Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка комп'ютерної 2d-гри «Magical adventure»»

Виконав: студент 3 курсу
групи 1ПІ-19мс2
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Колісниченко Г.М.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д. І.
(прізвище та ініціали)

Рецензент:. к.т.н., ст. вик. КН Озеранський В.С.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«____» _____ 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
25 березня 2022 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Колісниченку Георгію Миколайовичу

1. Тема роботи – розробка комп'ютерної 2d-гри «Magical adventure».

Керівник роботи: Войтко Вікторія Володимирівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 року № 66

2. Строк подання студентом роботи 13 червня 2022 р.

3. Вихідні дані до роботи: середовище розробки Visual Studio Code, мова розробки Python, фреймворк – Pygame, операційна система – Windows 10, базові алгоритми для розробки 2d-ігор у жанрі екшн-пригода.

4. Зміст розрахунково-пояснювальної записки: вступ, аналіз використання систем прийняття рішень в ігрових додатках, обґрунтування доцільності розробки; теоретичні основи розробленого методу; варіантний аналіз та обґрунтування вибору засобів реалізації системи; проєктування структур даних і модулів ігрової системи; розробка програмного коду ігрової системи; тестування роботи ігрової системи;

5. Перелік графічного матеріалу: мета, об'єкт та предмет дослідження; задачі дослідження; методи дослідження; наукова новизна; практичне значення;

обґрунтування доцільності розробки; методи та моделі реалізації ігрової системи; програмна реалізація ігрового додатку; тестування; висновки, апробація, публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Катєльніков Д. І., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання 25 березня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
	Обґрунтування вибору методу розробки та постановка задач дослідження	26.03.2022 – 30.03.2022	Вик.
	Аналіз методів розв’язання поставленої задачі	31.03.2022 – 05.04.2022	Вик.
	Розробка методів і моделей реалізації комп’ютерної 2d-гри	06.04.2022 – 20.04.2022	Вик.
	Варіантний аналіз та обґрунтування вибору засобів реалізації ігрової системи	21.04.2022 – 25.04.2022	Вик.
	Розробка програмного коду ігрової системи	26.04.2022 – 15.05.2022	Вик.
	Тестування роботи ігрової системи	16.05.2022 – 25.05.20221	Вик.
	Оформлення матеріалів до захисту БДР	26.05.2022 – 10.06.2022	Вик.

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

Керівник бакалаврської дипломної роботи _____ Катєльніков Д.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається зі 127 сторінок формату А4, на яких є 61 рисунків, 3 таблиця, список використаних джерел містить 23 найменувань

У бакалаврській дипломній роботі запропоновано ігрову систему, орієнтовану на прийняття тактичних рішень у битвах з ворожими істотами та дослідження користувачем локацій.

Розроблено моделі ігрової системи, які реалізовані в демонстраційному ігровому додатку «Magical adventure».

Програмний продукт було створено з використанням мови програмування Python та фреймворку Pygame. Для розробки графічного інтерфейсу Tiled. В якості середовища розробки програмного забезпечення було обрано Visual Studio Code.

Розроблені метод та модель гри можуть бути використані при розробці сучасних ігрових систем, у яких передбачена робота з штучним інтелектом.

Ключові слова: комп’ютерна гра, 2D, екшн-пригода, Pygame.

ABSTRACT

The bachelor's thesis consists of 127 pages of A4 format, on which there are 61 figures, 3 table, the list of used sources contains 23 items.

The bachelor's thesis offers a game system focused on making tactical decisions in battles with enemy creatures and user exploration of locations.

Developed models of the game system implemented in the demonstration game application "Magical adventure".

The software was created using the Python programming language and the Pygame framework. To develop the Tiled GUI. Visual Studio Code was chosen as the software development environment.

The developed methods and models can be used in the development of modern game systems, which provide work with artificial intelligence.

Keywords: computer game, 2D, action adventure, Pygame.

ЗМІСТ

ВСТУП 8
1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ 12
1.1 Аналіз стану 2d-ігор 12
1.2 Порівняльний аналіз аналогів 15
1.3 Аналіз методів розв’язання задачі 20
1.4 Постановка задач для 2d-гри «Magical adventure» 25
1.5 Висновки 25
2 РОЗРОБКА МЕТОДУ, МОДЕЛІ Й АЛГОРИТМІВ РОБОТИ ІГРОВОЇ ПРОГРАМИ 26
2.1 Аналіз даних 26
2.2 Розробка структури інтерфейсу 2d-гри «Magical adventure» 30
2.3 Розробка методу спільного руху гравець-ворог 33
2.4. Розробка моделі роботи ігрової системи 34
2.5 Розробка алгоритмів роботи 2d-гри «Magical adventure» 35
2.6 Висновки 38
3 РОЗРОБКА 2D-ГРИ «MAGICAL ADVENTURE» 40
3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу 40
3.2 Розробка графічних матеріалів 43
3.3 Розробка основних модулів додатку 50
3.4 Висновки 64
4 ТЕСТУВАННЯ ІГРОВОГО ДОДАТКУ 66
4.1 Вибір методів тестування програмного забезпечення 66
4.2 Тестування розробленого додатку 69
4.3 Висновки 81
ВИСНОВКИ 83
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 84

ДОДАТКИ 87
Додаток А – Технічне завдання 88
Додаток Б – Протокол перевірки на плагіат 91
Додаток В – Лістинг програми 92
Додаток Г – Графічна частина 127

ВСТУП

Обґрунтування вибору теми дослідження. Комп’ютери та їх використання швидко й широко зростали у всьому світі. Завдяки різному потенціалу вони використовуються для вирішення багатьох завдань. Це допомагає вирішувати задачі, з якими стикається людина в повсякденному житті. Тому вони значно впливають на наше життя. Вплив використання комп’ютера на життя людини, очевидно, визначається як економія грошей, часу та зусиль. Щоб зрозуміти глибину втручання комп’ютера в життя суспільства, слід подивитися на розвиток подій у сферах зв’язку, освіти, комунальних послуг та охорони здоров’я.

Протягом останніх 3 десятиліть комп'ютер був визнаний найбільш успішним винаходом, що змінює життя і вирішує проблеми в житті людини. Сьогодні там, де працює бізнес, знайдеться комп’ютер. Подивившись на освіту, охорону здоров’я, транспорт чи сектор зв’язку, ми бачимо вплив і застосування комп’ютера. Важко вижити в бізнесі без використання комп’ютера прямо чи опосередковано в цьому сучасному світі [1].

Також комп’ютери стали невід’ємною частиною індустрії розваг. Вони використовуються для створення спецефектів у фільмах. Вони використовуються для редагування фільмів, створення повнометражних мультфільмів, мультимедійних презентацій тощо. Комп'ютери використовуються для моделювання ігор і для випробовування різних ігрових стратегій. Таким чином, використання комп’ютерів у сфері розваг – фільмів, музики, презентацій, спорту та ігор – зросло за останнє десятиліття, і тепер комп’ютери відіграють важливу роль у цих сферах.

Комп'ютерні ігри з кожним днем здобувають все більшу популярність і у дітей, і у дорослих. Всі люди різні, і з цієї причини кожна людина має свої улюблені ігри. Хтось любить стріляти, хтось брати участь в автоперегонах, хтось – будувати і т. д. Комп’ютерні ігри дозволяють розвивати мислення та тренувати

швидкість реакції гравця в екстремальних умовах. Тому актуальною є розробка комп'ютерної 2d-гри «Magical adventure».

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

Мета та завдання дослідження. Метою бакалаврської дипломної роботи є підвищення можливостей тренування тактичного мислення користувача шляхом розробки та використання 2d-гри жанру екшн-пригода з компонентами розвитку зорової уваги та прийняття тактичних рішень, яка сприятиме тренуванню швидкості прийняття рішень гравцем в екстремальних умовах.

Основними **задачами** роботи є:

- аналіз стратегій тренування тактичного мислення;
- удосконалення методу розвитку гри з використанням принципу спільного руху гравець-ворог;
- розробка моделей ігрової програми для тренування швидкості прийняття тактичних рішень;
- розробка інтуїтивно зрозумілого користувацького інтерфейсу програмного продукту;
- розробка комп'ютерного додатку «Magical adventure»;
- тестування створеного програмного продукту.

Об'єкт дослідження – процес створення комп'ютерних ігор.

Предмет дослідження –засоби реалізації комп'ютерних ігор, орієнтованих на розвиток реакції та зорової уваги користувача у процесі прийняття тактичних рішень.

Методи дослідження. У процесі досліджень використовувались методи дослідження:

- методи теорії алгоритмів для побудови алгоритмів функціонування ігрової системи;
- методи створення графічних зображень для розробки ігрових сцен;

- методи та технології розробки комп’ютерних ігор для реалізація 2d-гри у жанрі екшн-пригоди;
- методи тестування для перевірки працездатності створеного програмного продукту.

Наукова новизна отриманих результатів.

- Подальшого розвитку отримав метод розвитку гри, який, на відміну від існуючих, базується на використанні принципу спільного руху гравець-ворог, що дозволяє швидко створювати і додавати нові види ворогів в ігрове середовище, оскільки вони наслідують деякі методи-функції персонажа гравця, що прискорює процес розробки та ускладнення екшн-ігор.
- Подальшого розвитку дістала модель ігрової системи, яка, на відміну від існуючих, орієнтована на використання компонентів тактичного мислення за допомогою збільшення характеристик персонажа, що забезпечить розвиток різноманітності ігрового процесу і створить зручні умови тренінгу для користувача.

Практична цінність отриманих результатів. Практична цінність полягає у кінцевій реалізації комп’ютерної 2d-гри «Magical adventure» у жанрі екшн-пригода.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У науковій роботі [2], опублікованій у співавторстві, автору належать алгоритми роботи програми.

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на Всеукраїнській науково-технічній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2022). Секція Інформаційні технології та комп‘ютерна інженерія».

Публікації. Основні результати дослідження опубліковані в науковій роботі [2] – тезах доповіді на Всеукраїнській науково-технічній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2022). Секція Інформаційні технології та комп‘ютерна інженерія».

Аналіз. У пояснювальній записці до бакалаврської дипломної роботи було розглянуто 4 розділи та було використане 23 літературне джерело.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану 2d-ігор

2D ігри – ігри з видом зверху, тобто камера розташована прямо перпендикулярно або під кутом над персонажем, або ігри з видом збоку чи від третьої особи, коли ігровий процес постає у вигляді панорами. Спочатку 2D графіка використовувала бітове зображення, пізніше еволюціонувала до 8-бітної, а потім і до 15/16 біт, так званого HighColor, а вже потім 24 біти, відомої як TrueColor і, нарешті, 48-бітне зображення – DeepColor.

Ігрова індустрія з 2D ігор стартувала і досі цей вид залишається актуальним як серед деяких невеликих компаній, так і серед інді-розробників. Серед 2D ігрових проєктів поширене використання піксельної графіки через простоту використання. З такою графікою малим групам розробників або розробникам-одинакам легше створити анімації, промалювати рівні та інше, приділивши більше уваги таким тонкощам, як опрацювання сюжету та персонажів. Однак не варто забувати, що на світі є багато шедеврів ігрової індустрії, які внесли нові фарби в ігрову індустрію, створили нові сеттинги та всесвіти, що використовуються і досі. Деякі з них, звичайно, перекочували в руки іншої компанії, яка частково занапастила, а частково дала нове життя програмам – і все це через банкрутство початкових авторів – серія ігор Fallout (рис. 1.1). Загалом можна сказати, що було б бажання, вміння і фантазія – одна маленька компанія зможе створити щось велике, що могло б бути не зневаженим залежністю від прибутку.

Рисунок 1.1 – Серія ігор Fallout

Недоліки таких ігор полягають лише у візуальній складовій, оскільки в 21-му столітті багато хто просто не переносить виду піксельної графіки. Але групі недосвідчених розробників, особливо, якщо це їх дебютний проєкт, це можна пробачити, тому що їхні можливості обмежені ресурсами та навичками. Також варто відмітити, що багато сучасних сеттингів стартували саме з 2D: Grand Theft Auto, The Elder Scrolls, Warcraft, Warhammer 4000, Fallout та інші. Наприкінці 90-х років отримав широку популярність стиль псевдо-3D, який все ще залишався 2D, створюючи ілюзію 3D. Найвідомішими представниками такої стилістики є Doom, Doom 2, Wolfenstein 3D (рис 1.2). Остання, незважаючи на назву, яка є рекламним ходом, гра двомірна. Головним недоліком такої графіки є неможливість деталізувати окремі деталі. Плоскі моделі одного персонажа іноді доводиться малювати кілька разів, оскільки сам двигун гри не дозволяє зробити одну цільну модель. Також не варто забувати про анімації персонажам – у багатьох випадках вони рвані і виглядають як картинки, що змінюються, а не рух [3].

Рисунок 1.2 – Гра Wolfenstein 3D

Останнім часом індустрія відео зазнає періоду так званого «оказуалювання» (від слова «casual»), тобто ігровий процес спрощується для нових гравців, яких може відлякати складний геймплей. Часто це викликано піратством – розробники не хочуть старатися для тих, хто не заплатить, а частково й через самих гравців – ігри з цікавим, але складним геймплеєм просто не окупаються і тому вигідніше робити простенький шутер замість продуманої та складної гри.

Однак буває і таке, що розробники просто полінувалися або керівники компанії хочуть великих заробітків з продажу в короткі терміни, через що кінцевий продукт від тих, хто в цій компанії працює, виходить у світ сируватим, місцями недоробленим і з «дірявим» сюжетом, що не тішить як самих творців, так і керівників компаній, які ризикують у такому разі залишитися без гроша в кишені, що, до речі, траплялося досить рідко, тож, як не дивно, подібні проєкти окуплюються, нехай і не задовольняючи бажання керівництва.

Також абсолютно кожен, хто хоч раз у своєму житті тримав у руках мишку з клавіатурою, геймпад, джойстик або хоча б ігрове кермо, знають про велике протистояння консолей та персональних комп’ютерів (ПК). Але, якщо подивитися об'єктивно, то у кожної ігрової програми знайдуться певні переваги та недоліки. Проте піратство є головною проблемою розробників ігор. Консольні ексклюзиви мають величезну перевагу для розробників – складно «спіратити» гру. А для більшості гравців – любителів безкоштовних ігор – це величезний мінус. Про це

часто замовчують, адже не прийнято відверто говорити про любов до безкоштовного.

1.2 Порівняльний аналіз аналогів

Існує досить велике різноманіття 2d-ігор жанру «Action-adventure». До найбільш відомих відносять:

– Unreal Element World;
– Crypt of the NecroDancer;
– Dead Hearts;
– Rise of the Third Power.

Unreal Element World (рис. 1.3) – екшн-пригоди з механіками рогалика та з видом камери зверху, майже як у Unreal Gold. Гру можна завантажити торрентом Unreal Element World.

Обирайте того героя, який найбільше підходить за набором якостей і здібностей, а потім вирушайте на пошуки дивного спорядження, щоб зробити свого протагоніста ще сильнішим, ніж він був до цього.

На відміну від інших представників даного жанру, тут гра заохочуватиме до використання елементів навколишнього середовища, якими можна користуватися в бою.

Комбінуйте всі знайдені речі, і тоді ефект від цього буде в рази суттєвішим.

Ще одна головна відмінність цієї гри від інших roguelike-розваг – це те, що гра не має значної динамічності, а навпаки, допоможе злегка сповільнитися [4].

Всі доступні предмети не вічні, а також здатні зношуватися, тому необхідно стежити за їх станом і вчасно займатися ремонтом.

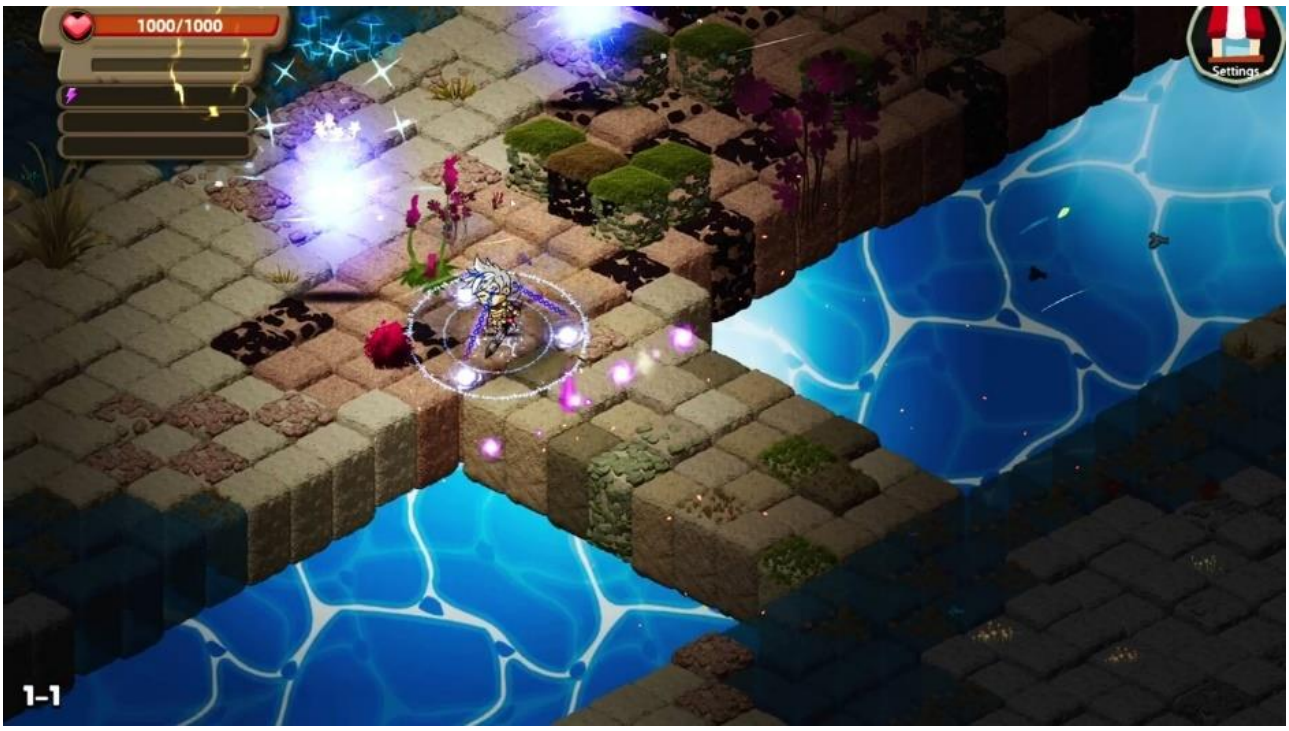

Рисунок 1.3 – Гра Unreal Element World

Crypt of the NecroDancer (рис. 1.4) є сумішшю ритм-ігри та слешера з двомірною піксельною графікою.

Спробуйте на собі цю незвичайну розвагу, яка вже встигла отримати кілька престижних нагород. Переміщуйтеся в такт із музикою та вибивайте правильний ритм на черепах ваших повалених супротивників! Все це дійство відбувається під чудовий музичний супровід музиканта Денні Барановські.

Вивчайте простори похмурих підземель, які кишять різноманітними смертоносними істотами, починаючи від танцюючих скелетонів та зомбі і закінчуючи величними драконами [5].

Пам'ятайте, що рухатися потрібно під музику, інакше на вас чекає крах. Для зручного керування можна вибрати клавіатуру і мишу або ж геймпад, але, якщо хочете випробувати дивовижні та реалістичні емоції, то можна запастися й повноцінним танцюючим килимком, який можна підключити до комп'ютера за допомогою USB.

Рисунок 1.4 – Гра Crypt of the NecroDancer

Dead Hearts (рис. 1.5) є сумішшю аніме-пригоди та рольової гри у стилістиці темного фентезі. Гравцю доведеться перетворитися на героя з ім'ям Зеро, який проживає в стародавніх і високих горах. Якось уночі до нього в поселення вторгаються елітні солдати з найближчої імперії Арайнія. Єдиними, кому вдалося вижити, є гравець, а також його сестра і подруга дитинства.

Тепер гравцю потрібно боротися за своє життя та вивчати простори пустелі, яка здатна привести до процвітаючого міста.

Ігровий процес є досить динамічним, а всі бої з супротивниками відбуватимуться у реальному часі з видом перспективи нагорі.

Прокачуйте рівень свого протагоніста, шукайте унікальні предмети спорядження та срібло, а також покращуйте всілякі навички та зброю, щоб з кожним разом ставати все сильнішими [6].

Рисунок 1.5 – Гра Dead Hearts

Rise of the Third Power (рис. 1.6) – це любовний лист у дні слави RPG у консольному стилі з сучасними зручностями, такими як автозбереження та поєднанням найкращих елементів ігрового процесу та письма в японському та західному стилях.

Історія. Приєднуйтесь до групи з восьми осіб, кожен зі своїми унікальними амбіціями, перспективами та особистостями, коли вони приступають до самогубної місії, щоб повалити імператора Аркадія, Дмитра Нораскова.

Політичні інтриги. У міру того, як світ Рін оговтується від Великої війни, баланс сил залишається спотвореним, а вакуум сили все ще чекає на заповнення. Партію гравця охоплюватимуть і допомагатимуть інтриги, брехуни та зрадники, поки годинник тікає до повторення найбільшої війни в історії людства. [7].

Рисунок 1.6 – Гра Rise of the Third Power

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльні характеристика аналогів

Критерії	Unreal Element World	Crypt of the NecroDancer	Dead Hearts	Rise of the Third Power	Magical adventure
Інтуїтивне управління персонажом	+	+	+	-	+
Музичний супровід і звукові ефекти	+	+	+	+	+
Мінімалістичні візуальні ефекти	-	+	+	-	+
Система підвищення характеристиків персонажа	+	-	-	+	+
Мінімальні системні вимоги	-	-	+	+	+
Абсолютно безкоштовна гра	-	-	-	-	+

Відповідно до таблиці 1.1.з порівняльними характеристиками розробка власної 2d-гри є доцільною. Отриманий продукт зможе виправити недоліки аналогів і забезпечити новий ігровий досвід користувачу.

1.3 Аналіз методів розв'язання задачі

Ігри створюються за допомогою рушіїв – набору інструментів, щоо дозволяє працювати з графікою, фізикою, скриптами та іншим.

Ігровій рушій Unity зображений на рисунку 1.7

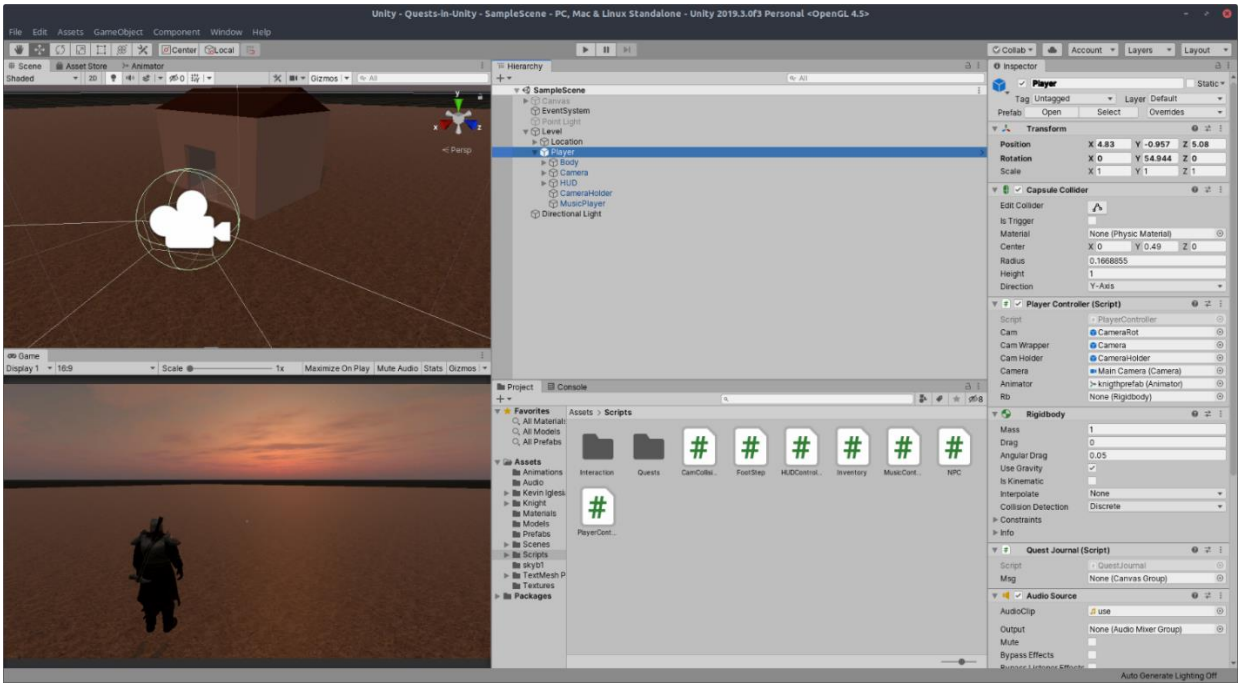

Рисунок 1.7 – Ігровий рушій Unity

У лівому верхньому кутку – ігрова сцена, на яку можна додавати об'єкти, рухати їх, прибирати і таке інше.

