ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Комп'ютерна гра «Intra» з використанням алгоритмів теорії графів»

Ступінь вищої освіти – бакалавр

Спеціальність – 122 «Комп’ютерні науки»

Освітня програма «Комп’ютерні науки»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: здобувач 4 курсу
групи КН-21
Максим ЛОТОЦЬКИЙ

Керівник: старший викладач кафедри
комп’ютерних наук
Олег ЛУКУТІН

Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.
Здобувач: _____
(підпис)

м. Київ – 2025 рік

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри
комп’ютерних наук
_Сергій МІЧКІВСЬКИЙ
«_____»____20___р

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лотоцький Максим Павлович

Тема роботи	Комп'ютерна гра «Intra» з використанням алгоритмів теорії графів
Номер та дата наказу про затвердження теми	№121-7 від 24 грудня 2024 року
Коротка постановка завдання	Розробка комп’ютерної гри «Intra» з використанням алгоритмів теорії графів для динамічного пошуку шляхів.
Посилання на джерела інформації (не більше п’яти найменувань, які рекомендує науковий керівник)	Кузьменко, І. М. Теорія графів [Електронний ресурс]: навчальний посібник для здобувачів ступеня бакалавра за спеціальністю 122 "Комп'ютерні науки" / І. М. Кузьменко ; КПІ ім. Ігоря Сікорського. - Електронні текстові дані (1 файл: 1,25 Мбайт). - Київ: КПІ ім. Ігоря Сікорського, 2020. - 71 с. URL: https://ela.kpi.ua/handle/123456789/35854 1. Jeff Erickson Algorithms [Електронний ресурс]: a free electronic version of a self-published textbook Algorithms URL: https://jeffe.cs.illinois.edu/teaching/algorithms/
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має містити теоретичне, системотехнічне або експериментальне дослідження за темою роботи, яку слід розглядати як складне спеціалізоване завдання або практичну проблему в галузі комп’ютерних наук, яка характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів інформаційних технологій.

Дата видачі завдання 27 грудня 2024 р.

Керівник Олег ЛУКУТІН

Здобувач освітнього ступеня бакалавра Максим ЛОТОЦЬКИЙ

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
	Підготовчий етап		
1	Вибір напряму дослідження	02.12.2024 р.	*виконано*
2	Формування теми та призначення керівника	16.12.2024 р.	*виконано*
3	Затвердження теми кваліфікаційної роботи	23.12.2024 р.	*виконано*
4	Затвердження завдання на кваліфікаційну роботу	27.12.2024 р.	*виконано*
	Основний етап		
5	Розробка концепції кваліфікаційної роботи	13.01.2025 р.	*виконано*
6	Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів	20.01.2025 р.	*виконано*
7	Затвердження розширеної постановки завдання. Підготовка та подання керівникові розділу 1 кваліфікаційної роботи	10.03.2025 р.	*виконано*
8	Проєктування. Підготовка та подання керівникові розділу 2 кваліфікаційної роботи	24.03.2025 р.	*виконано*
9	Підготовка доповіді для експертизи стану виконання кваліфікаційної роботи (проміжний контроль)	31.03-04.04.2025 р.	*виконано*
10	Реалізація. Підготовка та подання керівникові розділу 3 кваліфікаційної роботи	07.04.2025 р.	*виконано*
11	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи	14.04.2025 р.	*виконано*
12	Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівникові доопрацьованого варіанту кваліфікаційної роботи	21.04.2025 р.	*виконано*
	Завершальний етап		
13	Представлення рукопису для перевірки на плагіат	28.04-04.05.2025 р.	*виконано*
14	Підготовка презентації та доповіді на передзахист	05.05-11.05.2025 р.	*виконано*
15	Передзахист кваліфікаційної роботи	12.05-16.05.2025 р.	*виконано*
16	Доопрацювання роботи за результатами передзахисту	19.05-06.06.2025 р.	*виконано*
17	Експертиза роботи керівником та зовнішнім експертом	09.06-15.06.2025 р.	*виконано*
18	Доопрацювання доповіді та презентації для захисту	09.06-15.06.2025 р.	*виконано*
19	Захист кваліфікаційної роботи	16.06-22.06.2025 р.	*виконано*

Керівник _____Олег ЛУКУТІН

Здобувач освітнього ступеня бакалавра _____Максим ЛОТОЦЬКИЙ

Лотоцький М.П. Комп'ютерна гра «Intra» з використанням алгоритмів теорії графів.

