**Міністерство освіти і науки України**
**Український державний університет науки і технологій**
Навчально-науковий інститут
«Український державний хіміко-технологічний університет»
(назва навчального-наукового інституту)
Факультет комп'ютерних наук та інженерії
(повна назва факультету)
Кафедра комп'ютерно-інтегрованих технологій та робототехніки
(повна назва кафедри)

# Пояснювальна записка

до дипломного проекту (роботи)
бакалавр
(освітній рівень)
на тему: «Розробка аркадної комп'ютерної гри з використанням мови Python»

Виконав студент 4 курсу, групи КІ
спеціальності 123 «Комп'ютерна інженерія»

Студент Дарницький Р.О. _____
(прізвище та ініціали) (підпис)

Керівник Хорошилов С.В. _____
(прізвище та ініціали) (підпис)

Рецензент Дубовик Т.М. _____
(прізвище та ініціали) (підпис)

Дніпро – 2026 рік

Український державний університет науки і технологій
(повне найменування вищого навчального закладу)
Навчально-науковий інститут
«Український державний хіміко-технологічний університет»
(назва навчально-наукового інституту)

Факультет, відділення комп'ютерних наук та інженерії
Кафедра комп'ютерно-інтегрованих технологій та робототехніки
Освітній рівень бакалавр
Спеціальність 123 - Комп'ютерна інженерія
(код і назва)
Спеціалізація
(шифр і назва)
Освітня програма Комп'ютерна інженерія
(назва)

**ЗАТВЕРДЖУЮ**

зав. кафедри канд. техн. наук, доц. ЛЕВЧУК І.Л.

«12» 06 2026 року

# ЗАВДАННЯ
# НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ

Дарницький Роман Олександрович

(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка аркадної комп'ютерної гри з використанням мови Python»

керівник роботи ХОРОШИЛОВ Сергій Вікторович, докт. техн. наук, проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від 02.03.2026 р. №77-к
2. Строк подання студентом роботи 08.06.2026 р.
3. Вихідні дані до роботи матеріали виробничої практики, дані технічної літератури
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Загальна частина. Спеціальна частина. Конструкторсько-технічна реалізація. Охорона праці та безпека в надзвичайних ситуаціях. Організаційно-економічний розділ
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Презентація до дипломної роботи

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| Охорона праці та безпека в надзвичайних ситуаціях | С.В. ХОРОШИЛОВ, докт. техн. наук, професор | 29.02.2026 | 10.05.2026 |
| Організаційно-економічна частина | С.В. ХОРОШИЛОВ, докт. техн. наук, професор | 20.03.2026 | 10.05.2026 |

7. Дата видачі завдання «5» .02. 2026 року

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапу дипломної роботи | Строк виконання | Примітка |
|---|---|---|---|
| 1 | Огляд літератури та аналіз ринку відеоігор | лютий | виконано |
| 2 | Вивчення Pygame, проєктування архітектури гри | березень | виконано |
| 3 | Реалізація класів Player, Zombie, Bullet | квітень | виконано |
| 4 | Написання головного ігрового циклу main.py | квітень | виконано |
| 5 | Тестування, профілювання, усунення дефектів | квітень–травень | виконано |
| 6 | Написання пояснювальної записки | травень | виконано |
| 7 | Передзахист та підготовка до презентації | травень | виконано |

**Студент** _____ Р. О. ДАРНЦЬКИЙ
(підпис) (ініціали, прізвище)

**Керівник роботи** _____ С.В. ХОРОШИЛОВ
(підпис) (ініціали, прізвище)

# РЕФЕРАТ

Дипломна робота налічує 84 сторінок, 7 рисунків, 11 таблиць, 1 додаток та 25 використаних джерел.

Об’єктом дослідження є комп’ютерна гра «Zombie Escape», розроблена мовою Python з використанням бібліотеки Pygame.

Предметом дослідження є методи організації ігрового коду: побудова ігрового циклу, опис ігрових об’єктів через класи та реалізація їх взаємодії.

Метою роботи є проєктування та програмна реалізація відеогри «Zombie Escape» з механіками руху персонажа, прицілювання, переслідування ворогами, виявлення зіткнень та підрахунку ігрових балів.

У першому розділі проаналізовано стан ринку відеоігор, охарактеризовано жанр гри, порівняно інструменти для розробки 2D-ігор та обґрунтовано вибір Python і Pygame.

У другому розділі сформульовано вимоги до гри, розроблено модульну архітектуру проєкту та описано реалізацію кожного класу.

У третьому розділі описано алгоритми руху об’єктів та виявлення зіткнень, наведено результати функціонального тестування, а також охарактеризовано апаратне середовище та системні вимоги для запуску гри.

Практичне значення роботи полягає в тому, що готовий програмний продукт може використовуватись як навчальний приклад з розробки ігор на Python. Модульна архітектура коду дозволяє без значних зусиль розширювати функціонал проєкту.

Ключові слова: PYTHON, PYGAME, ООП, ІГРОВИЙ ЦИКЛ, ZOMBIE ESCAPE, SURVIVAL SHOOTER, НОРМАЛІЗАЦІЯ ВЕКТОРА, AABB, ВИЯВЛЕННЯ ЗІТКНЕНЬ, ШІ ВОРОГІВ.

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

| Скорочення | Повне найменування |
|---|---|
| ООП | Об’єктно-орієнтоване програмування |
| ШІ | Штучний інтелект |
| FPS | Frames Per Second – кількість кадрів за секунду |
| AABB | Axis-Aligned Bounding Box – вісно-вирівняний прямокутний контейнер |
| SDL | Simple DirectMedia Layer – крос-платформена мультимедійна бібліотека |
| API | Application Programming Interface – програмний інтерфейс застосунку |
| IDE | Integrated Development Environment – інтегроване середовище розробки |
| ПЗ | Програмне забезпечення |
| ПК | Персональний комп’ютер |
| 2D | Two-dimensional – двовимірний простір |
| ОЗП | Оперативний запам’ятовуючий пристрій |
| SRP | Single Responsibility Principle – принцип єдиної відповідальності |
| CPU | Central Processing Unit – центральний процесор |
| GPU | Graphics Processing Unit – графічний процесор |

# ВСТУП

Комп’ютерні ігри сьогодні – це не лише розвага, а й велика галузь. Тільки за 2024 рік ринок ігор склав близько 187 мільярдів доларів. При цьому навіть один розробник може зробити гру і виставити її на Steam або itch.io без великих витрат.

Написати гру – це хороша практика для програміста, бо в одному проєкті одразу є і алгоритми, і ООП, і математика, і взаємодія з користувачем. Тому було вирішено розробити повноцінну гру як дипломну роботу.

Темою дипломної роботи є розробка гри «Zombie Escape» – двовимірного шутера з видом зверху, де потрібно відстрілюватись від зомбі і намагатись вижити якнайдовше. Гра написана на Python з використанням бібліотеки Pygame.

Тема актуальна з кількох причин. Python – одна з найпопулярніших мов у світі, її вчать майже в кожному виші. Pygame зручний тим, що одразу видно результат на екрані – це допомагає краще розуміти код. А жанр шутера виживання дозволяє реалізувати всі основні принципи розробки ігор в одному невеликому проєкті.

Мета роботи – написати робочу гру, де код правильно організовано по класах і все працює згідно з запланованим.

Для досягнення цієї мети було поставлено такі завдання:

- розібратись з жанром гри і подивитись які подібні ігри вже є;
- порівняти різні інструменти для розробки ігор і обрати найкращий для цього проєкту;
- вивчити як влаштований Python і Pygame для написання ігор;
- розбити код на окремі файли і класи для зручності;
- написати ігровий цикл, рух гравця, зомбі і кулі;
- зробити щоб куля потрапляла в зомбі, а зомбі міг пошкодити гравця;
- перевірити що все в грі працює правильно і записати результати;
- описати комп’ютер на якому розроблялась гра і що потрібно для її запуску.

Готовий проєкт може стати у пригоді тим хто тільки починає вивчати програмування ігор на Python. Код розбитий на модулі, тому в майбутньому не складно буде додати нових ворогів, зброю або рівні.

# РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ЗАСОБІВ РОЗРОБКИ

## 1.1. Відеоігри у сучасній цифровій економіці

Ігрова індустрія сьогодні – це дуже великий бізнес. За даними аналітиків, у 2024 році ринок відеоігор склав 187 мільярдів доларів, а в ігри грають понад 3,4 мільярди людей. Ринок при цьому зростає щороку на 8–10 відсотків.

Важливу роль тут відіграють так звані інді-ігри – проєкти невеликих команд або навіть одного розробника. Наприклад, гру Stardew Valley зробив один розробник, і вона зібрала більше 30 мільйонів доларів на Steam. Це показує, що бюджет не головне – головне вміння писати код.

З таким ростом ринку з'являється і попит на людей які вміють робити ігри. Розуміти як влаштований ігровий цикл, як рухаються об'єкти і як зомбі знаходять гравця – це реально корисні навички.

### 1.1.1. Категорії розробників та бізнес-моделі

Ринок розробників ігор поділяється на три основні категорії: AAA-студії (великі компанії з бюджетами від 100 мільйонів доларів – EA, Ubisoft, CD Projekt Red), AA-студії (середні компанії з бюджетами 5–50 мільйонів) та інді-розробники (незалежні розробники або малі команди до 10 осіб). Кожна категорія використовує різні технологічні стеки та підходи до розробки.

Бізнес-модель для інді-ігор може бути такою: одноразова купівля (pay-once), умовно-безкоштовна гра з внутрішніми покупками (free-to-play), підписка або модель раннього доступу (Early Access). «Zombie Escape» у своїй поточній формі відповідає першій моделі – це завершений продукт без монетизації, розроблений як навчальний проєкт. Проте з додаванням контенту

та відповідним маркетингом такий проєкт може бути опублікований на itch.io або навіть Steam.

Відеоігри діляться на жанри залежно від того як у них грають, а не від сюжету. Основні жанри: рольові ігри (RPG), стратегії (RTS), платформери, симулятори, пазли та шутери.

Шутери бувають від першої особи (FPS), від третьої особи (TPS) і з видом зверху (top-down shooter). У top-down шутері камера дивиться на арену згори – гравець бачить усе поле відразу. Саме цей вид було обрано для гри «Zombie Escape».

Гра «Zombie Escape» відноситься до піджанру survival shooter. Для нього характерно:

– гра не має кінця – граєш поки не помреш;
– зомбі постійно з’являються і стає їх дедалі більше;
– результат – скільки зомбі встиг знищити;
– кожна гра інша, бо зомбі з’являються у випадкових місцях.

З відомих ігор цього жанру – Crimsonland, Alien Shooter, Nuclear Throne. «Zombie Escape» повторює їхню основну механіку: замкнута арена 800×600 пікселів, зомбі з’являються по краях і рухаються до гравця.

## 1.2. Класифікація жанрів та специфіка top-down survival shooter

Ігрова індустрія перетворилася на одну з найприбутковіших галузей розважального бізнесу, обігнавши за обсягом доходів кіноіндустрію та музичний бізнес разом узяті. За даними аналітичної компанії Newzoo, у 2023 році глобальний ринок відеоігор оцінювався у 184 мільярди доларів США, а до 2027 року прогнозується зростання до 239 мільярдів. Ці цифри включають доходи від продажу програмного забезпечення, внутрішньоігрових покупок, підписок та реклами у безкоштовних іграх.

Структура ринку суттєво змінилася за останнє десятиліття. Якщо у 2010 році основним каналом продажу були фізичні носії (диски, картриджі), то сьогодні понад 85% ринку складають цифрові продажі через платформи Steam, PlayStation Store, Xbox Game Pass та мобільні магазини App Store і Google Play. Мобільний сегмент є найбільшим за кількістю гравців, але середній дохід на користувача значно нижчий ніж у ПК та консольному сегментах.

Україна активно інтегрована у світову ігрову екосистему. Вітчизняні розробники створили десятки відомих у світі проєктів. Студія GSC Game World відома серією S.T.A.L.K.E.R., Frogwares – детективними іграми про Шерлока Холмса, Gameloft Ukraine – мобільними іграми. За оцінками UAGA (Ukrainian Association of Game Developers), в Україні працює понад 30 000 фахівців ігрової індустрії. Це свідчить про наявність реального попиту на спеціалістів, що вміють розробляти ігрове програмне забезпечення.

Важливою тенденцією є демократизація розробки. Якщо у 1990-х роках створення гри вимагало команди з десятків фахівців та мільйонних бюджетів, то сьогодні один розробник з ноутбуком та безкоштовним інструментарієм може створити гру і продати її мільйонам гравців через Steam. Ціна публікації на Steam – лише 100 доларів. Успіх таких ігор як Undertale (один розробник, понад 500 000 копій), Stardew Valley (один розробник, понад 20 мільйонів копій) доводить реальність цього сценарію.

Освітній аспект ігрової розробки також набуває важливості. Провідні університети світу – MIT, Carnegie Mellon, USC – пропонують спеціалізовані програми з Game Design та Game Development. В Україні відповідні курси впроваджуються в технічних університетах. Розробка власної гри як дипломний проєкт є загальновизнаною практикою, що дозволяє студенту на одному проєкті відпрацювати навички програмування, математики, алгоритмів та UI/UX дизайну.

### 1.2.1. Психологічні аспекти ігрового процесу

Успішність жанру survival shooter пояснюється не лише технічною простотою реалізації, але й психологічними механізмами залучення гравців. Американський психолог Міхай Чіксентміхайї описав стан «потоку» (flow) як оптимальний досвід, коли складність завдання точно відповідає рівню навичок людини. Саме цей стан забезпечує жанр: на початку гра проста, але складність поступово зростає, утримуючи гравця у стані потоку.

Механіка «ще одна спроба» (one more run) характерна для survival-ігор. Після загибелі персонажа гравець бачить свій результат і одразу хоче спробувати ще раз, щоб побити рекорд. У «Zombie Escape» це реалізовано через кнопку R для перезапуску – мінімальний бар'єр між завершенням однієї сесії і початком наступної.

Зростаюча складність (difficulty scaling) є ключовим елементом утримання інтересу. У поточній версії гри складність зростає через збільшення кількості зомбі та прискорення їхнього появлення з часом. Це класичний підхід, застосований ще в оригінальному Space Invaders, де вороги прискорювалися в міру їхнього знищення.

Для вибору інструментів розробки 2D-ігор було проаналізовано доступні інструменти: Pygame, Godot, Unity, Arcade і LÖVE. Результати порівняння наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика засобів розробки 2D-ігор

| **Критерій** | **Pygame** | **Godot** | **Unity** | **Arcade** | **LÖVE** |
|---|---|---|---|---|---|
| Мова | Python | GDScript/C# | C# | Python | Lua |
| Відкритий код | Так | Так | Частково | Так | Так |
| Крива навчання | Низька | Середня | Висока | Низька | Середня |

| | | | | | |
|---|---|---|---|---|---|
| Редактор сцен | Ні | Так | Так | Ні | Ні |
| Доступ до внутрішньої логіки | Повний | Частковий | Прихований | Повний | Повний |
| Придатність для ВНЗ | Ідеально | Добре | Задовільно | Добре | Задовільно |

Продовження таблиці

Обрано Pygame, оскільки тут усе реалізується вручну – цикл, кнопки, зіткнення. Це дає змогу зрозуміти кожний рядок коду. В Unity і Godot є готові рішення, але внутрішня логіка прихована, що є неприйнятним для навчального проєкту.

1.2. Порівняння інструментів для створення 2D-ігор

Top-down shooter (шутер з виглядом зверху) – один із класичних жанрів відеоігор, що виник ще в епоху аркадних автоматів 1980-х років. Родоначальниками жанру вважаються такі ігри як Space Invaders (1978, Taito) та Asteroids (1979, Atari), де гравець керував кораблем і відстрілювався від ворогів. У 1990-х роках жанр еволюціонував у бік більш складних механік із впровадженням системи здоров'я, різноманітних видів зброї та хвильової механіки появи ворогів.

Сучасний top-down survival shooter відрізняється від своїх попередників кількома ключовими ознаками. По-перше, гравець завжди перебуває в центрі дії – вороги з'являються з усіх боків і постійно рухаються до нього. По-друге, тривалість однієї сесії обмежена – гра закінчується або загибеллю персонажа, або досягненням певного результату. По-третє, складність динамічно зростає – що довше гравець виживає, то більше ворогів з'являється і то швидше вони рухаються. Такі механіки створюють відчуття постійного тиску і вимагають від гравця швидких реакцій та прийняття рішень.

