': 100,'energy':60,'attack': 10,'magic': 4,'speed': 5}
                self.max_stats = {'health': 300, 'energy': 140, 'attack': 20, 'magic' : 10, 'speed': 10}
                self.upgrade_cost = {'health': 100, 'energy': 100, 'attack': 100, 'magic' : 100,
'speed': 100}
                self.health = self.stats['health']
                self.energy = self.stats['energy']
                self.exp = 0
                self.speed = self.stats['speed']

                # статистика
                self.vulnerable = True
                self.hurt_time = None
                self.invulnerability_duration = 500

                # імпортувати звук
                self.weapon_attack_sound = pygame.mixer.Sound('../audio/sword.wav')
                self.weapon_attack_sound.set_volume(0.4)

        def import_player_assets(self):
                character_path = '../graphics/player/'
                self.animations = {'up': [],'down': [],'left': [],'right': [],
                        'right_idle':[],'left_idle':[],'up_idle':[],'down_idle':[],
                        'right_attack':[],'left_attack':[],'up_attack':[],'down_attack':[]}

                for animation in self.animations.keys():
                        full_path = character_path + animation
                        self.animations[animation] = import_folder(full_path)

        def input(self):
                if not self.attacking:
                        keys = pygame.key.get_pressed()
```

```
                # введення руху
                if keys[pygame.K_UP]:
                        self.direction.y = -1
                        self.status = 'up'
                elif keys[pygame.K_DOWN]:
                        self.direction.y = 1
                        self.status = 'down'
                else:
                        self.direction.y = 0

                if keys[pygame.K_RIGHT]:
                        self.direction.x = 1
                        self.status = 'right'
                elif keys[pygame.K_LEFT]:
                        self.direction.x = -1
                        self.status = 'left'
                else:
                        self.direction.x = 0

                # введення атаки
                if keys[pygame.K_SPACE]:
                        self.attacking = True
                        self.attack_time = pygame.time.get_ticks()
                        self.create_attack()
                        self.weapon_attack_sound.play()

                # магічний вхід
                if keys[pygame.K_LCTRL]:
                        self.attacking = True
                        self.attack_time = pygame.time.get_ticks()
                        style = list(magic_data.keys())[self.magic_index]
                        strength = list(magic_data.values())[self.magic_index]['strength'] +
self.stats['magic']
                        cost = list(magic_data.values())[self.magic_index]['cost']
                        self.create_magic(style,strength,cost)

                if keys[pygame.K_q] and self.can_switch_weapon:
                        self.can_switch_weapon = False
                        self.weapon_switch_time = pygame.time.get_ticks()

                        if self.weapon_index < len(list(weapon_data.keys())) - 1:
                                self.weapon_index += 1
                        else:
                                self.weapon_index = 0

                        self.weapon = list(weapon_data.keys())[self.weapon_index]

                if keys[pygame.K_e] and self.can_switch_magic:
                        self.can_switch_magic = False
                        self.magic_switch_time = pygame.time.get_ticks()
```

```
                        if self.magic_index < len(list(magic_data.keys())) - 1:
                                self.magic_index += 1
                        else:
                                self.magic_index = 0

                        self.magic = list(magic_data.keys())[self.magic_index]

        def get_status(self):

                # неактивний стан
                if self.direction.x == 0 and self.direction.y == 0:
                        if not 'idle' in self.status and not 'attack' in self.status:
                                self.status = self.status + '_idle'

                if self.attacking:
                        self.direction.x = 0
                        self.direction.y = 0
                        if not 'attack' in self.status:
                                if 'idle' in self.status:
                                        self.status = self.status.replace('_idle','_attack')
                                else:
                                        self.status = self.status + '_attack'
                else:
                        if 'attack' in self.status:
                                self.status = self.status.replace('_attack','')

        def cooldowns(self):
                current_time = pygame.time.get_ticks()

                if self.attacking:
                        if current_time - self.attack_time >= self.attack_cooldown +
weapon_data[self.weapon]['cooldown']:
                                self.attacking = False
                                self.destroy_attack()

                if not self.can_switch_weapon:
                        if current_time - self.weapon_switch_time >=
self.switch_duration_cooldown:
                                self.can_switch_weapon = True

                if not self.can_switch_magic:
                        if current_time - self.magic_switch_time >=
self.switch_duration_cooldown:
                                self.can_switch_magic = True

                if not self.vulnerable:
                        if current_time - self.hurt_time >= self.invulnerability_duration:
                                self.vulnerable = True

        def animate(self):
                animation = self.animations[self.status]
```

```
                # петля над індексом кадру
                self.frame_index += self.animation_speed
                if self.frame_index >= len(animation):
                        self.frame_index = 0

                # встановити зображення
                self.image = animation[int(self.frame_index)]
                self.rect = self.image.get_rect(center = self.hitbox.center)