Пояснювальна записка кваліфікаційної роботи за спеціальністю 122 - Комп'ютерні науки (освітня програма - Комп'ютерні науки) СО Бакалавр. - ВНЗ "Університет економіки та права "КРОК", Навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра комп'ютерних наук, Київ, 2025.

Розглянуто проблеми розробки алгоритмів у комп'ютерних іграх на прикладі розробки гри "Intra". Реалізовано комп'ютерну гру "Intra" з використанням Unity та застосуванням алгоритмів теорії графів для пошуку шляхів.

Ключові слова: комп'ютерна гра, Unity, алгоритми теорії графів, пошук шляхів.

Рис. 22. Бібліограф.: 14 найм.

Lototskyi.M.P. Computer game "Intra", using graph theory algorithms for pathfinding.

Project explanatory note by specialty 122 - Computer science. - "KROK" University, Educational and Scientific Institute of information and communication technologies, Department of Computer Science, Kyiv, 2025.

The problems of developing algorithms in computer games are considered on the example of the development of the game "Intra". The computer game "Intra" was implemented using Unity and the application of graph theory algorithms for pathfinding.

Keywords: computer game, Unity, graph theory algorithms, pathfinding.

Fig. 22. Bibliography: 14 Items.

ЗМІСТ

ВСТУП

Актуальність теми. Комп'ютерні ігри є однією з найбільших галузей індустрії розваг та інформаційних технологій. Вони є важливими не лише у сфері розваг, а й у розвитку штучного інтелекту, графіки. Особливу роль у цьому відіграють алгоритми теорії графів, що широко використовуються для створення ефективних механік навігації, що у свою чергу дозволяє існувати складним ігровим світам. Тема розробки комп'ютерної гри "Intra", з використанням алгоритмів теорії графів, є актуальною, оскільки дозволяє дослідити та впровадити сучасні методи пошуку шляхів та оптимізації ігрових процесів.

Проблемна ситуація. Сучасна ігрова індустрія потребує використання пошукових алгоритмів, що дозволяють створювати цікаві й непередбачувані ситуації у грі. Використання алгоритмів теорії графів відкриває нові можливості для розробки ефективних систем навігації та інтерактивних середовищ.

Мета дослідження полягає в розробці комп’ютерної гри «Intra» з використанням алгоритмів теорії графів для динамічного пошуку шляхів.

Завдання дослідження:

аналіз існуючих методів використання алгоритмів теорії графів у комп'ютерних іграх (A*/Дейкстри/BFS/DFS);

проектування архітектури комп’ютерної гри "Intra" з використанням діаграм;

розробити програмне забезпечення комп’ютерної гри Intra.

Об'єктом дослідження є ігровий процес навігації у віртуальному світі за допомогою теорії графів.

Предмет дослідження – комп'ютерні ігри, в яких використовуються алгоритми теорії графів для пошуку оптимальних шляхів.

Методи дослідження – алгоритми теорії графів, зокрема алгоритми пошуку шляхів (A*, Дейкстри, Хвильовий).

Практичне значення. Розроблена комп'ютерна гра "Intra", що застосовує алгоритми теорії графів для визначення шляхів навігації персонажів.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та списку використаних джерел (14 найменувань). Пояснювальна записка містить 22 рисунків. Загальний обсяг пояснювальної записки складає 40 сторінок, основний зміст викладено на 37 сторінках.

РОЗДІЛ 1
ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ КОМП'ЮТЕРНОЇ ГРИ "INTRA" З ВИКОРИСТАННЯМ АЛГОРИТМІВ ТЕОРІЇ ГРАФІВ

1.1 Предметна область

Комп'ютерні ігри є однією з найпопулярніших та найприбутковіших форм індустрії розваг у сучасному суспільстві [1]. Вони поєднують у собі елементи мистецтва, програмування та дизайну, створюючи абсолютно унікальний досвід за рахунок інтерактивності та взаємодії з навколишнім світом. Одним із важливих аспектів розробки ігор є керування персонажами, їх навігації ігровим світом і динамічна реакція навколишнього світу на дії гравця. У цьому контексті особливу роль становить застосування алгоритмів теорії графів, які забезпечують ефективний пошук маршрутів та розв'язання складних навігаційних завдань.