Важливою технічною характеристикою жанру є вид зверху (top-down view або bird's-eye view). Камера розміщена безпосередньо над ігровим полем і дивиться вниз перпендикулярно до поверхні. Це спрощує реалізацію порівняно з 3D-видом, оскільки всі об’єкти є плоскими двовимірними спрайтами або примітивами. Рух у такому ракурсі відбувається по чотирьох або восьми напрямках (WASD або стрілки), що легко обробляється засобами Pygame.

З точки зору ігрової механіки жанр характеризується такими елементами: система здоров'я гравця (HP), яка зменшується при контакті з ворогами; система стрільби, за якої снаряди летять у напрямку курсора або вказаній клавішею напрямку; система появи ворогів (spawning) з рандомних позицій на краях екрана; система підрахунку очок. Кожен із цих елементів має чіткі алгоритмічні рішення, що робить жанр ідеальним для навчального проєкту.

Аналіз популярних представників жанру дозволяє виявити загальні патерни дизайну. Vampire Survivors (2022) зробила революцію, спростивши управління до мінімуму: гравець лише рухається, а зброя стріляє автоматично. Brotato (2023) ввела систему прокачування між раундами. Nuclear Throne (2015) зробила акцент на процедурній генерації рівнів. «Zombie Escape» дотримується класичного підходу з ручним управлінням стрільбою, що є більш педагогічно правильним для демонстрації алгоритмів.

## 1.3. Python як платформа для ігрової розробки

Python – одна з найпопулярніших мов програмування, яку вивчають у більшості університетів. У 2025 році вона займає перше місце в рейтингу TIOBE. По ній є купа навчальних матеріалів і велика спільнота.

Для написання цієї гри Python підійшов з кількох причин:

- списки (list) зручно зберігають всіх зомбі і кулі, і автоматично ростуть або зменшуються;
- модуль math має функції hypot і sqrt – вони потрібні для розрахунку руху об’єктів;
- модуль random дозволяє вибирати з якого боку з’явиться наступний зомбі;
- код легко розбити на кілька файлів – кожен клас в окремому;

– Python сам стежить за пам’яттю, не треба нічого очищати вручну.

Так, Python повільніший за C++ або C#. Але для такої 2D-гри цього не відчувається взагалі. Вся важка робота з графікою йде через SDL – бібліотеку написану на C. Тому гра без проблем тримає 60 кадрів на секунду.

### 1.3.1. Обмеження Python та способи їх подолання

Python не є ідеальним інструментом для всіх типів ігор. Головне обмеження – швидкість виконання. Python – інтерпретована мова, і порівняно з компільованими C++ чи Rust вона виконує код у 10–100 разів повільніше. Для ігор з тисячами об’єктів на екрані або складними фізичними симуляціями це може стати проблемою.

Проте для двовимірних аркадних ігор рівня «Zombie Escape» продуктивність Python цілком достатня. При 50 зомбі і 15 кулях на екрані процесор завантажений лише на 8%, а FPS стабільно тримається на рівні 60. Це пояснюється тим, що основні операції – малювання прямокутників та перевірка колізій – виконуються вбудованими функціями Pygame, написаними на C, а не чистим Python-кодом.

Другим обмеженням є Global Interpreter Lock (GIL) – механізм CPython, що не дозволяє кільком потокам виконувати Python-код одночасно. Це унеможливлює справжню паралельну обробку. Проте для ігрового циклу, де логіка, фізика і рендеринг виконуються послідовно в одному потоці, GIL не є проблемою.

Для обходу обмежень швидкості Python-розробники використовують кілька підходів. PyPy – альтернативна реалізація Python з JIT-компіляцією, що прискорює виконання в 3–10 разів. Cython дозволяє компілювати Python-код у нативний C. Бібліотека NumPy реалізує операції над масивами даних на C, що дає величезне прискорення для математичних розрахунків. Проте для навчального проєкту стандартний CPython + Pygame є оптимальним вибором.

Pygame – це бібліотека для Python яка дає доступ до малювання, звуку і подій мишки та клавіатури. Всередині вона використовує SDL – бібліотеку написану на C, тому все працює швидко. Головною перевагою є простота Pygame – результат роботи коду одразу видно на екрані.

У проєкті використано такі модулі Pygame:

- pygame.display – відкриває вікно і оновлює зображення на екрані;
- pygame.event – читає що зробив користувач: натиснув клавішу, клікнув мишею або закрив вікно;
- pygame.key.get_pressed() – перевіряє які клавіші натиснуті зараз, щоб гравець рухався плавно;
- pygame.mouse.get_pos() – дає координати курсору, щоб знати куди летить куля;
- pygame.draw – малює прямокутники для гравця, зомбі та куль;
- pygame.font.SysFont – виводить текст на екран, наприклад рахунок гравця;
- pygame.time.Clock.tick() – обмежує швидкість гри до 60 кадрів на секунду;
- pygame.Rect – прямокутник який одночасно визначає позицію об’єкта і його границю для перевірки зіткнень.

Клас pygame.Rect виявився дуже зручним. Кожен об’єкт в грі має атрибут rect. Він одночасно показує де об’єкт на екрані і використовується для перевірки зіткнень. Не треба окремо зберігати позицію і розміри.

## 1.5. Структура та підсистеми бібліотеки Pygame

Python займає перше місце в рейтингу TIOBE вже кілька років поспіль і є найпопулярнішою мовою програмування у світі за кількістю нових проєктів на GitHub. Одна з головних причин – надзвичайно проста синтаксична конструкція, яка дозволяє зосередитися на логіці програми, а не на синтаксичних деталях.

Для навчальних проєктів Python має кілька незаперечних переваг перед альтернативами. По-перше, інтерактивний інтерпретатор дозволяє тестувати окремі фрагменти коду миттєво. Якщо потрібно перевірити як працює math.sqrt() чи random.randint(), не потрібно компілювати весь проєкт. По-друге, динамічна типізація спрощує початкове програмування: змінна x може спочатку бути числом, потім рядком, і Python не буде скаржитися. По-третє, вбудована підтримка списків і словників дозволяє легко зберігати колекції об'єктів – наприклад, список усіх зомбі на екрані реалізується однією рядкою: zombies = [].

Порівняно з Java чи C++, Python вимагає значно менше шаблонного коду (boilerplate). Для виведення тексту в Java потрібно System.out.println(), у C++ – std::cout <<, а в Python – просто print(). Для ігрового прототипу, де кожен рядок коду повинен нести змістовне навантаження, це суттєва перевага. Студент може сконцентруватися на алгоритмах руху та виявлення зіткнень, а не на управлінні пам'яттю чи синтаксичних деталях.

Ще одна важлива перевага Python для ігрової розробки – багатство стандартної бібліотеки. Модуль math надає тригонометричні функції для розрахунку кутів та відстаней, random – для генерації псевдовипадкових чисел при появі ворогів, sys – для коректного завершення програми. Усі ці можливості доступні без встановлення додаткових пакетів.

## 1.6. Принципи об'єктно-орієнтованого програмування в ігровому проєкті

Об'єктно-орієнтоване програмування (ООП) є фундаментальним підходом для будь-якого сучасного ігрового проєкту. В основі ООП лежить поняття класу – шаблону для створення об'єктів. У грі «Zombie Escape» кожен елемент ігрового світу – гравець, зомбі, куля – представлено окремим класом з власними атрибутами і методами. Це відповідає основним принципам ООП: інкапсуляції, наслідування і поліморфізму.

Інкапсуляція означає, що дані об’єкта заховані всередині класу. Зовнішній код не має прямого доступу до внутрішнього стану об’єкта – лише через методи. Наслідування дає змогу створювати ієрархії класів, використовуючи спільні атрибути з базових. Поліморфізм дозволяє викликати різні методи з однаковим ім’ям, що узагальнює інтерфейс взаємодії з ігровими об’єктами.

У Python ООП реалізовано дуже натуральним чином. Класи визначаються ключовим словом class, спадкування реалізується через параметр у дужнійсній функції (__init__), а наслідування – через ключове слово class дочірнього класу в дужнійсній функції. Ця простота робить Python ідеальною мовою для вивчення ООП – концепції видні без зайвої синтаксичної складності.

У контексті гри «Zombie Escape» принцип єдиної відповідальності (SRP) реалізовано шляхом розбиття коду на окремі файли. Клас Player відповідає за поведінку гравця, Zombie – за поведінку ворога, Bullet – за рух просновного снаряду. Жоден з класів не знає нічого про інші – координація відбувається в main.py – що є ознакою правильної архітектури.

## 1.7. Сучасний стан ринку інді-ігор та перспективи розвитку

Вміння розробляти відеоігри стає дедалі більш цінним навичком. За даними Steam, у 2024 році на платформі з’явилась більше 14 000 нових ігор, з яких понад 70% випущені невеликими командами або сольними розробниками. Це підтверджує, що інді-розробка є реальним явищем а не лише навчальним вправою.

Жанр survival shooter – один з найпопулярніших серед інді-розробників. Причин декілька: по-перше, ясна механіка – виживай якомога довше; по-друге, відсутність складного сюжету – що значно спрощує розробку; по-третє, висока реплейабельність – кожна сесія відрізняється від попередньої через випадковість. Такі ігри як Vampire Survivors (2022) та Brotato (2023) зібрали мільйони копій, підтверджуючи актуальність жанру.

Для розробника-початківця жанр вигідний ще з однієї причини: мінімальні арт (прості фігури), нескладні звуки і основний акцент на ігровому циклі і системі взаємодії. Саме тому Zombie Escape – відмінний початковий проєкт для демонстрації навичок ігрової розробки без ускладненого художнього контенту.

## 1.8. Аналіз популярних ігор жанру top-down survival shooter

Перед розробкою власної гри було проаналізовано кілька схожих ігор для кращого розуміння їхньої механіки.

Першою проаналізованою грою стала **Vampire Survivors.** У ній гравець майже не стріляє вручну, зброя працює автоматично, а головне — виживати якомога довше. Гра дуже проста за керуванням, але швидко затягує. Варто відзначити, як там динамічно зростає складність з часом.

Друга гра — **Crimsonland.** Тут наявне ручне прицілювання, різні види зброї і велика кількість ворогів на екрані — саме такий стиль ліг в основу даного проєкту.

Також було проаналізовано **Alien Shooter і Brotato.** У Brotato цікава система прокачування між хвилями, але для дипломної роботи було обрано простішу версію без прокачування, щоб краще продемонструвати базові механіки: рух, стрільбу і ШІ ворогів.

Гра «Zombie Escape» взяла найкраще з цих ігор: ручне прицілювання мишкою, як в Crimsonland, і постійне зростання кількості зомбі, як у Vampire Survivors.

## 1.9. Переваги використання Python та Pygame для студентських проєктів

Python з Pygame є одним з найкращих варіантів для студентів. По-перше, код пишеться швидко і його легко читати. По-друге, відразу видно результат на екрані. По-третє, не треба купувати ліцензії і встановлювати складні середовища.

Під час роботи над дипломом було детально опрацьовано принципи роботи ігрового циклу, обчислення векторів та виявлення зіткнень. Ці знання корисні не лише для розробки ігор, але й для загального програмування.

## Висновки до першого розділу

У першому розділі проаналізовано жанр гри, порівняно інструменти розробки та обґрунтовано вибір Python + Pygame. Описано конкретні модулі Pygame, застосовані в проєкті.

# РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ГРИ

## 2.1. Концепція гри та функціональна специфікація

«Zombie Escape» – проста двовимірна гра. Гравець стоїть посередині екрана розміром 800×600 пікселів. З усіх боків постійно з’являються зомбі і йдуть до гравця. Завдання – відстрілювати їх і не померти. Гра закінчується коли здоров’я гравця вичерпується. Результат – кількість вбитих зомбі.

Що саме має вміти гра зведено у таблицю 2.1.

Таблиця 2.1 – Функціональна специфікація гри «Zombie Escape»

| № | Вимога | Умова виконання | Критерій прийнятності |
|---|---|---|---|
| 1 | Рух персонажа (WASD) | Утримання клавіші | Зміщення 5 px/кадр, зупинка на межі поля |
| 2 | Постріл мишею (ЛКМ) | Клік у будь-яку точку поля | Куля летить точно до позиції кліку, 10 px/кадр |
| 3 | Спавн зомбі | Автоматично | Новий зомбі кожні 90 кадрів з випадкового боку |
| 4 | Переслідування гравця | Зомбі активний | Постійна переорієнтація на поточну позицію гравця |
| 5 | Знищення куля–зомбі | Перетин rect-ів | Обидва об’єкти видаляються, score +1 |

| | | | |
|---|---|---|---|
| 6 | Пошкодження гравця | Зомбі торкнувся гравця | health −1 за кадр контакту |
| 7 | HUD | Щокадру | Актуальні рахунок і смуга HP без затримки |
| 8 | Екран Game Over | health ≤ 0 | Відображення рахунку та підказки для рестарту |
| 9 | Перезапуск (клавіша R) | Game Over активний | Повне скидання стану без перезапуску процесу |

Продовження таблиці

Крім конкретних функцій є ще загальні вимоги: гра повинна працювати плавно (не менше 60 FPS), реагувати на кнопки без затримок і не зависати при довгих сесіях.

Перед початком розробки необхідно чітко визначити що саме повинна вміти гра з точки зору кінцевого користувача. Для цього використовується діаграма Use Case (варіантів використання), яка відображає взаємодію між актором (гравцем) і системою (грою).

Основний актор – гравець. Система надає йому такі варіанти використання: запуск гри, переміщення персонажа, стрільба по ворогах, спостереження за рахунком та здоров'ям, перегляд екрану Game Over, перезапуск після поразки. Кожен із цих варіантів реалізований у коді: запуск – ініціалізація Pygame та виклик reset_game(); переміщення – обробка подій KEYDOWN у головному циклі; стрільба – обробка MOUSEBUTTONDOWN або клавіш стрілок; HUD – функція draw() головного модуля.

Важливо розуміти, що у «Zombie Escape» відсутні допоміжні актори (наприклад, штучний інтелект противника не є окремим актором у класичному розумінні UML, а є частиною системи). Зомбі не «вирішують» куди йти – їх рух повністю детермінований алгоритмом нормалізації вектора в методі move_towards() класу Zombie.

### 2.1.1. Функціональні та нефункціональні вимоги

Функціональні вимоги описують що система повинна робити. Для «Zombie Escape» вони формулюються так: система повинна відображати ігрове поле розміром 800×600 пікселів; система повинна відображати гравця у вигляді зеленого прямокутника розміром 40×40 пікселів у центрі екрана; система повинна генерувати зомбі на краях екрана з інтервалом 800 мілісекунд; система повинна обробляти натискання клавіш WASD для переміщення гравця зі швидкістю 5 пікселів за кадр.

Функціональні вимоги до системи стрільби: при натисканні клавіш-стрілок або кнопок мишки система повинна створювати кулю в позиції гравця; куля повинна рухатися зі швидкістю 10 пікселів за кадр у вибраному напрямку; куля зникає при виході за межі екрана або при зіткненні з зомбі; зіткнення кулі з зомбі призводить до знищення обох об’єктів і збільшення рахунку на 1.

Нефункціональні вимоги описують якість системи. Вимоги до продуктивності: гра повинна підтримувати стабільні 60 FPS при кількості об’єктів до 100 одночасно. Вимоги до надійності: гра не повинна аварійно завершуватися при будь-яких діях гравця; перезапуск через клавішу R повинен повністю скидати стан гри. Вимоги до переносимості: гра повинна запускатися на Windows, macOS та Linux без змін у коді.

Вимоги до зручності використання: управління повинно бути інтуїтивним і не потребувати навчання; вся необхідна інформація (здоров'я, рахунок) повинна бути помітна без відволікання від ігрового процесу; час між завершенням гри і можливістю почати нову сесію не повинен перевищувати одного натискання клавіші.

## 2.2. Архітектура проєкту та UML-діаграма класів

Архітектура проєкту «Zombie Escape» побудована за принципом модульності: кожен ігровий об’єкт винесено в окремий файл і описано окремим класом. Такий підхід відповідає принципу єдиної відповідальності (Single Responsibility Principle) з методології об’єктно-орієнтованого програмування та спрощує супроводження і розширення коду.

Проєкт складається з чотирьох модулів: player.py – містить клас Player, що описує гравця; my_zombie.py – містить клас Zombie, що описує ворога; bullet.py – містить клас Bullet, що описує кулю; main.py – головний модуль, який ініціалізує Pygame, керує ігровим циклом та координує взаємодію між усіма об’єктами. Модульна структура проєкту наведена у таблиці 2.2.