Нижче розташоване ігрове вікно – в ньому можна побачити, як виглядатиме готова гра. Можна навіть натиснути кнопку Play і пограти.

Далі можна побачити ієрархію об'єктів на сцені, файловий менеджер та вкладку Inspector – у ній є різні настройки для вибраного об'єкта. Крім того,

можна зайти в налаштування проєкту та вказати бажані показники для гравітації, освітлення, тіней, якості графіки та іншого.

Також у рушія є підтримка скриптів та API. Скрипти допомагають писати команди, які виконуватимуться грою весь час або після якихось дій гравця. API допомагає спростити написання скриптів. Тобто ви не проводите складних математичних розрахунків, щоб змінити положення або обертання об'єкта – ви просто пишете команду на кшталт «Unity, поверни об'єкт А на 5 градусів по осі X» [8].

Можна використовувати готовий рушій або написати свій – у обох варіантів є плюси та мінуси.

Переваги готового рушія:

– Велика кількість готових інструментів, щоб реалізувати свої ідеї.

– Спільнота. Інші розробники, які користуються цим же рушієм, можуть допомогти при розробці, головне задати правильне запитання.

– Налагоджене портування ігор на інші платформи.

– Є безкоштовні варіанти рушіїв.

– Власні магазини рушіїв з готовими скриптами, моделями, ефектами і т.д.

Недоліки готового рушія:

– Можлива наявність бага, який в силах виправити лише розробник рушія.

– Ліцензія, іноді доводиться ділитися частиною прибутку.

– Розробники можуть зупинити підтримку рушія.

– Надмірна кількість не потрібних функцій саме при вашій розробці, що впливає на розмір гри.

Переваги самописного рушія:

– Тільки ті інструменти що потрібні вам.

– Хороша оптимізація, в результаті відсутності непотрібних модулів і інструментів.

– Краща інтуїтивність.

Недоліки готового рушія:

– Довгий період написання рушія.

– Велика вартість.

– Помилки у проєктуванні можуть бути критичними як для ігри, так і для рушія.

Список сучасних рушіїв:

– Unity;

– Unreal Engine 4;

– CryEngine 3;

– Source, Source 2;

– Creation Engine;

– Godot;

– Cocos2D;

– Game Maker Studio 2;

– RPG Maker і т.д.

Мови програмування. З їх допомогою відбувається опис для комп'ютера умов і команди: якщо А, зроби Б, а якщо В, зроби Д. Незважаючи на те що рушій беруть на себе велику частину роботи, програмувати доведеться багато [9].

Переміщення по меню, перехід між локаціями, керування персонажем, рух камери, зміна музики, діалоги, система квестів – все це та багато іншого потрібно буде запрограмувати. Не кажучи вже про штучний ігровий інтелект.

Конкретний рушій, підтримує конкретні мови. Наприклад, у Unity підтримуються C# та JavaScript (його модифікація, яка називається UnityScript), а в UE4 - C++ [10].

Найпопулярніші мови програмування для розробки ігор:

– C#;

– C++;

– Java;

– Python;

– Swift;

– Objective-C;
– JavaScript;
– PHP;
– Lua.

Фреймворки дають змогу використовувати мову, щоб написати гру без двигуна. Наприклад, на JavaScript створюються браузерні ігри, на C++ або C# – ігри для комп'ютерів, Java – для пристроїв на Android, і так далі.

Для цих мов є бібліотеки для роботи з графікою, або цілі фреймворки для створення ігор. Фреймворк – це каркас, майже готовий додаток. Розробник просто дописує для цього каркаса якісь додаткові функції, підганяючи його цим під свої потреби [11].

Використання бібліотек чи фреймворків поруч із написанням власного движка дає максимальну свободу дій при розробці. Але зникає можливість користуватися графічним інтерфейсом, а всі налаштування та параметри доводиться писати за допомогою коду.

Список найпопулярніших фреймворків:

– XNA та MonoGame для C#;
– PyGame для Python;
– Pixi.JS та Babylon.JS для JavaScript;
– SceneKit та SpriteKit для Swift.

Бібліотеки, на відміну від фреймворків, не дають майже готову програму, але надають певні інструменти. Найпростіший приклад – бібліотека Math (математика), яка є практично у кожній мові програмування.

Використання Math дозволяє без проблем зводити числа в мірі, знаходити коріння, шукати модулі, вираховувати синуси, косинуси тощо. Ви не реалізуєте все це самостійно, а просто викликаєте потрібну функцію та передаєте їй параметри [12].

У розробці ігор використовують складніші бібліотеки, які дозволяють працювати з графікою або фізикою. Наприклад, графічні бібліотеки дають змогу раструвати ігрові об'єкти.

Тобто розробник не пише для відеокарти інструкцію, які пікселі виводити їй. Натомість він додає в гру спрайти (зображення) або 3D-моделі, а графічна бібліотека сама вираховує, як це має виглядати на моніторі.

Фізичні бібліотеки беруть роботу з фізикою: прискорення вільного падіння, закон збереження енергії, імпульси, вектори – найпростіші приклади.

Аналогічні бібліотеки є для роботи зі звуком і т.д.

Список графічних бібліотек:

– OpenGL;

– WebGL;

– DirectX.

Список фізичних бібліотек:

– Havok;

– PhysX.

Втім, якщо використовувати якийсь рушій, то особливо замислюватися про це не потрібно.

Критерії вибору інструмента для розробки такі:

1. Спільнота – чим більше людей користуються рушієм/фреймворком, тим активніше його розвиватимуть і тим більше навчальних матеріалів по ньому.

2. Ігри, що вийшли – по них можна оцінити можливості рушія/фреймворку.

3. Складність – простота інтерфейсу (рушій), мова програмування що використовується (рушій і фреймворк).

4. Підтримувані платформи – якщо планується розробка мобільних ігор, необхідно переконатися, що обраний рушій (фреймворк) підтримує таке портування.

5. Спрямованість рушія (фреймворку) на 2d чи 3d.

1.4 Постановка задач для 2d-гри «Magical adventure»

Проаналізувавши переваги та недоліки існуючих 2d-ігор жанру «Action-adventure», було сформульовано такі задачі бакалаврської дипломної роботи:

- провести аналіз стратегій тренування тактичного мислення;
- удосконалити метод розвитку гри з використанням принципу спільного руху гравець-ворог;
- розробити модель ігрової програми для тренування швидкості прийняття тактичних рішень;
- розробити інтуїтивно зрозумілий користувацький інтерфейс програмного продукту;
- розробити комп'ютерний додаток «Magical adventure»;
- провести тестування створеного програмного продукту.

.

1.5 Висновки

У першому розділі було розглянуто стан 2d-ігор в ігровій індустрії на сьогоднішній день. Було проведено порівняння відомих 2d-ігор, таких як: Unreal Element World, Crypt of the NecroDancer, Dead Hearts та Rise of the Third Power. У результаті порівняння було виявлено, що досліджувані ігри набули значної популярності серед гравці завдяки своїй 2d стилістиці та невеликому навантаженню на ресурси комп'ютера. Проаналізувавши переваги та недоліки відомих 2d-ігор, було виявлено, що вони частково не відповідаю основним критеріям 2d-ігор жанру екшн-пригода. Таким чином було доведено доцільність розробки власного програмного рішення. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власної 2d-гри жанру екшн-пригода.

2 РОЗРОБКА МЕТОДУ, МОДЕЛІ Й АЛГОРИТМІВ РОБОТИ ІГРОВОЇ ПРОГРАМИ

2.1 Аналіз даних

Як відомо, першою грою з прокачуванням (принаймні з тих, які справді відомі багатьом) є шахи. Пішак, що дійшов до ворожого краю поля, підвищує рівень і покращується до будь-якої іншої фігури на вибір гравця. Проте варто зауважити, що шахи (як і шашки) – гра досить абстрактна, хоч і є номінально відображенням протистояння двох армій.

Інша справа, коли йдеться про гру, де гравець керує окремими персонажами і вони не беруть участь в одній окремій битві, а досліджують підземелля, подорожують світом і займаються не лише боєм, а й іншими справами. І тут ігровий час може обчислюватися тижнями і місяцями. Більше того, гра Dungeons&Dragons, яка першою стала розвивати цей напрямок, за задумом авторів повинна була дозволити гравцям відчути себе героями творів жанру Sword&Sorcery, до якого належать романи Роберта Говарда та Едгара Берроуза. І, як ви, напевно, знаєте, на початку цих романів герой найчастіше є звичайним нубом [13].

Цей герой вирушає на пошуки пригод (або як варіант на пошуки вбивці своїх близьких), набирається на шляху досвіду, крутіє і в самому кінці стає легендарним воїном і перемагає головного злидня. І цей аспект самовдосконалення по ходу сюжету потрібно було відобразити в D&D.

І це треба було зробити досить просто. У настільних іграх гравці (і майстер гри) самі повинні стежити за тим, що відбувається, все пам'ятати (або записувати) і якщо більшу частину часу вони займатимуться розрахунками, то захоплюючих пригод це не додасть. Тож творці гри просто ввели кілька (десятків) рівнів крутості для персонажів. А досвід зробили простою числовою характеристикою, величина якої зростає в міру вчинення подвигів та інших справ. При наборі певної кількості досвіду герой переходить більш високий рівень – все просто [14].

Така система розвитку називається лінійною і вона стоїть окремо, не тому, що вона дуже відрізняється від інших систем, а навпаки – вона є спрощеним варіантом усіх систем прокачування одночасно. А решта систем походить від неї.

Якщо розглядати системи прокачування з погляду математики, тут не обійтися без такого поняття, як граф. «Граф» у математиці (для тих хто ще не знає чи вже не пам'ятає) – це набір точок (званих вершинами), які якимось чином з'єднані лініями (їх називають ребрами). Існує ціла «теорія графів», що вивчає закономірності, властивості, які різні графи мають. Найчастіше розглядаються різні способи «обходу» графів, коли ми є об'єкт, який може переміщатися з однієї вершини в іншу за умови, що ці вершини з'єднує ребро.

Крім простих графів, де важливо лише які вершини з'єднані, а які ні, є й накручені. Зокрема ребра можуть бути спрямованими – тобто з однієї вершини графа можна потрапити в іншу, але назад тим самим шляхом повернутися не можна. У ребр і вершин також можуть бути різні характеристики, які знов-таки можуть бути пов'язані з тим, чи можна перейти в певну вершину чи ні

Власне, основа кожної системи прокачування – це граф (або як його часто, хоч і не зовсім правильно, називають дерево) розвитку. У деяких іграх (наприклад, серії Final Fantasy Tactics) ми переміщуємо по графу самого персонажа. В інших ми поступово отримуємо «окуляри таланту» (або як вони називаються у кожній конкретній грі), які і поміщаємо на вершини графа, що відповідають потрібним навичкам .

Проте, більшість гравців вивчення графів прокачування важливе лише з погляду оптимального шляху отримання потрібних навичок. І набагато більше їх займає інша сторона процесу – дії, які персонаж має вжити для того, щоб «просунутися» потрібним шляхом. Саме тому на мій погляд сортувати системи прокачування потрібно насамперед за цією ознакою. У такому разі їх можна поділити на три типи [15].

1. Свідомий вибір. Системи з явним вибором, безперечно, є найпопулярнішими системами прокачування на сьогоднішній день. У грі з такою

системою гравець іноді отримує нагороду – ресурс поліпшення. Не важливо, коли і як він його отримує (збирає сам, автоматично отримує при досягненні нового рівня або просто за сюжетом) і як саме цей ресурс називається (бали навички, бали здібностей, таланти або просто позначається цифрою), суть залишається незмінною. Купує на нього покращення для свого персонажа.

Плюси і мінуси. По-перше, це легкість проходження гравця своїм бажанням. Вважає гравець, що йому потрібно збільшити силу та кількість здоров'я – він бере і збільшує, хоче придбати навичку гіпнозу для впливу на NPC – набуває та впливає.

По-друге, розподіл характеристик та навичок вручну дозволяє «тонко налаштувати» свого персонажа під себе. Якщо вибраний стиль проходження та улюблена тактика вимагають певних здібностей, то можна розвинути їх максимально, заощадивши саме на тому, що вам не потрібно.

Мінуси систем з вибором безлічі варіантів розвитку. Багато з них розробники не зможуть протестувати. Звідси проблеми з балансом, коли один із варіантів практично марний, інший навпаки – надсильний.

Інший мінус – відхід реалізму виглядає прямо скажемо незначним.

2. За законами природи. Друга категорія систем прокачування заснована на виведеному Ламарком колись «принципі вправи органів». Ламарк вважав, що еволюція відбувається за рахунок того, що тварини в ході свого життя тренують ті органи, якими часто користуються, а їхнє потомство вже отримує такі «натреновані» органи у спадок. Нісенітниця насправді давно вже доведено, що це не так.

Однак, якщо говорити не про еволюцію, то тренування справді допомагає покращувати навички. регулярно бігатимете – зможете бігати швидше і далі, будете вправлятися в стрільбі – станете влучнішим. Немає нічого дивного, що у багатьох іграх розвиток персонажів обставлений саме так.

Плюси і мінуси. Першим плюсом систем з природним розвитком персонажів можна назвати реалізм та правдоподібність. Стріляє персонаж з

автомата – підвищує влучність, зламує електронні замки – покращує навички злому, а не навпаки.

Звідси випливає і другий плюс – інтуїтивно зрозумілий розвиток. В іграх з вибором навичок цей вибір нерідко залишає людину у скруті.

Головний мінус «природних» систем – це природний розвиток. Не можна в них просто взяти та покращити володіння магією чи дипломатією. Треба обов'язково тренуватися, причому саме у цій конкретній галузі.

3. Збери їх усіх. Третій тип прокачування у певному сенсі близький до другого, оскільки тут гравець теж має вчинити певні дії для отримання необхідної навички. Однак, на відміну від «природного зростання», тут поліпшення навичок відбуваються не по порядку, а залежно від конкретних дій. Випив зілля - отримав посилення персонажа вночі, здійснив магічний ритуал - став невразливим для певного виду магії, зібрав бонус – збільшив кількість здоров'я.

Звичайно, дане прокачування використовується в першу чергу в екшенах, платформерах і подібних іграх. Збір магічних карт та амулетів у франшизі Castlevania, виконання додаткових завдань у Painkiller, пошук покращень для енергетиків/плазмідів у серії BioShock. Однак і в CRPG є відповідні приклади (про них нижче).

Плюси і мінуси. Один із плюсів, як і за природного розвитку, є реалістичність. Якщо вам дали нового покемона, то тепер у бою ви можете викликати цього покемона, а якщо вас навчили заклинання заморозки, то можете заморожувати ворогів. Логічно, що саме так, а чи не навпаки.

Інший плюс – стимул для гравців досліджувати світ. Дуже часто ми проходимо вздовж по сюжету, не згортаючи зі шляху, тому що все, що нас чекає осторонь (принаймні в сенсі геймплею) – це лише додаткова досвід для персонажу.

Однак, цей плюс може обернутися і мінусом – не всі гравці належать до категорії «дослідників». Багатьом з них просто хочеться чергового рівня крутості або чергової глави і вони зовсім не мають бажання досліджувати світ вздовж і

поперек. Саме тому сховані або одержувані під час виконання унікальних завдань бонуси найчастіше зроблені необов'язковими.

Було обрано систему прогресу персонажа типу «Свідомий вибір». Наприклад вона реалізована в грі «Dark Souls». За знищення ворогів гравець отримує певну кількість досвіду (бали досвіду) і може самостійно обирати яку з характеристик персонажа необхідно збільшити. Ця система прогресу є доволі популярною і зрозуміло.

2.2 Розробка структури інтерфейсу 2d-гри «Magical adventure»

При проєктуванні користувацького інтерфейсу потрібно враховувати низку факторів: колір не повинен заважати сприймати текст; інформативні елементи мають знаходися в полі зору та правильних місцях, їх колір немає зливатися з кольором фону, зокрема, функції, виклик яких необхідно забезпечити, параметри, які необхідно показати, а також безліч інших, важливих з точки зору зручності користувачу.

Розроблюваний ігровий додаток призначений для використання на персональних комп’ютерах, тому у користувача в наявності буде великого монітору, а також пристроїв вводу типу миша та клавіатура.

Для виконання проєктування користувацького інтерфейсу складено список необхідних сцен для відображення елементів інформативного та контролюючого характеру:

1. Сцена «Гра» відображує ігровий процес.
2. Сцена «Характеристики» відображає меню підвищення характеристик гравця.

На основі висунутих вимог розроблено схематичні зображення користувацького інтерфейсу ігрового додатку.

Схему сцени «Гра» наведено на рисунку 2.1.

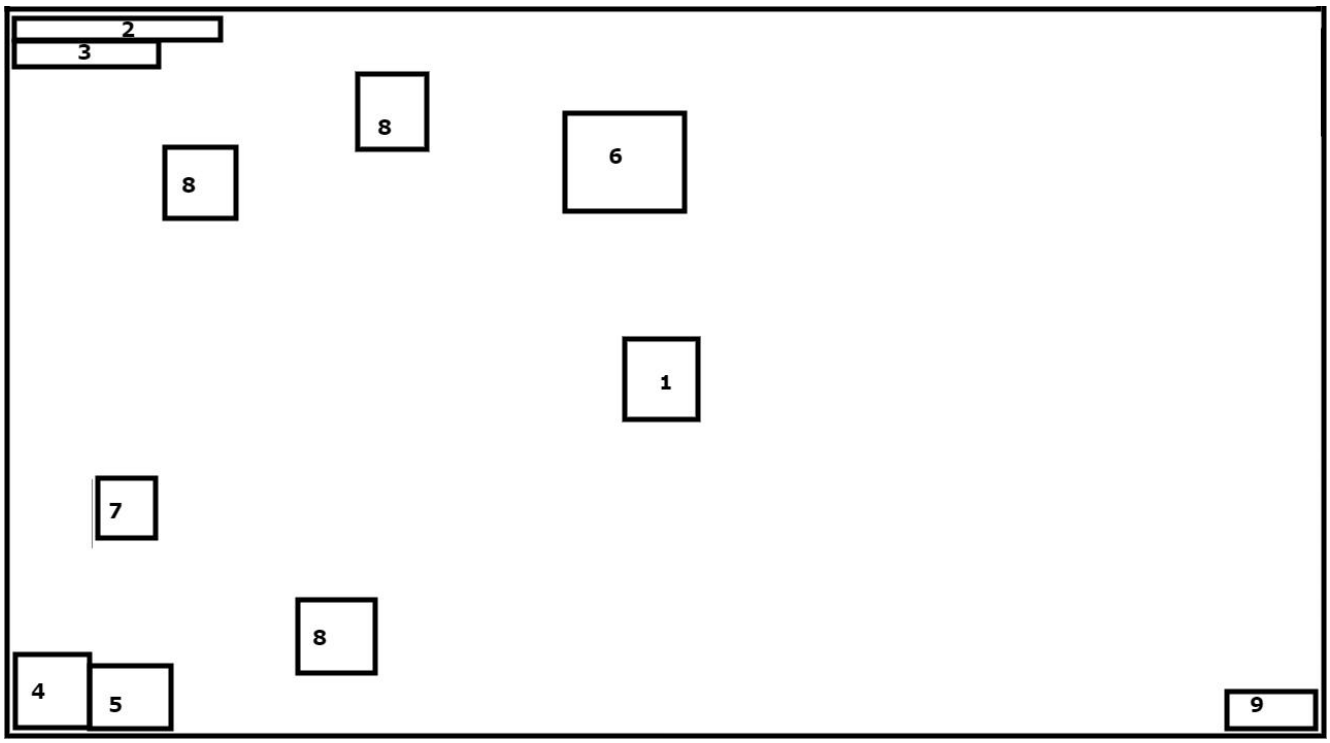

Рисунок 2.1 – Схема користувацького інтерфейсу сцени «Гра»

Опис елементів схеми відповідно до їх нумерації на рисунку 2.1:

1. Персонаж гравця в стартовій позиції, гравець може рухатись на Схід, Захід, Південь та Північ.
2. Шкала здоров’я персонажа, червоного кольору.
3. Шкала енергії персонажа, синього кольору.
4. Іконка відображення обраної зброї персонажа.
5. Іконка відображення обраного заклинання персонажа.
6. Перешкода (дерево) яку не можливо знищити.
7. Перешкода (кущ) яку можна знищити.
8. Вороги.
9. Іконка відображення поточної кількості досвіду.

Сцена «Характеристики» відображує вікно з п’ятьма стовпцями з повзунками на фоні сцени «Гра».

Схему сцени «Характеристики» наведено на рисунку 2.2.

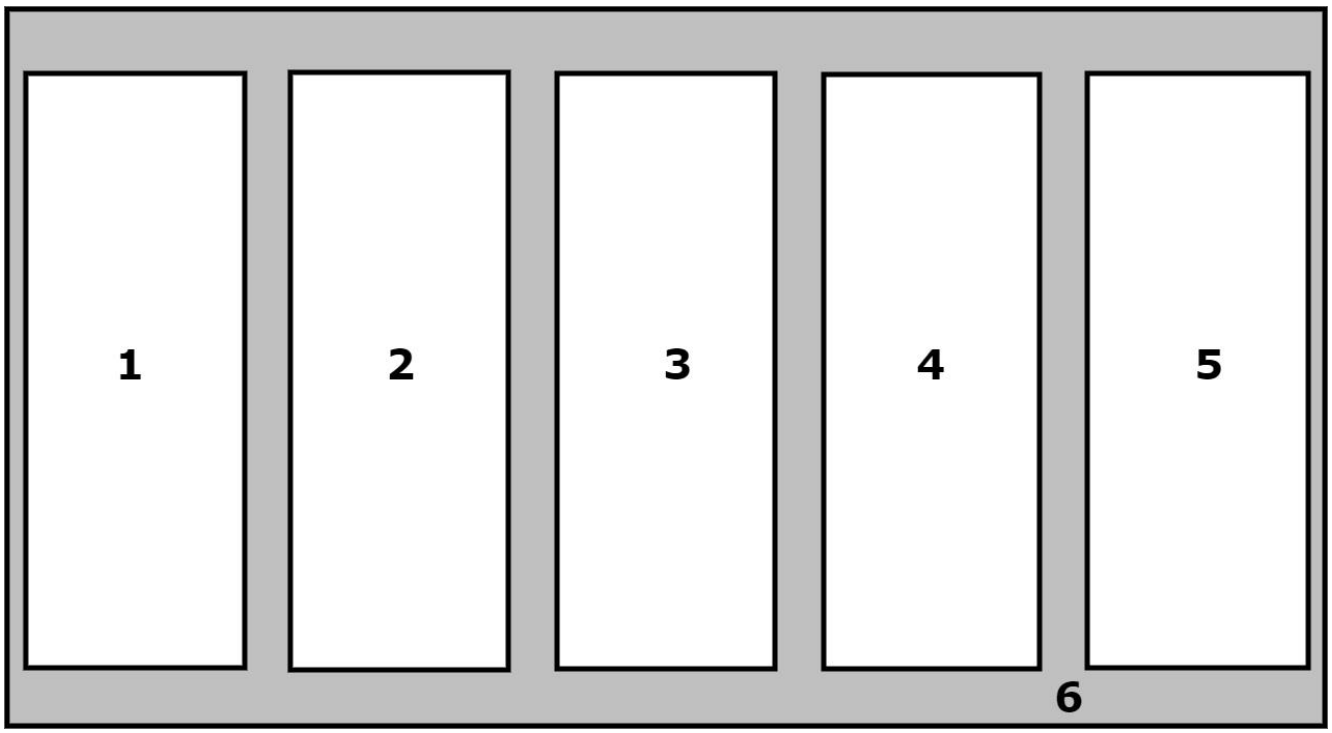

Рисунок 2.2 – Схема користувацького інтерфейсу сцени «Характеристики»

Меню відкривається на кнопку «M» клавіатури. Опис елементів схеми відповідно до їх нумерації на рисунку 2.3:

1. Блок з повзунком характеристики «HEALTH».
2. Блок з повзунком характеристики «ENERGY».
3. Блок з повзунком характеристики «ATTACK».
4. Блок з повзунком характеристики «MAGIC».
5. Блок з повзунком характеристики «SPEED».
6. Фон сцени «Гра».

Розробка графічного інтерфейсу веб-системи була проведена у середовищі розробки Pygame і Tiled – кросплатформний відкритий редактор тайтлових карт для ігор.