                # мерехтіння
                if not self.vulnerable:
                        alpha = self.wave_value()
                        self.image.set_alpha(alpha)
                else:
                        self.image.set_alpha(255)

        def get_full_weapon_damage(self):
                base_damage = self.stats['attack']
                weapon_damage = weapon_data[self.weapon]['damage']
                return base_damage + weapon_damage

        def get_full_magic_damage(self):
                base_damage = self.stats['magic']
                spell_damage = magic_data[self.magic]['strength']
                return base_damage + spell_damage

        def get_value_by_index(self,index):
                return list(self.stats.values())[index]

        def get_cost_by_index(self,index):
                return list(self.upgrade_cost.values())[index]

        def energy_recovery(self):
                if self.energy < self.stats['energy']:
                        self.energy += 0.01 * self.stats['magic']
                else:
                        self.energy = self.stats['energy']

        def update(self):
                self.input()
                self.cooldowns()
                self.get_status()
                self.animate()
                self.move(self.stats['speed'])
                self.energy_recovery()
```

settings.py

```
# налаштування гри
WIDTH    = 1280
HEIGTH   = 720
FPS      = 60
TILESIZE = 64
```

```
HITBOX_OFFSET = {
        'player': -26,
        'object': -40,
        'grass': -10,
        'invisible': 0}

#ui
BAR_HEIGHT = 20
HEALTH_BAR_WIDTH = 200
ENERGY_BAR_WIDTH = 140
ITEM_BOX_SIZE = 80
UI_FONT = '../graphics/font/joystix.ttf'
UI_FONT_SIZE = 18

# загальні кольори
WATER_COLOR = '#71ddee'
UI_BG_COLOR = '#222222'
UI_BORDER_COLOR = '#111111'
TEXT_COLOR = '#EEEEEE'

# кольори інтерфейсу користувача
HEALTH_COLOR = 'red'
ENERGY_COLOR = 'blue'
UI_BORDER_COLOR_ACTIVE = 'gold'

# меню оновлення
TEXT_COLOR_SELECTED = '#111111'
BAR_COLOR = '#EEEEEE'
BAR_COLOR_SELECTED = '#111111'
UPGRADE_BG_COLOR_SELECTED = '#EEEEEE'

# зброя
weapon_data = {
        'sword': {'cooldown': 100, 'damage': 15,'graphic':'../graphics/weapons/sword/full.png'},
        'lance': {'cooldown': 400, 'damage': 30,'graphic':'../graphics/weapons/lance/full.png'},
        'axe': {'cooldown': 300, 'damage': 20, 'graphic':'../graphics/weapons/axe/full.png'},
        'rapier':{'cooldown': 50, 'damage': 8, 'graphic':'../graphics/weapons/rapier/full.png'},
        'sai':{'cooldown': 80, 'damage': 10, 'graphic':'../graphics/weapons/sai/full.png'}}

#магія
magic_data = {
        'flame': {'strength': 5,'cost': 20,'graphic':'../graphics/particles/flame/fire.png'},
        'heal' : {'strength': 20,'cost': 10,'graphic':'../graphics/particles/heal/heal.png'}}

# ворог
monster_data = {
        'squid': {'health': 100,'exp':100,'damage':20,'attack_type': 'slash',
'attack_sound':'../audio/attack/slash.wav', 'speed': 3, 'resistance': 3, 'attack_radius': 80,
'notice_radius': 360},
        'raccoon': {'health': 300,'exp':250,'damage':40,'attack_type': 'claw',
'attack_sound':'../audio/attack/claw.wav','speed': 2, 'resistance': 3, 'attack_radius': 120,
'notice_radius': 400},
```

```
        'spirit': {'health': 100,'exp':110,'damage':8,'attack_type': 'thunder',
'attack_sound':'../audio/attack/fireball.wav', 'speed': 4, 'resistance': 3, 'attack_radius': 60,
'notice_radius': 350},
        'bamboo': {'health': 70,'exp':120,'damage':6,'attack_type': 'leaf_attack',
'attack_sound':'../audio/attack/slash.wav', 'speed': 3, 'resistance': 3, 'attack_radius': 50,
'notice_radius': 300}}
```

support.py

```
from csv import reader
from os import walk
import pygame

def import_csv_layout(path):
        terrain_map = []
        with open(path) as level_map:
                layout = reader(level_map,delimiter = ',')
                for row in layout:
                        terrain_map.append(list(row))
                return terrain_map

def import_folder(path):
        surface_list = []

        for _,__,img_files in walk(path):
                for image in img_files:
                        full_path = path + '/' + image
                        image_surf = pygame.image.load(full_path).convert_alpha()
                        surface_list.append(image_surf)

        return surface_list
```

tile.py

```
import pygame
from settings import *

class Tile(pygame.sprite.Sprite):
        def __init__(self,pos,groups,sprite_type,surface =
pygame.Surface((TILESIZE,TILESIZE))):
                super().__init__(groups)
                self.sprite_type = sprite_type
                y_offset = HITBOX_OFFSET[sprite_type]
                self.image = surface
                if sprite_type == 'object':
                        self.rect = self.image.get_rect(topleft = (pos[0],pos[1] - TILESIZE))
                else:
                        self.rect = self.image.get_rect(topleft = pos)
                self.hitbox = self.rect.inflate(0,y_offset)
```

ui.py

```
import pygame
from settings import *
```

```
class UI:
	def __init__(self):

		# загальне налаштування
		self.display_surface = pygame.display.get_surface()
		self.font = pygame.font.Font(UI_FONT,UI_FONT_SIZE)

		# налаштування бару
		self.health_bar_rect =
pygame.Rect(10,10,HEALTH_BAR_WIDTH,BAR_HEIGHT)
		self.energy_bar_rect =
pygame.Rect(10,34,ENERGY_BAR_WIDTH,BAR_HEIGHT)