Гра "Intra" є платформером, в якому гравець досліджує рівні, що містять множинні шляхи та перешкоди. Для забезпечення взаємодії з ігровим середовищем використано алгоритми теорії графів, що дозволяють ефективно визначати маршрути переміщення супротивників. Зокрема, у грі застосовуються алгоритми пошуку шляхів, що гарантують швидке й ефективне знаходження оптимального маршруту у просторах гри.

Важливо відзначити, що алгоритми навігації в іграх мають широкий спектр застосування. Вони використовуються не лише для моделювання поведінки неігрових персонажів (NPC), що дозволяє створювати динамічні та реалістичні сценарії, але і для визначення оптимальних маршрутів у певних ситуаціях і жанрах (такі як визначення оптимального шляху по мапі або Point and Click adventure). Завдяки цьому ігровий процес стає більш захопливим і комфортним для користувачів.

Окрім цього, використання алгоритмів теорії графів відкриває нові можливості у створенні рівнів, оскільки дозволяє динамічно змінювати структуру ігрового середовища відповідно до дій гравця без негативного

впливу на логіку дії неігрових персонажів. Це дозволяє реалізувати нелінійний ігровий процес, де користувач має можливість обирати різні стратегії проходження.

Таким чином, дослідження та впровадження алгоритмів теорії графів у розробку комп'ютерної гри "Intra" є важливим кроком у вдосконаленні механік платформерів та створенні інтерактивних світів.

1.2 Аналіз конкурентів

Animal Well (рис. 1.1) – інді (indie) гра в жанрі метроїдванія, що поєднує платформерні елементи з дослідженням нелінійного світу [2].

Гра вирізняється складними головоломками, секретами та навігацією заплутаними рівнями, що вимагає використання навігаційних алгоритмів для різних типів ворогів, що переслідують гравця.

Переваги:

- великий ігровий світ;
- висока інтерактивність з навколишнім середовищем;
- динамічна система переслідування гравця.

Недоліки:

- відсутність бойової системи.

Рисунок 1.1 - Гра Animal Well

Джерело: [2]

Hollow Knight (рис. 1.2) – ще один приклад метроїдванії, що поєднує в собі жанри платформеру та метроїдванії - нелінійну структуру дослідження світу [3].

Використання алгоритмів графів дозволяє створювати різноманітних літаючих ворогів, які переслідують гравця та займають певні позиції для атаки.

Переваги:

- великий ігровий світ у жанрі метроїдванія;
- зручний інтерфейс;
- чудовий стиль графіки, якість музики.

Недоліки:

- деякі вороги не переслідують гравця на достатню відстань.

Рисунок 1.2 - Гра Hollow Knight

Джерело: [3]

The Binding of Isaac (рис. 1.3) – гра жанру Roguelike з видом згори, що є одним з найкращих представників свого жанру [5].

Завдяки унікальному розміщенню камери (вид зверху вниз), сегментного та випадкового створення карти, ігрової площі яка складається з «клітин», абсолютна більшість ворогів може використовувати алгоритми графів задля відстеження та переслідування гравця.

Переваги:

- неймовірно якісний геймплей у жанрі roguelike;
- велика кількість унікального контенту;
- динамічна система переслідування гравця.

Недоліки:

- залежність від удачі за рахунок жанру Roguelike.

Рисунок 1.3 - Гра The Binding of Isaac

Джерело: [4]

За результатами аналізу аналогів, виділимо переваги комп'ютерної гри що розробляється:

- динамічна система переслідування гравця;
- наявність бойової системи.

1.3 Постановка задачі

Метою кваліфікаційної роботи є розробка комп’ютерної гри Intra, в якій алгоритми теорії графів застосовуються для побудови адаптивної навігації ворогів у внутрішньоігровому середовищі. Головним завданням є реалізація ефективної поведінки неігрових персонажів, які здатні динамічно орієнтуватися в просторі гри, реагуючи на перешкоди та дії гравця.