2.3 Розробка методу спільного руху гравець-ворог

Сутність цього методу полягає у тому, щоб створити швидко ворогів за допомогою методів, які використовує персонаж гравця, а саме метод переміщення і зіткнення. Це допоможе швидше наповнити ігрове середовище різноманітними ворогами, оскільки не буде необхідності витрачати час на створення окремих методів для них.

Спільне використання методів ворогом і гравцем можна назвати міні-шаблоном. За допомогою такого міні-шаблону можна прискорити створення 2d-ігор жанру екшн.

Запропонований метод включає послідовність дій:

1. Переміщення гравця реалізується за допомогою отримання вхідних даних від клавіатури, а переміщення ворога, в свою чергу, реалізується на вхідних даних від переміщення гравця. У ворога є радіус зору, якщо гравець буде в ньому знаходитися, ворог почне наближатися до гравця, якщо гравець є поза радіусом, ворог буде стояти на початковій позиції.

2. Якщо ворог буде знаходиться в притул до гравця, то відбудеться анімація атаки і гравець втратить частину шкали здоров’я.

3. Від ворога можливо втекти, якщо швидко вийти з радіусу його зору.

4. Для того, щоб ворог і гравець використовували методи переміщення і зіткнення, було створено файл «entity.py». В класі Entity були розміщенні методи «move» і «collision».

Взаємодія класів «Player» і «Enemy» з класом «Entity» продемонсторована на рисунку 2.3.

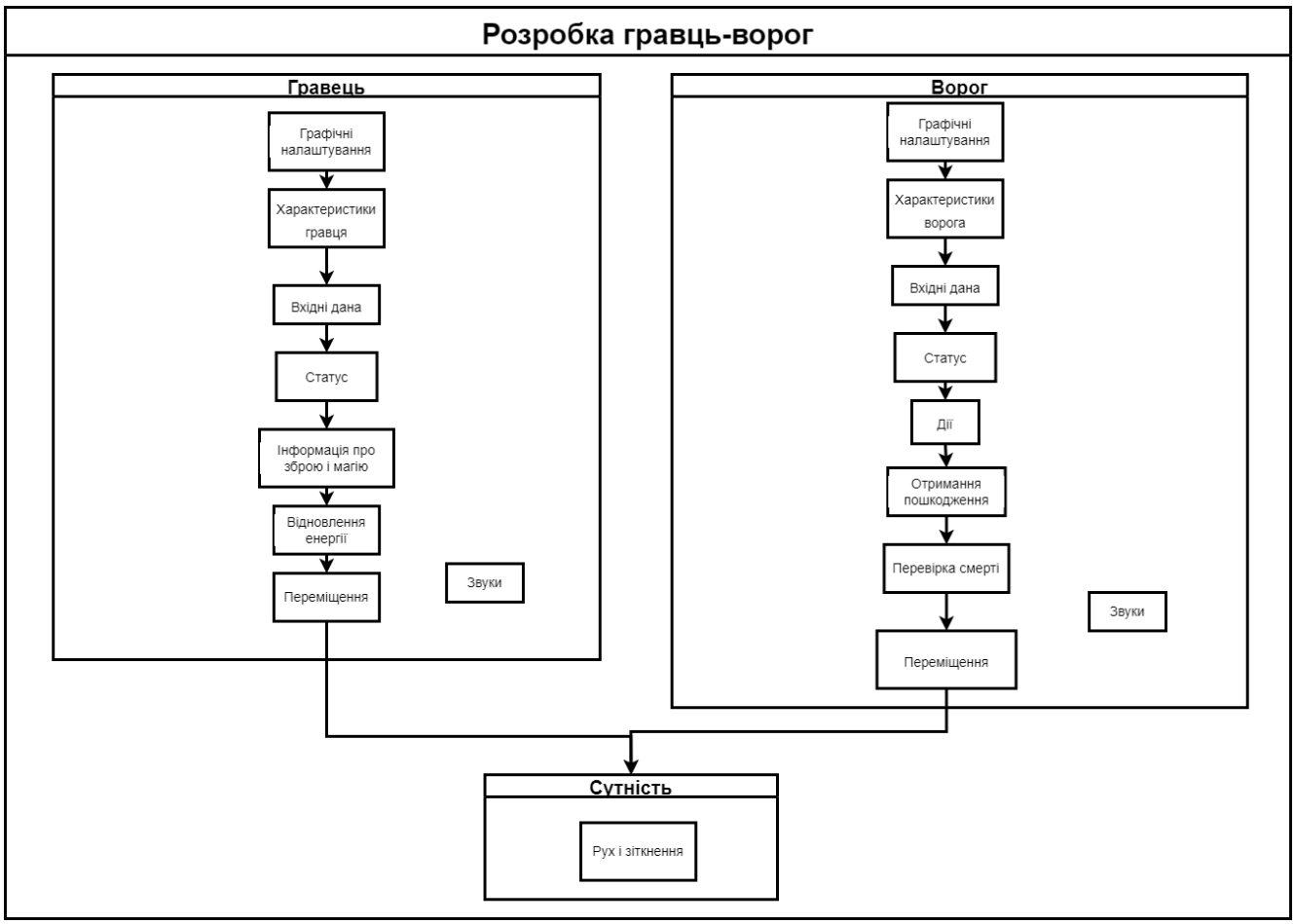

Рисунок 2.3 – Метод спільного руху гравець-ворог

2.4. Розробка моделі роботи ігрової системи

Для тренування тактичного мислення за допомогою 2d-гри жанру екшн-пригода розроблено модель ігрової системи (рис. 2.4). Ця модель передбачає, що гравець при зустрічі з ворогом має його перемогти за допомогою зброї чи магії, в той же час 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. За знищення ворога гравцю буде нараховуватися певна кількість балів досвіду. Ці бали гравець зможе витратити на покращення 5 базових характеристик персонажа, що допоможе в майбутньому знищувати більш небезпечних ворогів і зменшує ймовірність програшу.

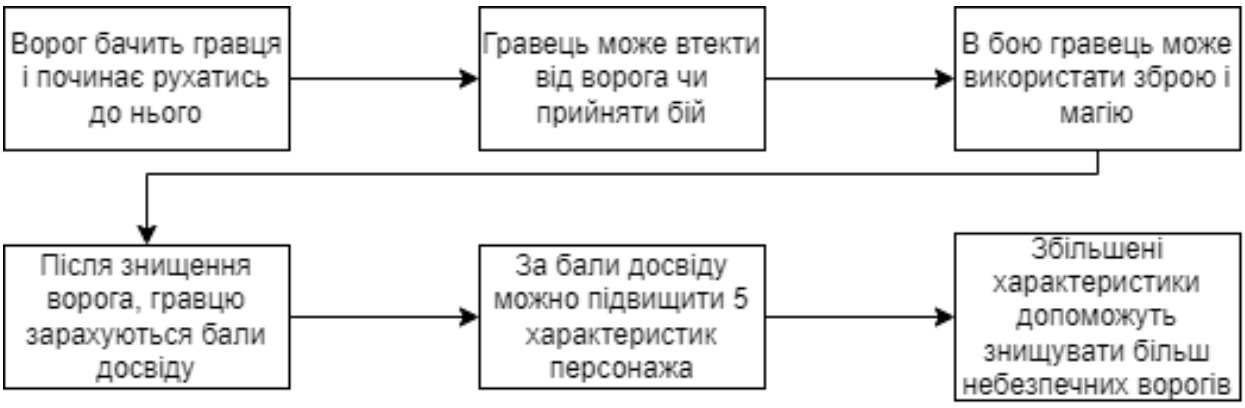

Рисунок 2.4 – Модель роботи ігрової системи

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, вплине на реіграбельність гри.

2.5 Розробка алгоритмів роботи 2d-гри «Magical adventure»

Алгоритмів роботи комп’ютерної 2d-гри «Magical adventure» починається із появи гравця в ігровій локації. Коли гравець почне досліджувати локацією, він обов’язкова зустріне ворогів різного виду. У гравця є можливість обрати зброю і змінювати її під час бою. Також гравець може знищити ворога за допомогою магії. В крайньому випадку гравець може втекти від ворога за рахунок своєї швидкості переміщення.

Якщо гравець переможе ворога, йому нарахуються бали досвіду. Гравець має вибір: збільшувати характеристики свого персонажу чи ні. Він має можливість збільшувати як одну характеристику, так і декілька одночасно, головне, щоб для цього у нього було достатньо балів досвіду.

Після цього гравець продовжуватиме досліджувати локацію, зіштовхуючись з більш важкими ворогами. Якщо гравець програє бій з ворогом, гра закінчить. Блок-схема алгоритму роботи 2d-гри «Magical adventure» зображена на рисунку 2.5.

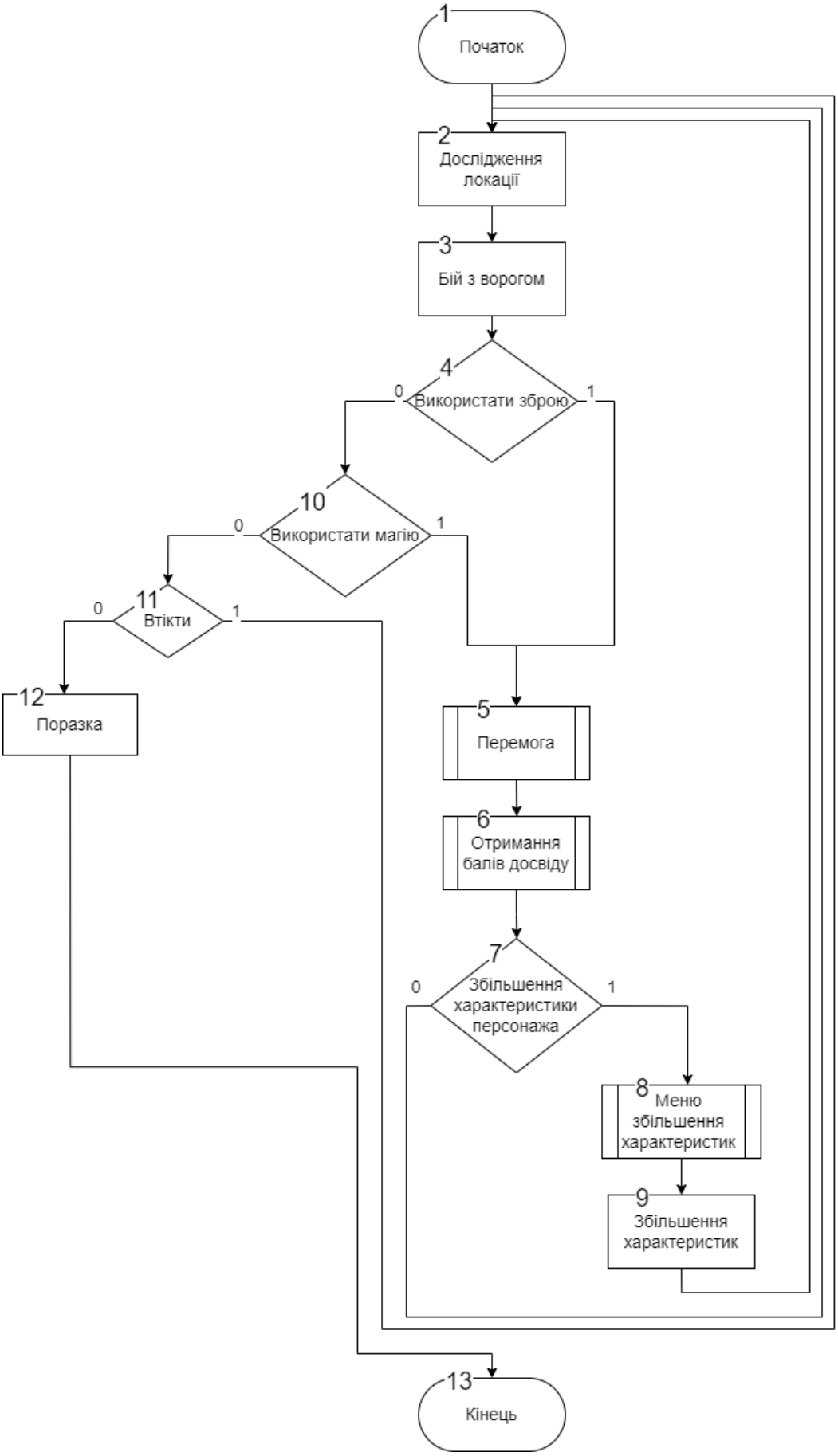

Рисунок 2.5 – Блок-схема загального алгоритму програми

Функціональна модель персонажу зображена на рисунку 2.6.

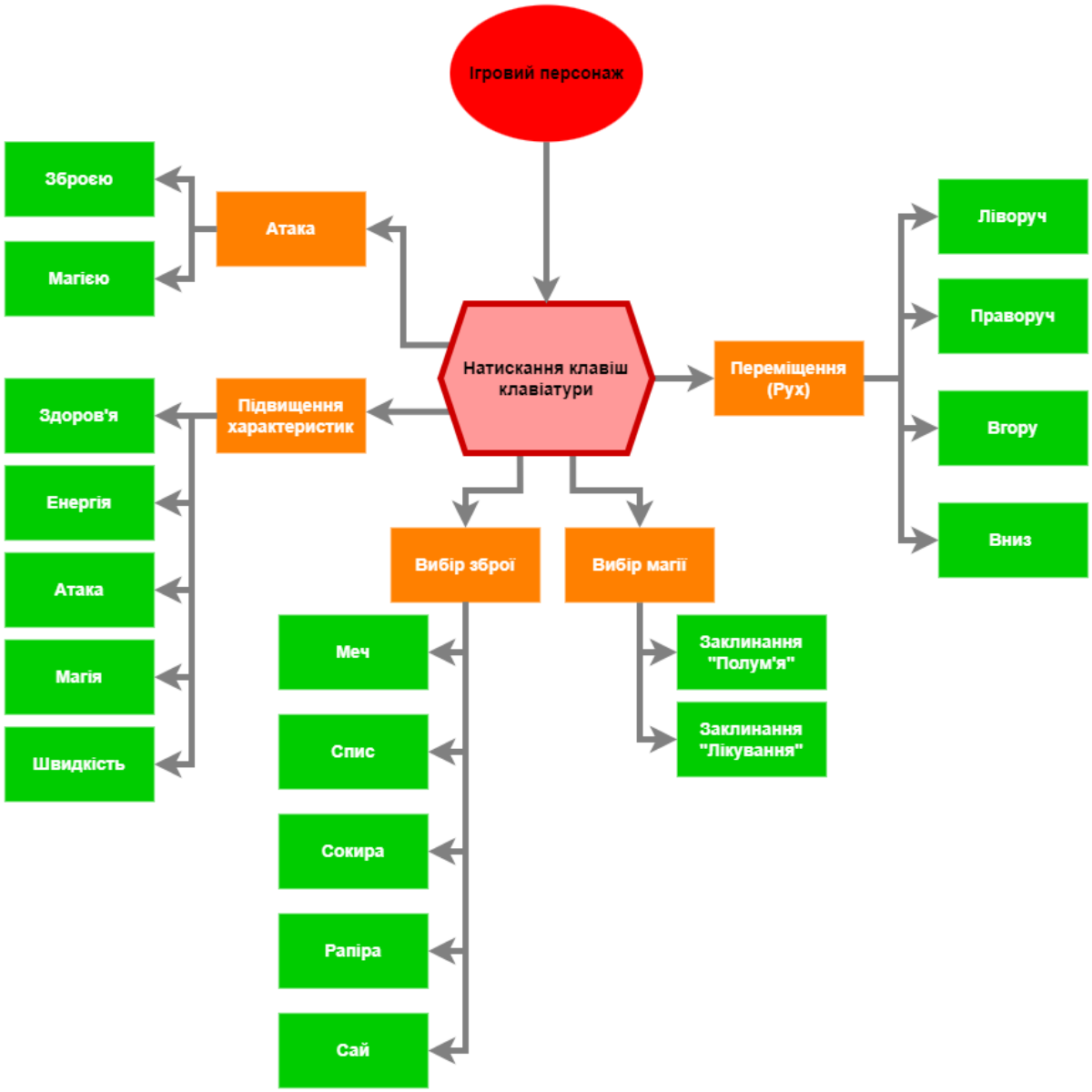

Рисунок 2.6 – Функціональна модель персонажу

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.

2.6 Висновки

У другому розділі розглянуто структуру інтерфейсу ігрового додатку. Розроблено метод роботи ігрового додатку з принципом спільного руху гравець-ворог, що використовує одні методи переміщення і зіткнення як для гравця так, і для ворога, за допомогою чого можна зменшити час створення нових ворогів і в цілому час розробки 2d-гри.

Подальшого розвитку дістала модель роботи ігрової системи, яка орієнтована на використання компонентів тактичного мислення за допомогою збільшення характеристик персонажа, що забезпечить розвиток різноманітності ігрового процесу і створить зручні умови тренінгу для користувача, орієнтовані на тренування тактичного мислення.

Розроблено блок-схему алгоритму роботи комп’ютерного ігрового додатку, та описано процес роботи програми, також реалізовано функціональну модель ігрового персонажу з переліком можливостей користувача.

3 РОЗРОБКА 2D-ГРИ «MAGICAL ADVENTURE»

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу

Над важливою частиною розробки програмного продукту є правильне обрання необхідних інструментів. А саме вибір між ігровим рушієм і фреймворком, та вибір мови програмування. Вибір має ґрунтуватися на масштабності проєкту і платформах на яких він має працювати, також немало важливим є характеристики ПК на якому програмний продукт буде розроблятися. Так як для використання потужних інструментів необхідні високі характеристики ПК (процесор, відеокарта, ОЗП і т.д.).

Python – динамічна інтерпретована об’єктно-орієнтована скриптова мова програмування із строгою динамічною типізацією [16]. Вона переваги:

1. Гнучкість – основна перевага мови.
2. Можливість розширення, існують бібліотеки і фреймворки під будь-який тип завдань і потреб.
3. Простота синтаксису. З синтаксису було прибрано все зайве, код чистий і зрозумілий без зайвих дужок і виразів.
4. Інтерпретованість. Інтерпретатор Python існує для всіх популярних платформ і за замовчуванням входить в більшість дистрибутивів Linux, а значить є на більшості серверів «з коробки».
5. PEP – єдиний стандарт для написання коду, що робить код підтримуваним і читабельним навіть при переході від одного програміста до іншого.
6. Open Source – код інтерпретатора Python є відкритим, що дозволяє будь-кому, хто зацікавлений у розвитку мови взяти участь в його розробці і поліпшити його.

7. Ком’юніті (спільнота) – навколо Python утворилося досить дружнє і приємне ком’юніті, яке готове прийти на допомогу будь-якому починаючому або вже вмілому розробнику і розібратися в його проблемі.

Виходячи з характеристик мови програмування і характеристик мого ноутбуку було обрано фреймворк, а не ігровий рушій, а саме фреймворк Pygame.

Pygame – це набір модулів Python, призначених для написання відеоігор. Pygame додає функціональність на додаток до чудової бібліотеки SDL. Це дозволяє створювати повнофункціональні ігри та мультимедійні програми на мові Python[17].

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. Код містить підтримку AmigaOS, Dreamcast, Atari, AIX, OSF/Tru64, RISC OS, SymbianOS та OS/2, але вони офіційно не підтримуються. Ви можете використовувати його на ручних пристроях, ігрових консолях та комп’ютері One Laptop Per Child (OLPC).

3. Він 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 все одно 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.

В якості IDE була обрана Visual Studio Code. Visual Studio Code – засіб для створення, редагування та зневадження сучасних веб-застосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X [18].

Visual Studio Code такі переваги:

1. Крос-платформний. Працює на Windows, Linux та macOS, так що ви можете працювати незалежно від платформи, на якій базується ваш пристрій;

2. Підтримує багато мов програмування. А саме Python, JavaScript, HTML, CSS, TypeScript, C++, Java, PHP, Go, C#, PHP, SQL, Ruby, Objective-C та багато інших.

3. Надає мовну документацію. Сайт містить документи, що стосуються спільних мов, які підтримує Visual Studio Code. Деякі з них - C++, C#, CSS, Go, Python, PHP, Java та багато іншого.

4. Налагодження. VSCode постачається з вбудованим налагоджувачем, який також є однією з його ключових функцій. Це допомагає прискорити цикл редагування, компіляції та налагодження будь-якого програміста.

5. IntelliSense. Це функція, яка використовується програмістами для інтелектуального завершення коду, інформації про параметри, контенту, швидкої інформації та натяків на код.

6. Палітра команд. Вона дозволяє отримати доступ до всіх функцій VS Code, включаючи всі ярлики ключових слів. Крім того, ця палітра також дозволяє отримати доступ до багатьох команд.

7. Функції керування кодом. Visual Studio Code також надає функції управління кодом, такі як Go to Definition, Peek Definition, Find all References і rename Symbol.

3.2 Розробка графічних матеріалів

Важливою частиною будь-якої гри є візуалізація ігрового процесу, локації, персонажа, ворогів тощо. Для розробки графічних зображень використовуються графічні редактори. Графічний редактор – це програма, призначена для створення та зміни графічного зображення на екрані комп'ютера, а також збереження у вигляді графічного файлу.

Графічний редактор, який має додаткові інтелектуальні засоби, називають графічним процесором. Такі програми дозволяють обробляти зображення за допомогою різноманітних графічних ефектів, перетворювати формат, палітру, масштаб, працювати з багатошаровими зображеннями, отримувати зображення зі сканера та іншої цифрової техніки тощо. Будь-який графічний редактор включає текстовий редактор і дозволяє набирати тексти [19].

Для розробки графічних матеріалів я обрав графічний редактор GIMP. GIMP – це кросплатформний-векторний редактор зображень, доступний для GNU/Linux, macOS, Windows та інших операційних систем[20].

Він має такі переваги:

– є вільним ПЗ;

– є високоякісним додатком для фоторетуші та дозволяє створювати оригінальні зображення;

– є високоякісним додатком для створення екранної та веб-графіки;

– є платформою для створення потужних та сучасних алгоритмів обробки графіки вченими та дизайнерами;

– дозволяє автоматизувати виконання повторюваних дій;

– легко розширюємо за рахунок простої встановлення доповнень.

У векторній графіці об'єкти створюються шляхом поєднання різних об'єктів. Прості об'єкти, такі як кола, лінії, сфери, куби тощо називається примітивами, і використовуються при створенні складніших об'єктів, тому необхідно вміти створювати різноманітні векторні об'єкти, щоб надалі редагувати їх, створюючи довільні композиції.

Графічні матеріали розробляються в програмах GIMP та Tiled. Tiled – це редактор 2D-рівень, який допомагає розробляти контент для гри. Його основною функцією є редагування тайтлових карт різних форм, але він також підтримує безкоштовне розміщення зображень, а також потужні способи анотувати створений рівень додатковою інформацією, що використовується грою. Tiled фокусується на загальній гнучкості, намагаючись залишатися інтуїтивно зрозумілим [21].

Що стосується тайлових карт, він підтримує прямі прямокутні тайлові шари, а також проектовані ізометричні, шахові ізометричні та шахові шестикутні шари. Набір тайлів може бути одним зображенням, що містить безліч тайлів, або набором окремих зображень. Для підтримки певних методів імітації глибини плитки та шари можуть бути зміщені на задану відстань, а порядок їхнього рендерингу може бути налаштований.

Основна сцена «Гра» складається з великою кількості графічних елементів. Першим створеним зображенням цієї сцени є Floor.png (набір тайтлів). Воно містить тайтли різної місцевості (лісна, пустельна та засніжена) води, мосту, стін, ворогів і мітки гравця.

Процес створення зображення Floor.png в програмі GIMP, наведено на рисунку 3.1.