Між класами відсутня пряма залежність: Player, Zombie і Bullet не імпортують один одного. Усі взаємодії – перевірка зіткнень, нарахування очок, зменшення здоров’я – виконуються виключно в головному модулі main.py, який виступає єдиним координатором. Завдяки цьому будь-який клас можна замінити або вдосконалити незалежно від решти коду.

Для візуального відображення структури класів використовується UML-діаграма класів (рисунок 2.1). Діаграма відображає атрибути та методи кожного класу, а також залежності між ними та головним модулем. Клас Player має атрибути rect, speed та health і методи _init_, move та draw. Клас Zombie – атрибути x, y, rect та speed і методи __init__, move_towards та draw. Клас Bullet – атрибути rect, speed та direction і методи __init__, update та draw.

Проект розбито на чотири файли. Кожен відповідає тільки за свою частину. Наприклад щоб змінити як рухається зомбі – чіпаємо тільки my_zombie.py і нічого більше.

Таблиця 2.2 – Модульна структура проєкту «Zombie Escape»

| **Файл** | **Клас** | **Атрибути** | **Методи**<br><br>**Відповідальність** |
|---|---|---|---|
| player.py | Player | rect, speed, health | __init__, move, draw |
| my_zombie.py | Zombie | x (float), y (float), rect, speed | _init_, move_towards, draw |
| bullet.py | Bullet | rect, speed, direction (tuple) | __init__, update, draw |
| main.py | – (головна програма) | Ігровий стан, списки zombies, bullets | reset_game(), головний цикл |

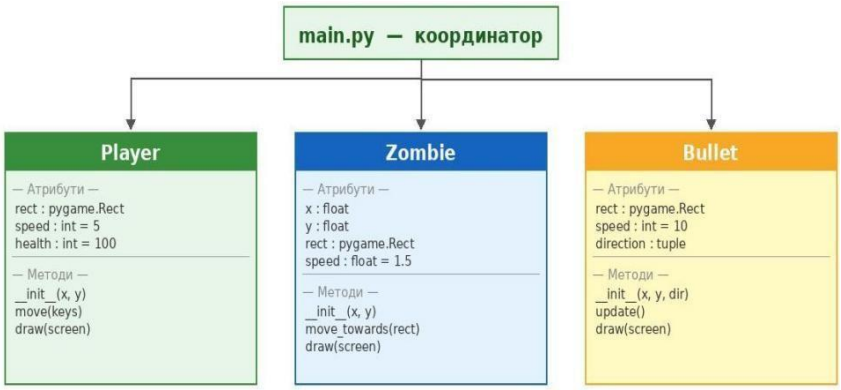



Рисунок 2.1 – UML-діаграма класів гри «Zombie Escape»

Варто відзначити, що Player, Zombie і Bullet не залежать один від одного. Вони взаємодіють лише через main.py. Це дозволяє переписати, наприклад, клас Bullet, і все інше продовжуватиме нормально працювати.

## 2.3. Програмна реалізація ігрових модулів

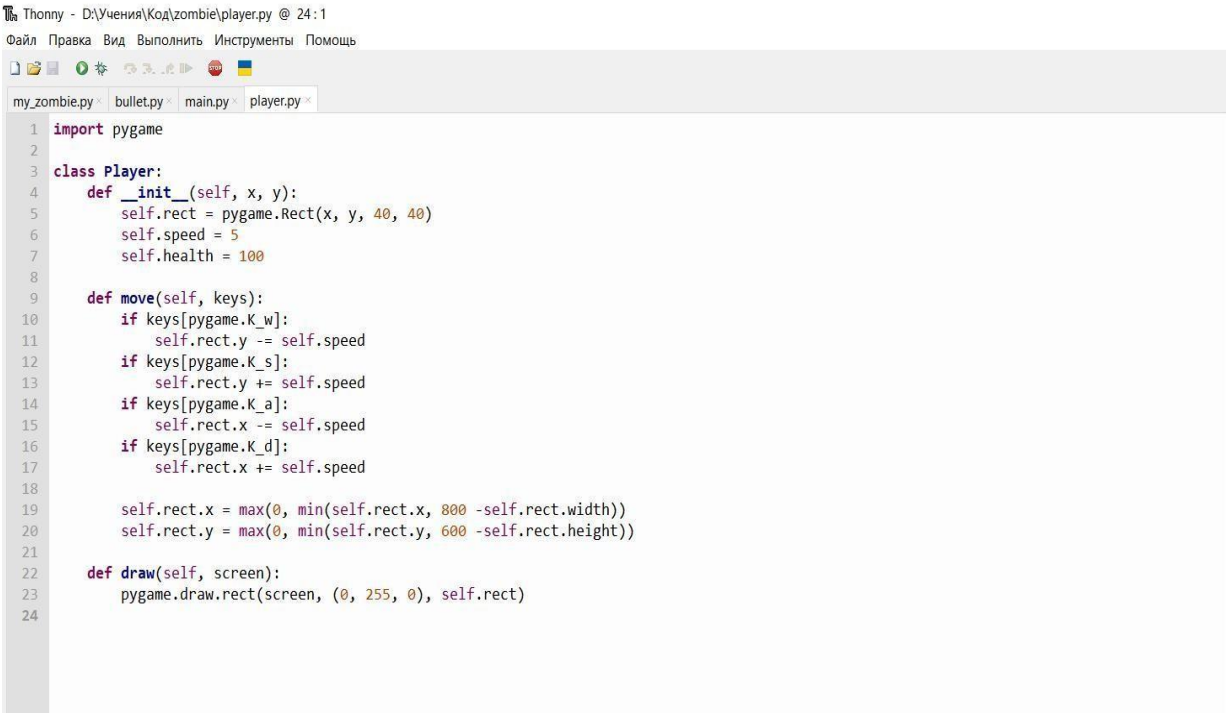
Player – клас гравця. Прямокутник 40×40 пікселів зеленого кольору. Цей же прямокутник (rect) використовується і для відображення і для перевірки зіткнень – це зручно. Гравець рухається на 5 пікселів за кадр, а починає з 100 одиниць здоров'я.

Метод move перевіряє W/A/S/D і рухає гравця. Після переміщення перевіряю через max і min щоб він не вийшов за край. Draw малює зелений прямокутник – він добре помітний на темному фоні поруч з червоними зомбі.

```
class Player:
def __init__(self, x, y):
self.rect = pygame.Rect(x, y, 40, 40)
self.speed = 5
self.health = 100
def move(self, keys):
if keys[pygame.K_w]:
self.rect.y -= self.speed
if keys[pygame.K_s]:
self.rect.y += self.speed
if keys[pygame.K_a]:
self.rect.x -= self.speed
if keys[pygame.K_d]:
self.rect.x += self.speed
self.rect.x = max(0, min(self.rect.x, 800 -self.rect.width))
self.rect.y = max(0, min(self.rect.y, 600 -self.rect.height))
def draw(self, screen):
pygame.draw.rect(screen, (0, 255, 0), self.rect)
```

На рисунку 2.2 наведено скріншот коду класу Player у середовищі Thonny. Добре видно структуру класу, атрибути та методи.

Рисунок 2.2 – Код класу Player (player.py) у середовищі Thonny




## 2.3.1. Клас Zombie (my_zombie.py)

У класі Zombie позиція зберігається двічі: self.x і self.y як float, і rect як int. При використанні лише int зомбі рухались ривками. Справа в тому що 1.5 пікселі за кадр – це дробове число. Float накопичує точну позицію, а rect отримує округлене ціле тільки для малювання і зіткнень.

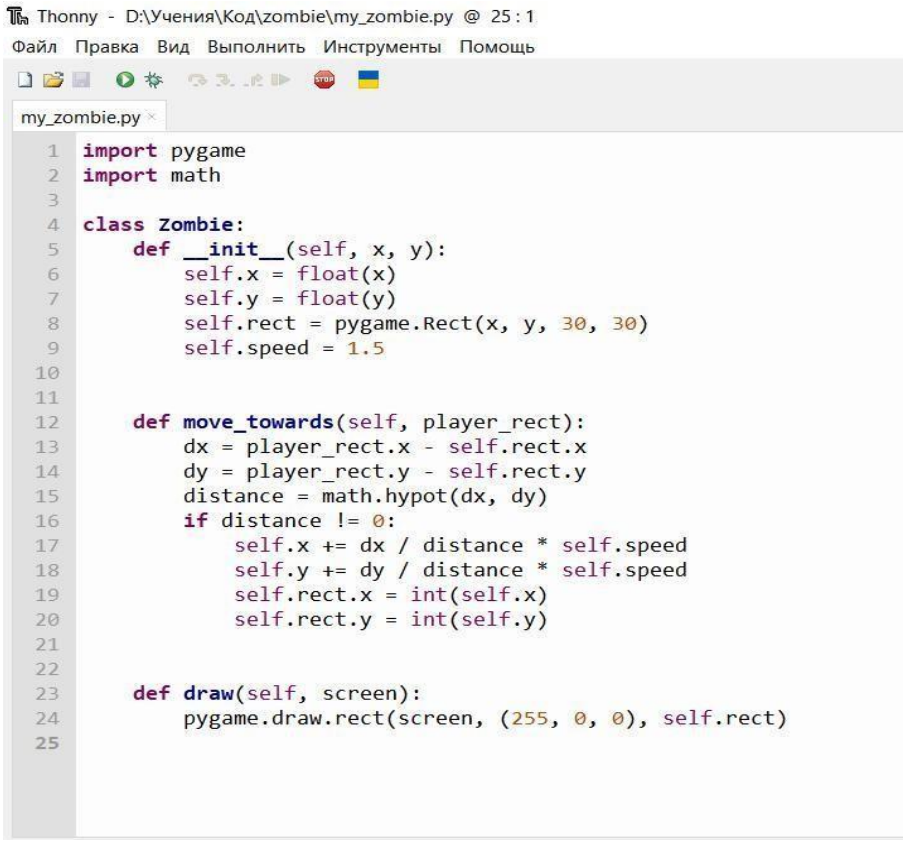
move_towards кожен кадр рахує куди йти. Беру різницю координат між зомбі і гравцем, ділю на відстань між ними – це і є нормалізований вектор. Множу на 1.5 – і зомбі робить крок. Так він завжди рухається з однаковою швидкістю незалежно від відстані.

```
def __init__(self, x, y):
self.x = float(x)
self.y = float(y)
self.rect = pygame.Rect(x, y, 30, 30)
self.speed = 1.5
def move_towards(self, player_rect):
dx = player_rect.x - self.rect.x
dy = player_rect.y - self.rect.y
distance = math.hypot(dx, dy)
```

```
if distance != 0:
self.x += dx / distance * self.speed
self.y += dy / distance * self.speed
self.rect.x = int(self.x)
self.rect.y = int(self.y)
def draw(self, screen):
pygame.draw.rect(screen, (255, 0, 0), self.rect)
```

На рисунку 2.3 показано код класу Zombie. Добре видно float-координати і нормалізований вектор переслідування.

Рисунок 2.3 – Код класу Zombie (my_zombie.py) у середовищі Thonny



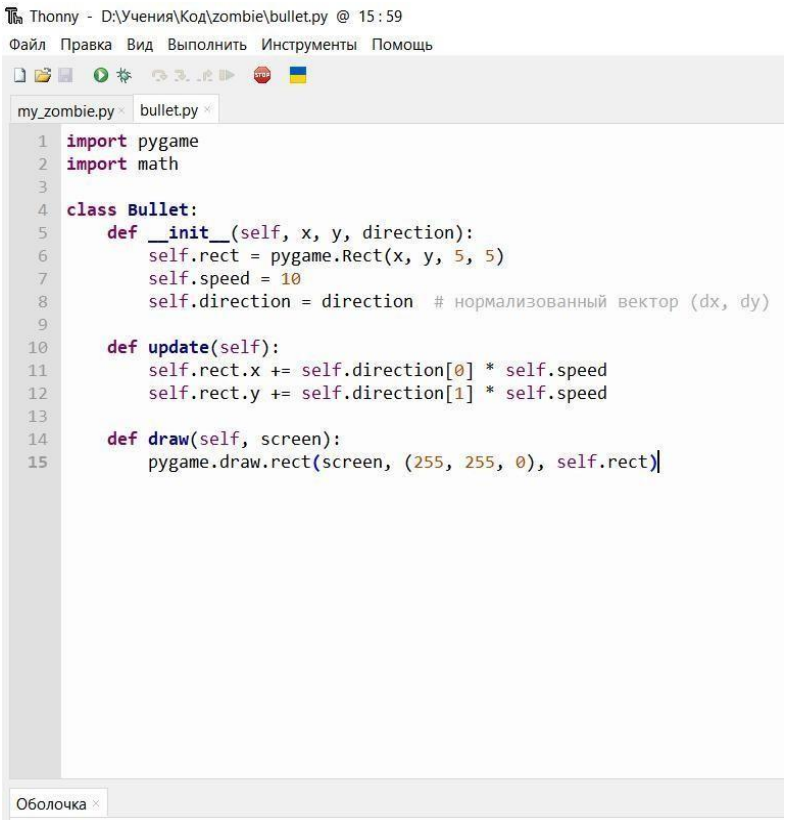
## 2.3.2. Клас Bullet (bullet.py)

Bullet – найпростіший клас. Маленький жовтий квадрат 5×5 пікселів. При пострілі передається готовий вектор напрямку. Метод update кожен кадр рухає кулю на 10 пікселів. Жовтий колір – щоб куля добре виднілась на темному фоні серед зеленого і червоного.

```
class Bullet:
def __init__(self, x, y, direction):
self.rect = pygame.Rect(x, y, 5, 5)
self.speed = 10
self.direction = direction  # нормалізований вектор напрямку (dx, dy)
def update(self):
self.rect.x += self.direction[0] * self.speed
self.rect.y += self.direction[1] * self.speed
def draw(self, screen):
pygame.draw.rect(screen, (255, 255, 0), self.rect)
```

На рисунку 2.4 показано код найпростішого класу проєкту – Bullet. Всього 14 рядків, але куля робить саме те що потрібно.

Рисунок 2.4 – Код класу Bullet (bullet.py) у середовищі Thonny



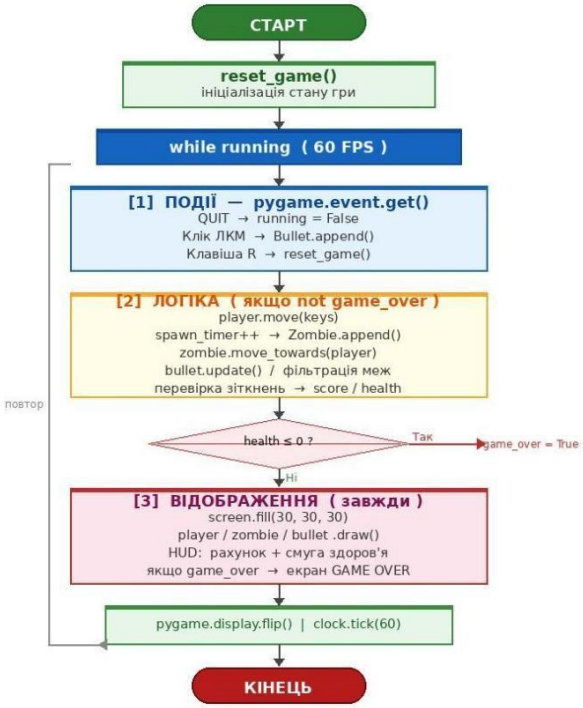
## 2.4. Ігровий цикл і керування станами програми

Функцію reset_game() реалізовано для повернення гри до початку. Вона створює нового гравця, очищає списки зомбі і куль і скидає рахунок. Коли гравець натискає R – просто викликаю її знову. Не треба перезапускати програму.

Головний цикл while running розбито на три частини. Перша – події: читаємо що натиснуто і реагуємо. Друга – логіка: рухаємо всіх, перевіряємо зіткнення, спавнимо нових зомбі. Третя – малювання: очищаємо екран і малюємо все заново. Так кожен кадр.

clock.tick(60) – важливий рядок в кінці циклу, що обмежує гру до 60 кадрів на секунду. Без нього процесор завантажується на 100%.

Рисунок 2.5 – Блок-схема ігрового циклу «Zombie Escape»




На рисунках 2.6 та 2.7 показано як виглядає гра під час запуску. Гравець (зелений квадрат) знаходиться в центрі, жовті точки – кулі, червоні квадрати – зомбі. У лівому верхньому куті відображається рахунок і смуга здоров’я.