Функціональні вимоги:

- користувач повинен мати змогу пересуватися в ігровому світі та взаємодіяти з ним;
- змога відродження персонажу у випадку поразки за допомогою системи чекпойнтів;
- можливість атакувати, отримувати пошкодження та перемагати ворогів (бойова система);
- динамічна система навігації неігрових персонажів на основі алгоритмів теорії графів.

Нефункціональні вимоги:

- застосунок повинен швидко завантажуватися;
- підтримувати функціонування у 60 fps;
- інтуїтивне управління та дизайн інтерфейсу;

Очікується отримати додаток-гру у якому будуть реалізовані такі функції як:

- ігровий світ;
- пересування та колізія з навколишнім середовищем;
- система навігації неігрових персонажів на основі алгоритмів графів;
- інтерфейс;
- відстеження параметрів/статусу ігрового персонажа (здоров’я);
- бойова система.

Висновки до розділу 1

У розділі було розглянуто стан ігрової індустрії, яку роль відіграють алгоритми теорії графів у створенні ігор. Також сплановано створення комп’ютерної гри Intra з акцентом на використання алгоритмів пошуку шляху для навігації неігрових персонажів.

Було проаналізовано існуючі рішення та конкурентів, зокрема такі ігри, як Animal Well та Hollow Knight.

Було визначено основні вимоги для завдання кваліфікаційної роботи.

РОЗДІЛ 2
ПРОЄКТУВАННЯ

2.1 Проєктування структури

Після запуску гри, програмне забезпечення починає обробку внутрішньоігрової фізики та інших ігрових даних.

Користувачу надається управління головним персонажем, в результаті чого він отримує можливість переміщатися, атакувати, отримувати пошкодження, активувати чекпойнти та відновлювати раніше втрачене здоров’я.

На рис. 2.1 представлена Use-case diagram.

Рисунок 2.1 - Use-case diagram

Джерело: розроблено автором

2.2 Моделювання даних

На рис. 2.2 представлена діаграма класів (Class diagram)