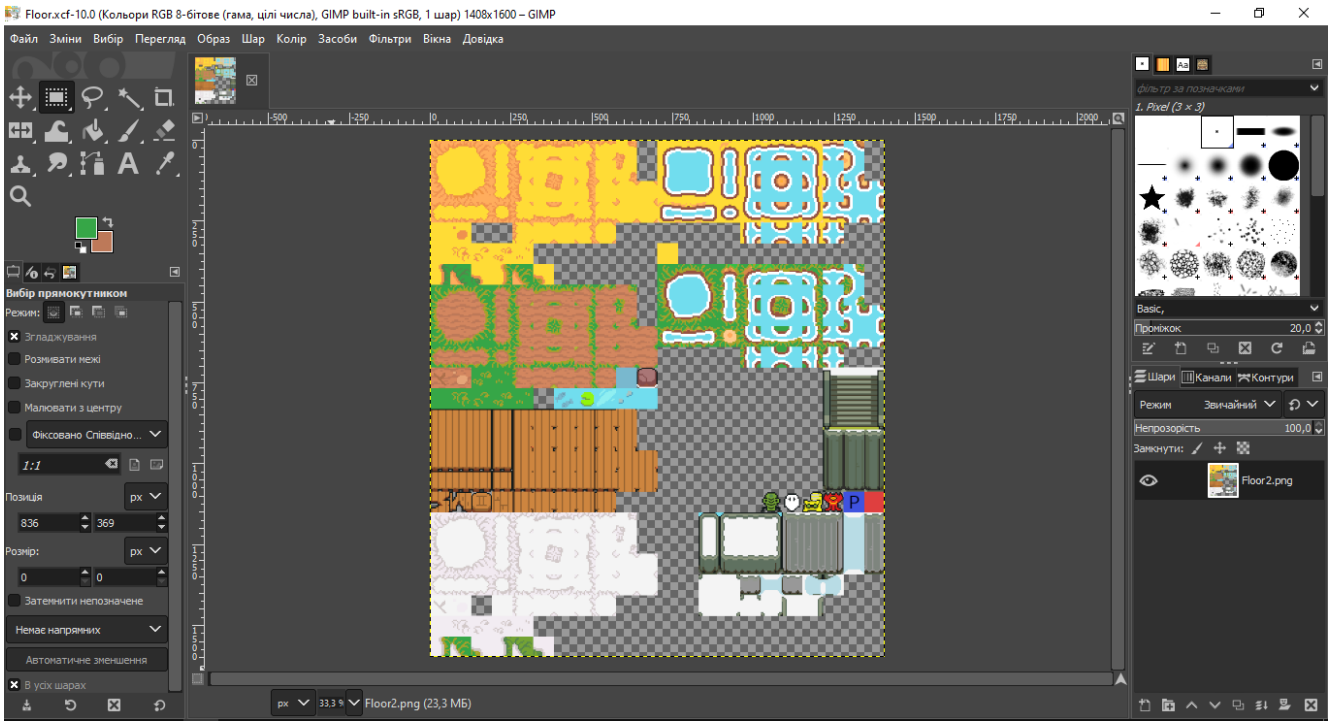

Рисунок 3.1 – Графічний елемент «Підлога»

Другим створеним зображенням було details.png (набір тайтлів), воно містить декоративні деталі такі як камінці, траву, невеликі квітки і т.п. Процес створення зображення details.png в програмі GIMP, наведено на рисунку 3.2.

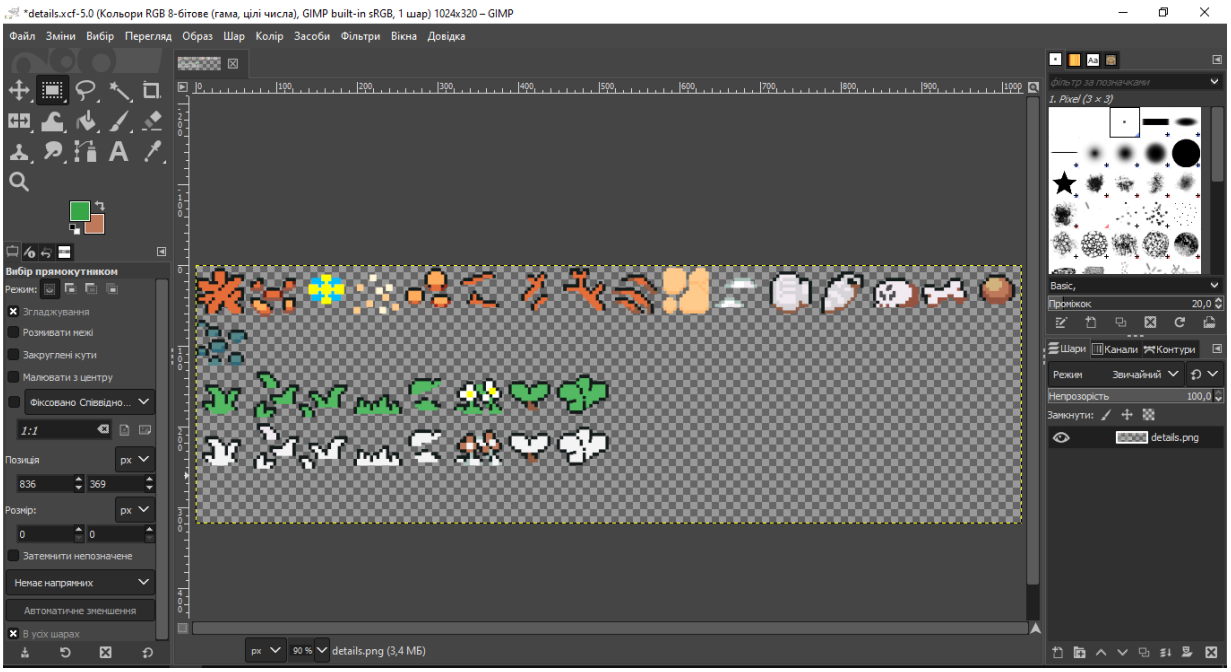

Рисунок 3.2 – Графічний елемент «Деталі»

За допомогою зображень Floor.png і details.png в програмі Tiled було створено головне фонове зображення – ground.png.

Перші два зображення є, так би мовити, пазлами (наборами тайлів), за допомогою яких можна швидко створити нове зображення.

Процес створення зображення details.png в програмі GIMP, наведено на рисунках 3.3-3.4.

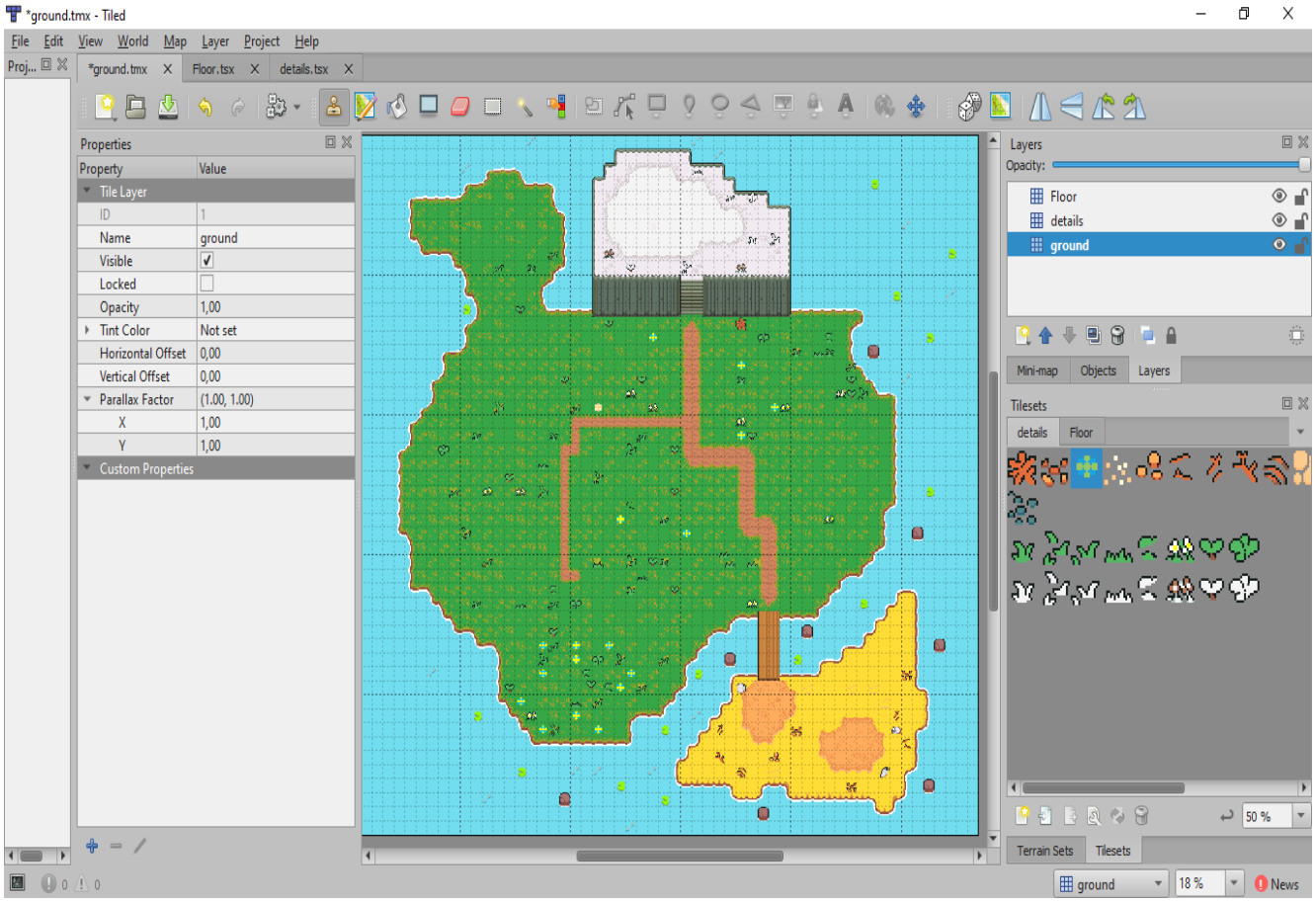

Рисунок 3.3 – Графічний елемент «Земля»

Зображення головного героя було розроблено в GIMP, також розроблені зображення в 4 проекціях (ліворуч, праворуч, верз, вниз) для анімування персонажа.

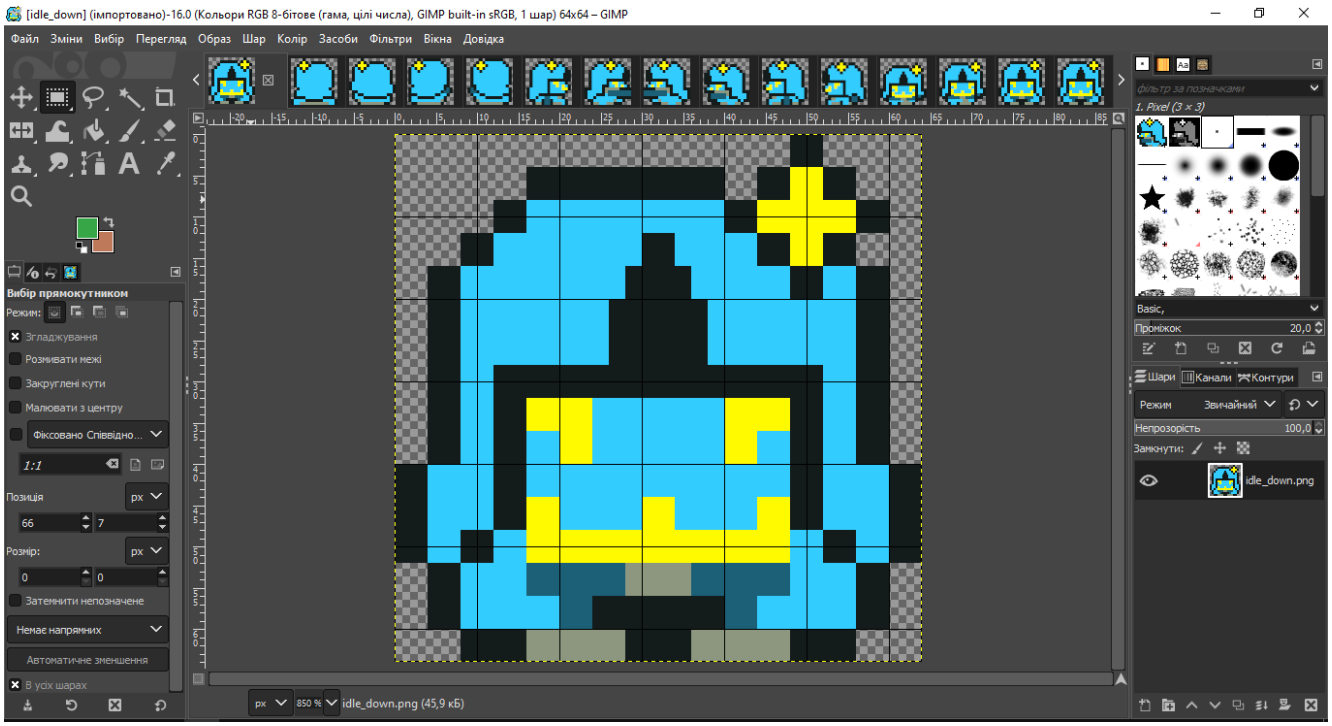

Рисунок 3.4 – Група графічних елементів «Персонаж»

Ворогами гравця будуть 4 ворога (монстри), а саме bamboo, raccoon, spirit та squid. Процес створення зображення ворогів персонажа наведено на рисунку 3.5.

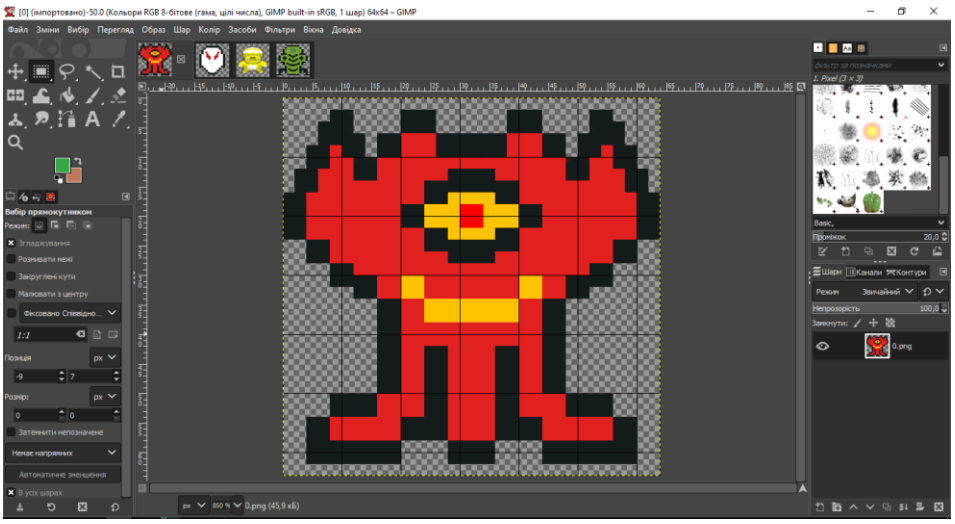

Рисунок 3.5 – Група графічних елементів «Вороги»

Окрім декоративних елементів, які є частинами головного фону сцени «Гра» і не будуть впливати на переміщення гравця, були розроблені перешкоди за допомогою векторних об’єктів в декількох варіантів залежно від місцевості

ігрової локації (ліс, пустеля, засніжена область). Через які гравець не зможе пройти, такі як: дерева, великий камінь, статуї, колони і т.п.

Процес створення зображення перешкод наведено на рисунку 3.6.

Рисунок 3.6 – Група графічних елементів «Об’єкти»

У персонажа на вибір буде 5 одиниць різної зброї для того щоб ввести бій, а саме меч, спис, сокира, рапіра та сай. Використовуючи векторні об’єкти розроблялось зображення повного вигляду зброї для іконки інтерфейсу, та 4 його проекції за напрямком нанесення удару для анімації удару. Процес створення зображення зброї наведено на рисунку 3.7

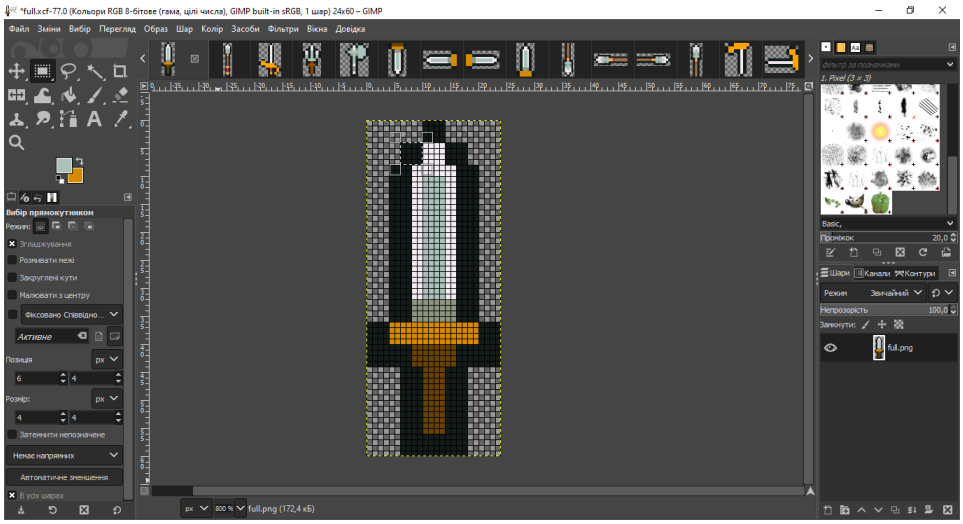

Рисунок 3.7 – Група графічних елементів «Зброя»

Крім зброї гравець може використовувати магію, а саме заклинання «Полум’я» та «Лікування». За допомогою векторного об’єкта «Коло» було створено зображення полум’я і аура лікування для іконок інтерфейсу і група зображення для анімування використання цих заклинань. Процес створення зображення зброї наведено на рисунках 3.8 і 3.9.

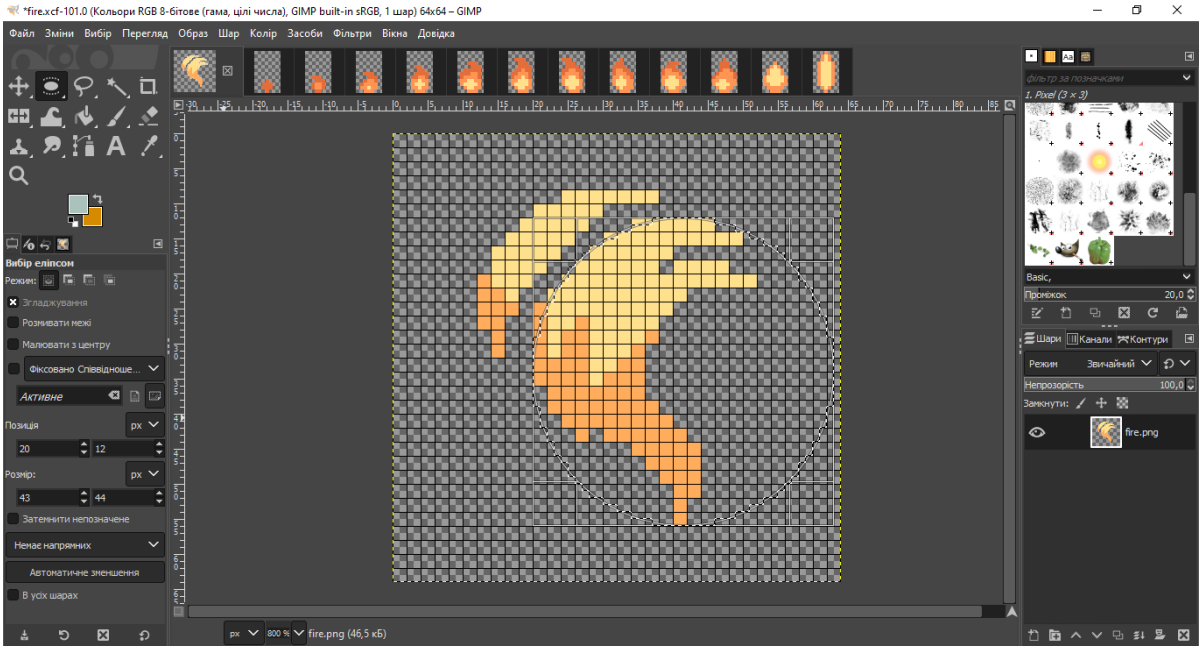

Рисунок 3.8 – Група графічних елементів «Полум’я»

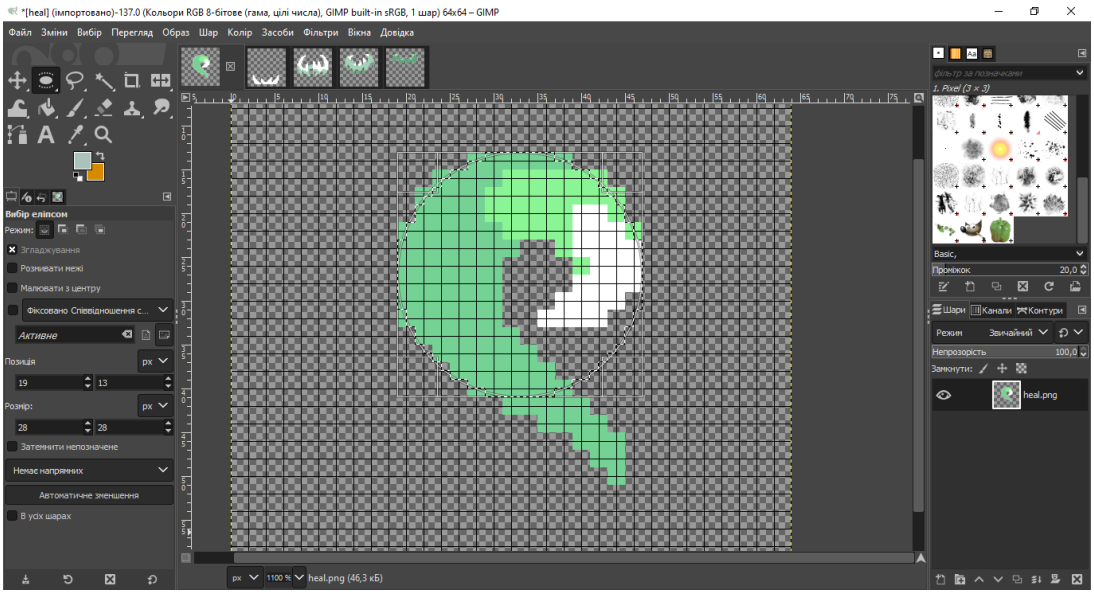

Рисунок 3.9 – Група графічних елементів «Лікування»

Також було розроблено графічні ефекти об’єктів, такі як знищення ворогів та їх атак і т.п. за допомогою векторних об’єктів «Коло» та «Квадрат». Щоб наситити графічну складову гри, чим більше деталей та ефектів тим цікавіше спостерігати за тим що відбувається на моніторі під час гри (рис. 3.10).

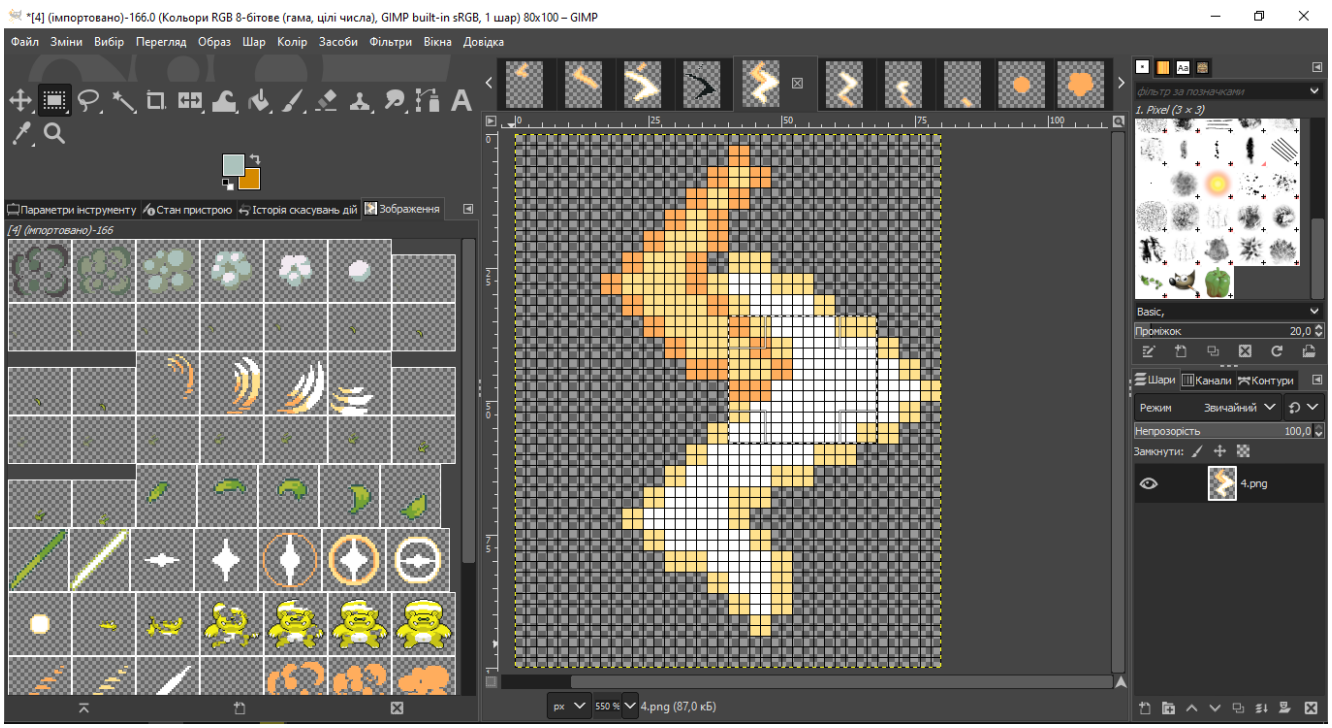

Рисунок 3.10 – Група графічних елементів «Частинок»

Після цього необхідно зберегти графічні елементи у вигляді, придатному для використання в ігровій системі. Для Pygame це графічні формати PNG. Формат PNG дозволяє зберегти прозорість, тому саме він буде використаний у рамках розробки графічних елементів гри «Magical adventure».