		# конвертувати словник зброї
		self.weapon_graphics = []
		for weapon in weapon_data.values():
			path = weapon['graphic']
			weapon = pygame.image.load(path).convert_alpha()
			self.weapon_graphics.append(weapon)

		# конвертувати магічний словник
		self.magic_graphics = []
		for magic in magic_data.values():
			magic = pygame.image.load(magic['graphic']).convert_alpha()
			self.magic_graphics.append(magic)


	def show_bar(self,current,max_amount,bg_rect,color):
		# малюємо фон
		pygame.draw.rect(self.display_surface,UI_BG_COLOR,bg_rect)

		# перетворення характеристик в піксель
		ratio = current / max_amount
		current_width = bg_rect.width * ratio
		current_rect = bg_rect.copy()
		current_rect.width = current_width

		# малювання планки
		pygame.draw.rect(self.display_surface,color,current_rect)
		pygame.draw.rect(self.display_surface,UI_BORDER_COLOR,bg_rect,3)

	def show_exp(self,exp):
		text_surf = self.font.render(str(int(exp)),False,TEXT_COLOR)
		x = self.display_surface.get_size()[0] - 20
		y = self.display_surface.get_size()[1] - 20
		text_rect = text_surf.get_rect(bottomright = (x,y))

		pygame.draw.rect(self.display_surface,UI_BG_COLOR,text_rect.inflate(20,20))
		self.display_surface.blit(text_surf,text_rect)

	pygame.draw.rect(self.display_surface,UI_BORDER_COLOR,text_rect.inflate(20,20),3)
```

```
        def selection_box(self,left,top, has_switched):
                bg_rect = pygame.Rect(left,top,ITEM_BOX_SIZE,ITEM_BOX_SIZE)
                pygame.draw.rect(self.display_surface,UI_BG_COLOR,bg_rect)
                if has_switched:

        pygame.draw.rect(self.display_surface,UI_BORDER_COLOR_ACTIVE,bg_rect,3)
                else:
                        pygame.draw.rect(self.display_surface,UI_BORDER_COLOR,bg_rect,3)
                return bg_rect

        def weapon_overlay(self,weapon_index,has_switched):
                bg_rect = self.selection_box(10,630,has_switched)
                weapon_surf = self.weapon_graphics[weapon_index]
                weapon_rect = weapon_surf.get_rect(center = bg_rect.center)

                self.display_surface.blit(weapon_surf,weapon_rect)

        def magic_overlay(self,magic_index,has_switched):
                bg_rect = self.selection_box(80,635,has_switched)
                magic_surf = self.magic_graphics[magic_index]
                magic_rect = magic_surf.get_rect(center = bg_rect.center)

                self.display_surface.blit(magic_surf,magic_rect)

        def display(self,player):

        self.show_bar(player.health,player.stats['health'],self.health_bar_rect,HEALTH_COLOR
)

        self.show_bar(player.energy,player.stats['energy'],self.energy_bar_rect,ENERGY_COLO
R)

                self.show_exp(player.exp)

                self.weapon_overlay(player.weapon_index,not player.can_switch_weapon)
                self.magic_overlay(player.magic_index,not  player.can_switch_magic)
```

upgrade.py

```
import pygame
from settings import *

class Upgrade:
        def __init__(self,player):

                # загальне налаштування
                self.display_surface = pygame.display.get_surface()
                self.player = player
                self.attribute_nr = len(player.stats)
                self.attribute_names = list(player.stats.keys())
                self.max_values = list(player.max_stats.values())
                self.font = pygame.font.Font(UI_FONT, UI_FONT_SIZE)
```

```
                # створення предмета
                self.height = self.display_surface.get_size()[1] * 0.8
                self.width = self.display_surface.get_size()[0] // 6
                self.create_items()

                # система відбору
                self.selection_index = 0
                self.selection_time = None
                self.can_move = True

        def input(self):
                keys = pygame.key.get_pressed()

                if self.can_move:
                        if keys[pygame.K_RIGHT] and self.selection_index < self.attribute_nr -
1:
                                self.selection_index += 1
                                self.can_move = False
                                self.selection_time = pygame.time.get_ticks()
                        elif keys[pygame.K_LEFT] and self.selection_index >= 1:
                                self.selection_index -= 1
                                self.can_move = False
                                self.selection_time = pygame.time.get_ticks()

                        if keys[pygame.K_SPACE]:
                                self.can_move = False
                                self.selection_time = pygame.time.get_ticks()
                                self.item_list[self.selection_index].trigger(self.player)

        def selection_cooldown(self):
                if not self.can_move:
                        current_time = pygame.time.get_ticks()
                        if current_time - self.selection_time >= 300:
                                self.can_move = True

        def create_items(self):
                self.item_list = []

                for item, index in enumerate(range(self.attribute_nr)):
                        # горизонтальне положення
                        full_width = self.display_surface.get_size()[0]
                        increment = full_width // self.attribute_nr
                        left = (item * increment) + (increment - self.width) // 2

                        # вертикальне положення
                        top = self.display_surface.get_size()[1] * 0.1

                        # створити об'єкт
                        item = Item(left,top,self.width,self.height,index,self.font)
                        self.item_list.append(item)

        def display(self):
```

```
                self.input()
                self.selection_cooldown()

                for index, item in enumerate(self.item_list):