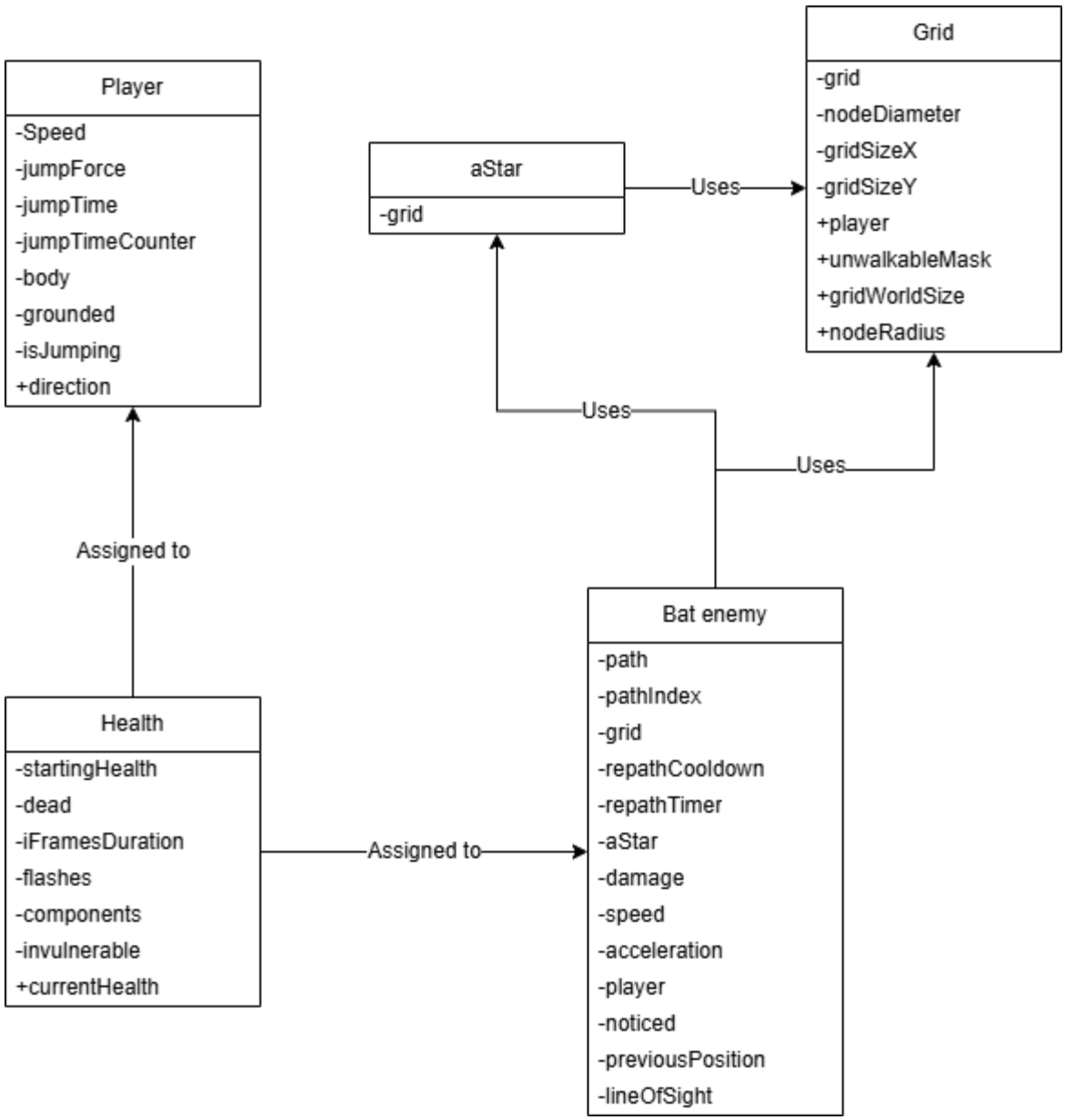

Рисунок 2.2 - Class diagram

Джерело: розроблено автором

Знак «-» позначає приватні змінні.

Знак «+» позначає публічні змінні.

Клас Player в основному визначає пересування гравця. Клас Health слідкує за поточним станом здоров’я як гравця, так і ворогів і визначає коли їх потрібно деактивувати у випадку падіння здоров’я до 0. Клас Bat enemy використовує обидва класи aStar та Grid, щоб проводити динамічну навігацію від поточного місцезнаходження до позиції гравця.

2.3 Моделювання процесів

Також побудовано діаграму діяльності (Activity Diagram), що описує цикл гри: запуск, обробка введення, перевірка зіткнень, обробка подій (рис. 2.3).

Рисунок 2.3 - Activity diagram

Джерело: розроблено автором

2.4 Моделювання процесів: пошук шляхів за допомогою теорії графів

У програмному забезпеченні має бути реалізована механіка переміщення ворогів по складному ігровому середовищу, що потребує

алгоритмів пошуку шляху. Моделювання такого процесу виконується за допомогою теорії графів: ігровий простір подається у вигляді сітки (ґратки), де кожна клітинка відповідає вузлу графа, а можливі переходи між ними — ребрам. Клітини/вершини недоступні для переміщення позначаються окремо як недоступні вершини і не використовуються у знаходженні шляху, таким чином алгоритм оминає перешкоди.

Тож у даному проєкті буде розглянуто наступні алгоритми пошуку шляху:

1. Алгоритм хвильового пошуку (або ж алгоритм Лі):

Це метод пошуку шляху, який базується на алгоритмі пошуку в ширину (BFS). Його принцип роботи складається у тому, що визначається цільова клітинка (наприклад, позиція гравця) і вона отримує значення 0. Кожна сусідня клітинка яка не є перешкодою отримує значення, яке на одиницю більше за значення поточної клітинки (як на рис. 2.4). Цей процес продовжується, або поки всі можливі клітини будуть досягнуті, або поки не буде досягнута ціль, після чого будується зворотній шлях.