3.3 Розробка основних модулів додатку

Основою (фундаментом) гри є її рівень (ігрова локація). Розробка гри починається саме з неї. Так як клас level.py буде контейнером для всього що буде в грі (персонажа, ворогів, об’єктів). Частина коду класа Level зображено на рис. 3.11.

```python
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

class Level:
    def __init__(self):

        # отримати поверхню дисплея
        self.display_surface = pygame.display.get_surface()
        self.game_paused = False

        # налаштування групи спрайтів
        self.visible_sprites = YSortCameraGroup()
        self.obstacle_sprites = pygame.sprite.Group()

        # спрайти атаки
        self.current_attack = None
        self.attack_sprites = pygame.sprite.Group()
        self.attackable_sprites = pygame.sprite.Group()

        # налаштування спрайту
        self.create_map()

        # інтерфейс користувача
        self.ui = UI()
        self.upgrade = Upgrade(self.player)

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

Рисунок 3.11 – Частина коду загальних налаштувань класу Level

Створення карти відбувається за рахунок макетів створених в програмі Tiled. В кінці роботи в програмі отримано файли map_FloorBlocks.csv, map_Grass.csv, map_Objects.csv та map_Entities.csv. Ці файли містять інформацію де який об’єкт має знаходитись на карті. Скрип в текстовому вигляді наведений на рис. 3.12.

```python
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')
    }
```

Рисунок 3.12 – Частина коду функції create_map

Імпорт графіки, з папок з необхідними графічними матеріалами зображений на рисунку 3.13

```
graphics = {
    'grass': import_folder('../graphics/Grass'),
    'objects': import_folder('../graphics/objects')
}
```

Рисунок 3.13 – Частина коду імпорту налаштування класу Level

Експорт поверхні гравця і монстрів за рахунок їх індексу який їм надала програма Tiled. Це пришвидшує роботу по наповненню карти об’єктами. Частину її коду наведено на рисунку 3.14:

```
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)
```

Рисунок 3.14 – Частина коду імпорту поверхонь гравця і монстрів за рахунок індексів

Також в класі level.py створено функцію камери, яка буде слідкувати за персонажем гравця під час його переміщення по карті. Частина коду її наведена на рисунку 3.15.

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

        # загальне налаштування
        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()

        # створення підлоги
        self.floor_surf = pygame.image.load('../graphics/tilemap/ground.png').convert()
        self.floor_rect = self.floor_surf.get_rect(topleft = (0,0))
```

Рисунок 3.15 – Частина коду функції камери

Створена функція пошкодження гравця в класі level.py. Частина коду її наведена на рисунку 3.16.

```
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_sprites])
```

Рисунок 3.16 – Частина коду функції пошкодження гравця

Створена функція руху гравця і ворогів, частина її коду наведена на рисунку 3.17.

```
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()
```

Рисунок 3.17 – Частина коду функції пошкодження гравця

Визначення функцію створення атаки і магія в якості заклинань двох заклинань.

Частина коду створення атаки і магії наведена на рисунку 3.18.

```
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])
```

Рисунок 3.18 – Частина коду функції створення атаки і магії

Для створення гравця та його можливостей було створено клас player.py.

На сам перед графіка ініціалізація зображення персонажу і регулювання його розмірів.

Частина коду, що реалізує загальні налаштування персонажа гравця, наведена на рисунку 3.19.

Визначена функція введення даних про переміщення персонажа гравця за допомогою натискань клавіш «Left», «Right», «Up» та «Down» (клавіші з символом ←, →,↑ і ↓).

Частина її коду наведена на рисунку 3.20.

Визначена функція анімування персонажа.

Частина її коду наведена на рисунку 3.21

```python
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)
```

Рисунок 3.19 – Загальні налаштування класу player

```
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
```

Рисунок 3.20 – Частина коду функції введення даних за допомогою клавіш «Left», «Right», «Up» та «Down»

```
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)
```

Рисунок 3.21 – Частина коду функції анімування персонажа

Визначена функція автоматичного відновлення енергії персонажа, яку той може використати для застування заклинань. Частина її коду наведена на рис. 3.22.

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

Рисунок 3.22 – Частина коду функції автоматичного відновлення енергії персонажа

Визначена функція імпорту активів персонажа, а саме графічні матеріали його руху та атаки в 4 сторони. Частина її коду наведена на рисунку 3.23.

```
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)
```

Рисунок 3.23 – Частина коду функції імпорту графічних матеріалів персонажа

Визначення функції перевірки стасу гравця має перевіряти чи гравець рухається, чи стоїть на місці, чи він атакує.

Частина її коду наведена на рисунку 3.24.

```
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','')
```

Рисунок 3.24 – Частина коду функції отримання стану персонажа

Визначено функції оновлення, оновлення даних функцій вхідних даних, перезарядки, отримання статусу, анімування, переміщення та відновлення енергії. Частина коду наведена на рисунку 3.25.

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

Рисунок 3.25 – Частина коду функції оновлення

Визначення функцій отримання повного значення пошкодження яке може нанести зброя і магія. Частина її коду наведена на рисунках 3.26.

```
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
```

Рисунок 3.26 – Частина коду функції отримання повного пошкодження від зброї та магії

Для створення ворогів було виділено окремий клас Enemy (файл enemy.py). Код загальних налаштувань класу Enemy наведений на рисунку 3.27.

```
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)
```

Рисунок 3.27 – Загальні налаштування класу Enemy

Визначено функцію імпорту графіки ворогів трьох станів: бездіяльність, переміщення та атака з відповідної папки. Код імпорту графіки ворогів наведений на рисунку 3.28.

```python
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)
```

Рисунок 3.28 – Код імпорту графіки ворогів

Визначено функцію отримання напрямку відстані від гравця. Це необхідно для реалізації алгоритму руху ворога до гравця. Код отримання напрямку відстані від гравця наведений на рисунку 3.29.

```python
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)
```

Рисунок 3.29 – Код отримання напрямку відстані від гравця

Визначено функцію дій ворога в залежності від того як далеко від нього знаходиться гравець. Код функції дій ворога по відношенню до гравця наведено на рисунку 3.30.

```python
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()
```

Рисунок 3.30 – Код функції дій ворога по відношенню до гравця

Для реалізації і відображення ігрового інтерфейсу користувача було створено клас UI (ui.py). Основними елементами інтерфейсу є шкала здоров’я і енергії, іконок зброї і заклинань, та відображення кількості балів досвіду. Код загальних налаштувань класу наведений на рисунку 3.31.

```python
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)
```

Рисунок 3.31 – Код загальних налаштувань класу

Налаштування статус бара було визначено функцію show_bar, це необхідно для майбутньої реалізації відображення інформації про рівень здоров’я та енергії гравця. Частина коду функції налаштування статус бара наведена на рис.3.32

```python
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)
```

Рисунок 3.32 – Частина коду функції show_bar

Також необхідно що гравець бачив кількість балів досвіду яку він отримав за вбивство ворогів. Код відображення отриманих балів досвіду гравцем наведений на рисунку 3.33.

```
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)
```

Рисунок 3.33 – Код відображення отриманих балів досвіду гравцем

Необхідно реалізувати відображення обрано на цей момент зброї і заклинання гравцем. Код функції відображення обраної зброї і заклинання наведений на рисунку 3.34.

```
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)
```

Рисунок 3.34 – Код функції відображення обраної зброї і заклинання

Так як гравець і ворог кардинально не відрізняються було вирішено що вони можуть використовувати спільно дві функції, а саме функцію руху і зіткнення. Функції створенні за допомогою бібліотеки math, яка дозволяє працювати арифметичних та тригонометричних операцій у Python. Ці дві функції (переміщення і зіктнення) знаходяться в класі Entity (entity.py), в класах player.py та enemy.py знаходяться лише загальні налаштування руху. Код функції переміщення і зіткнення наведений на рисунку 3.34.

```
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
```

Рисунок 3.34 – Код функції переміщення і зіткнення

Меню збільшення характеристик персонажа є однією із головніших частиною моделі взаємодії гравець-ворог. Гравцеві необхідно отримати досвід за вбивство ворога. Гравець за рахунок цього досвіду може покращити характеристики персонажа, зі збільшенням значення характеристик кількість балів досвіду зменшується (певна кількість витрачається). Після цього має відбуватися оновлення графічного інтерфейсу. Меню збільшення реалізовано в класі Upgrade (upgrade.py).

Обрахунок вартості і величини на відбудеться збільшення характеристик гравця визначений в функції trigger, її код наведений на рисунку 3.35.

```
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)
```

Рисунок 3.35 – Код функції trigger обрахунку витрат і збільшення характеристик

Весь лістинг ігрового додатку наведений в додатку В

3.4 Висновки

У третьому розділі було проведено аналіз і обґрунтування вибору засобів для реалізації програмного засобу. Мовою програмування обрано Python, фреймворк Pygame в якості набору інструментів для розробки ігор, IDE – Visual Studio Code. Для розробки графічних матеріалів графічний редактор GIMP та плиточний редактор рівнів Tiled для експорту файлів .csv які містять дані про находження всіх об’єктів на карті (гравець, ворог, дерева і т.д.).

Було розроблено необхідну кількість графічних матеріалів різних видів, а саме:

– зображення персонажа гравця з 4 сторін і зображення для анімування атаки в 4 сторін та переміщення;

– зображення ворогів (монстрів) і зображення для анімування їх переміщення;

– зображення зброї гравця з 4 сторін;

– зображення заклинань та зображення для анімування їх застосування;

– зображення невеликих деталей які не впливають на переміщення гравця, щоб наповнити карту;

– зображення об’єктів, а саме дерева, статуї, колони, кущи та каміння;

– зображення ефектів атаки ворогів та їх смерті.

Розроблені основні модулі додатку, а саме:

– level.py є контейнером для всього що знаходиться на карті (гравець, ворог, об’єкти і т.д.)

– player.py реалізовані характеристики гравця, анімація персонажа, налаштування його руху, його статус, введення даних з клавіатури для переміщення і т.д.

– enemy.py реалізовані анімація ворога, перевірка його смерті, отримання пошкодження від гравця, дії, отримання дистанції до гравця і т.д

– ui.py реалізовано ігровий інтерфейс гравця, а саме шкалу індикації стану здоров’я та енергії, іконки зброї та заклинань, і відображення кількості балів досвіду.

Також розроблено спільний рух гравець-ворог. За рахунок того що персонаж гравця і ворог не дуже відрізняються, було реалізовано спільне використання функцій переміщення і зіткнення. Ці методи реалізовані в файлі entity.py, гравець і ворог використовуються їх за рахунок імпорту.

Було розроблено меню збільшення характеристик гравця в файлі (класі Upgrade) upgrade.py. Створено графічні відображення кожної характеристики персонажа засобами Pygame. Реалізована система отримання гравцем балів досвіду після знищення ворога і система обміну їх на підвищення 5 характеристик: «Здоров’я», «Енергія», «Атака», «Магія», «Швидкість». В цілому така модель сприяє розвитку тактичного мислення і появі бажання у гравця зіграти ще раз в гру та спробувати іншу комбінацію збільшення характеристик.

4 ТЕСТУВАННЯ ІГРОВОГО ДОДАТКУ

4.1 Вибір методів тестування програмного забезпечення

Тестування ігор, підмножина розробки ігор, є процесом тестування програмного забезпечення контролю якості відеоігор. Основна функція тестування ігор – виявлення та документування дефектів програмного забезпечення. Тестування програмного забезпечення для інтерактивних розваг – це високотехнологічна область, що вимагає комп'ютерних знань, аналітичної компетентності, навичок критичної оцінки та витривалості. [22]

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.

Тестування чорної скриньки. Тестування без будь-яких знань про внутрішню роботу програми називається тестуванням чорної скриньки. Тестувальник не звертає уваги на архітектуру системи та не має доступу до вихідного коду. Як правило, при виконанні тесту чорної скриньки тестувальник буде взаємодіяти з інтерфейсом системи, надаючи вхідні дані і перевіряючи вихідні дані, не знаючи, як і де обробляються вхідні дані [23].

У таблиці 4.1 перераховані переваги та недоліки тестування методом «чорної скриньки».

Таблиця 4.1 – Переваги і недоліки методики тестування «Чорної скриньки»

Переваги	Недоліки
Добре підходить і ефективний для великих сегментів коду.	Обмежене покриття, оскільки фактично виконується лише вибрана кількість тестових сценаріїв.
Доступ до коду не потрібен.	Неефективне тестування через те, що тестувальник має лише обмежені знання про програму.
Чітко відокремлює точки зору користувача від точки зору розробника завдяки чітко визначеним ролям.	Сліпе покриття, оскільки тестер не може націлюватися на конкретні сегменти коду або зони, схильні до помилок.
Велика кількість тестувальників середньої кваліфікації можуть протестувати програму, не знаючи реалізації, мови програмування чи операційних систем.	Тестові приклади важко спроектувати.

Тестування білої скриньки – це детальне дослідження внутрішньої логіки та структури коду. Тестування білої скриньки також називається тестуванням скла

або тестуванням відкритої коробки. Для того, щоб провести тестування додатка за білою скринькою, тестувальник повинен знати внутрішню роботу коду.

Тестеру потрібно зазирнути у вихідний код і з’ясувати, яка одиниця/частина коду поводиться неадекватно.

У таблиці 4.2 наведено переваги та недоліки тестування «білої скриньки».

Таблиця 4.2 – Переваги і недоліки методики тестування «білої скриньки»

Переваги	Недоліки
Оскільки тестер знає вихідний код, дуже легко з'ясувати, який тип даних може допомогти в ефективному тестуванні програми.	Через те, що для виконання тестування білої скриньки потрібен досвідчений тестер, витрати збільшуються.
Це допомагає оптимізувати код.	Певні часи це є неможливим для того, щоб зробити всі торти і кинути до вичерпаних помилок, які можуть створювати проблеми, як багато способів, які йдуть.
Можна видалити зайві рядки коду, які можуть призвести до прихованих дефектів.	Підтримувати тестування «білої скриньки» складно, тому що для цього потрібні спеціалізовані інструменти, такі як аналізатори коду та засоби налагодження.
Завдяки знанню тестувальником коду максимальне охоплення досягається при написанні тестового сценарію.	.

Порівняючи переваги і недоліки «чорної скриньки» та «білої скриньки» та з урахуванням мети програмного додатки, обрано метод «чорної скриньки» для тестування програмного забезпечення.

4.2 Тестування розробленого додатку

Найперше і найголовніше що необхідно протестувати це запуск програми. Після запуску файла main.py запускається програма.

Очікуваний результат. Створення вікна яке буде відображати гравця (персонажа), певна ділянка карти зі усіма об’єктами і почне відтворюватися звукова тема гри. Результат запуску програми відображений на рисунку 4.1.

Рисунок 4.1 – Початкова сцена гри після запуска

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

Очікуваний результат. Персонаж змінить свою позицію на локації. Результат переміщення відображений на рисунку 4.2.

Рисунок 4.2 – Персонаж перемістився після натискання необхідних клавіш

Після вдалого тесту переміщення гравця. Необхідно протестувати переміщення ворогів до гравця, якщо гравець знаходиться недалеко від ворога.

Очікуваний результат. Ворог має йти на зближення до гравця і зменшити дистанцію з ним. Результат переміщення ворогів наведений на рисунку 4.3.

Рисунок 4.3 – Ворог перемістився до гравця

Для знищення ворогів гравець може використати зброю натискаючи клавішу «Space».

Очікуваний результат. Анімація атаки мечем та знищення ворога. Ворог зникає змінюючи і зображення змінюється на знищину версію ворога певного виду його зникнення з зображенням. Результат переміщення ворогів наведений на рисунку 4.4.

Рисунок 4.4 – Результат гравець атакує і знищує ворога

Тест пройдено вдало. Ворог при атаці має наносити пошкодження, що має зменшувати шкалу здоров’я.

Очікуваний результат. Після атаки ворога шкала здоров’я (червона лінія) має зменшитись на певну частину.

Результат переміщення ворогів наведений на рисунку 4.5.

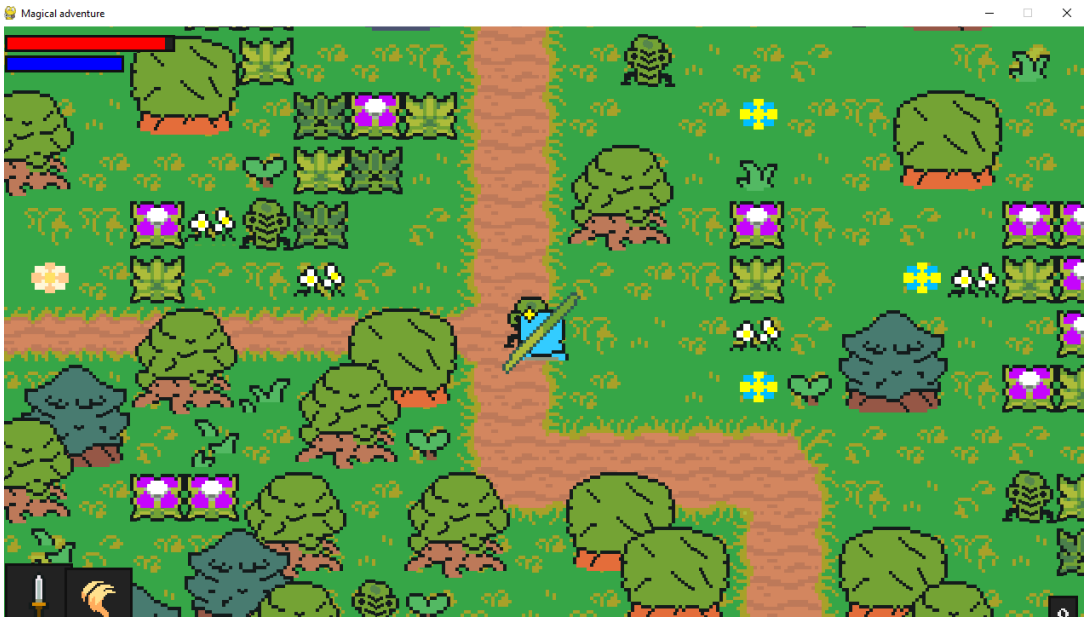

Рисунок 4.5 – Результат ворог наносить атаку по гравцю шкала здоров’я зменшується

Для відновлення здоров’я гравець може застосувати заклинання «Лікування» натискаючи на клавішу «Ctrl».

Очікуваний результат. При застосуванні заклинання «Лікування» цьому шкала енергії буде зменшуватися, а шкала здоров’я збільшуватись з анімацією використання заклинання. Результат відновлення здоров’я наведений на рис. 4.6.

Рисунок 4.6 – Результат відновлення здоров’я, за допомогою заклинання «Лікування»

Для знищення ворога гравець може використати заклинання «Полум’я» натискаючи клавішу «Ctrl», воно наносить пошкодження і має анімацію застосування, за використання заклинання витрачається енергія, шкала енергії зменшується.

Очікуваний результат при застосуванні заклинання «Полум’я» відбувається анімація і наноситься пошкодження ворогу якщо він під нього потрапляє, і за використання має зменшуватись шкала енергії. Результат відновлення здоров’я наведений на рисунку 4.7.

Рисунок 4.7 – Результат застосування «Полум’я» для знищення ворога

Гравець за вбивство ворога отримує бали досвіду які він може використати для збільшення характеристик.

Очікуваний результат. За вбивство ворога «Бамбук» гравець отримає 120 балів здоров’я.

Результат відновлення здоров’я наведений на рисунку 4.8.

Рисунок 4.8 – Результат нарахування балів досвіду за вбивство ворога «Бамбук»

Для збільшення характеристик персонажа є спеціальне меню яке викликається клавішою «M» (англійського алфавіту).

Очікуваний результат. На натискання клавіши «M» відображається спеціальне меню збільшення характеристик, а саме 5 блоків кожне зі свої заголовком «HEALTH», «ENERGY», «ATTACK», «MAGIC» і «SPEED».

Результат відновлення здоров’я наведений на рисунку 4.9.

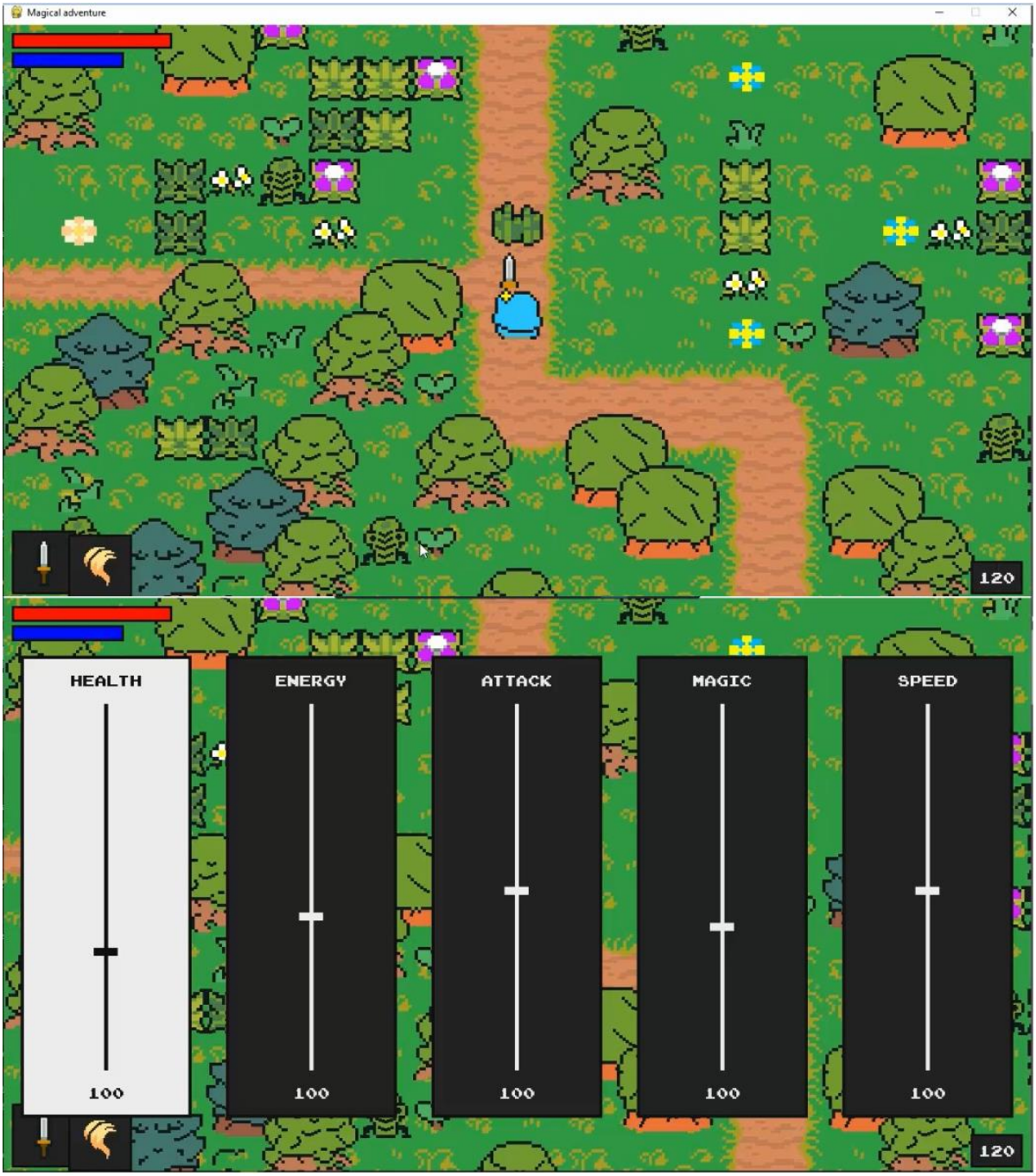

Рисунок 4.9 – Результат нарахування балів досвіду за вбивство ворога «Бамбук»

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

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

Значення лічильника балів досвіду має бути 40 (100 балів за к після двох збільшень характеристик. Результат тесту наведений на рисунку 4.10.