                        # отримати атрибути
                        name = self.attribute_names[index]
                        value = self.player.get_value_by_index(index)
                        max_value = self.max_values[index]
                        cost = self.player.get_cost_by_index(index)

        item.display(self.display_surface,self.selection_index,name,value,max_value,cost)

class Item:
        def __init__(self,l,t,w,h,index,font):
                self.rect = pygame.Rect(l,t,w,h)
                self.index = index
                self.font = font

        def display_names(self,surface,name,cost,selected):
                color = TEXT_COLOR_SELECTED if selected else TEXT_COLOR

                # назва
                title_surf = self.font.render(name,False,color)
                title_rect = title_surf.get_rect(midtop = self.rect.midtop +
pygame.math.Vector2(0,20))

                # вартість
                cost_surf = self.font.render(f'{int(cost)}',False,color)
                cost_rect = cost_surf.get_rect(midbottom = self.rect.midbottom -
pygame.math.Vector2(0,20))

                #малюй
                surface.blit(title_surf,title_rect)
                surface.blit(cost_surf,cost_rect)

        def display_bar(self,surface,value,max_value,selected):

                # налаштування креслення
                top = self.rect.midtop + pygame.math.Vector2(0,60)
                bottom = self.rect.midbottom - pygame.math.Vector2(0,60)
                color = BAR_COLOR_SELECTED if selected else BAR_COLOR

                # налаштування бару
                full_height = bottom[1] - top[1]
                relative_number = (value / max_value) * full_height
                value_rect = pygame.Rect(top[0] - 15,bottom[1] - relative_number,30,10)

                # малюємо елементи
                pygame.draw.line(surface,color,top,bottom,5)
                pygame.draw.rect(surface,color,value_rect)
```

```
        def trigger(self,player):
                upgrade_attribute = list(player.stats.keys())[self.index]

                if player.exp >= player.upgrade_cost[upgrade_attribute] and
player.stats[upgrade_attribute] < player.max_stats[upgrade_attribute]:
                        player.exp -= player.upgrade_cost[upgrade_attribute]
                        player.stats[upgrade_attribute] *= 1.2
                        player.upgrade_cost[upgrade_attribute] *= 1.4

                if player.stats[upgrade_attribute] > player.max_stats[upgrade_attribute]:
                        player.stats[upgrade_attribute] = player.max_stats[upgrade_attribute]

        def display(self,surface,selection_num,name,value,max_value,cost):
                if self.index == selection_num:

        pygame.draw.rect(surface,UPGRADE_BG_COLOR_SELECTED,self.rect)
                        pygame.draw.rect(surface,UI_BORDER_COLOR,self.rect,4)
                else:
                        pygame.draw.rect(surface,UI_BG_COLOR,self.rect)
                        pygame.draw.rect(surface,UI_BORDER_COLOR,self.rect,4)

                self.display_names(surface,name,cost,self.index == selection_num)
                self.display_bar(surface,value,max_value,self.index == selection_num)
```

weapon.py

```
import pygame

class Weapon(pygame.sprite.Sprite):
        def __init__(self,player,groups):
                super().__init__(groups)
                self.sprite_type = 'weapon'
                direction = player.status.split('_')[0]

                #графіка
                full_path = f'../graphics/weapons/{player.weapon}/{direction}.png'
                self.image = pygame.image.load(full_path).convert_alpha()