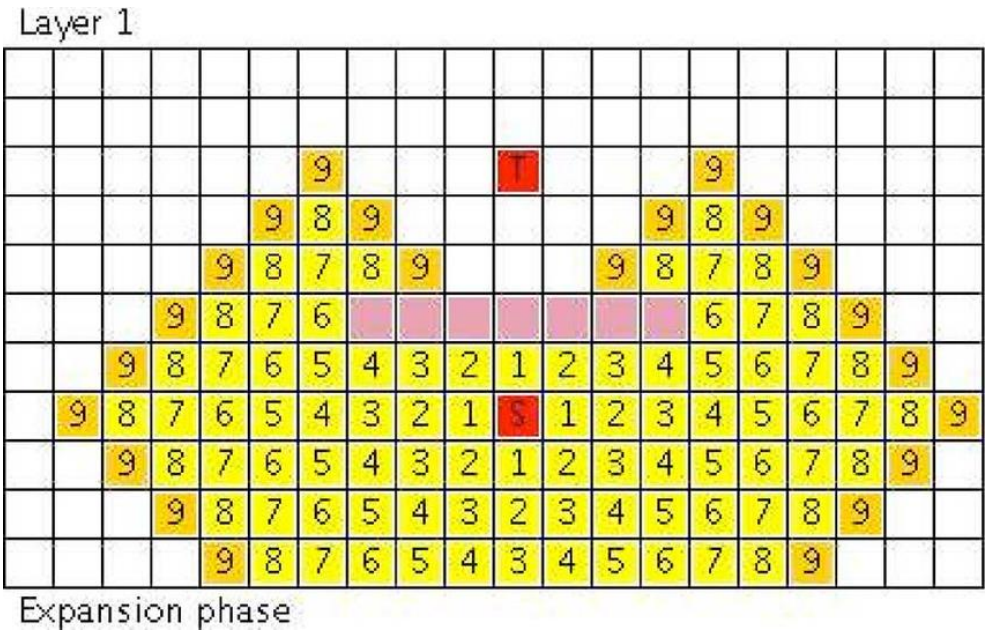

Рисунок 2.4 Алгоритм Лі у стадії поширення

Джерело: [6]

Переваги:

- простий у реалізації;
- гарантує знайдення найкоротшого шляху.

Недоліки:

- не враховує пріоритетів/напрямку;
- використовує значні ресурси при пошуку на великих графах.

2. Алгоритм Дейкстри:

Алгоритм Дейкстри знаходить найкоротші шляхи від початкового вузла до всіх інших або до цільової вершини у графі з невід'ємними вагами ребер [7].

Принцип роботи полягає у тому, що усі вузли отримують початкову відстань ∞, окрім стартового (0). Далі, сусіднім вузлам призначається відстань в залежності від ваги яку має з'єднуване ребро. Вибирається вузол з мінімальною відстанню та для кожного його сусіда оновлюється відстань, що є сумою ваги ребра та відстанню обраної вершини. Процес повторюється, поки не будуть опрацьовані всі вузли або знайдено ціль.

Переваги:

- гарантовано знаходить найкоротший шлях.

Недоліки:

- не має напрямку до цілі;
- алгоритм обробляє зайві вузли;
- важчий у реалізації порівняно з хвильовим;
- все ще використовує значні ресурси при пошуку на великих графах.

На рис. 2.5 можна побачити процес виконання алгоритму Дейкстри. Починаючи від вершини C, визначається відстань для вершин A, B, D – 1, 7, 2 відповідно. З найменшої вершини A оновлюються сусідні вершини, тому змінюємо значення на менше для вузла B – 4. Тепер вершиною з найменшою відстанню є D, але для сусіднього вузла B є вже менше значення за 7, тому

залишаємо його та встановлюємо значення для E – 9. Наступна найменша вершина є B, оновлюємо відстань для вузла E на 5. Алгоритм виконано.