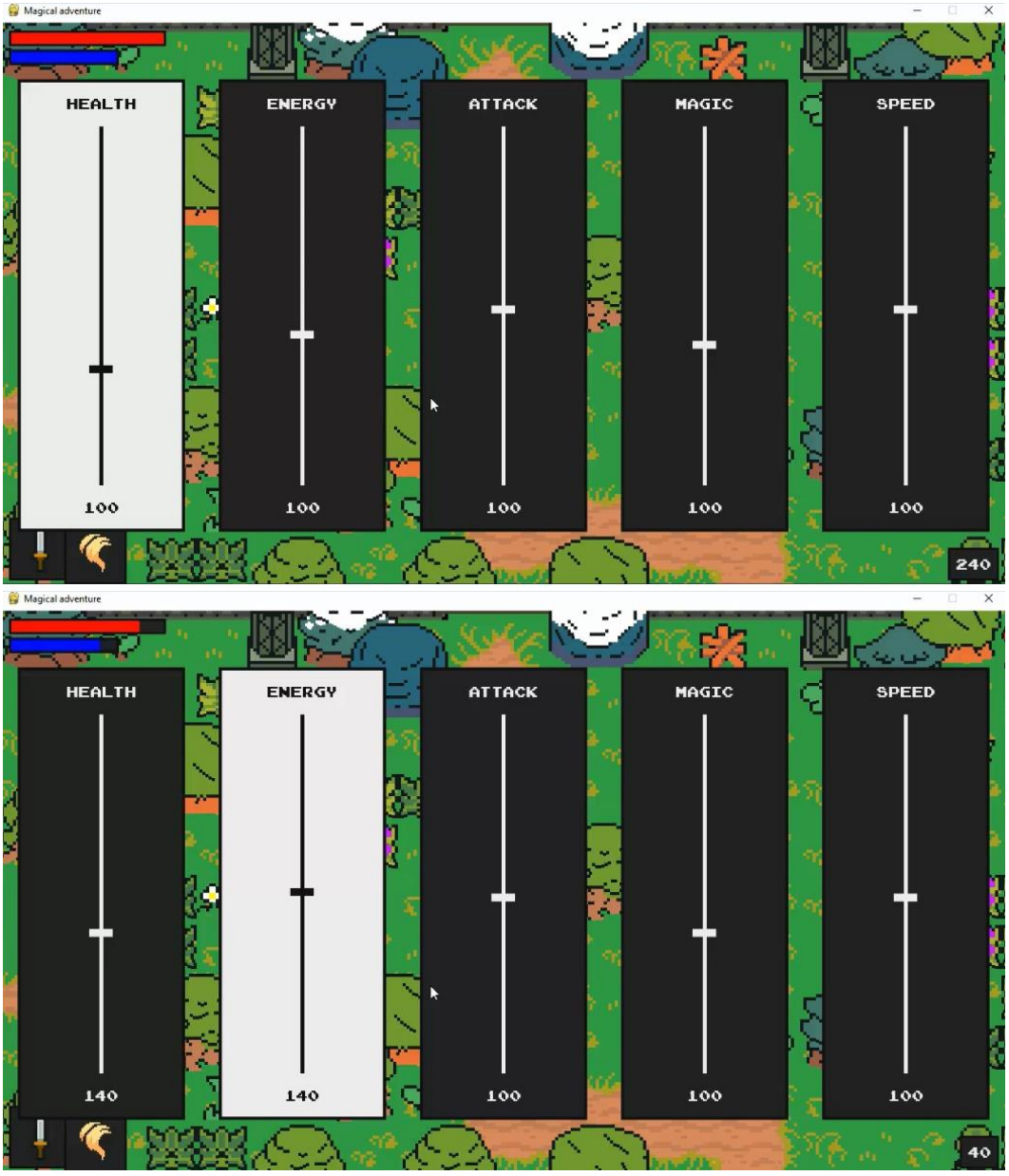

Рисунок 4.10 – Результат збільшення характеристик

Крім ворога «Бамбук» є ще декілька, найскладніший з них це «Єнот». Він має багато здоров’я і наносить багато пошкоджень гравцю. У «Єнот» є анімація атаки і смерті.

Очікуваний результат. Ворог «Єнот» має слідкувати за гравцем якщо той знаходиться поруч з ним, його атака має значно зменшувати здоров’я (шкалу

здоров’я) гравця. Під сам атаки має відбуватися анімація атаки. Також у нього присутня анімація смерті. Результат тесту наведений на рисунку 4.11.

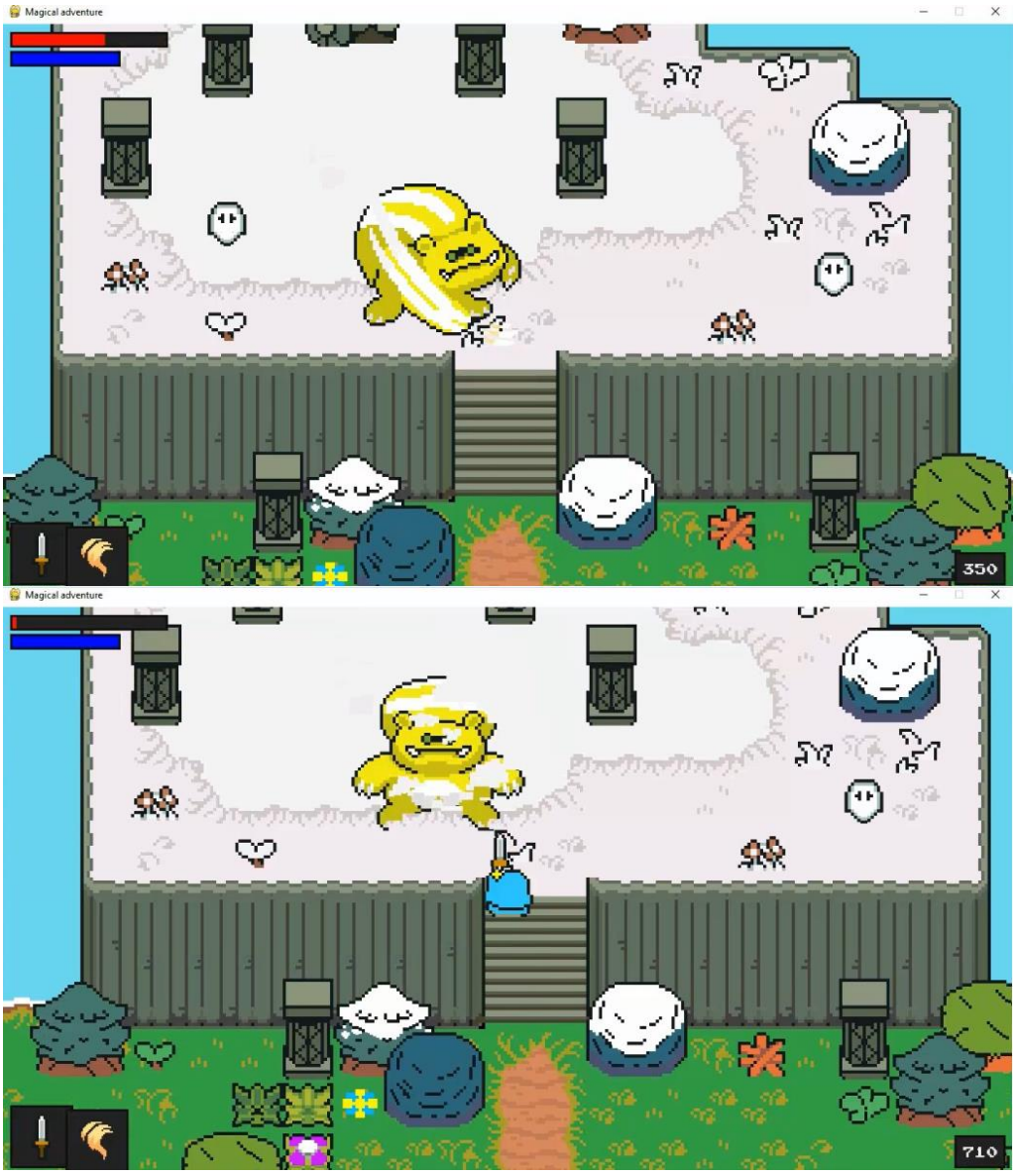

Рисунок 4.11 – Результат переміщення, атаки «Єнот» і анімації атаки та смерті

Ще одним ворогом є «Дух». Він має велику швидкість переміщення що робить втечу від нього майже не можливою. Його атака супроводжується анімацією блискавки. Смерть «Духа» теж має анімацію.

Очікуваний результат. «Дух» рухатись на до гравця якщо той знаходиться поруч. Його атака має зменшувати здоров’я гравця та супроводжуватись

анімацією блискавки. Смерть «Духа» теж супроводжується відповідною анімацією. Результат тесту наведений на рисунку 4.12.

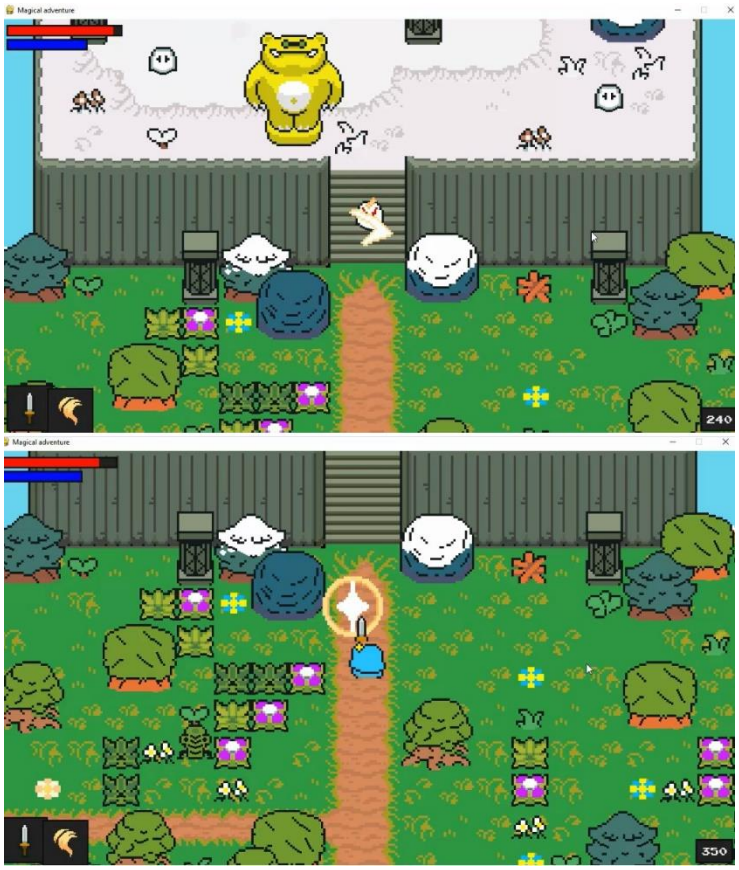

Рисунок 4.12 – Результат «Дух» наносить удар відбувається анімація атаки, зменшення здоров’я гравця і відображення анімації смерті

Тест пройдено успішно. Останнім ворогом гравця є «Кальмар», так як інші він може переміщуватись і атакувати гравця, має анімацію атаки та анімацію смерті. Він може нанести більше пошкодження ніж «Бамбук» і «Дух», але менше ніж «Єнот».

Очікуваний результат. Ворог «Кальмар» має рухатись в бік гравця якщо той буде поруч, має атаку з анімацією удару і зменшувати здоров’я гравця. У випадку його знищення має бути анімація.

Результат тесту наведений на рисунку 4.13.

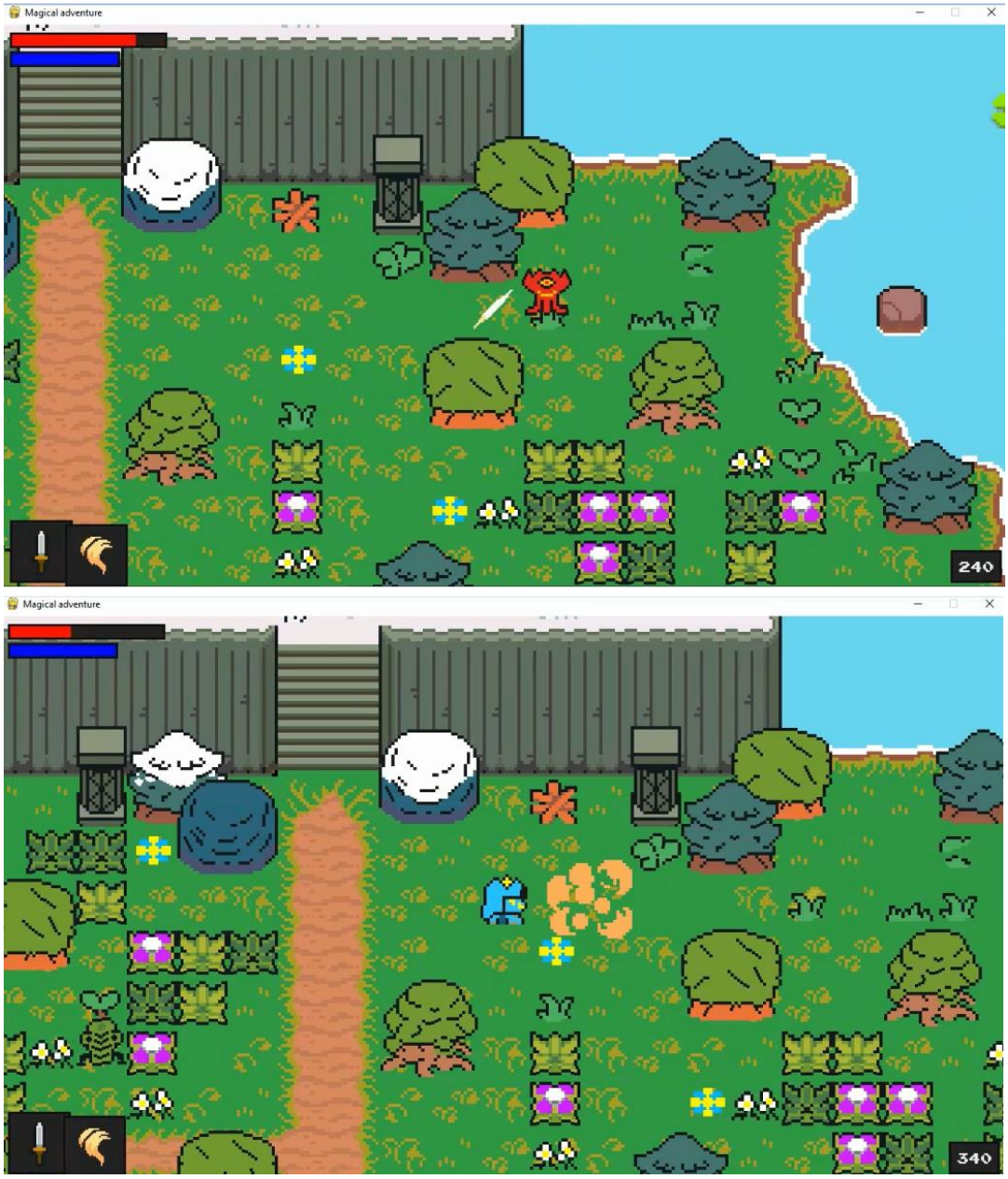

Рисунок 4.13 – Результат «Кальмар» наносить удар відбувається анімація атаки, зменшення здоров’я гравця і відображення анімації смерті

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

Очікуваний результат. Натискаючи клавішу «Q» відбувається зміна іконки зброї, іконка по контуру підсвічується жовтим кольором, 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 наведено на рисунку 4.14.

Рисунок 4.14 – Результат зміни зброї та заклинання

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

Очікуваний результат. При збільшенні характеристики на максимум, атака ворога «Єнота» буде менше зменшувати шкалу здоров’я гравця.

Результат зміни зброї і заклинання наведено на рисунку 4.15.

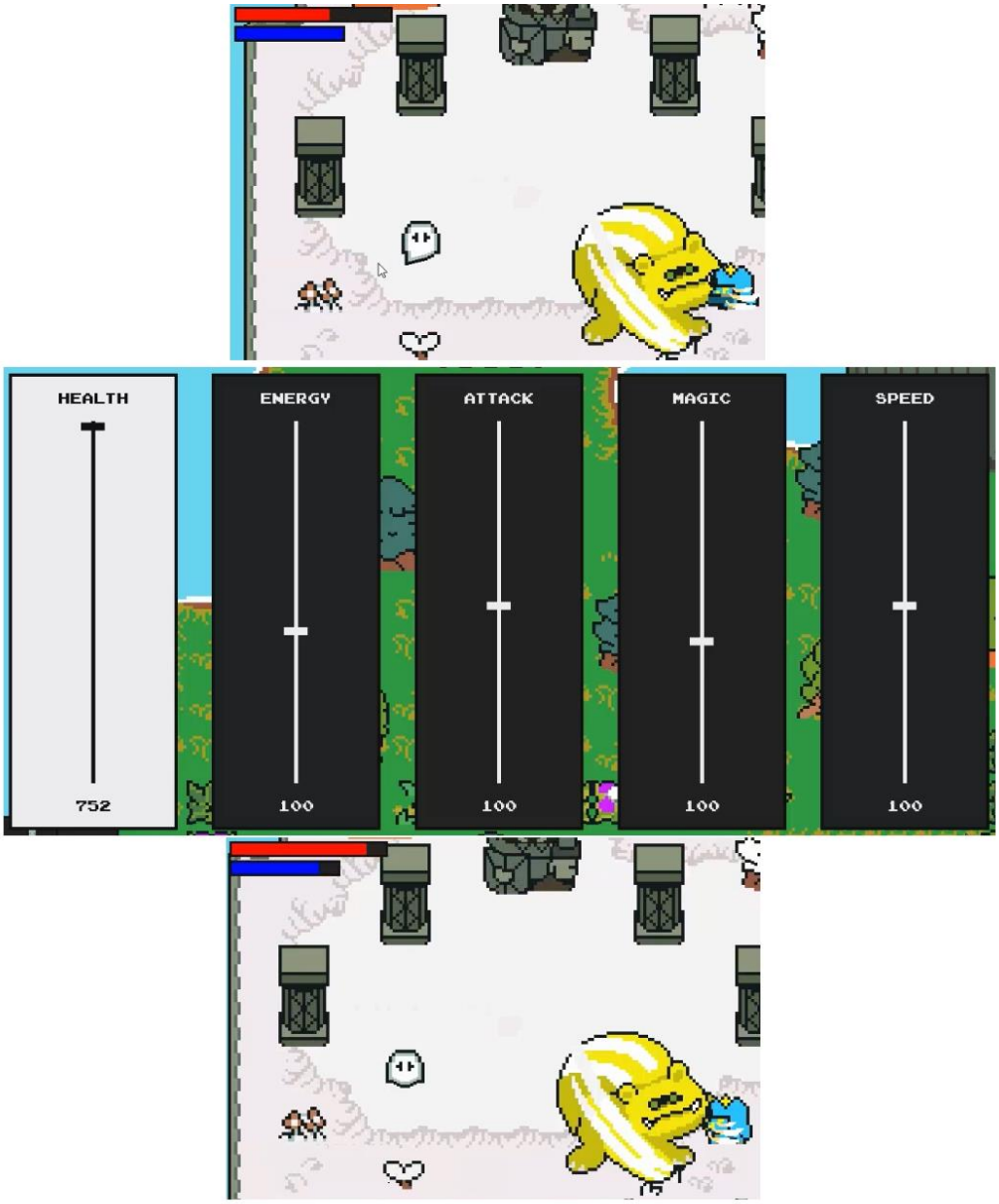

Рисунок 4.15 – Результат збільшення здоров’я гравця

4.3 Висновки

У четвертому розділі бакалаврської дипломної роботи було проаналізовано методики тестування і проведено порівняння їх переваг, і недоліків в результаті було обрано метод тестування «чорної скриньки». Відповідно до обраної методики були проведені тести до розроблюваної комп’ютерної гри та підтверджено повну працездатність.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено комп’ютерну 2d-гру «Magical adventure» у жанрі екшн-пригода. Для розробки було використано середовище програмної розробки Visual Studio Code та фреймворк Pygame.

У бакалаврській дипломній роботі проаналізовано стан задачі на сьогоднішній день. Розглянуто основні аналоги програмних продуктів та у порівнянні з власною програмою продуктом було виявлено їхні недоліки. На основі порівняння сформульовано основні завдання бакалаврської дипломної роботи для розробки комп’ютерної 2d-гри у жанрі екшн-пригода. Під час аналізу технологій розробки було обґрунтовано вибір мови програмування Python. Також було розглянуто переваги використання фреймворку Pygame.

Запропоновано метод реалізації ігрової системи, який, на відміну від існуючих, дозволяє швидко створювати і додавати нові види ворогів в ігрове середовище, оскільки вони наслідують деякі методи-функції персонажа гравця, за допомогою розроблених програмних засобів та прискорює процес розробки та ускладнення екшн-ігор

Розроблено модель роботи ігрової системи, яка орієнтована на використання компонентів тактичного мислення за допомогою збільшення характеристик персонажа, що забезпечить розвиток різноманітності ігрового процесу і створить зручні умови тренінгу для користувача.

Спроєктовано користувацький інтерфейс ігрового додатку, розроблена схема алгоритму роботи ігрового додатку та функціональна модель персонажа. Виконано розробку програмного коду ігрової системи «Magical adventure», призначеної для розважального характеру та тренування тактичного мислення гравця. Розроблено графічний матеріал для наповнення ігрового додатку.

Виконано аналіз та порівняння методів тестування, обрано метод тестування «чорної скриньки». За обраним методом проведено тестування розробленого комп’ютерного ігрового додатку згідно поставлених задач, яке підтвердило його повну працездатність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. COMPUTERS AND OUR LIFE: HOW HAVE COMPUTERS CHANGED OUR LIFE? [Електронний ресурс] – Режим доступу до ресурсу: https://www.lopol.org/article/computers-and-our-life-how-have-computers-changed-our-life.

2. 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

3. Video games are fun. Here's why, and how they hook us. [Електронний ресурс] – Режим доступу до ресурсу https://www.idtech.com/blog/what-makes-video-games-fun.

4. Unreal Element World [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/1757250/Unreal_Element_World/

5. Crypt of the NecroDancer [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/247080/Crypt_of_the_NecroDancer/

6. Dead Hearts [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/1465130/Dead_Hearts/

7. Rise of the Third Power [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/698700/Rise_of_the_Third_Power/

8. What Is Game Development? [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/what-is-game-development/.

9. Розробка комп’ютерних ігор, з чого почати? [Електронний ресурс] – Режим доступу до ресурсу: https://blog.ithillel.ua/articles/rozrobka-kompiuternykh-ihor-z-choho-pochaty

10. How to Make a Video Game: The Best Game Development Software for 2022 [Електронний ресурс] – Режим доступу до ресурсу: https://www.pcmag.com/picks/how-to-make-a-video-game-the-best-game-development-software.

11. Як створити гру? Огляд 5 етапів на прикладі «Гри для участі» муніципалітету Гельсинки [Електронний ресурс] – Режим доступу до ресурсу: https://uk.soringpcrepair.com/programs-for-make-games/

12. The Difference Between a Framework and a Library [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/the-difference-between-a-framework-and-a-library-bd133054023f/

13. Level Up [Електронний ресурс] – Режим доступу до ресурсу: https://www.giantbomb.com/level-up/3015-475/

14. The Limitations of Leveling Systems in Game Design [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@GWBycer/the-limitations-of-leveling-systems-in-game-design-f06a2a14c026

15. Experience point [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Experience_point

16. Head First Python: A Brain-Friendly Guide 2nd Edition / Paul Barry, O'Reilly Media, 2016 – 622 ст.

17. Python, PyGame, and Raspberry Pi Game Development 2nd ed. Edition / Sloan Kelly, Apress, 2019 – 414 ст.

18. Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com

19. BEST FREE GRAPHIC DESIGN SOFTWARE IN 2022 [Електронний ресурс] – Режим доступу до ресурсу: https://fixthephoto.com/best-free-graphic-design-software.html

20. About GIMP [Електронний ресурс] – Режим доступу до ресурсу: https://www.gimp.org/about/

21. About Tiled [Електронний ресурс] – Режим доступу до ресурсу: https://doc.mapeditor.org/en/stable/manual/introduction/#about-tiled

22. Effective Methods for Software Testing: Includes Complete Guidelines, Checklists, and Templates 3rd Edition / William E. Perry, Wiley, 2006 – 1008 ст.

23. Software Testing – Methods [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/software_testing/software_testing_methods.htm

ДОДАТКИ

Додаток А – Технічне завдання

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

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
31 березня 2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка комп'ютерної 2d-гри «Magical adventure»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доц. Д.І. Катєльніков
31 березня 2022 р.
Виконав:
_____студент гр. 1ПІ-19мс2 Г.М. Колісниченко
31 березня 2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка комп’ютерної 2d-гри «Magical adventure».