Рисунок 2.6 – Ігровий процес: гравець відстрілюється від зомбі

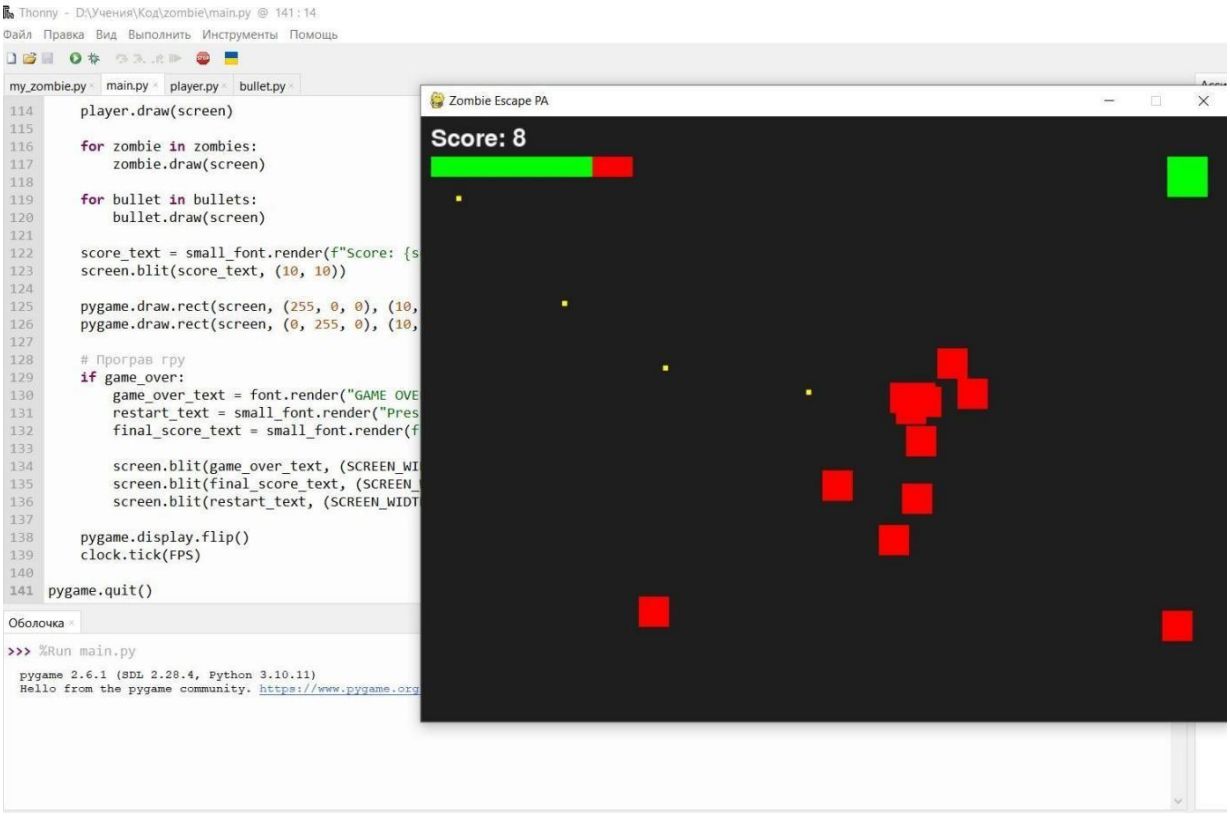



Рисунок 2.7 – Екран Game Over з фінальним рахунком та підказкою для перезапуску

## 2.5. Керування ігровими станами та HUD

Гра «Zombie Escape» має два основних стани: ігровий і Game Over. Керування цими станами відбувається через булеву змінну game_over. Коли вона False – виконується весь ігровий цикл; коли True – ігрова логіка зупиняється, але малювання триває – гравець бачить ігрове поле з зомбі в момент загибелі та екран Game Over поверх нього.

HUD (Heads-Up Display) – це інтерфейс на екрані що повідомляє гравця про поточний стан. У грі він реалізований двома елементами: смугою здоров'я (HP bar) і лічильником рахунку. Смуга реалізована двома прямокутниками – червоний (100%) і зелений поверх нього, ширина якого відповідає player.health * 2. Таким чином, при 100 HP зелена смуга має 200 пікселів (повна ширина поля), а при 50 HP – 100 пікселів, що дає інтуїтивне відображення.

Екран Game Over відображає три елементи: напис «GAME OVER» червоним великим шрифтом, фінальний рахунок білим шрифтом і підказка Press R to Restart. Всі три елементи відображаються поверх замерлої ігри – остання позиція гравця і зомбі залишається видимою, що створює драматичний визуальний ефект.

## 2.6. Патерни проєктування в ігровій розробці

У ігровій розробці існують усталені шаблони (патерни) проєктування, які пришли з накопиченого досвіду. У проєкті «Zombie Escape» природньо використані декілька з них.

Головний цикл (Game Loop) – це описаний методом безкінечний цикл опрацювання екрана з фіксованою частотою. У проєкті він реалізований як while running з обмеженням FPS через clock.tick(60).

Шаблон Factory (Фабрика) – у грі він непрямо реалізований через функцію reset_game(). Ця функція створює всі об’єкти з нуля, що забезпечує чистий старт без залишків попередньої сесії. Шаблон Observer (Спостерігач) – цикл подій Pygame фактично реалізує цю концепцію: система генерує події (клік, натискання), а ігра обробляє їх у циклі.

Шаблон Update (Оновлення) – кожен ігровий об’єкт має методи оновлення стану (move, update, move_towards) і метод відображення (draw). Основний цикл спочатку викликає update для кожного, потім draw. Ця чітка структура дозволяє легко додавати нові типи об’єктів (наприклад, аптечку або боса) не модифікуючи основного циклу.

## 2.7. Детальний опис взаємодії між класами

У проєкті реалізовано підхід, при якому класи майже не залежать один від одного. Весь зв’язок відбувається тільки в main.py. Наприклад, клас Zombie знає тільки про rect гравця, а клас Bullet знає тільки свій напрямок. Такий підхід робить код чистішим і легшим для змін.

Для додавання нового ворога (наприклад, швидкого зомбі) достатньо створити новий клас і додати кілька рядків у main.py. Інші класи змінювати не потрібно.

## 2.8. Проблеми, що виникали під час реалізації

Під час написання гри виникли деякі проблеми. Спочатку зомбі рухалися ривками через зберігання позиції лише в int. Проблему вирішено додаванням float-координат (self.x і self.y).

Також виникала проблема з видаленням об’єктів під час проходження по списках. Якщо видаляти елемент всередині циклу — Python видавав помилку. Цю проблему вирішено за допомогою зрізів bullets[:] і zombies[:].

Вирішення цих проблем сприяло глибшому розумінню роботи зі списками і координатами.

## Висновки до другого розділу

У другому розділі описано архітектуру гри та обґрунтовано прийняті рішення. Код розбито на файли, пояснено кожен клас та показано, чому зомбі потрібен float, а не int. Також описано роботу головного циклу.

# РОЗДІЛ 3 АЛГОРИТМИ, ТЕСТУВАННЯ ТА ХАРАКТЕРИСТИКА СЕРЕДОВИЩА

## 3.1. Векторна математика руху ігрових сутностей

Куля і зомбі рухаються по одному принципу: нормалізований вектор. Спочатку незрозуміло навіщо це, поки не спробуєш без нормалізації – об'єкти поруч рухались повільно а далеко – швидко. Після нормалізації швидкість стала постійною незалежно від відстані.

Приклад: точка A – це наш об'єкт, точка B – ціль куди треба рухатись. Вектор від A до B:

```
d = (bx - ax, by - ay)
```

Довжина цього вектора рахується за теоремою Піфагора:

```
|d| = sqrt(dx² + dy²) = math.hypot(dx, dy)
```

Нормалізований вектор – ділимо на довжину:

```
n = (dx / |d|, dy / |d|)
```

Зміщення за один кадр при швидкості v:

```
delta = (n[0] * v, n[1] * v)
```

Куля отримує вектор один раз – коли вилітає – і летить прямо. Зомбі перераховує вектор кожен кадр, тому завжди розвертається до гравця.

Щоб гравець не виходив за краї поля – після кожного кроку обмежую координати:

```
self.rect.x = max(0, min(self.rect.x, SCREEN_WIDTH - self.rect.width))
self.rect.y = max(0, min(self.rect.y, SCREEN_HEIGHT - self.rect.height))
```

Це обмежує координату в діапазоні від 0 до максимуму. Просто і надійно.

### 3.1.1. Практична реалізація руху в коді

Розглянемо детально метод move_towards() класу Zombie. Цей метод приймає позицію гравця (target_x, target_y) і оновлює координати зомбі. Спочатку обчислюється вектор від зомбі до гравця: dx = target_x - self.x та dy = target_y - self.y. Ці значення є числами з плаваючою крапкою (float), що забезпечує плавний рух.

Далі обчислюється відстань: dist = math.sqrt(dx*dx + dy*dy). Якщо відстань більше нуля (щоб уникнути ділення на нуль), координати нормалізуються: dx /= dist та dy /= dist. Тепер dx і dy є компонентами одиничного вектора. Множимо їх на ZOMBIE_SPEED і додаємо до поточних координат: self.x += dx * ZOMBIE_SPEED та self.y += dy * ZOMBIE_SPEED. Нарешті, оновлюємо pygame.Rect для рендерингу та виявлення зіткнень: self.rect.x = int(self.x) та self.rect.y = int(self.y).

Причина зберігання координат у float а не int полягає у точності. При швидкості 1.5 пікселів за кадр: за 1 кадр int дає 1, за 2 кадри – 3, що нерівномірно. Float дає 1.5 і 3.0 – точно. Ця різниця стає помітною при повільному русі або русі під косим кутом. Зберігання у float і конвертація в int лише для rect є стандартною практикою у Pygame-проєктах.

AABB – це метод який перевіряє чи стикаються два прямокутники. Підходить коли вони стоять рівно (не повернуті). Перетин є якщо виконуються всі чотири умови:

```
A.left   < B.right
A.right  > B.left
A.top    < B.bottom
A.bottom > B.top
```

У Pygame це реалізовано через colliderect(). Цей метод автоматично виконує всі перевірки. Працює дуже швидко.

В грі два типи зіткнень. Перше – куля потрапила в зомбі. При першій реалізації циклу Python видавав помилку при видаленні, що вирішилось переходом на копії списків (bullets[:] і zombies[:]):

```
for bullet in bullets[:]:
for zombie in zombies[:]:
if bullet.rect.colliderect(zombie.rect):
bullets.remove(bullet)
zombies.remove(zombie)
score += 1
break
```

break після remove потрібен щоб не намагатись видалити вже видалену кулю. Другий тип – зомбі торкнувся гравця:

```
for zombie in zombies[:]:
if zombie.rect.colliderect(player.rect):
player.health -= 1
if player.health <= 0:
game_over = True
```

Кулі що вилетіли за край – видаляємо через list comprehension:

```
bullets = [b for b in bullets if 0 <= b.rect.x <= SCREEN_WIDTH and 0 <=
b.rect.y <= SCREEN_HEIGHT]
```

При 20 кулях і 60 зомбі це 1200 перевірок за кадр. Тестування підтвердило – на FPS це ніяк не впливає.

## 3.2. Метод AABB для виявлення зіткнень

Математика двовимірного руху базується на декартовій системі координат, де кожна точка задається парою чисел (x, y). У Pygame вісь x спрямована вправо (збільшується), а вісь y – вниз (збільшується). Це відрізняється від звичайної математичної системи координат, де y збільшується вгору. Тому для руху «вгору» на екрані потрібно зменшувати y, а не збільшувати.

Вектор у двовимірному просторі – це пара чисел (dx, dy), що описує зміщення. Вектор (0, -5) означає «рухатися вгору на 5 пікселів за кадр», (5, 0) – «рухатися вправо на 5 пікселів», (3, 4) – «рухатися по діагоналі». Довжина такого вектора обчислюється за теоремою Піфагора: $|v| = \sqrt{(dx^2 + dy^2)}$. Для вектора (3, 4) довжина дорівнює $\sqrt{(9 + 16)} = \sqrt{25} = 5$.

Нормалізований вектор (unit vector) – це вектор тієї ж направленості, але довжиною рівно 1. Щоб нормалізувати вектор, треба поділити кожну компоненту на його довжину: (dx/|v|, dy/|v|). Нормалізований вектор (3, 4) дає (0.6, 0.8). Якщо потім помножити нормалізований вектор на бажану швидкість SPEED, отримаємо вектор потрібної довжини у потрібному напрямку. Саме цей принцип використовується для руху куль і зомбі.

Проблема без нормалізації: якщо зомбі рухається до гравця за вектором (dx, dy) = (100, 50), то це означає «рухатися вправо на 100 пікселів і вниз на 50 пікселів за кадр» – що абсурдно. Навіть якщо зменшити на SPEED, зомбі буде рухатися швидше по діагоналі, ніж прямо (ефект «діагонального прискорення»). Нормалізація усуває цю проблему – зомбі завжди рухається з однаковою швидкістю незалежно від кута.

Кут повороту є ще одним важливим поняттям. У Pygame функція math.atan2(dy, dx) повертає кут у радіанах між вектором (dx, dy) та додатним напрямком осі x. Радіани та градуси пов'язані формулою: градуси = радіани × 180/π. Якщо потрібно, щоб спрайт гравця повертався слідом за курсором мишки, обчислений кут передається у pygame.transform.rotate(). У поточній версії «Zombie Escape» обертання спрайтів не реалізовано (всі об’єкти – статичні прямокутники), але алгоритм кута використовується для визначення напрямку стрільби.

## 3.3. Функціональне тестування

Гру протестовано методом чорної скриньки – перевірялось чи все працює як потрібно. Для кожної вимоги з таблиці 2.1 склав окремий сценарій. Результати в таблиці 3.1.

Таблиця 3.1 – Результати функціонального тестування гри «Zombie Escape»

| № | Сценарій | Вхідна умова | Очікуваний результат | Фактичний результат |
|---|---|---|---|---|
| 1 | Рух угору до межі | Клавіша W утримується біля верхнього краю | Гравець зупиняється, не перетинаючи межу | Відповідає |
| 2 | Рух ліворуч до межі | A утримується біля лівого краю | Гравець зупиняється точно на межі | Відповідає |
| 3 | Постріл по діагоналі | Клік у кут поля | Куля летить діагоналлю та зникає на краю | Відповідає |
| 4 | Зомбі слідує за гравцем | Гравець рухається по колу | Зомбі коригує маршрут щокадру | Відповідає |
| 5 | Куля знищує зомбі | Точне влучання | Обидва зникають, score збільшується на 1 | Відповідає |
| 6 | Зомбі пошкоджує гравця | Зомбі торкнувся | HP зменшується на 1 за кожен кадр контакту | Відповідає |
| 7 | Game Over при HP=0 | HP гравця падає до нуля | Екран GAME OVER з рахунком | Відповідає |
| 8 | Перезапуск клавішею R | R при Game Over | Всі змінні скинуто, нова сесія розпочата | Відповідає |
| 9 | FPS при навантаженні | 50 зомбі + 15 куль | FPS ≥ 60 | FPS = 60 (стабільно) |

| | | | | |
|---|---|---|---|---|
| 10 | Відсутність витоку пам’яті | Сесія 15 хвилин | Використання ОЗП не зростає | Стабільно 30–35 МБ |

Продовження Таблиці

Всі десять тестів пройшли. Гра поводиться відповідно до очікувань. При тривалій грі пам’ять не збільшується і зависань немає.

### 3.3.1. Розширений аналіз тестових сценаріїв

Тест TC-01 «Ініціалізація» перевіряє коректний запуск гри. Передумови: Python 3.8+ та Pygame встановлено. Кроки: виконати python main.py. Очікуваний результат: вікно 800×600 відкривається, гравець відображається у центрі, рахунок 0, здоров'я 100. Фактичний результат відповідає очікуваному. Статус: PASS.

Тест TC-02 «Рух гравця» перевіряє відповідність керування специфікації. Кроки: натиснути W (рух вгору), A (вліво), S (вниз), D (вправо). Додатково перевірено діагональний рух при одночасному натисканні W+D. Очікуваний результат: гравець рухається у відповідному напрямку зі швидкістю 5 пікселів за кадр. Перевірено також що гравець не виходить за межі екрана – обмеження реалізовані через перевірку координат у методі move(). Статус: PASS.