                # розміщення
                if direction == 'right':
                        self.rect = self.image.get_rect(midleft = player.rect.midright +
pygame.math.Vector2(0,16))
                elif direction == 'left':
                        self.rect = self.image.get_rect(midright = player.rect.midleft +
pygame.math.Vector2(0,16))
                elif direction == 'down':
                        self.rect = self.image.get_rect(midtop = player.rect.midbottom +
pygame.math.Vector2(-10,0))
                else:
                        self.rect = self.image.get_rect(midbottom = player.rect.midtop +
pygame.math.Vector2(-10,0))
```

ДОДАТОК Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОДИ ОПТИМІЗАЦІЇ ГРАФІЧНИХ РУШІЇВ 2D ІГОР

Рисунок Г.1 – Титульний слайд

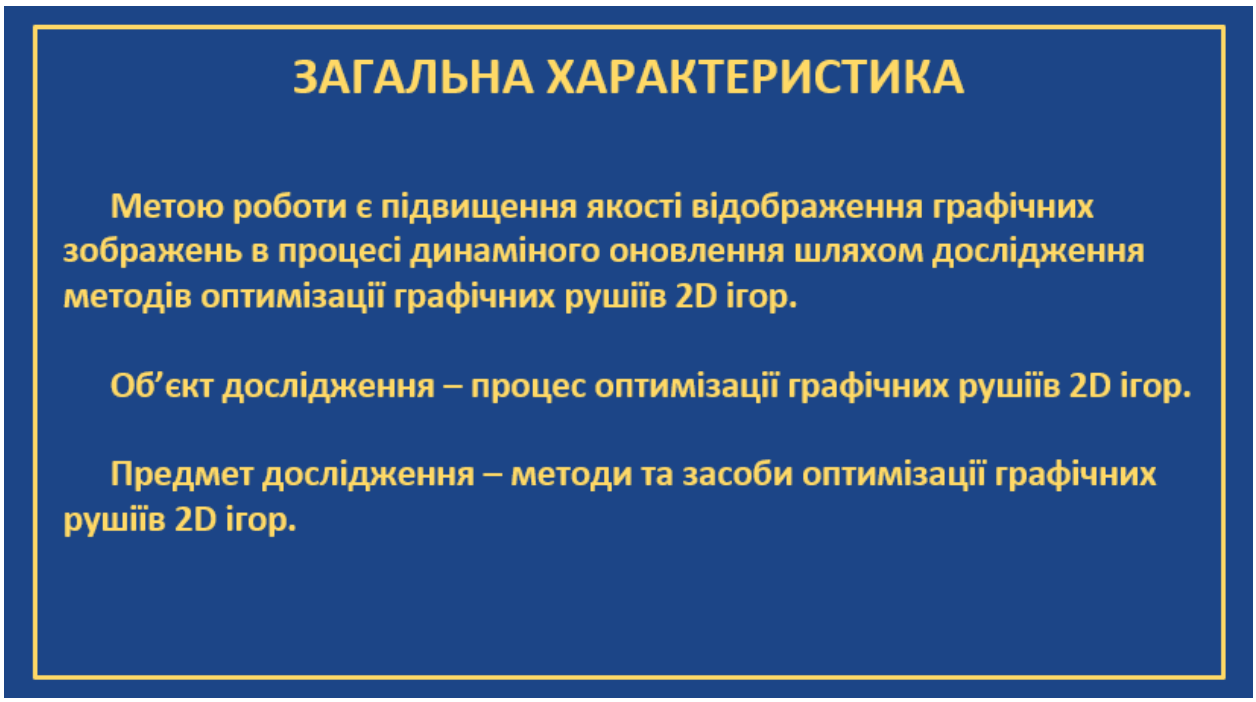

Рисунок Г.2 – Мета, об’єкт і предмет роботи

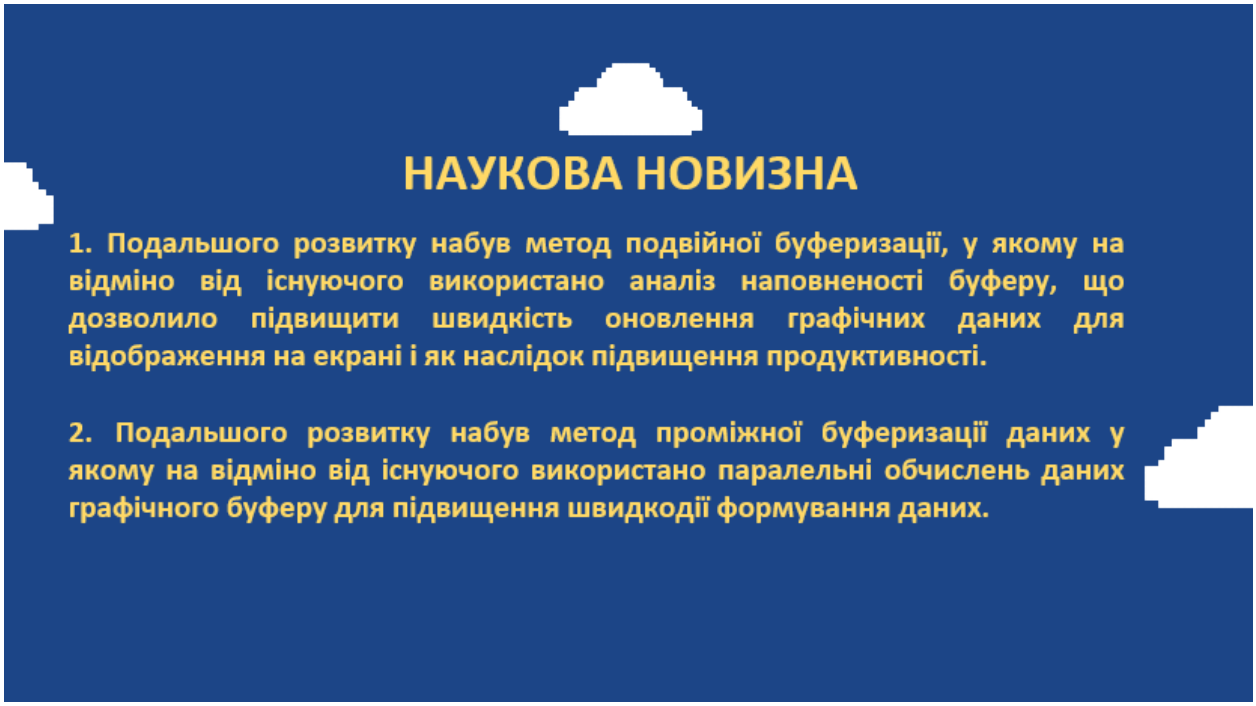

Рисунок Г.3 – Наукова новизна одержаних результатів

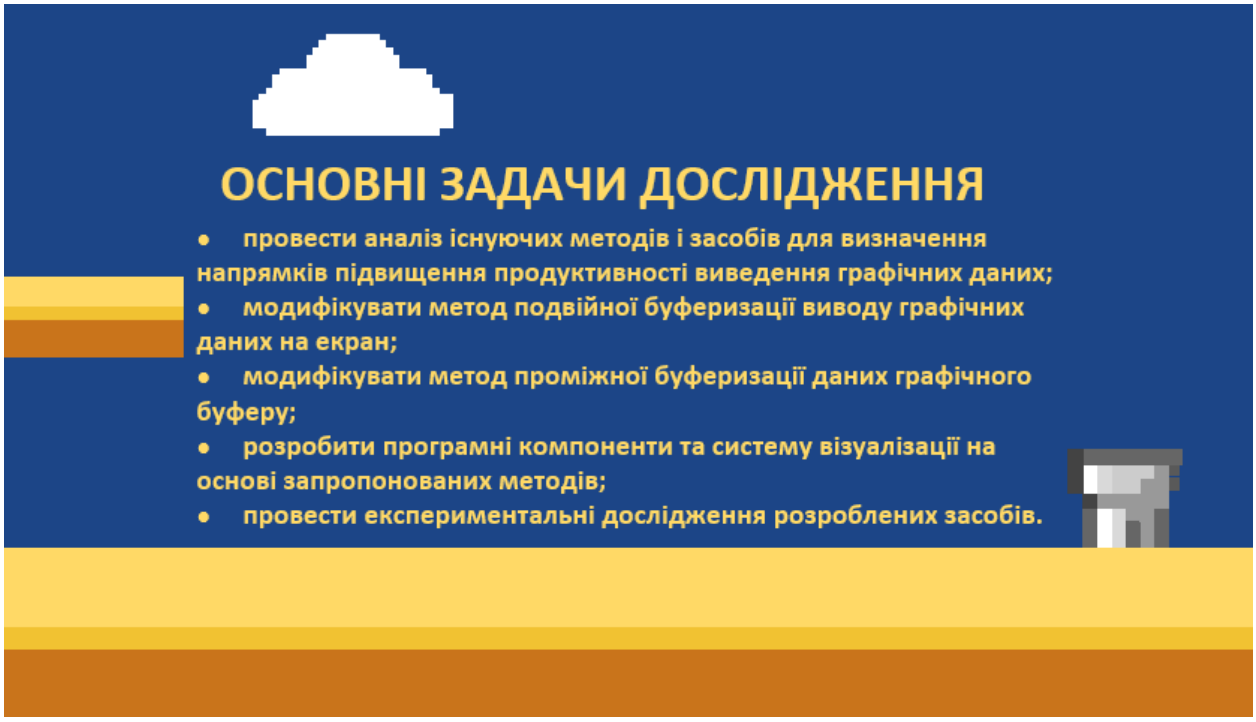

Рисунок Г.4 – Основні задачі дослідження

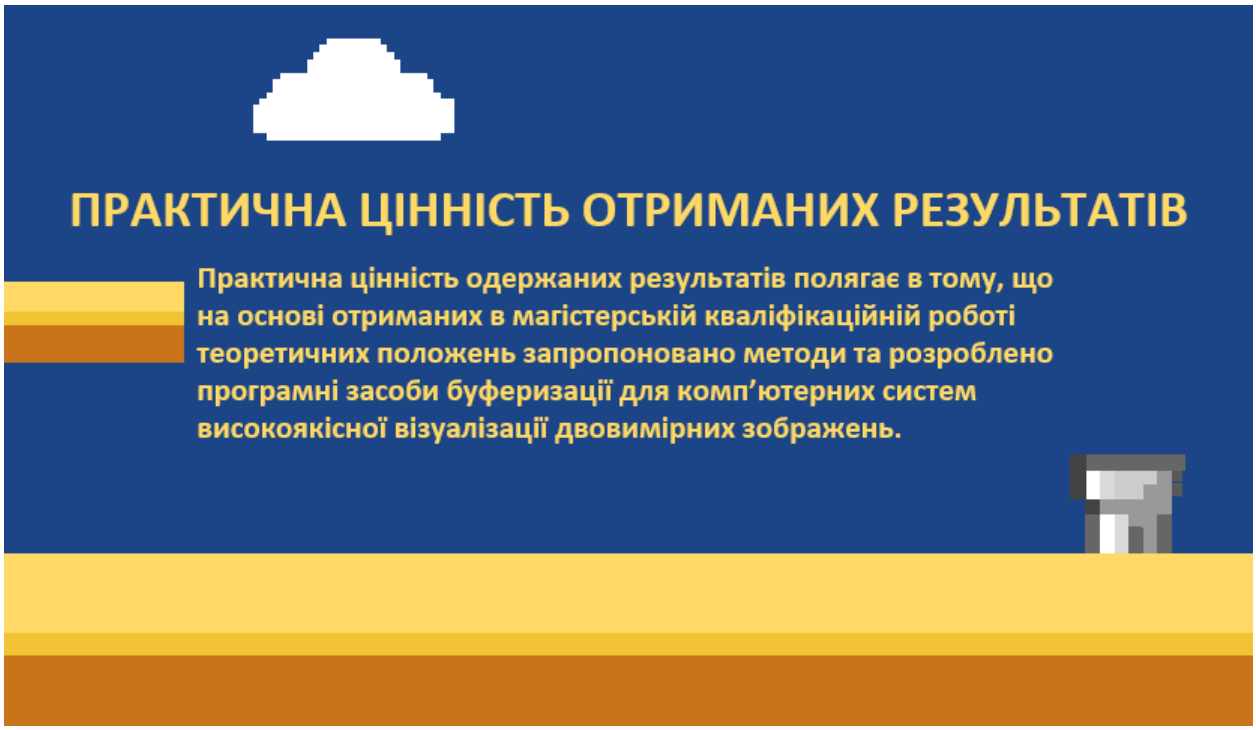