Галузь застосування – комп'ютерних ігри.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 12 від « 7 » лютого 2022 р.

3. Мета та призначення розробки.

Метою бакалаврської дипломної роботи є підвищення можливостей тренування тактичного мислення користувача шляхом розробки та використання 2d-гри жанру екшн-пригода з компонентами розвитку зорової уваги та прийняття тактичних рішень, яка сприятиме тренуванню швидкості прийняття рішень гравцем в екстремальних умовах.

Призначення роботи – комп’ютерна 2d-гра для проведення тренувань тактичного мислення в ігровій формі.

4. Вихідні дані для проведення НДР

1. The Art of Game Design: A Book of Lenses, Third Edition / Jesse Schell – A K Peters/CRC Press, 2019. – 654 ст.
2. Python Crash Course: A Hands-On, Project-Based Introduction to Programming / Eric Matthes – No Starch Press, 2015. – 623 ст.
3. Beginning Game Development with Python and Pygame: From Novice to Professional (Beginning From Novice to Professional) / Will McGugan – Apress, 2007. – 340 ст.
4. Invent Your Own Computer Games with Python, 4th Edition / Al Sweigart – No Starch Press, 2016. – 376 ст.

5. Технічні вимоги

Склад комплексу технічних засобів

Вхідні дані – Для розв’язку задачі буде використовуватися сучасний ноутбук на операційній системі Windows 10.

Вимоги до складових частин комплексу технічних засобів.

Для запуску програми апаратне забезпечення повинне відповідати мінімальним вимогам:

- ноутбук з процесором Pentium 2217U (2 ядра, 1,8 ГГц);
- 4 Гбайт оперативної пам’яті;
- Інтегрована графіка Intel HD Graphics 630;
- монітор з розширенням 1280 на 720 пікселів;
- Windows 10;

6. Конструктивні вимоги.

Дизайн програми має відповідати естетичним та ергономічним вимогам, програма повинна бути зручною в користуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка комп’ютерної 2d-гри «Magical adventure»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: доц. каф. ПЗ Катєльніков Д.І.

Оригінальність	92,7%
Схожість	7,3%

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 Георгій Миколайович

Керівник роботи _____ Катєльніков Денис Іванович

Додаток В – Лістинг програми

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_e
xp):

                # 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

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_sprites])

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()
                        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_COLOR)

        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 – Графічна частина

ГРАФІЧНА ЧАСТИНА
РОЗРОБКА КОМП’ЮТЕРНОЇ 2D-ГРИ «MAGICAL ADVENTURE»

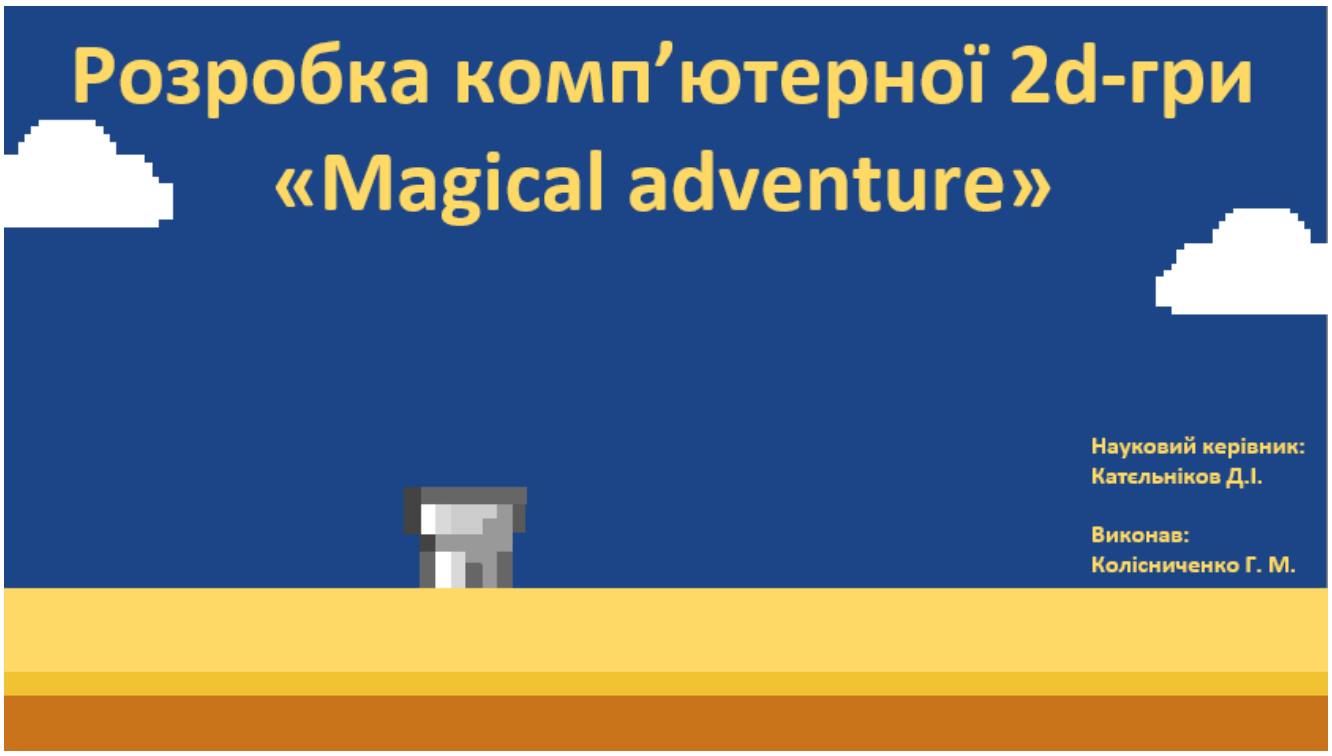

Рисунок Г.1 – Назва роботи

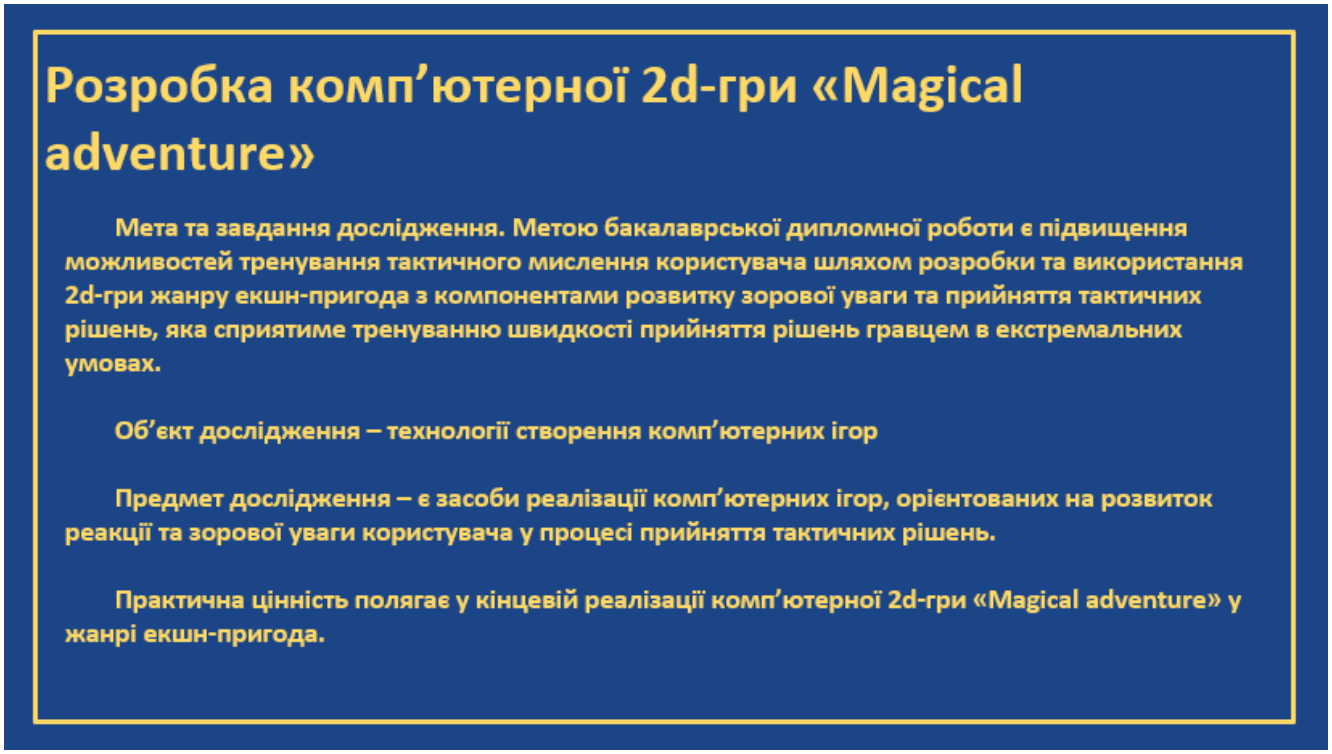

Рисунок Г.2 – Мета, об’єкт і предмет дослідження

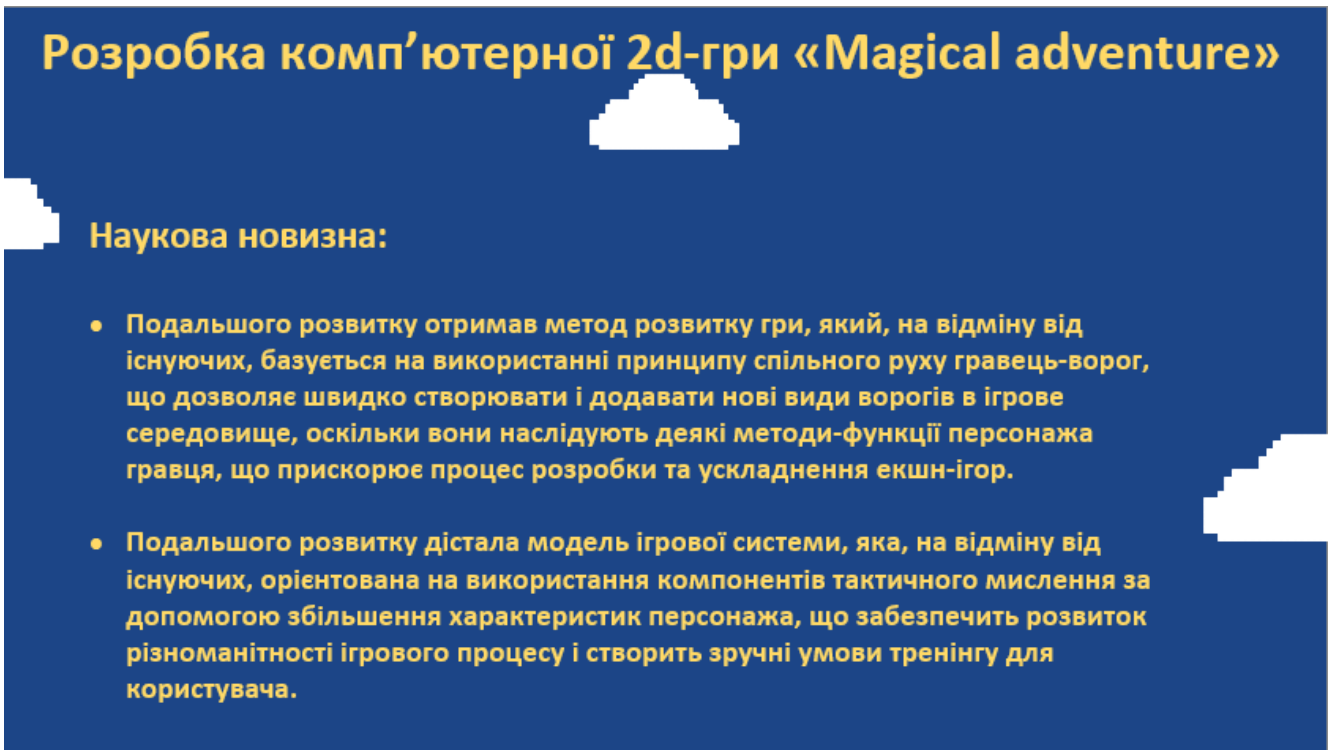

Рисунок Г.3 – Наукова новизна

Рисунок Г.4 – Задачі бакалаврської дипломної роботи

Рисунок Г.5 – Аналоги

Результати порівняльного аналізу аналогів

Критерії	Unreal Element World	Crypt of the NecroDancer	Dead Hearts	Rise of the Third Power	Magical adventure
Інтуїтивне управління персонажом	+	+	+	-	+
Музичний супровід і звукові ефекти	+	+	+	+	+
Мінімалістичні візуальні ефекти	-	+	+	-	+
Система підвищення характеристиків персонажа	+	-	-	+	+
Мінімальні системні вимоги	-	-	+	+	+
Абсолютно безкоштовна гра	-	-	-	-	+