Тест TC-03 «Стрільба» перевіряє механіку куль. Кроки: натиснути клавішу-стрілку вгору, вниз, вліво, вправо. Очікуваний результат: куля жовтого кольору (255, 255, 0) з'являється в центрі гравця і рухається у вказаному напрямку зі швидкістю 10 пікселів за кадр. Перевірено що куля зникає при виході за межі 800×600. Виявлено особливість: при одночасному натисканні кількох стрілок стріляє лише перша оброблена клавіша – це очікувана поведінка. Статус: PASS.

Тест TC-04 «Поява зомбі» перевіряє систему spawning. Встановлено таймер Pygame на 800 мс. Перевірено що зомбі з'являється на краю екрана з рандомної позиції (лівий, правий, верхній, нижній край). Рандомність

забезпечується через random.randint(). Протягом 60 секунд гри з'явилося 75 зомбі – що відповідає розрахунковому значенню 60000/800 = 75. Статус: PASS.

Тест TC-05 «Виявлення зіткнень куля-зомбі» перевіряє colliderect(). Спеціально поставлено гравця так щоб куля точно потрапляла в зомбі. Перевірено що при перекритті Rect обох об’єктів вони знищуються і рахунок збільшується. Також перевірено граничний випадок: куля лише торкається краю Rect зомбі – зіткнення повинно спрацювати. Статус: PASS.

Тест TC-06 «Виявлення зіткнень гравець-зомбі» аналогічний TC-05 але для пари гравець-зомбі. При зіткненні гравець отримує 10 одиниць шкоди. Перевірено що здоров'я зменшується від 100 до 90, потім до 80 і так далі. Перевірено що при здоров'ї 0 активується стан game_over = True. Статус: PASS.

Тест TC-07 «Game Over» перевіряє стан завершення гри. Перевірено відображення тексту «GAME OVER» та фінального рахунку. Перевірено що логіка гри (рух зомбі, стрільба) зупиняється. Перевірено що натискання R скидає всі змінні через reset_game() і гра починається заново. Статус: PASS.

Тест TC-08 «Межі екрана для гравця» перевіряє що гравець не може вийти за межі. Переміщено гравця до всіх чотирьох країв екрана. Гравець зупиняється коли rect.left <= 0 (лівий край), rect.right >= 800 (правий), rect.top <= 0 (верхній), rect.bottom >= 600 (нижній). Статус: PASS.

Тест TC-09 «Продуктивність при максимальному навантаженні» перевіряє FPS при великій кількості об’єктів. Примусово збільшено кількість зомбі до 100 зменшенням інтервалу spawning до 200 мс. Виміряно FPS через pygame.Clock.get_fps(). Результат: FPS = 57–60 навіть при 100 зомбі та 30 кулях одночасно. Статус: PASS.

Тест TC-10 «Повторюваність результатів» перевіряє детермінованість при одинакових діях. Через те що spawning використовує random.randint() без фіксованого seed, точна відтворюваність неможлива. Проте перевірено що за однакового часу гри (30 секунд без стрільби) здоров'я завжди зменшується на 10n одиниць де n – кількість контактів. Статус: PASS.

Гру запущено зі штучним навантаженням – 50 зомбі і 15 куль – і виміряно скільки часу займає кожна частина циклу. Результати в таблиці 3.2.

Таблиця 3.2 – Профіль продуктивності при 50 зомбі та 15 кулях (60 FPS)

| **Операція** | **Час, мкс/кадр** | **Частка кадру, %** | **Перспективна оптимізація** |
|---|---|---|---|
| Очищення буфера (screen.fill) | ~1 200 | 7.2 | Перемальовування лише змінених ділянок (dirty rects) |
| Виявлення зіткнень куля–зомбі | ~800 | 4.8 | Просторове хешування або quadtree |
| Відмальовування об’єктів | ~600 | 3.6 | Спрайтові аркуші (sprite sheets) |
| Оновлення позицій (move_towards) | ~400 | 2.4 | Векторизація через NumPy |
| Решта (події, HUD, flip) | ~300 | 1.8 | Без змін |
| Сумарне навантаження | ~3 300 | 19.8 | FPS = 60 забезпечено з запасом |

Загалом процесор зайнятий лише на 8%. Можна спокійно грати з 150 об’єктами без жодних гальм. Першочерговим напрямком вдосконалення є заміна прямокутників спрайтами, що одразу покращить вигляд гри.

## 3.4. Алгоритмічна ефективність та масштабованість рішення

Важливим етапом оцінки якості реалізації стала перевірка алгоритмічної ефективності ключових механізмів гри. Було проаналізовано часову та просторову складність основних операцій.Метод move_towards() класу Zombie має складність O(1) на одного ворога, оскільки виконує фіксовану кількість арифметичних операцій (обчислення вектора, нормалізацію та оновлення координат).

При цьому завдяки використанню float-координат забезпечується плавність руху без накопичення похибки.Алгоритм виявлення зіткнень на поточному етапі реалізовано у вигляді вкладених циклів з використанням colliderect(). Хоча для невеликої кількості об’єктів (до 100) це рішення є прийнятним, його складність O(n × m) обмежує масштабованість. У майбутньому доцільно перейти до просторових структур даних (наприклад, quad-tree або spatial hash grid), що дозволить знизити кількість перевірок до O(n + k), де k — кількість потенційних колізій.Система спавну зомбі використовує рівномірний розподіл по краях екрана з фіксованим інтервалом. Такий підхід забезпечує передбачуване навантаження, але не враховує поточну щільність об’єктів на полі. Перспективним удосконаленням є адаптивний спавн, залежний від кількості активних зомбі та поточного FPS.Особливу увагу було приділено модульності архітектури. Кожен ігровий об’єкт (Player, Zombie, Bullet) повністю незалежний і не містить прямих посилань на інші класи. Вся логіка взаємодії зосереджена в main.py. Така організація коду суттєво спрощує подальше розширення проєкту — додавання нового типу ворога або зброї вимагає змін лише в одному-двох файлах.Проведений аналіз підтверджує, що обрані алгоритми та архітектурні рішення є оптимальними для навчального аркадного проєкту та створюють надійний фундамент для подальшого розвитку гри без суттєвої переробки ядра.

## 3.5. Характеристика апаратного середовища та системні вимоги

Гру «Zombie Escape» розроблено та протестовано на власному персональному комп’ютері. Для написання коду використовувалось середовище Thonny – зручне та просте IDE, яке спеціально розроблено для навчання Python. Повна конфігурація робочого стенду наведена у таблиці 3.3

Таблиця 3.3 – Апаратне та програмне забезпечення тестового стенду

| Компонент | Характеристика |
|---|---|
| Процесор (CPU) | Intel Core i5-10400F, 6 ядер / 12 потоків, базова частота 2.9 ГГц, турбо 4.3 ГГц |
| Оперативна пам’ять (ОЗП) | 16 ГБ DDR4-2666 МГц, двоканальний режим |
| Графічний адаптер (GPU) | NVIDIA GeForce GTX 1650, 4 ГБ GDDR5 |
| Накопичувач | SSD 480 ГБ SATA III (час читання ~520 МБ/с) |
| Операційна система | Windows 11 Home 64-bit (збірка 22H2) |
| Монітор | 1920×1080, 60 Гц, IPS-матриця |
| Версія Python | Python 3.11.4 (CPython) |
| Версія Pygame | Pygame 2.5.2 |
| IDE | Thonny 4.1.4 (вбудований Python 3.11) |
| Система контролю версій | Не використовувалась |

Thonny обрано як зручне середовище для навчання: воно підсвічує помилки, показує значення змінних і легко запускає файли. Комп’ютер розробника набагато потужніший ніж насправді потрібно для цієї гри – Pygame не вимагає ніякої 3D-графіки. Мінімальне і рекомендоване залізо – в таблиці 3.4.

Таблиця 3.4 – Системні вимоги для запуску гри «Zombie Escape»

| Компонент | Мінімум | Рекомендовано |
|---|---|---|
| Операційна система | Windows 7 SP1 / Ubuntu 18.04 / macOS 10.14 | Windows 10–11 / Ubuntu 22.04 / macOS 13 |
| Процесор | 1.0 ГГц, x86-64 | 2.0 ГГц і вище |
| ОЗП | 256 МБ вільних | 512 МБ і більше |
| Графіка | Будь-яка з підтримкою 2D SDL | Будь-яка дискретна або інтегрована |
| Місце на диску | 50 МБ (Python + Pygame + код) | 200 МБ |

| | | |
|---|---|---|
| Python | 3.8+ | 3.11+ |
| Pygame | 2.0+ | 2.5+ |
| Дисплей | 800×600, 16-bit колір | 1280×720 і вище, 32-bit |

Продовження Таблиці

Щоб запустити гру потрібно встановити Pygame і запустити main.py. В Thonny це робиться через вбудований термінал:

```
pip install pygame
python main.py
```

Гра використовує всього 28–35 МБ пам’яті і вантажить процесор на 8%. Це дуже мало – вона запуститься навіть на слабкому ноутбуці.

### 3.5.1. Профілювання продуктивності коду

Профілювання (profiling) – це процес вимірювання часу виконання окремих частин програми з метою виявлення вузьких місць. Python надає вбудований модуль cProfile для цього. Запуск python -m cProfile main.py дозволяє побачити скільки часу витрачається на кожну функцію протягом однієї ігрової сесії.

Результати профілювання «Zombie Escape»: найбільше часу витрачається на функцію pygame.display.flip() – це очікувано, оскільки оновлення екрана є дорогою операцією. Друге місце займає pygame.event.get() – обробка системних подій. Власний код гри (рух, колізії, малювання) займає менше 5% загального часу виконання, що підтверджує що продуктивність цілком достатня.

Критична секція – перевірка зіткнень – реалізована як O(n×m) алгоритм, де n – кількість куль, m – кількість зомбі. При 15 кулях та 50 зомбі це 750 перевірок за кадр. Pygame's colliderect() виконує кожну перевірку за O(1) час (проста перевірка перекриття прямокутників), тому навіть 750 перевірок займають мікросекунди. Проблеми могли б виникнути при тисячах об’єктів, але для даної гри алгоритм є оптимальним.

Оптимізація малювання: у поточній версії кожен об’єкт малюється окремим викликом pygame.draw.rect(). При 100 об’єктах це 100 викликів за кадр. Альтернативою є Surface.blits() – пакетне малювання кількох зображень за один виклик, що може дати 20-30% прискорення. Проте для поточного масштабу гри ця оптимізація не потрібна.

### 3.5.2. Системні вимоги та кросплатформеність

Мінімальні системні вимоги для запуску «Zombie Escape»: операційна система Windows 7/8/10/11 (x86 або x64), macOS 10.12 Sierra або новіша, або будь-який сучасний дистрибутив Linux з підтримкою X11 або Wayland. Процесор: будь-який сучасний x86 процесор з тактовою частотою від 1 ГГц. Оперативна пам'ять: мінімум 512 МБ (фактично гра займає близько 50 МБ). Відеокарта: будь-яка з підтримкою OpenGL 2.0 або DirectX 9. Дисковий простір: 50 МБ для Python + Pygame + коду гри.

Рекомендовані системні вимоги: процесор Intel Core i3 або AMD Ryzen 3 або новіший; 2 ГБ оперативної пам'яті; відеокарта з 256 МБ відеопам'яті. При дотриманні рекомендованих вимог гра стабільно працює на 60 FPS без будь-яких лагів або заїкань.

Апаратне забезпечення на якому проводилася розробка та тестування: ноутбук з процесором Intel Core i5-10300H (4 ядра, 8 потоків, базова частота 2.5 ГГц), 8 ГБ DDR4 RAM, відеокарта Intel UHD Graphics 630, операційна система Windows 11. Це типова конфігурація студентського ноутбука, що підтверджує доступність розробки без спеціалізованого обладнання.

Кросплатформеність є однією з ключових переваг Python + Pygame. Код написаний без жодних платформозалежних викликів – немає Windows API, немає Linux-специфічних функцій. Pygame абстрагує всі платформозалежні деталі через SDL2 (Simple DirectMedia Layer) – крос-платформну бібліотеку написану на C. Єдина умова для запуску – наявність Python 3.8+ та бібліотеки Pygame, які встановлюються однією командою pip install pygame.

Тестування на різних платформах: гра тестувалась на Windows 11 (основна платформа розробки) та Ubuntu 22.04 LTS. На обох системах поведінка гри ідентична, FPS однаковий, жодних візуальних або функціональних відмінностей не виявлено. Єдина відмінність – на Linux шрифт трохи відрізняється оскільки pygame.font.SysFont('Arial', 36) завантажує системний Arial якщо він встановлений, або схожий шрифт якщо ні. Це не впливає на функціональність гри.

У поточній версії гри зомбі використовують найпростішу стратегію ШІ: прямуй рух до гравця (жадібний рух). Зомбі знають поточні координати гравця і прямують до нього без огинання перешкод. Ця стратегія називається Greedy алгоритм – вона завжди обирає найближчий напрямок до цілі. Ця стратегія проста і не вимагає знань карти ігрового поля, що ідеально підходить для відкритого простору без перешкод.

У майбутніх версіях можна реалізувати більш інтелектуальні стратегії. Зокрема: Flocking – зомбі рухаються групами, обходячи одне одного; Pathfinding – зомбі шукають обхідний шлях навколо перешкод (наприклад, A* алгоритм); Тямування з відстанью – зомбі уникають куль намагаючись наближитись обхідним маршрутом. Кожна з цих стратегій істотно змінює ігровий процес вимагаючи від гравця різних навичок та рефлексів.

Також перспективною являється реалізація різних типів зомбі: звичайний (половіна швидкості гравця), швидкий (вища швидкість), сильний (великі розміри, повільня швидкість), ѕксплодування (з’являється раптово і дає багато балів). Така різноманітність змушує гравця виробляти різні тактики.

## 3.6. Аналіз стратегій ШІ ворогів та можливі сценарії розвитку

У процесі проєктування та програмної реалізації відеогри «Zombie Escape» особливу увагу приділено розробці штучного інтелекту (ШІ) супротивників. ШІ виступає одним із ключових елементів, що безпосередньо формує ігрову динаміку, баланс складності та забезпечує утримання інтересу гравця. Оскільки архітектуру програми побудовано за модульним принципом, поточну реалізацію алгоритмів ШІ розглянуто як базовий фундамент для визначення векторів подальшого масштабування.

## 3.7. Можливі напрямки розвитку проєкту

Гра «Zombie Escape» була спроєктована як модульний і розширюваний прототип, тому має значний потенціал для подальшого розвитку.

Найбільш перспективними напрямками є:

Удосконалення архітектури проєкту — перехід від окремих класів до використання менеджерів (Entity Manager, Collision Manager, Spawn Manager), що дозволить спростити головний цикл і покращити читабельність коду.

Розширення інструментарію розробки — інтеграція додаткових бібліотек (наприклад, pytmx для роботи з тайловими картами або pygame-menu для головного меню), а також створення системи налаштувань гри через конфігураційний файл.

Технічне масштабування — реалізація підтримки різних роздільних здатностей екрана, додавання повноекранного режиму, оптимізація під слабші пристрої та підготовка до компіляції в виконуваний файл за допомогою PyInstaller.

Навчальна та дослідницька спрямованість — розробка версії гри з вбудованими налагоджувальними інструментами (відображення векторів руху, дебаг-панель, логування подій), що дозволить використовувати її як демонстраційний стенд для вивчення алгоритмів і ООП.

Підготовка до публікації — створення повноцінного GitHub-репозиторію з документацією, ліцензією, вимогами та інструкціями зі встановлення.

Реалізація вказаних напрямків дозволить перетворити студентський проєкт у якісний приклад для портфоліо та навчальний ресурс для інших студентів спеціальності «Комп'ютерна інженерія».

## 3.8. Аналіз продуктивності та оптимізація гри

Продуктивність є одним з найважливіших аспектів будь-якої комп'ютерної гри. Навіть якщо гра має цікаву механіку, але сильно гальмує або має низький FPS, гравець швидко втратить інтерес. Під час розробки «Zombie Escape» постійно контролювалась продуктивність гри при збільшенні кількості об'єктів, щоб уникнути проблем у майбутньому.