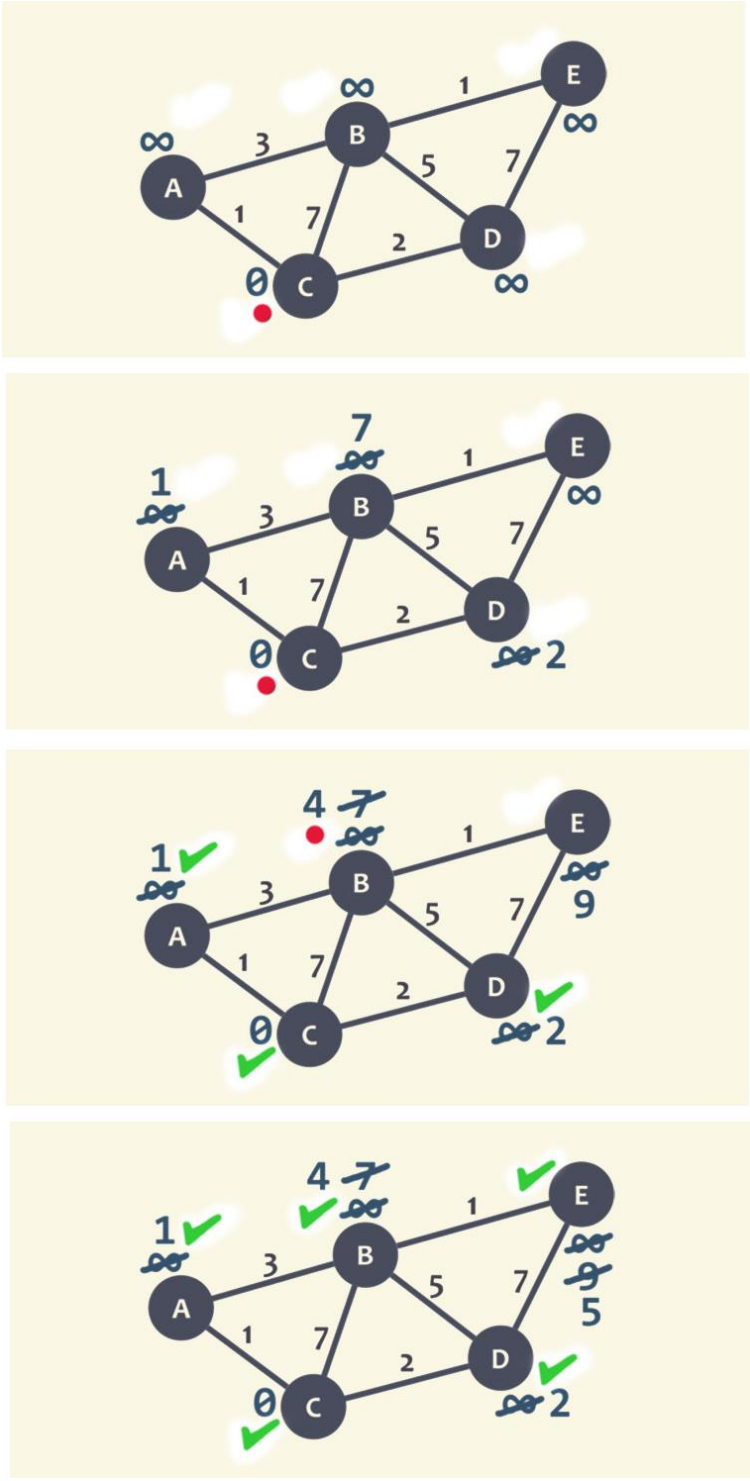

Рисунок 2.5 Приклад роботи алгоритму Дейкстри

Джерело: [7]

3. Алгоритм A* (A-Star):

A* це евристичний алгоритм пошуку, який використовує функцію вартості:

f(n) = g(n) + h(n)

де:

g(n) — відстань від початку шляху до вузла **n**;

h(n) — відстань від вузла **n** до цілі [8].

За допомогою цієї функції алгоритм визначає ціну сусіднім клітинам з поточної/обраної. Далі клітини/вершини що мають нижче значення f перевіряються першими та оновлюють їх сусідів. Якщо існує декілька клітин з однаковим значенням f, обирається клітина з найменшим значенням h, тобто клітина що найближча до кінця шляху.

На рис. 2.6 зображено A-Star алгоритм у дії, де синя клітина – початкова, червона – вже перевірена, зелена – сусідня клітина g, h, F score яких вирахували, жовта клітина – ціль шляху.