Рисунок Г.5 – Практична цінність результатів

ПОРІВНЯННЯ МЕТОДІВ ОПТИМІЗАЦІЇ

Критерії	Метод «Подвійна буферизація»	Метод «Керування оновленням екрану»	Метод «Спрайти та групи»
Плавність	1	0,5	0,5
Мінімізація мерехтіння	1	0,5	1
Затримка	1	1	0,5
Стабільність відображення	0,5	0,5	0,5
Синхронізація	1	0	1
Низьке навантаження ЦП	0,5	0	0,5
Універсальність	1	1	0
Всього	6	3,5	4

Рисунок Г.6 – Порівняння методів оптимізації

Рисунок Г.7 – Блок-схема загального алгоритму програми

Рисунок Г.8 – Блок-схема роботи алгоритм аналізу заповненості буфера

Рисунок Г.9 – Блок-схема роботи алгоритму аналізу змін даних в буферах з використанням паралельних обчислень

Рисунок Г.10 – Розробка програмної компоненти

Рисунок Г.11 – Тестування програмної компоненти

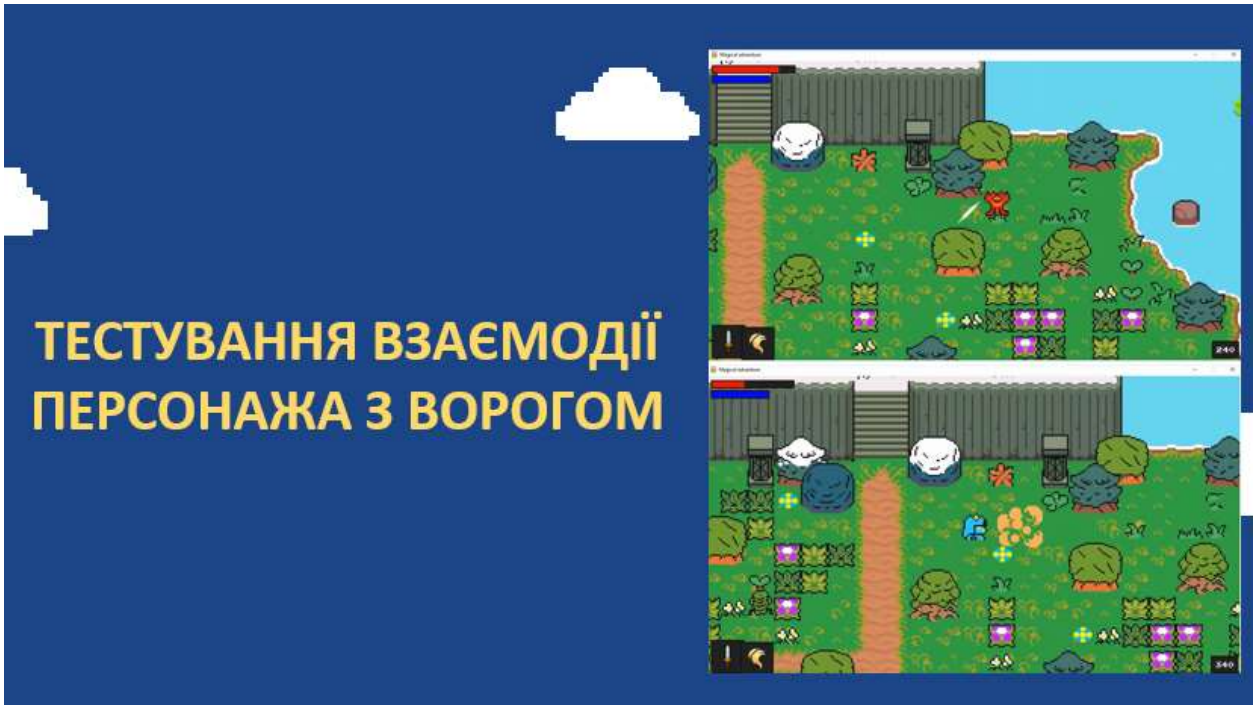

Рисунок Г.12 – Тестування взаємодії персонажа з ворогом

Рисунок Г.13 – Тестування зміни зброї і заклинань

Рисунок Г.14 – Тестування збільшення характеристики здоров'я гравця

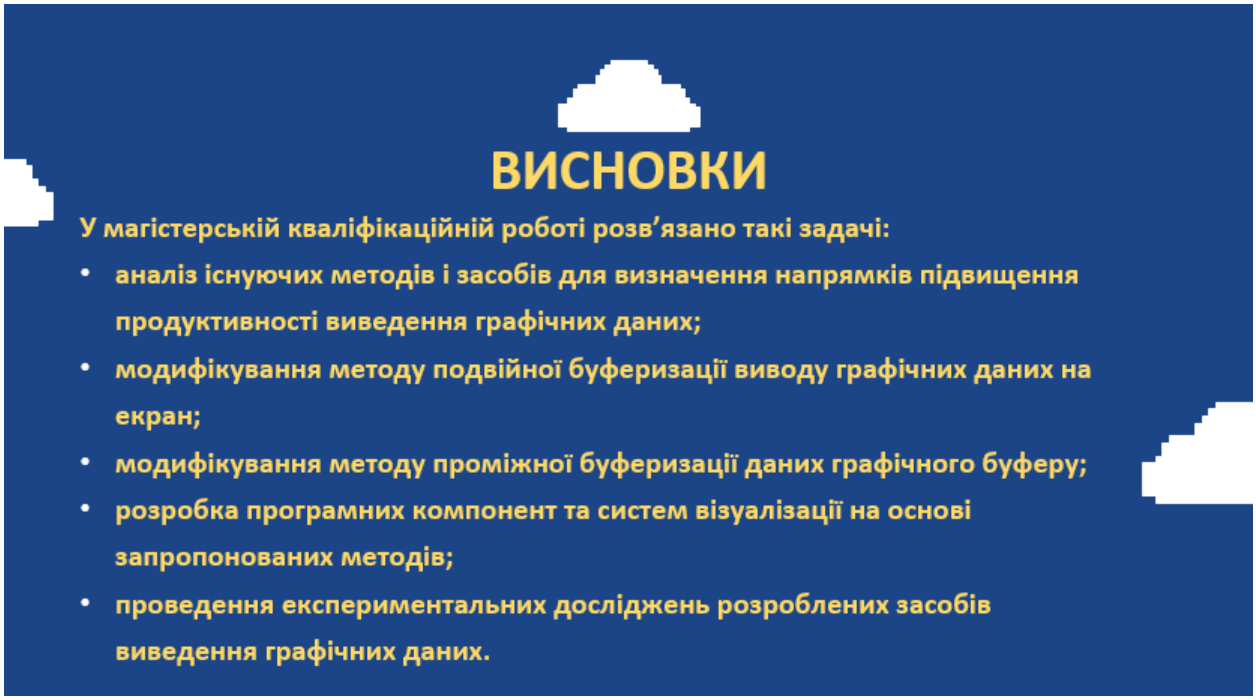

Рисунок Г.15 – Висновки

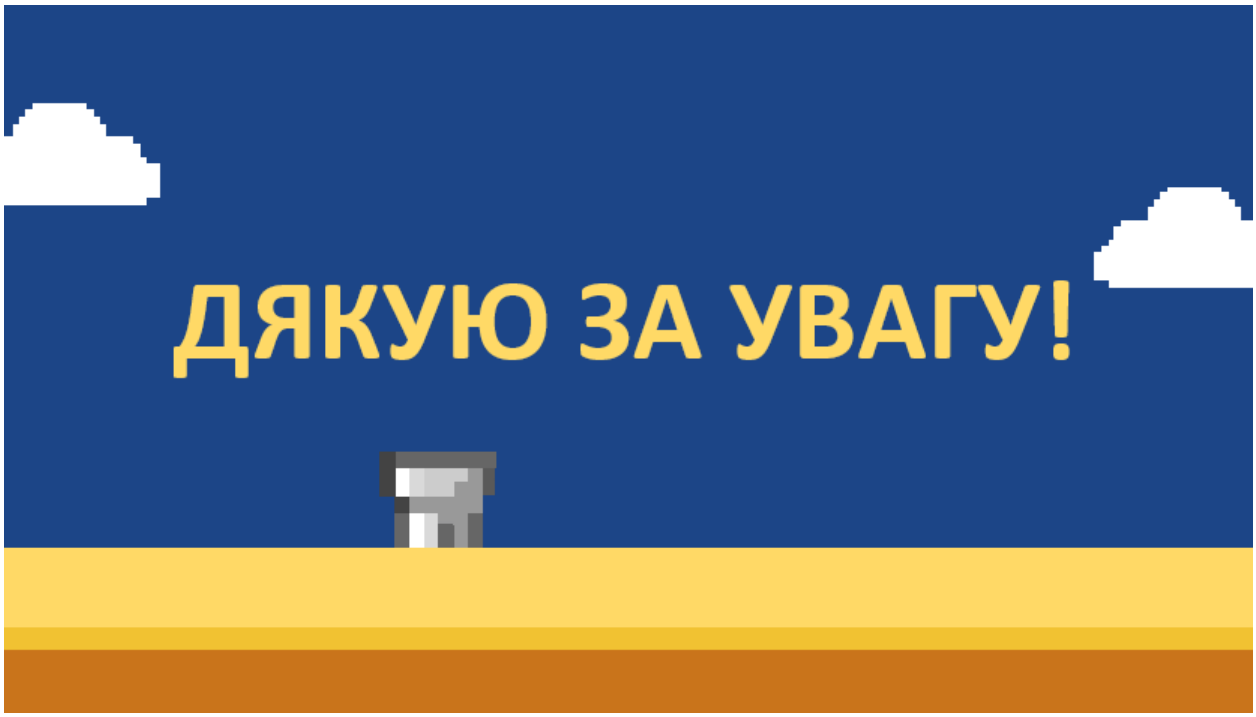

Рисунок Г.16 – Дякую за увагу