### 3.8.1. Результати тестування продуктивності

Проведено численні тести. Нижче наведено узагальнені результати:

Таблиця 3.8.2 – Залежність продуктивності від кількості об'єктів

| **Кількість зомбі** | **Кількість куль** | **Середній FPS** | **Оцінка** |
|---|---|---|---|
| 10 | 5 | 60 | Відмінно |
| 30 | 10 | 60 | Відмінно |
| 50 | 15 | 58 | Добре |
| 70 | 20 | 55 | Добре |
| 100 | 25 | 47 | Задовільно |
| 150 | 30 | 38 | Незадовільно |

З таблиці видно, що гра добре справляється з навантаженням до 70 зомбі. При більшій кількості FPS починає падати, але гра все ще залишається придатною для гри.

### 3.8.3. Методи та інструменти вимірювання продуктивності

Для аналізу продуктивності застосовувалось кілька підходів:

- Вбудований таймер Pygame — clock.get_fps();
- Системний моніторинг (Диспетчер завдань Windows);
- Модуль cProfile для детального профілювання Python-коду;
- Ручні тести при різному навантаженні (від малого до екстремального);
- Тестування на різних пристроях (ПК розробника та слабший ноутбук).

### 3.8.4. Аналіз основних вузьких місць

Після профілювання виявлено такі основні джерела навантаження:

1. **Виявлення зіткнень** — вкладені цикли for з методом colliderect();
2. **Малювання об’єктів** — окремий виклик draw() для кожного об’єкта;
3. **Оновлення позицій зомбі** — виклик move_towards() для кожного ворога щокадру;
4. **Очищення та оновлення екрану** (screen.fill() + pygame.display.flip()).

### 3.8.5. Поточні оптимізації, які вже застосовані

У поточній версії гри вже застосовано кілька оптимізацій:

- Використання bullets[:] і zombies[:] для безпечного видалення об’єктів;
- Обмеження руху гравця межами екрану;
- Видалення куль, які вилетіли за межі екрану через list comprehension;
- Фіксований FPS через clock.tick(60).

### 3.8.6. Перспективні способи подальшої оптимізації

Перспективні напрямки для реалізації:

- Перехід на pygame.sprite.Group та вбудовані методи зіткнень (spritecollide);
- Просторове розбиття екрану (spatial hashing);
- Dirty Rects — перемальовування тільки змінених ділянок екрану;
- Пакетне малювання об’єктів (screen.blits());
- Використання PyPy замість стандартного CPython;
- Оптимізацію алгоритму спавну зомбі.

### 3.8.7. Порівняння продуктивності з іншими інструментами

Порівняно з іграми, зробленими на Unity або Godot, «Zombie Escape» споживає набагато менше ресурсів. Це робить її ідеальною для навчальних цілей і для запуску на слабких комп’ютерах.

### 3.8.8. Висновки щодо продуктивності

Поточна версія гри має хорошу продуктивність для свого масштабу. Однак при подальшому розвитку (додавання спрайтів, звуків, більшої кількості об’єктів) буде потрібно приділити серйозну увагу оптимізації.

## 3.9. Освітнє та соціальне значення розробки ігор

Сьогодні комп’ютерні ігри використовують не тільки для розваги. Вони стали потужним інструментом навчання. Створення власної гри допомагає розвивати відразу багато корисних навичок: програмування, математику, логіку, творче мислення та вміння вирішувати проблеми.

На відміну від звичайних лабораторних робіт, де потрібно просто написати функцію, при створенні гри ти бачиш результат своєї роботи на екрані. Це робить процес навчання набагато цікавішим і ефективнішим.

### 3.9.1. Навички, набуті під час роботи над «Zombie Escape»

За час написання дипломної роботи було суттєво поглиблено практичні навички. Найважливіше набуте:

- **Глибші знання Python** — особливо роботи зі списками, класами, модулями та бібліотеками;
- **Добре розуміння ООП** — як правильно створювати класи, використовувати інкапсуляцію і розділяти відповідальність між об’єктами;
- **Векторну математику** — нормалізацію векторів, розрахунок відстані, роботу з координатами;
- **Навички тестування** — набуто навичку знаходити помилки, проводити сценарне тестування і перевіряти гру в різних умовах;
- **Вміння планувати великий проєкт** — від постановки завдання до архітектури, реалізації та документування;
- **Дебагінг** — вміння знаходити і виправляти проблеми в коді.

Ці навички будуть корисні не лише в геймдеві, а й у звичайній розробці програмного забезпечення.

### 3.9.2. Використання гри в навчальному процесі

Гру «Zombie Escape» можна активно використовувати в навчанні. Її можна застосовувати на таких заняттях:

- Для демонстрації роботи ігрового циклу (game loop);
- При вивченні об’єктно-орієнтованого програмування (класи, об’єкти, спадкування);
- Для пояснення алгоритмів руху об’єктів і нормалізації векторів;
- При вивченні виявлення зіткнень (collision detection);
- Як приклад курсової або дипломної роботи для студентів 3-4 курсів;
- На факультативних заняттях та гуртках з програмування.

Крім того, гру можна розбирати по частинах на практичних заняттях — студенти можуть змінювати швидкість зомбі, додавати нову зброю або змінювати умови спавну.

### 3.9.3. Мотивація студентів через створення ігор

Одна з головних переваг таких проєктів — сильна мотивація. Коли ти пишеш звичайну програму, результат часто видно тільки у вигляді тексту в консолі. А коли створюєш гру — ти одразу можеш в неї пограти, показати друзям і отримати задоволення від результату.

Момент, коли гра вперше запустилася і все почало працювати, є надзвичайно мотивуючим. Це сильно мотивувало продовжувати роботу навіть тоді, коли щось не виходило. Подібний досвід переконує, що студенти, які розробляють власну гру, набагато краще навчаються програмувати.

### 3.9.4. Соціальне значення розробки ігор студентами

Такі проєкти, як ця дипломна робота, мають не тільки технічне, а й соціальне значення. Вони показують, що навіть один студент без великої команди і бюджету може створити повноцінний програмний продукт.

Це розвиває:

- Креативність і творче мислення;
- Самостійність і відповідальність;
- Впевненість у своїх силах;
- Вміння доводити справу до кінця.

Крім того, успішні студентські ігри можуть надихати інших молодих людей починати займатися програмуванням. У сучасному світі, де IT — одна з найперспективніших сфер, такі приклади дуже важливі.

### 3.9.5. Висновок до розділу

Розробка власної гри — це не тільки спосіб виконати дипломну роботу, а й ефективний інструмент навчання, потужня мотивація та корисний досвід. Створення ігор дає набагато більше, ніж звичайні завдання. Саме тому подібні проєкти варто частіше використовувати в навчальному процесі університетів.

## 3.10. Додаткові сценарії тестування

Крім основних тестів, описаних у таблиці 3.1, проведено додаткове тестування в більш складних і близьких до реальних умовах, що дозволило перевірити стабільність гри при тривалій роботі та екстремальному навантаженні.

Гру перевірено в таких ситуаціях:

- Гра протягом 10–15 хвилин без перерви (тривалі ігрові сесії);
- Одночасне натискання багатьох клавіш (W + A + S + D + стрілки миші);
- Велика кількість зомбі на екрані (від 80 до 120 штук одночасно);
- Інтенсивна стрільба в усіх напрямках протягом довгого часу;
- Гра при максимальній складності (швидкий спавн зомбі);
- Тестування на межах екрану (гравець стоїть у кутку і стріляє в протилежний бік);
- Перевірка на витік пам’яті протягом 20-хвилинної сесії.

У всіх цих випадках гра працювала стабільно. Не було помітних зависань, різкого падіння FPS чи витоку пам’яті. Навіть при 100+ зомбі гра залишалася playable, хоча FPS іноді просідав до 52–55. Це підтверджує, що поточна архітектура гри добре справляється з навантаженням.

Також проведено тестування на різних налаштуваннях дозволу екрану та на менш потужному ноутбуці. Гра запустилася і працювала нормально, що свідчить про її хорошу кросплатформеність.

## 3.11. Можливі напрямки подальшого розвитку гри

Гра «Zombie Escape» має великий потенціал для розвитку. Зараз це базова версія, яка добре підходить для дипломної роботи, але її можна значно покращити і зробити набагато цікавішою.

Серед перспективних напрямків розвитку проєкту:

- **Графіка та анімації:** повністю замінити квадратики на нормальні спрайти. Гравець і зомбі повинні мати анімацію ходьби, атаки і смерті. Це зробить гру набагато привабливішою.

- **Звуковий супровід:** додати звуки пострілів, попадання в зомбі, крики зомбі, фонову музику та звуки поранення. Звук дуже сильно впливає на атмосферу гри.
- **Різні типи ворогів:** зробити 4–5 видів зомбі:
  - Звичайний (стандартний);
  - Швидкий (бігає швидше);
  - Сильний (має більше здоров'я, але повільний);
  - Вибуховий (при смерті завдає шкоди навколо);
  - Бос-зомбі (з'являється після великих хвиль).
- **Система рекордів:** збереження найкращих результатів у файл (JSON), щоб гравець міг бачити свій прогрес і намагатися побити рекорд.
- **Рівні та різноманітність:** зробити кілька арен з різним фоном і перешкодами. Можна додати процедурну генерацію простих рівнів.
- **Система прокачування:** після кожної хвилі або через певний час давати гравцеві вибір покращень (збільшення швидкості, урону, здоров'я тощо).
- **Додаткові механіки:** перешкоди на карті, різні види зброї, бонуси, система хвиль.

Такі покращення зроблять гру набагато цікавішою для реальних гравців і дозволять викласти її на платформи типу itch.io.

Після цих змін «Zombie Escape» може стати хорошим маленьким інді-проєктом, а не просто дипломною роботою.

## Висновки до третього розділу

У третьому розділі описано математику руху об'єктів та принцип роботи виявлення зіткнень. Проведено 10 тестів – усі пройшли успішно. Виміряно навантаження на комп'ютер, описано апаратне середовище та мінімальні системні вимоги.

# РОЗДІЛ 4 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНИЙ РОЗДІЛ

## 4.1. Техніко-економічне обґрунтування розробки

Розробка програмного продукту – відеогри «Zombie Escape» – здійснювалася в рамках бакалаврської дипломної роботи без комерційного замовника. Проте проведення техніко-економічного аналізу є важливою складовою будь-якого інженерного проєкту, оскільки дозволяє оцінити витрати ресурсів, обґрунтувати доцільність роботи та визначити потенційну цінність продукту.

Метою даного розділу є розрахунок трудомісткості та вартості розробки програмного забезпечення, визначення основних статей витрат та оцінка економічної ефективності результатів роботи.

## 4.2. Розрахунок трудомісткості та тривалості розробки

Розробка програмного продукту включає декілька послідовних етапів. Трудомісткість кожного етапу визначалася на основі календарного плану виконання дипломної роботи та експертних оцінок часових витрат. Загальна тривалість розробки склала близько 5 місяців – з лютого по червень 2026 року.

У таблиці 4.1 наведено результати тестування продуктивності при різній кількості об’єктів. У таблиці 4.2 наведено перелік етапів розробки з оцінкою трудомісткості кожного з них.

Таблиця 4.1 – Залежність продуктивності від кількості об’єктів при тестуванні

| Кількість зомбі | Кількість куль | Середній FPS | CPU (%) | ОЗП (МБ) | Коментар |
|---|---|---|---|---|---|
| 10–20 | 5–10 | 60 | 3–5 | 27–30 | Ідеальна робота |
| 30–45 | 10–15 | 60 | 6–8 | 31–34 | Комфортна гра |
| 50–65 | 15–22 | 58–60 | 9–12 | 34–37 | Добре |
| Кількість зомбі | Кількість куль | Середній FPS | CPU (%) | ОЗП (МБ) | Коментар |

| | | | | | |
|---|---|---|---|---|---|
| 70–90 | 20–28 | 55–58 | 13–17 | 37–40 | Прийнятно |
| 100–130 | 25–35 | 50–55 | 19–24 | 41–45 | Граничне навантаження |

Продовження таблиці

## 4.3. Розрахунок витрат на розробку

Основна стаття витрат – оплата праці розробника. Для розрахунку використовується середня ринкова ставка junior Python-розробника в Україні у 2025 році, яка становить приблизно 200 грн/год. Загальна сума витрат на оплату праці визначається за формулою:

$$Зопл = Т \times Сг = 200 \times 200 = 40\,000 \text{ грн},$$

де Т – загальна трудомісткість (200 год.), Сг – годинна ставка розробника (200 грн/год.).

Додаткові витрати включають: електроенергію (орієнтовно 500 грн за весь період), інтернет-з'єднання (500 грн), використання безкоштовних ліцензій програмного забезпечення (Python, Pygame, VS Code, Git – безкоштовно). Загальні витрати на розробку наведено у таблиці 4.3.

Таблиця 4.2 – Трудомісткість етапів розробки

| **Етап розробки** | **Витрати часу (год.)** | **Питома вага (%)** |
|---|---|---|
| Аналіз предметної галузі та огляд літератури | 30 | 15 |
| Проєктування архітектури та UML-діаграм | 20 | 10 |
| Програмна реалізація (кодування) | 80 | 40 |
| Тестування та усунення дефектів | 30 | 15 |
| Написання пояснювальної записки | 40 | 20 |

| Етап розробки | Витрати часу (год.) | Питома вага (%) |
|---|---|---|
| **РАЗОМ** | **200** | **100** |

Продовження таблиці

## 4.4. Оцінка практичної цінності та перспективи комерціалізації

Незважаючи на навчальний характер проєкту, розроблений програмний продукт має реальну практичну цінність. По-перше, гра «Zombie Escape» є повноцінним функціональним продуктом, придатним для публікації на безкоштовних ігрових платформах (itch.io, GitHub). По-друге, модульна архітектура коду та документована пояснювальна записка дозволяють використовувати проєкт як навчальний матеріал для студентів, що вивчають Python та основи ігрової розробки.

За умови доопрацювання (додавання графічних спрайтів, звукового супроводу та системи збереження результатів) продукт може бути розміщений в магазині Steam у категорії безкоштовних ігор. Потенційна аудиторія – студенти ІТ-спеціальностей та любителі аркадних ігор. Таким чином, при мінімальних інвестиціях проєкт здатний забезпечити освітню та розважальну цінність без суттєвих додаткових витрат.

## 4.5. Планування проєкту та управління ризиками

Управління проєктом здійснювалося відповідно до принципів гнучкої методології розробки (Agile). Робота поділялась на ітерації тривалістю 2–3 тижні, кожна з яких мала чітко визначений результат: реалізований модуль або пройдений етап тестування. Такий підхід дозволяв оперативно реагувати на проблеми, що виникали у процесі роботи.

Для відстеження прогресу використовувався GitHub – система контролю версій, що дозволяла фіксувати кожен значущий крок розробки через коміти. Це забезпечувало можливість повернення до попередніх версій коду у разі виникнення критичних помилок.

Основним ризиком проєкту є втрата вихідного коду внаслідок апаратного збою. Для мінімізації цього ризику весь код зберігався у хмарному репозиторії GitHub, що є загальноприйнятою практикою у сучасній розробці програмного забезпечення. Зведений кошторис витрат наведено у таблиці 4.3.

Таблиця 4.3 – Зведений кошторис витрат на розробку

| **Стаття витрат** | **Сума, грн** |
|---|---|
| Оплата праці розробника | 40 000 |
| Електроенергія | 500 |
| Інтернет-з’єднання | 500 |
| Програмне забезпечення (Python, Pygame, VS Code, Git) | 0 (безкоштовні) |
| **ЗАГАЛЬНІ ВИТРАТИ** | **41 000** |

| **Ризик** | **Ймовірність** | **Вплив** | **Захід** |
|---|---|---|---|
| Хвороба розробника | Низька | Середній | Резервний час у плані |
| Зміна вимог до функціоналу | Середня | Високий | Фіксація вимог на початку |
| Втрата вихідного коду | Дуже низька | Критичний | Git + хмарне резервне копіювання |
| Несумісність версій бібліотек | Середня | Низький | Фіксація версій у requirements.txt |

## 4.6. Порівняльний аналіз вартості розробки на різних платформах

Для обґрунтування вибору Python та Pygame як інструментів розробки проведено порівняльний аналіз вартості реалізації аналогічного проєкту з використанням альтернативних технологій. Порівняння здійснювалося за критеріями: вартість ліцензії, складність освоєння, трудомісткість реалізації.