Рисунок Г.6 – Результати порівняльного аналізу аналогів

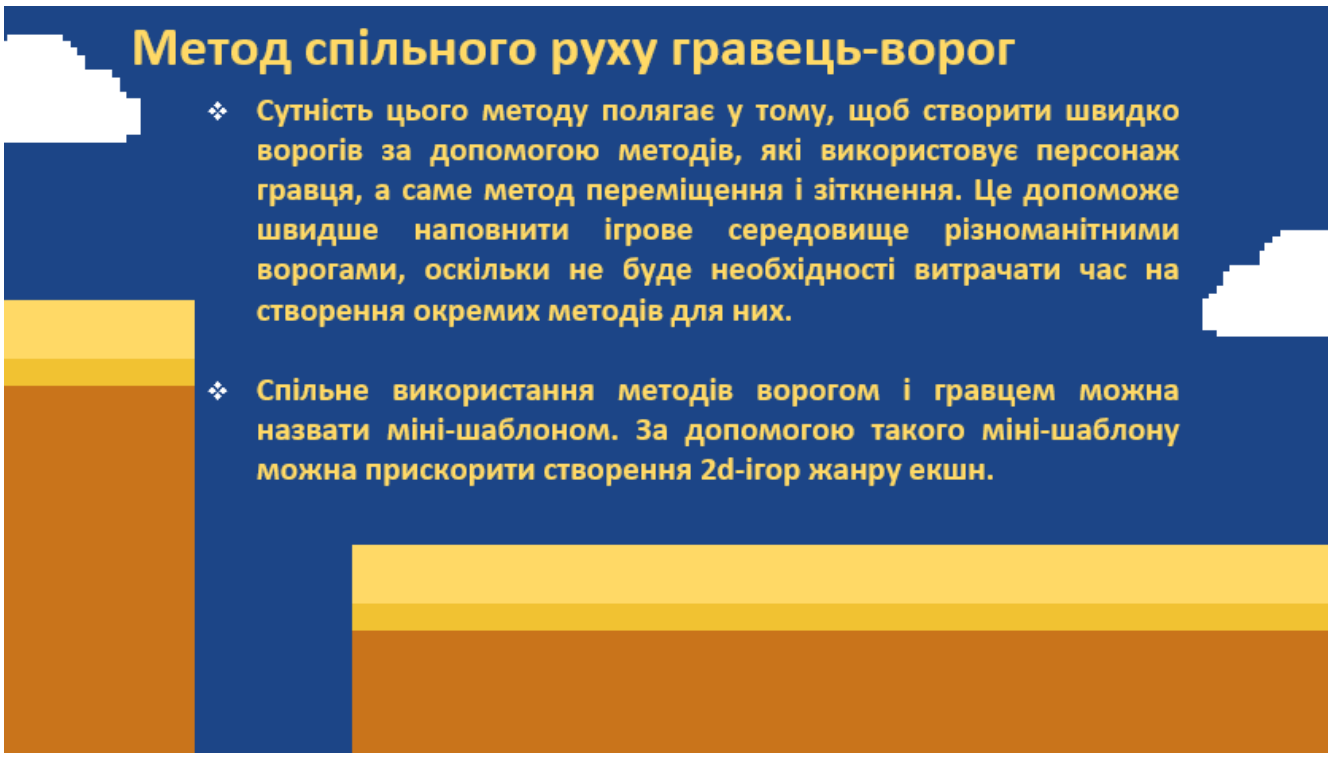

Рисунок Г.7 – Метод спільного руху гравець-ворог

Рисунок Г.8 – Схема методу спільного руху гравець-ворог

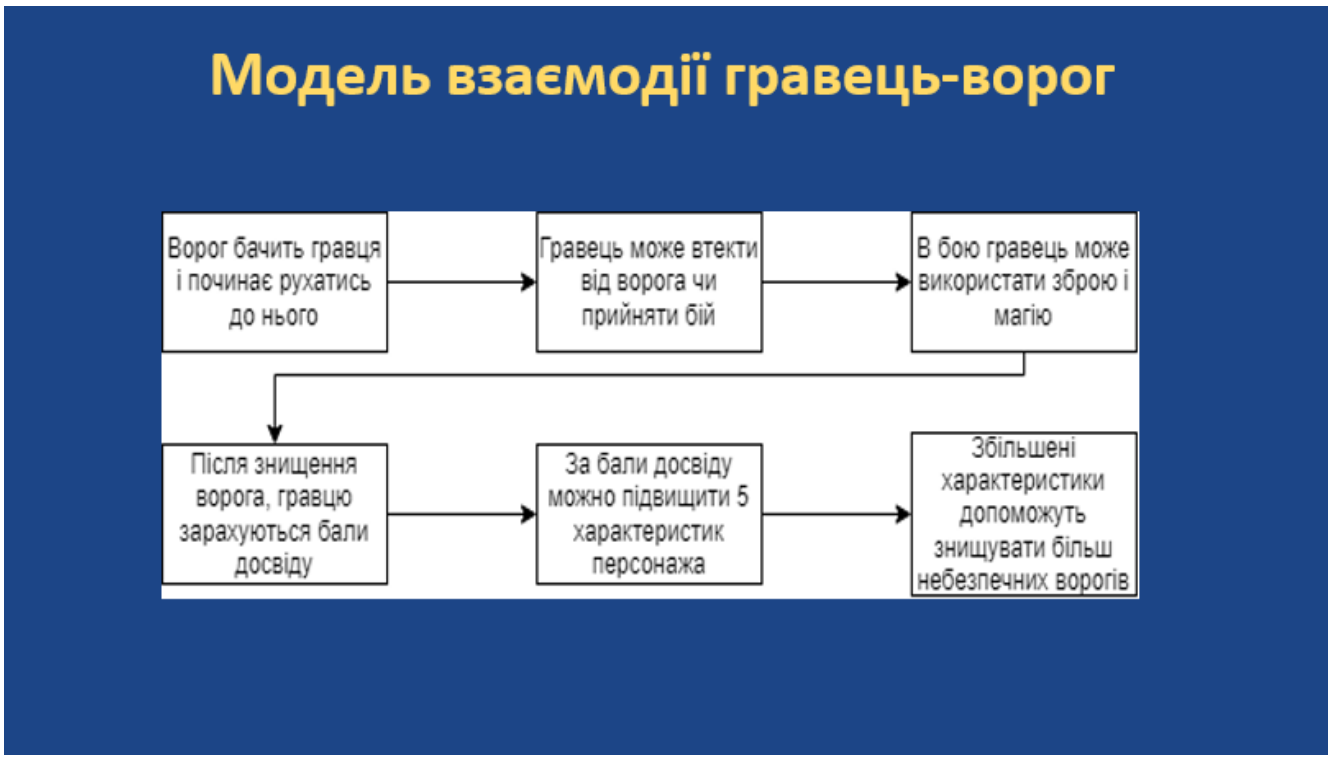

Рисунок Г.9 – Модель взаємодії гравець-ворог

Рисунок Г.10 – Блок-схема загального алгоритму програми

Рисунок Г.11 – Функціональна модель персонажа

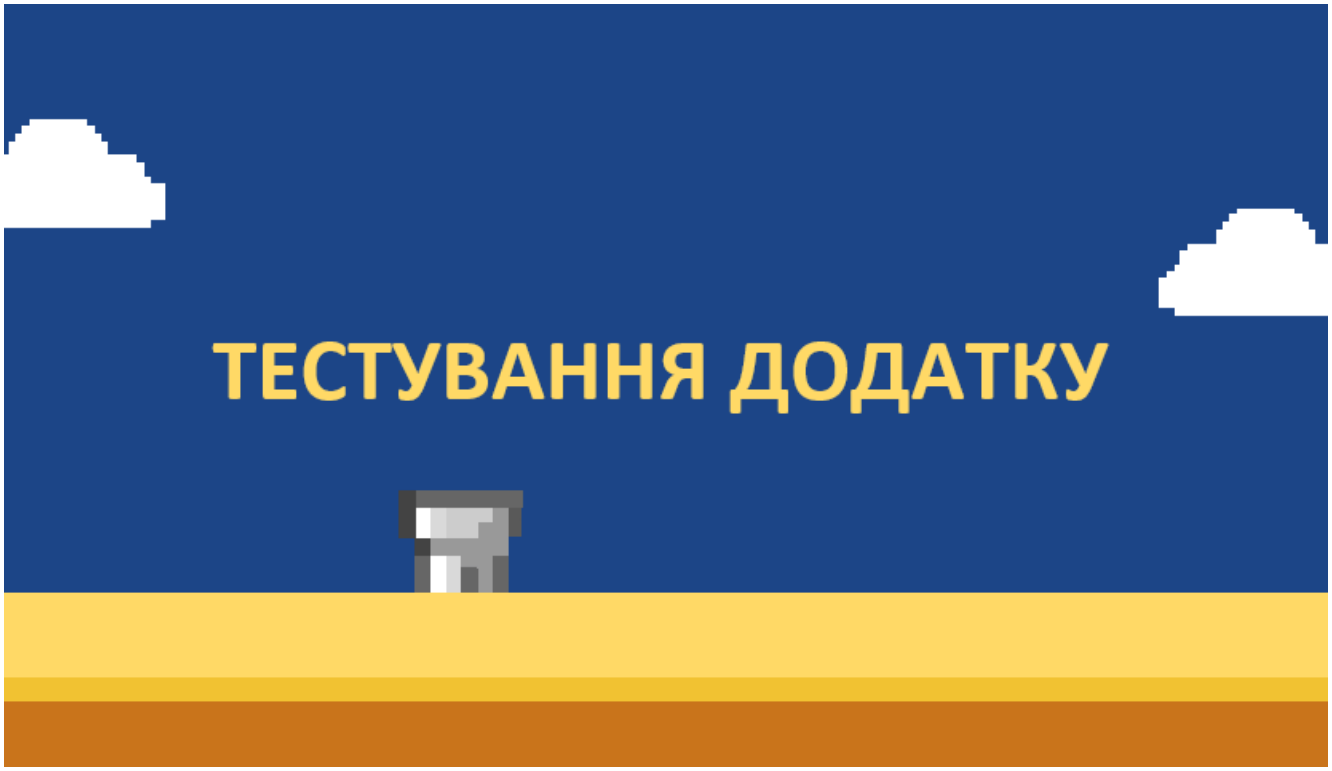

Рисунок Г.12 – Тестування додатку

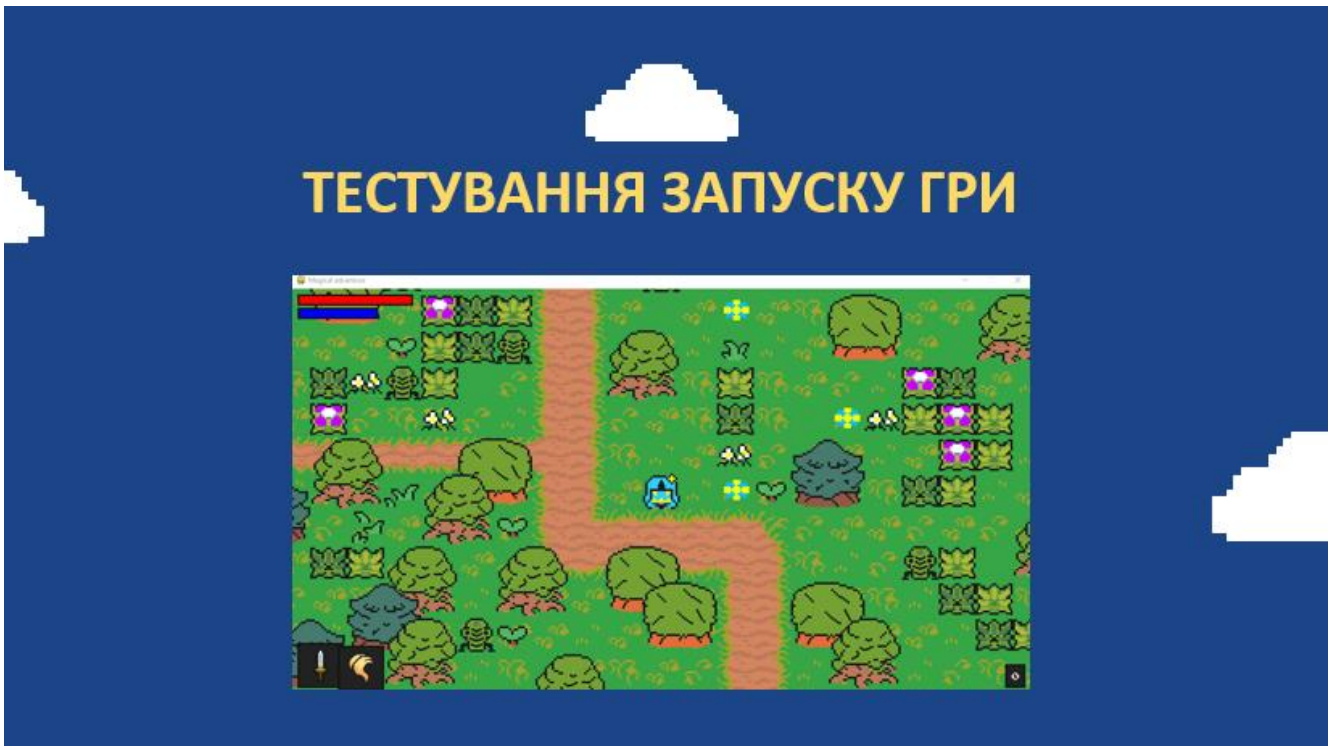

Рисунок Г.13 – Тестування запуску гри

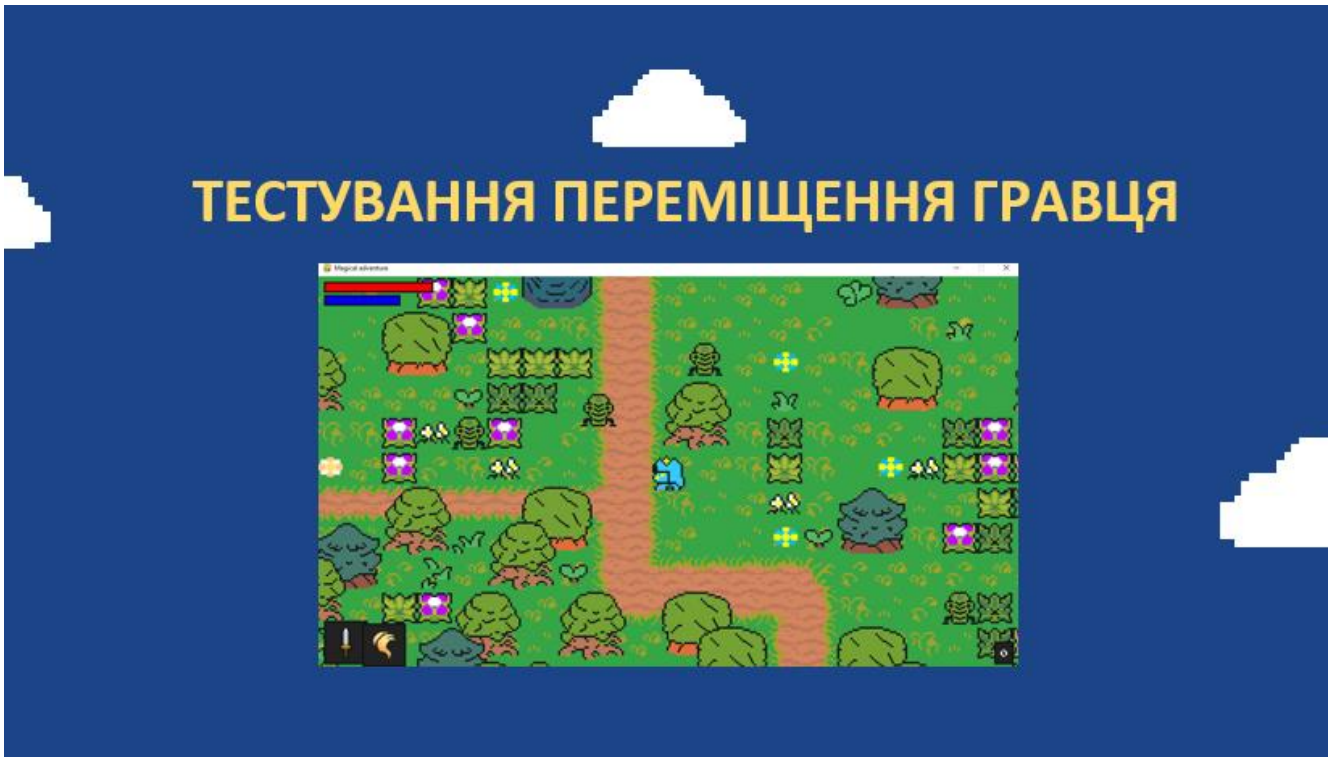

Рисунок Г.14 – Тестування переміщення гравця

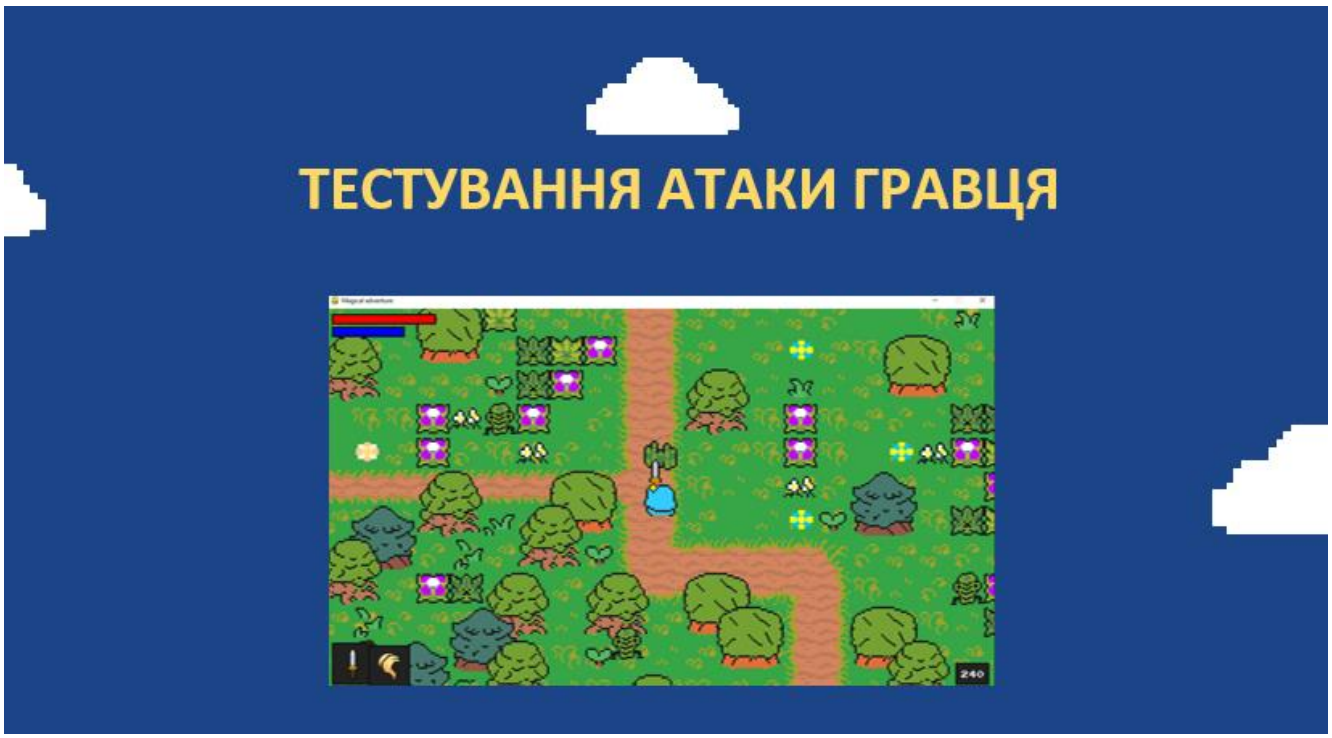

Рисунок Г.15 – Тестування атаки гравця

Рисунок Г.16 – Тестування атаки ворога

Рисунок Г.17 – Тестування заклинання «Лікування»

Рисунок Г.18 – Тестування заклинання «Полум’я»

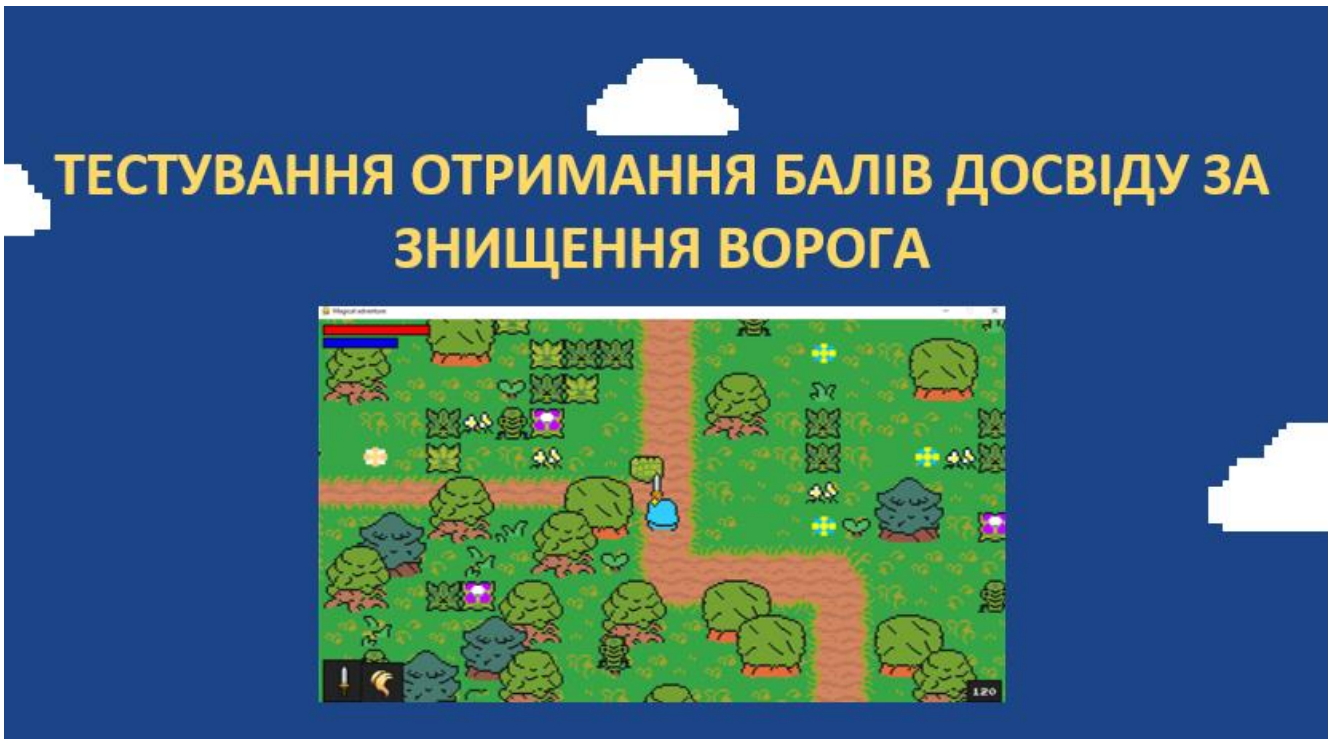

Рисунок Г.19 – Тестування отримання балів досвіду за знищення ворога

Рисунок Г.20 – Тестування відкриття меню збільшення характеристик

Рисунок Г.21 – Тестування роботи меню збільшення характеристик

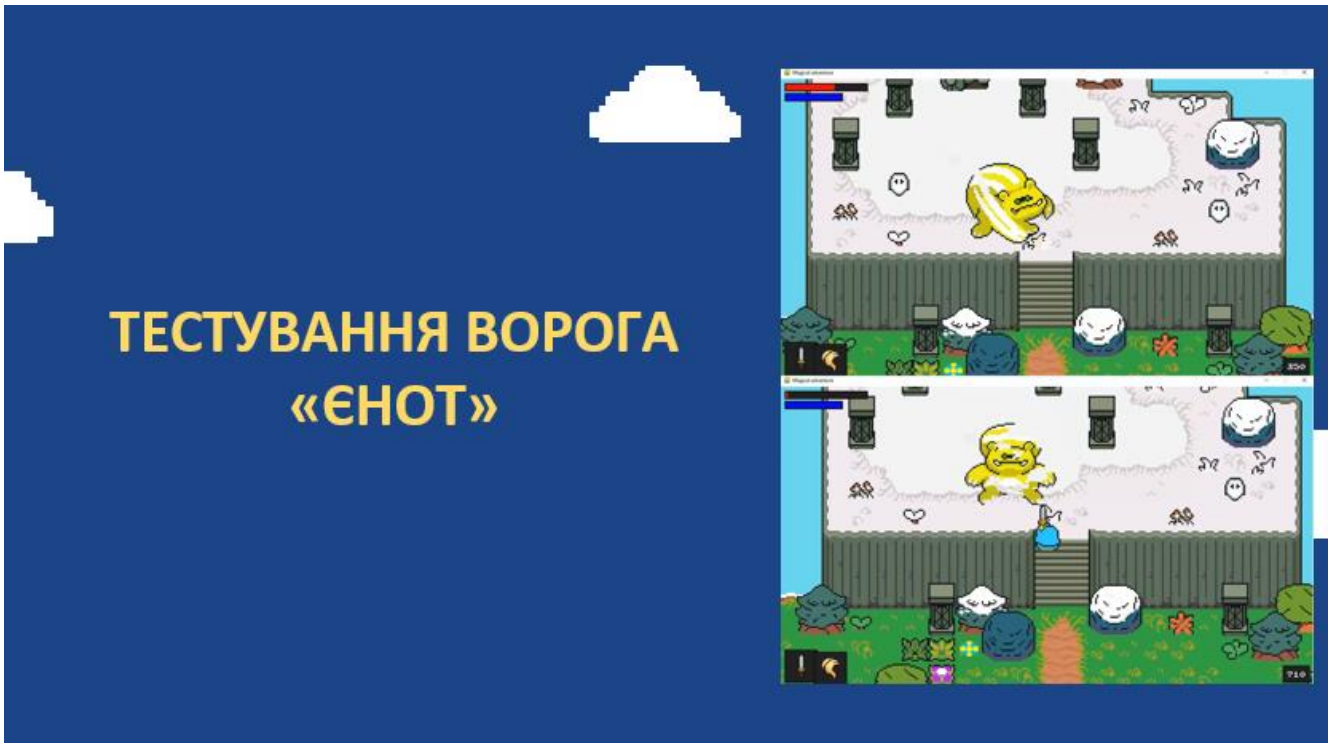

Рисунок Г.22 – Тестування ворога «Єнот»

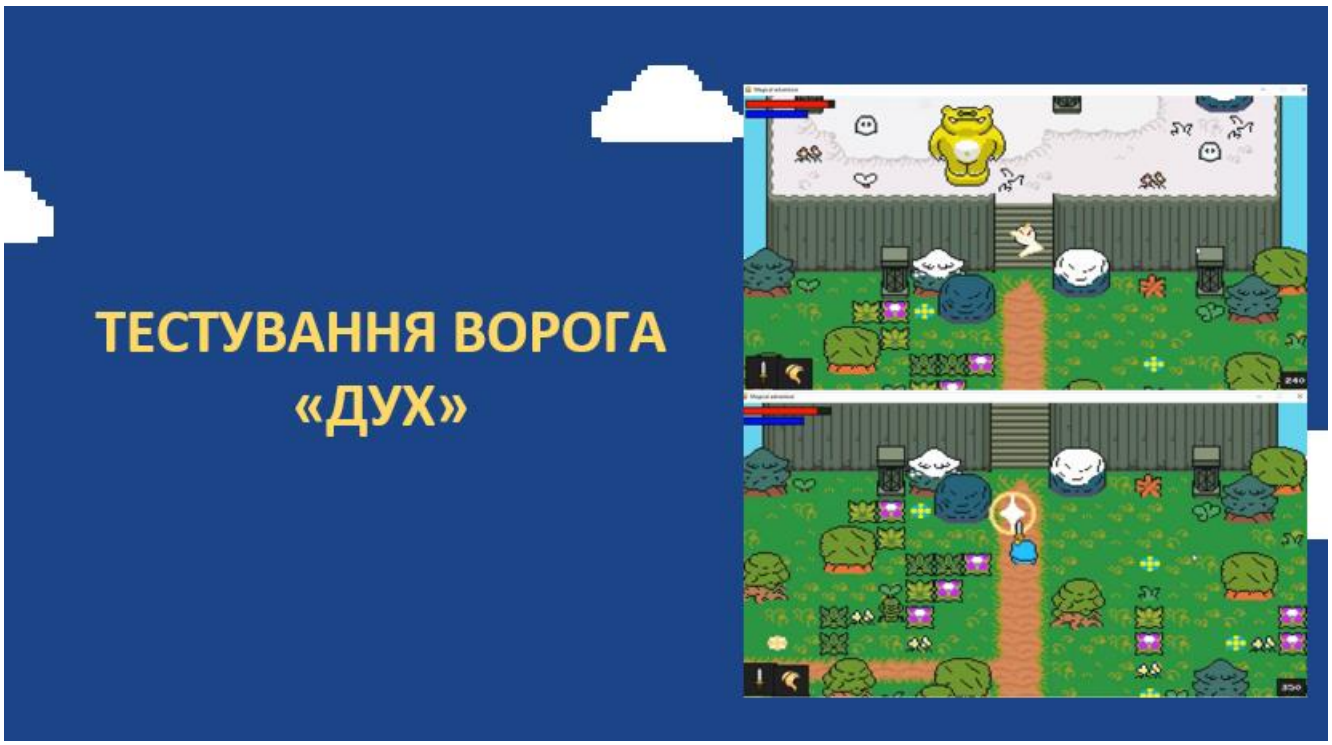

Рисунок Г.23 – Тестування ворога «Дух»

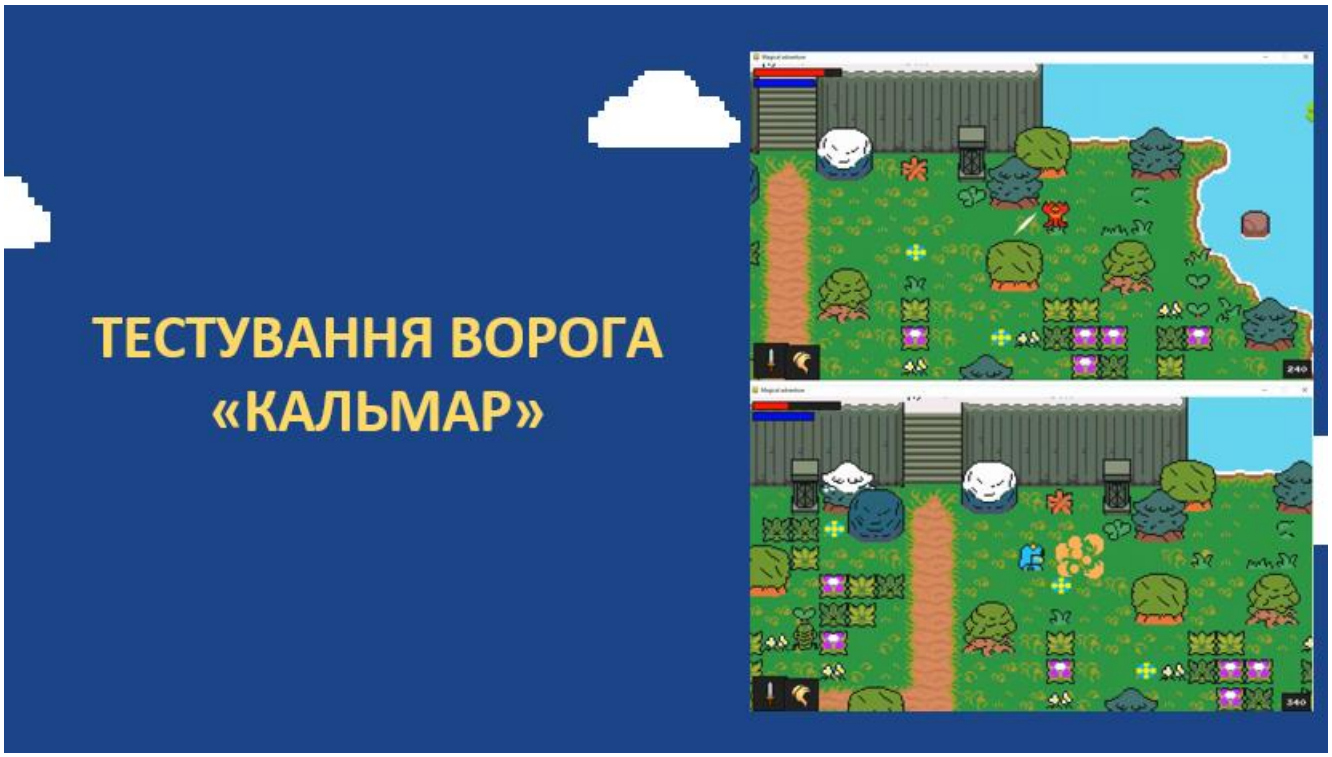

Рисунок Г.24 – Тестування ворога «Кальмар»

Рисунок Г.25 – Тестування зміни зброї і заклинань

Рисунок Г.26 – Тестування збільшення характеристики здоров’я гравця

Рисунок Г.27 – Висновки

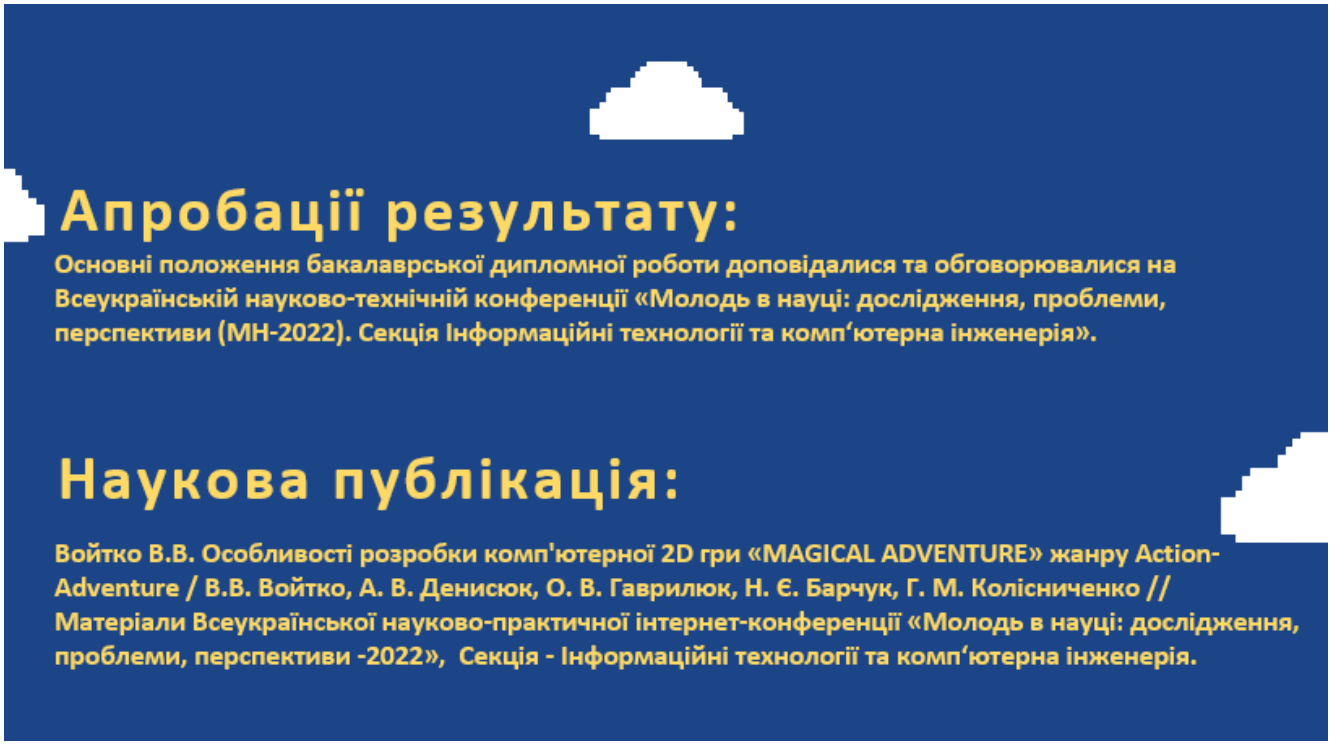

Рисунок Г.28 – Апробація результатів і публікація