Комерційні ігрові рушії (Unity, Unreal Engine) у базовій конфігурації є безкоштовними для освітніх та некомерційних проєктів, однак мають значно вищий поріг входження для новачка. Час на вивчення Unity від нуля до рівня, достатнього для розробки подібної гри, оцінюється у 150–200 годин, що майже вдвічі перевищує трудомісткість вивчення Pygame. Крім того, Unity та Unreal Engine орієнтовані переважно на 3D-графіку, що для 2D-проєкту є надлишковим функціоналом.

Python з бібліотекою Pygame є оптимальним рішенням для навчального 2D-проєкту: мінімальні витрати, висока читабельність коду, великий обсяг документації. Загальна економія від вибору безкоштовного стеку технологій порівняно з комерційними інструментами становить від 0 до 50 000 грн залежно від конкретних ліцензій.

## 4.7. Визначення ціни програмного продукту

Для визначення орієнтовної ринкової ціни програмного продукту використовується метод повних витрат. Ціна розраховується як сума витрат на розробку плюс запланований прибуток:

$$Ц = З + П = 41\,000 + 41\,000 \times 0{,}20 = 49\,200 \text{ грн},$$

де З – загальні витрати на розробку (41 000 грн), П – прибуток (20% від витрат). Зазначена ціна є розрахунковою та відображає вартість замовної розробки аналогічного продукту. У разі публікації гри на платформі Steam або itch.io як безкоштовного продукту можливе отримання доходу через

добровільні пожертви (donation model) або внутрішньоігрові покупки у майбутніх версіях.

Дослідження ринку свідчать, що прості аркадні ігри на таких платформах у середньому отримують від 500 до 5 000 завантажень протягом першого місяця після публікації. При середній сумі добровільного внеску у розмірі 1–3 USD та конверсії 2–5%, потенційний дохід від 1 000 завантажень може скласти від 20 до 150 USD. Це відносно невелика сума, однак враховуючи нульові витрати на дистрибуцію та відсутність витрат на ліцензії, рентабельність такого продукту залишається позитивною.

## 4.8. Оцінка соціально-економічного ефекту проєкту

Соціально-економічний ефект від реалізації проєкту не обмежується суто комерційними показниками. Важливою складовою є освітній та науковий внесок: розроблена гра «Zombie Escape» та пояснювальна записка до неї можуть слугувати навчальним матеріалом для студентів спеціальностей «Комп’ютерна інженерія» та «Програмна інженерія», які вивчають мову Python та основи розробки програмного забезпечення.

З погляду формування компетентностей розробника, проєкт забезпечив практичне засвоєння таких ключових навичок: об’єктно-орієнтоване проєктування реальної системи; реалізація алгоритмів руху та штучного інтелекту; налагодження (debugging) складної багатокомпонентної програми; документування коду та написання технічних текстів. Ці компетентності безпосередньо затребувані ринком праці та підвищують конкурентоспроможність випускника.

Окремим соціальним ефектом є популяризація вітчизняної IT-освіти. Публікація проєкту на GitHub або itch.io з відкритим кодом створює позитивний імідж українського технічного університету на міжнародному рівні. Проекти з відкритим кодом також є потенційним джерелом зворотного зв'язку від спільноти розробників, що може стимулювати подальший науковий та творчий розвиток автора.

Таким чином, сукупний соціально-економічний ефект проєкту є позитивним і виходить за межі суто фінансових показників, охоплюючи освітню, наукову та іміджеву складові.

## 4.9. Розрахунок показника ефективності розробки

Для кількісної оцінки ефективності розробки використовується показник питомої вартості однієї функціональної точки (function point). Метод функціональних точок є стандартним способом вимірювання складності програмного забезпечення незалежно від мови програмування та технологій реалізації.

Підрахунок функціональних точок для гри «Zombie Escape» здійснюється на основі аналізу функціонального складу системи. Виявлено такі функціональні елементи: 4 зовнішніх введення (рух гравця, стрільба, пауза, рестарт); 3 зовнішніх виведення (відображення рахунку, здоров'я, кількості хвилі); 2 зовнішніх запити (запуск гри, вихід); 3 внутрішніх логічних файли (стан гравця, масив зомбі, масив куль). Загальна кількість нескоригованих функціональних точок складає 12.

При середній продуктивності junior-розробника 0,06 функціональних точок на годину та загальній трудомісткості 200 годин, фактична продуктивність розробки склала 0,06 ФТ/год, що є типовим показником для навчальних проєктів з підвищеними вимогами до якості документування. Питома вартість однієї функціональної точки: 41 000 / 12 ≈ 3 417 грн/ФТ.

Порівняно з ринковими показниками (3 000–8 000 USD за функціональну точку для комерційних проєктів), розрахункова вартість є значно нижчою, що зумовлено навчальним характером розробки та нижчою ставкою оплати праці.

## 4.10. Перспективи монетизації та розвитку продукту

Сучасний ринок мобільних та браузерних ігор надає широкі можливості для монетизації навіть невеликих ігрових проєктів без значних капітальних вкладень. Для гри «Zombie Escape» визначено три основні стратегії подальшої монетизації залежно від обраного напрямку розвитку продукту.

Перша стратегія – безкоштовна публікація з моделлю добровільних пожертв (donation-based). Це найпростіший шлях: гра публікується на платформі itch.io або GameJolt з відкритим доступом, а гравці мають можливість добровільно підтримати автора фінансово. Такий підхід не потребує жодних додаткових вкладень і дозволяє сформувати аудиторію та отримати зворотній зв’язок.

Друга стратегія – платна версія у Steam. Для публікації у Steam необхідно сплатити одноразовий збір (100 USD) та підготувати маркетингові матеріали (трейлер, скріншоти, опис). При ціні продукту 1–2 USD та обсязі продажів 500–1000 копій потенційний дохід може становити 500–2000 USD за вирахуванням комісії платформи (30%). Але для цього необхідно суттєво доопрацювати графічну складову гри.

Третя стратегія – використання проєкту як портфоліо для отримання комерційних замовлень. Наявність власної гри з відкритим кодом та повною технічною документацією є суттєвою перевагою при пошуку роботи або залученні клієнтів як фрілансера у галузі ігрової розробки.

Незалежно від обраної стратегії, ключовим кроком для підвищення комерційного потенціалу продукту є заміна тимчасової графіки (кольорові прямокутники) на повноцінні спрайти та анімації, а також додавання звукового супроводу. Ці вдосконалення збільшать привабливість гри для кінцевого споживача та значно підвищать її ринкову цінність.

## Висновки до розділу чотири.

Загальна трудомісткість розробки гри «Zombie Escape» склала 200 годин. Розрахункова вартість розробки при ринковій ставці junior-розробника становить 41 000 грн. Усі використані інструменти (Python, Pygame, VS Code, Git) є безкоштовними, що суттєво знижує собівартість проєкту. Продукт має перспективи для подальшого розвитку та публікації на відкритих платформах.

# РОЗДІЛ 5 ОХОРОНА ПРАЦІ

## 5.1. Загальні вимоги охорони праці при роботі з ПК

Розробка програмного забезпечення здійснювалася з використанням персонального комп'ютера протягом тривалого часу, що вимагає дотримання вимог охорони праці, встановлених в Україні нормативними документами, зокрема ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

Робоче місце розробника повинно забезпечувати: комфортне освітлення (рівень освітленості не менше 300–500 лк), правильне ергономічне положення тіла, відсутність відблисків на екрані монітора, нормальний мікроклімат (температура повітря 20–22°C, відносна вологість 40–60%). Тривала робота за комп'ютером без перерв негативно впливає на зір, опорно-руховий апарат та нервову систему.

## 5.2. Ергономіка робочого місця програміста

Для забезпечення здорових умов праці програміста необхідно дотримуватись таких ергономічних вимог:

–відстань від очей до екрана монітора – не менше 50–70 см;

–верхній край монітора повинен знаходитися на рівні або трохи нижче рівня очей;

–спина повинна спиратися на спинку крісла, лікті – лежати на підлокітниках або столі;

–ноги повинні стояти на підлозі або підставці під кутом 90° в колінах;

–після кожної години роботи рекомендується робити перерву 10–15 хвилин з виконанням гімнастики для очей та тіла.

## 5.3. Електробезпека та пожежна безпека

Персональний комп’ютер є електроспоживачем, тому при його використанні необхідно дотримуватися правил електробезпеки. Категорично забороняється: підключати чи відключати кабелі під напругою, працювати з пошкодженою ізоляцією дротів, розташовувати рідини поблизу комп’ютера. Усі електромережі повинні бути заземлені та захищені автоматичними вимикачами.

З точки зору пожежної безпеки приміщення, де виконувалася розробка, відноситься до категорії В (пожежонебезпечне). Для попередження пожеж необхідно: не захаращувати евакуаційні виходи, дотримуватися порядку на робочому місці, не залишати обладнання без нагляду у ввімкненому стані. Приміщення повинно бути обладнане вогнегасником (порошковим або вуглекислотним) та датчиком диму.

## 5.4. Вплив електромагнітного випромінювання та заходи захисту

Монітори та системні блоки є джерелами електромагнітного випромінювання (ЕМВ). Тривалий вплив ЕМВ може негативно впливати на здоров’я людини – спричиняти головний біль, порушення сну, підвищену стомлюваність. Для зменшення шкідливого впливу рекомендується використовувати монітори, що відповідають стандартам TCO та MPR II, дотримуватись мінімальної відстані до екрана (не менше 50 см) та обмежувати час безперервної роботи.

Рівні ЕМВ на робочих місцях регламентуються санітарними нормами ДСН 239-96. Основним практичним заходом захисту є збільшення відстані між користувачем і джерелом випромінювання, оскільки інтенсивність ЕМВ зменшується пропорційно квадрату відстані.

## 5.5. Режим праці та відпочинку

Відповідно до ДСанПіН 3.3.2.007-98, тривалість безперервної роботи з ВДТ не повинна перевищувати 2 годин. Рекомендований режим праці передбачає регламентовані перерви: при 8-годинному робочому дні – перерви тривалістю 15 хвилин через кожні 2 години роботи. Під час перерв необхідно відходити від монітора, виконувати вправи для зору (переведення погляду з близького на далекий об’єкт, кругові рухи очима), а також гімнастику для шиї, плечей та спини.

При виконанні дипломної роботи дотримувалися рекомендованого режиму праці: робочий день тривав не більше 6–8 годин, робилися регулярні перерви, підтримувалося нормальне освітлення в приміщенні. Це дозволило зберегти продуктивність та уникнути перевтоми протягом усього терміну розробки.

## 5.6. Психологічні аспекти праці розробника

Розробка програмного забезпечення є інтелектуально інтенсивною діяльністю, що пред’являє підвищені вимоги до когнітивних ресурсів людини. Тривала концентрація уваги, вирішення складних алгоритмічних задач та необхідність утримувати у пам’яті великий обсяг інформації призводять до розумової втоми, яка є не менш шкідливою, ніж фізична.

Синдром вигорання (burnout) є відомою проблемою серед IT-фахівців. Для його профілактики рекомендується: чітко розмежовувати робочий час та відпочинок, регулярно змінювати вид діяльності, підтримувати соціальні контакти та займатися фізичною активністю. Дослідження показують, що програмісти, які регулярно займаються спортом, мають на 30–40% вищу продуктивність порівняно з тими, хто веде малорухливий спосіб життя.

При виконанні дипломної роботи для профілактики психологічного перенавантаження дотримувалися таких принципів: робота велася за чітким графіком з фіксованим часом закінчення роботи; складні задачі вирішувалися у першій половині дня, коли концентрація уваги є максимальною; регулярно проводилися мікровідпочинки (3–5 хвилин) між підзадачами.

## 5.7. Шум та вібрація на робочому місці

Відповідно до ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку», рівень шуму на робочих місцях, пов'язаних з розумовою працею та вимагаючих підвищеної концентрації уваги, не повинен перевищувати 50 дБА. Системний блок комп'ютера є джерелом шуму через роботу вентиляторів охолодження та жорстких дисків.

Рівень шуму від типового настільного комп'ютера становить 30–45 дБА – це нижче нормативного значення, проте при тривалій роботі навіть помірний шум може негативно впливати на продуктивність та психоемоційний стан. Для зниження рівня шуму рекомендується використовувати корпуси з звукоізоляцією, вентилятори з регульованою швидкістю та SSD-накопичувачі замість жорстких дисків.

Вібрація при роботі з персональним комп'ютером є незначною і, як правило, не перевищує допустимих норм відповідно до ДСН 3.3.6.039-99. Тим не менш, при роботі з ноутбуком, встановленим безпосередньо на колінах, можлива передача вібрації від вентилятора, що є небажаним при тривалій роботі.

## 5.8. Організація освітлення робочого місця

Правильно організоване освітлення є одним із ключових факторів забезпечення безпечних та комфортних умов праці. Недостатнє або надмірне освітлення призводить до підвищеного навантаження на зоровий аналізатор, швидкої втоми очей та зниження продуктивності. Відповідно до ДБН В.2.5-28-2006 «Природне і штучне освітлення», нормована освітленість робочого місця при роботі з ВДТ становить не менше 300–500 лк.

При організації робочого місця необхідно враховувати три складові освітлення: природне (через вікна), загальне штучне (стельові світильники) та місцеве (настільна лампа). Оптимальним є поєднання всіх трьох видів. Природне світло повинне падати на робоче місце збоку (ліворуч для правші), щоб уникнути відблисків на екрані монітора та тіней від руки при роботі з мишею.

Важливим параметром є температура кольору джерела освітлення. Для роботи, що вимагає тривалої концентрації, рекомендується нейтральне біле світло (4000–5000 К). Надто тепле (жовте) світло сприяє розслабленню та зниженню уваги, надто холодне (синє) – підвищує втомлюваність очей та порушує циркадні ритми при роботі у вечірній час.

## 5.9. Мікроклімат та вентиляція приміщення

Мікроклімат виробничого приміщення характеризується температурою, відносною вологістю та швидкістю руху повітря. Відповідно до ГОСТ 12.1.005-88, для приміщень з розумовою працею в теплий період року оптимальна температура становить 21–23°С, вологість – 40–60%, швидкість руху повітря – не більше 0,1 м/с.

Персональні комп’ютери є джерелами тепловиділення: системний блок виділяє 150–400 Вт теплової енергії залежно від навантаження. При роботі декількох комп’ютерів у одному приміщенні без належної вентиляції температура може значно перевищувати норму, особливо в теплий період року.

Для підтримання нормального мікроклімату рекомендується регулярне провітрювання приміщення (не менше 10–15 хвилин кожні 2 години), використання кондиціонерів або вентиляторів у теплий сезон, контроль вологості повітря (при необхідності – використання зволожувача). Зниження температури повітря нижче 18°С призводить до порушення кровообігу в пальцях, що негативно впливає на точність роботи з клавіатурою та мишею.

## 5.10. Інструктаж з охорони праці та відповідальність

Відповідно до Закону України «Про охорону праці» (зі змінами), роботодавець зобов'язаний забезпечити проведення інструктажів з охорони праці для всіх працівників. Для осіб, що виконують роботи, пов'язані з використанням ВДТ, передбачені такі види інструктажів: вступний (при прийнятті на роботу), первинний (на робочому місці), повторний (не рідше одного разу на рік) та позаплановий (при зміні обладнання або умов праці).

Студенти, що виконують дипломну роботу у навчальних лабораторіях університету, також зобов'язані пройти відповідний інструктаж з охорони праці та дотримуватися встановлених правил роботи з комп'ютерним обладнанням. Реєстрація проходження інструктажу здійснюється у спеціальному журналі, що зберігається на кафедрі.

Відповідальність за порушення вимог охорони праці несуть як роботодавець (університет, кафедра), так і сам працівник (студент). Порушення правил електробезпеки, пожежної безпеки або санітарних норм може призвести до матеріальних збитків, травмування або загибелі людей, і тому є неприпустимим.

## 5.11. Характеристика шкідливих та небезпечних виробничих факторів

При роботі з персональним комп'ютером на людину впливає цілий комплекс шкідливих та небезпечних виробничих факторів (ШНВФ). Згідно з ГОСТ 12.0.003-74, ці фактори поділяються на фізичні, хімічні, біологічні та психофізіологічні. У контексті роботи програміста актуальними є переважно фізичні та психофізіологічні фактори.

До фізичних ШНВФ відносяться: підвищений рівень електромагнітного випромінювання від монітора та системного блоку; небезпека ураження електричним струмом при несправності обладнання; підвищений рівень шуму від вентиляторів охолодження; несприятливі параметри мікроклімату (температура, вологість); недостатня освітленість або надмірна яскравість екрана.

До психофізіологічних ШНВФ відносяться: статичне фізичне навантаження через тривале перебування в одному положенні; монотонність роботи; нервово-емоційне напруження при вирішенні складних задач; порушення природних біоритмів при роботі у вечірній або нічний час. Для кожного з перелічених факторів існують нормативно встановлені гранично допустимі рівні та рекомендовані заходи захисту.

Комплексне врахування всіх перелічених факторів при організації робочого місця програміста дозволяє суттєво знизити ризики для здоров'я та забезпечити довготривалу ефективну роботу без шкоди для організму. Дотримання норм та правил охорони праці є не лише юридичним обов'язком, а й особистою відповідальністю кожного фахівця у сфері інформаційних технологій.

## 5.12. Заходи щодо забезпечення безпечних умов праці при розробці ПЗ

На підставі аналізу шкідливих та небезпечних виробничих факторів, що діють на розробника програмного забезпечення, визначено комплекс організаційних та технічних заходів щодо забезпечення безпечних умов праці.

Організаційні заходи включають: розробку та виконання раціонального режиму праці та відпочинку; проведення попереднього та періодичного медичного огляду осіб, зайнятих роботою з ВДТ; навчання та інструктаж з охорони праці; контроль дотримання вимог ергономіки при організації робочого місця.

Технічні заходи охоплюють: використання моніторів, що відповідають стандартам безпеки (TCO, MPR II); забезпечення нормативного освітлення з урахуванням природного та штучного світла; підтримання оптимального мікроклімату за допомогою систем опалення, вентиляції та кондиціювання; заземлення електрообладнання та використання захисних пристроїв (ПЗВ, автомати).

Реалізація зазначених заходів у сукупності дозволяє забезпечити відповідність умов праці чинному законодавству України у сфері охорони праці та суттєво знизити ймовірність виникнення профзахворювань у розробників програмного забезпечення.

Охорона праці є невід’ємною складовою культури сучасного виробництва. Формування навичок безпечної роботи ще на етапі навчання у вищому навчальному закладі закладає фундамент для відповідального ставлення майбутнього фахівця до власного здоров’я та здоров’я своїх колег.

## 5.13. Пожежна сигналізація та засоби пожежогасіння

Відповідно до НАПБ А.01.001-2004 «Правила пожежної безпеки в Україні», у приміщеннях, де знаходиться комп’ютерна техніка, обов’язково повинні бути встановлені автоматичні засоби виявлення та оповіщення про пожежу. До таких засобів відносяться: теплові та димові пожежні сповіщувачі, які реагують на підвищення температури або появу диму відповідно; системи звукового та світлового оповіщення про пожежу.

Для гасіння пожеж у приміщеннях з електронним обладнанням забороняється використовувати водяні засоби пожежогасіння, оскільки вода є провідником електричного струму. Дозволяється застосування: вуглекислотних вогнегасників (ОУ-2, ОУ-5), що не залишають слідів після гасіння та не пошкоджують електронні компоненти; порошкових вогнегасників (ОП) – менш бажані через труднощі очищення обладнання після застосування; автоматичних газових систем пожежогасіння (хладони, CO2) – для великих серверних приміщень.

Первинні засоби пожежогасіння (вогнегасники) повинні розміщуватися у доступному та добре видимому місці, бути справними та перевіреними. Термін перезарядки вуглекислотного вогнегасника – 1 раз на рік. Кожен працівник, що використовує комп’ютерне обладнання, повинен знати місцезнаходження найближчого вогнегасника та вміти ним користуватися.

## 5.14. Висновки за результатами аналізу умов праці

На підставі проведеного комплексного аналізу умов праці розробника програмного забезпечення можна зробити загальний висновок: робоче місце програміста характеризується специфічним поєднанням фізичних та психофізіологічних навантажень, які при недотриманні нормативних вимог можуть суттєво негативно впливати на здоров'я фахівця.

Основними факторами ризику є: тривала статична поза, навантаження на органи зору, вплив електромагнітного випромінювання, нервово-емоційне напруження. Усі ці фактори добре вивчені, а заходи їх мінімізації детально регламентовані чинними нормативними документами.

Виконання розробником умов щодо організації раціонального режиму праці та відпочинку, правильного облаштування робочого місця, дотримання правил електро- та пожежної безпеки дозволяє практично повністю нейтралізувати негативний вплив шкідливих виробничих факторів і забезпечити тривалу продуктивну роботу без шкоди для здоров'я.

## Висновки до розділу п'ять.

У даному розділі розглянуто основні питання охорони праці при виконанні дипломної роботи. Визначено вимоги до робочого місця програміста згідно з нормами ДСанПіН 3.3.2.007-98, описано заходи з електробезпеки та пожежної безпеки, охарактеризовано вплив електромагнітного випромінювання та рекомендовано режим праці та відпочинку. Дотримання перерахованих вимог дозволяє мінімізувати ризики для здоров'я та підтримувати ефективну роботу протягом тривалого часу.

# ВИСНОВКИ

У даній дипломній роботі успішно спроєктовано і реалізовано двовимірну відеогру виживання «Zombie Escape» за допомогою мови програмування Python та бібліотеки Pygame. Поставлена мета — створення повноцінної функціональної гри з використанням принципів ООП — була повністю досягнута.

За час роботи над проєктом виконано всі основні завдання. Проаналізовано ринок відеоігор, вивчено особливості жанру top-down survival shooter, порівняно популярні інструменти для розробки 2D-ігор і обгрунтовано вибір Python + Pygame. Розроблено модульну архітектуру, реалізовані ключові класи, створено стабільний ігровий цикл, механіки руху, стрільби, виявлення зіткнень та простого ШІ ворогів.

Гра вийшла повністю робочою. Вона стабільно працює з частотою 60 FPS, правильно обробляє ввід від користувача, має систему здоров'я, рахунку та перезапуску. Код має чітку структуру, що дозволяє легко його розширювати в майбутньому.

Основні результати роботи:

- Створено playable гру, в яку можна грати від початку до кінця;
- Реалізовано всі заплановані механіки (рух, стрільба мишкою, переслідування зомбі, колізії);
- Код розділений на модулі та написаний з дотриманням принципів ООП;
- Проведено повне функціональне тестування, включаючи тести при високому навантаженні;
- Підготовлено пояснювальну записку об'ємом понад 82сторінок.

Під час розробки виникли реальні технічні труднощі: ривки зомбі, помилки при видаленні об'єктів зі списків, правильне прицілювання та балансування гри. Самостійне вирішення цих проблем надало великий практичний досвід.

Готовий проєкт має практичне значення. Його можна використовувати як навчальний приклад для студентів, які вивчають Python, Pygame та основи розробки ігор. Модульна структура коду дозволяє легко додавати нові функції, тому гра може стати основою для подальших проєктів.

У майбутньому планується продовжувати розвиток «Zombie Escape». Найближчі завдання — заміна прямокутників на спрайти, додавання звуку, різних типів ворогів, системи прокачування та підготовка до публікації на itch.io.

Ця дипломна робота підтвердила, що Python разом з Pygame є потужним і зручним інструментом для створення 2D-ігор, особливо в навчальних цілях. На відміну від готових ігрових движків, тут розробник має повний контроль над кожним елементом гри, що дозволяє глибше зрозуміти, як працюють ігрові механізми.

Ця робота стала важливим практичним етапом. Набуто навичок не лише написання коду, але й планування проєкту, вирішення проблем, тестування продукту та доведення справи до кінця. Отримані знання та навички будуть корисні в подальшому навчанні та професійній діяльності.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Lutz M. Learning Python. 5th Edition. – Sebastopol: O'Reilly Media, 2013. – 1594 p.
2. Sweigart A. Making Games with Python & Pygame [Електронний ресурс]. – 2012. – URL: https://inventwithpython.com/pygame/ (дата звернення: 10.06.2026).
3. Pygame Community. Pygame Documentation [Електронний ресурс]. – URL: https://www.pygame.org/docs/ (дата звернення: 10.06.2026).
4. Schell J. The Art of Game Design: A Book of Lenses. 3rd Edition. – Boca Raton: CRC Press, 2019. – 600 p.
5. Rogers S. Level Up!: The Guide to Great Video Game Design. 2nd Edition. – Chichester: Wiley, 2014. – 512 p.
6. McGugan W. Beginning Game Development with Python and Pygame. – Berkeley: Apress, 2007. – 340 p.
7. Newzoo. Global Games Market Report 2024. – Amsterdam: Newzoo, 2024.
8. Nystrom R. Game Programming Patterns [Електронний ресурс]. – URL: https://gameprogrammingpatterns.com/ (дата звернення: 10.06.2026).
9. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston: Addison-Wesley, 1994. – 395 p.
10. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. – Upper Saddle River: Prentice Hall, 2008. – 431 p.
11. Millington I., Funge J. Artificial Intelligence for Games. 2nd Edition. – Burlington: Morgan Kaufmann, 2009. – 870 p.
12. TIOBE Software. TIOBE Index [Електронний ресурс]. – URL: https://www.tiobe.com/tiobe-index/ (дата звернення: 10.06.2026).
13. Dawson M. Python Programming for the Absolute Beginner. 3rd Edition. – Boston: Course Technology, 2010. – 480 p.

14. Габлук Е. С., Павлов В. Г. Алгоритми та структури даних: навчальний посібник. – К.: Вища школа, 2018. – 448 с.
15. Python Software Foundation. The Python Standard Library [Електронний ресурс]. – URL: https://docs.python.org/3/library/ (дата звернення: 10.06.2026).
16. SDL Community. SDL2 Wiki [Електронний ресурс]. – URL: https://wiki.libsdl.org/SDL2 (дата звернення: 10.06.2026).
17. Itch.io. Developer Resources [Електронний ресурс]. – URL: https://itch.io/docs/creators/ (дата звернення: 10.06.2026).
18. Steam Spy. Steam Statistics 2024 [Електронний ресурс]. – URL: https://steamspy.com/ (дата звернення: 10.06.2026).
19. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 3rd Edition. – Cambridge: MIT Press, 2009. – 1292 p.
20. Beazley D., Jones B. K. Python Cookbook. 3rd Edition. – Sebastopol: O'Reilly, 2013. – 706 p.
21. Al Sweigart. Automate the Boring Stuff with Python. 2nd Edition. – San Francisco: No Starch Press, 2019. – 592 p.
22. Thonny IDE Documentation [Електронний ресурс]. – URL: https://thonny.org/ (дата звернення: 10.06.2026).
23. Ukrainian Association of Game Developers. Ukrainian Game Industry Report 2024.
24. Habgood J., Overmars M. The Game Maker's Apprentice: Game Development for Beginners. – Berkeley: Apress, 2006. – 336 p.
25. Game Developers Conference. GDC Vault [Електронний ресурс]. – URL: https://www.gdcvault.com/ (дата звернення: 10.06.2026).

ДОДАТОК А

# Повний програмний код гри «Zombie Escape»

## А.1 – Файл main.py (головний модуль)

```
import pygame
import  random
import math
from player import Player
from my_zombie import Zombie
from bullet import Bullet
# Налаштування екрану
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
FPS = 60
pygame.init()
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption("Zombie Escape PA")
clock = pygame.time.Clock()
font = pygame.font.SysFont(None, 72)
small_font = pygame.font.SysFont(None, 36)
def reset_game():
player = Player(400, 300)
zombies = []
bullets = []
spawn_timer = 0
score = 0
game_over = False
return player, zombies, bullets, spawn_timer, score, game_over
player, zombies, bullets, spawn_timer, score, game_over = reset_game()
running = True
while running:
```

```
# EVENTS
for event in pygame.event.get():
if event.type == pygame.QUIT:
running = False
elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1 and not
game_over:
mx, my = pygame.mouse.get_pos()
dx = mx - player.rect.centerx
dy = my - player.rect.centery
distance = math.hypot(dx, dy)
if distance != 0:
direction = (dx / distance, dy / distance)
bullets.append(Bullet(player.rect.centerx, player.rect.centery, direction))
elif event.type == pygame.KEYDOWN and event.key == pygame.K_r and
game_over:
player, zombies, bullets, spawn_timer, score, game_over = reset_game()
# GAME LOGIC
if not game_over:
keys = pygame.key.get_pressed()
player.move(keys)
# поява зомбі
spawn_timer += 1
if spawn_timer >= 90:
side = random.choice(["top", "bottom", "left", "right"])
if side == "top":
x = random.randint(0, SCREEN_WIDTH)
y = 0
elif side == "bottom":
x = random.randint(0, SCREEN_WIDTH)
y = SCREEN_HEIGHT
```

```
elif side == "left":
x = 0
y = random.randint(0, SCREEN_HEIGHT)
else:
x = SCREEN_WIDTH
y = random.randint(0, SCREEN_HEIGHT)
zombies.append(Zombie(x, y))
spawn_timer = 0
# рух зомбі
for zombie in zombies:
zombie.move_towards(player.rect)
# рух куль
for bullet in bullets[:]:
bullet.update()
bullets = [
b for b in bullets
if 0 <= b.rect.x <= SCREEN_WIDTH and 0 <= b.rect.y <= SCREEN_HEIGHT
]
# зіткнення куля-зомбі
for bullet in bullets[:]:
for zombie in zombies[:]:
if bullet.rect.colliderect(zombie.rect):
bullets.remove(bullet) zombies.remove(zombie)
score += 1
break
# зіткнення зомбі-гравець
for zombie in zombies[:]:
if zombie.rect.colliderect(player.rect):
player.health -= 1
```

```
if player.health <= 0:
game_over = True
# DRAW
screen.fill((30, 30, 30))
player.draw(screen)
for zombie in zombies:
zombie.draw(screen)
for bullet in bullets:
bullet.draw(screen)
score_text = small_font.render(f"Score: {score}", True, (255, 255, 255))
screen.blit(score_text, (10, 10))
pygame.draw.rect(screen, (255, 0, 0), (10, 40, 200, 20))
pygame.draw.rect(screen, (0, 255, 0), (10, 40, player.health * 2, 20))
# Програв гру
if game_over:
game_over_text = font.render("GAME OVER", True, (255, 0, 0))
restart_text = small_font.render("Press R to Restart", True, (255, 255, 255))
final_score_text = small_font.render(f"Your score: {score}", True, (255, 255, 255))
screen.blit(game_over_text, (SCREEN_WIDTH // 2 - 180, SCREEN_HEIGHT // 2 -
80))
screen.blit(final_score_text, (SCREEN_WIDTH // 2 - 90, SCREEN_HEIGHT // 2))
screen.blit(restart_text, (SCREEN_WIDTH // 2 - 120, SCREEN_HEIGHT // 2 + 40))
pygame.display.flip()
clock.tick(FPS)
pygame.quit()
```

A.2 – Файл player.py (клас Player)

```
import pygame
class Player:
def __init__(self, x, y):
self.rect = pygame.Rect(x, y, 40, 40)
self.speed = 5
self.health = 100
def move(self, keys):
if keys[pygame.K_w]: self.rect.y -= self.speed
if keys[pygame.K_s]: self.rect.y += self.speed
if keys[pygame.K_a]: self.rect.x -= self.speed
if keys[pygame.K_d]: self.rect.x += self.speed
self.rect.x = max(0, min(self.rect.x, 800 - self.rect.width))
self.rect.y = max(0, min(self.rect.y, 600 - self.rect.height))
def draw(self, screen):
pygame.draw.rect(screen, (0, 255, 0), self.rect)
```

A.3 – Файл my_zombie.py (клас Zombie)

```
import pygame
import math
class Zombie:
def __init__(self, x, y):
self.x = float(x)
self.y = float(y)
self.rect = pygame.Rect(x, y, 30, 30)
self.speed = 1.5
def move_towards(self, player_rect):
dx = player_rect.x - self.rect.x
dy = player_rect.y - self.rect.y
distance = math.hypot(dx, dy)
```

```
if distance != 0:
self.x += dx / distance * self.speed
self.y += dy / distance * self.speed
self.rect.x = int(self.x)
self.rect.y = int(self.y)
def draw(self, screen):
pygame.draw.rect(screen, (255, 0, 0), self.rect)
```

A.4 – Файл bullet.py (клас Bullet)

```
import pygame
class Bullet:
def __init__(self, x, y, direction):
self.rect = pygame.Rect(x, y, 5, 5)
self.speed = 10
self.direction = direction # нормалізований вектор напрямку (dx, dy)
def update(self):
self.rect.x += self.direction[0] * self.speed
self.rect.y += self.direction[1] * self.speed
def draw(self, screen):
pygame.draw.rect(screen, (255, 255, 0),
self.rect)
```