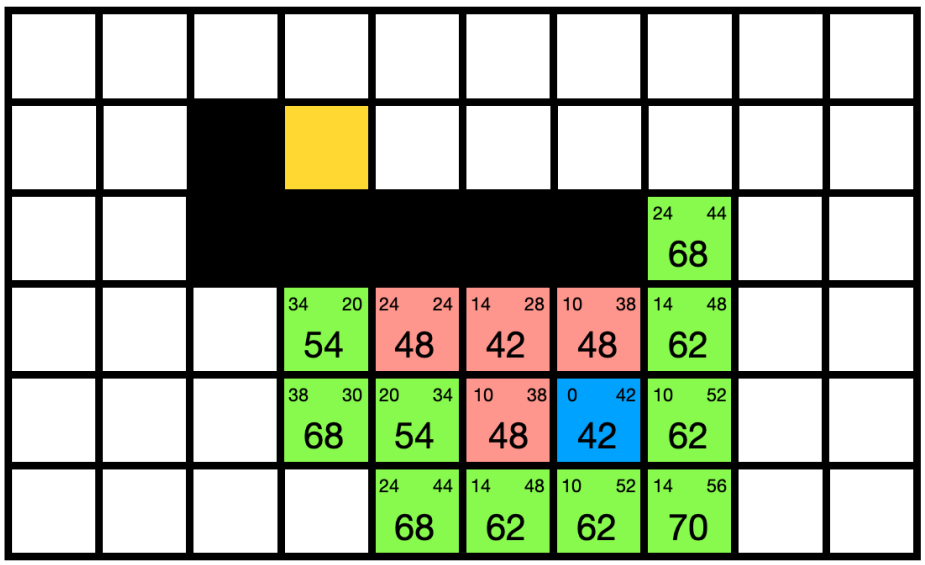

Рисунок 2.6 A-Star алгоритм у дії

Джерело: [9]

Алгоритм відстежує з якої клітини було отримане оновлення та при досягнені кінцевої цілі прокладає зворотній шлях згідно з цими даними.

Переваги:

- працює швидше, ніж BFS (алгоритми пошуку у ширину);
- завдяки евристичному підходу визначає напрям руху алгоритму.

Недоліки:

- складніша реалізація.

За рахунок евристичних методів, використаних у A-Star, цей алгоритм є швидшим за алгоритм Дейкстри [10] та оптимальним для вирішення поставленої задачі, а саме реалізації алгоритму пошуку шляхів для неігрових персонажів.

Висновки до розділу 2

У розділі було спроектовано структуру програмного забезпечення за допомогою діаграм.

Проведено аналіз трьох алгоритмів теорії графів та обрано оптимальний варіант.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1 Особливості реалізації

Середовищем розробки було обрано ігровий рушій Unity. Це один з найпотужніших рушіїв 2D та 3D інді індустрії, який був використаний у таких проєктах, як Cuphead, Outer Wilds, Hollow Knight, Subnautica, DREDGE, Rimworld, Ori & The Blind Forest та інші [11].

Значною перевагою Unity є наявність Unity Asset Store – офіційного маркетплейсу графічних, звукових, анімаційних та програмних ресурсів, використання яких може істотно покращити оформлення візуальної частини гри.

Для реалізації візуального оформлення проєкту були використані наступні безкоштовні ассети:

- Dragon Warrior – спрайти та анімація головного героя [12];
- Pixel Adventure 1 – елементи оточення [13];
- Enemy Galore 1 - Pixel Art - спрайти ворогів[14].

Unity використовує мову C# для програмування, яка є популярною об’єктно-орієнтованою мовою з широкою документаційною базою. Однією з найбільших переваг Unity та C# є підтримка багатоплатформної розробки – зокрема, для Windows, Linux та Android, що дозволяю зручно портувати проєкт на різні операційні системи.

Для написання коду було обрано інтегроване середовище розробки (IDE) Microsoft Visual Studio. Це є одним з найпопулярніших середовищ розробки, яке спрощує форматування коду, має підсвітку синтаксису та інтеграцію з Unity. Якщо інстальовано, Visual Studio є середовищем розробки для Unity за замовчуванням.

У випадках, коли Asset Store не містить необхідних ресурсів, використовувався онлайн редактор Piskel для зручного та швидкого створення простого піксель арту.

Ігровому рушію Unity притаманна компонентно-орієнтована архітектура програмного забезпечення - CBA (Component-Based Architecture). Сутність цієї архітектури полягає поділі системи на компоненти, кожен із яких виконує окрему функцію. Наприклад компонент PlayerMovement.cs відповідає за переміщення головного персонажа та містить відповідні дані (такі як швидкість, сила стрибку).

Основними перевагами цієї архітектури є:

- капсуляція – логіка кожного модуля є ізольованою
- модульність – компоненти легко оновлювати або замінювати на аналогічні по функціонуванню компоненти.
- повторне використання - один компонент може бути застосований до кількох об’єктів
- гнучкість – компоненти взаємодіють між собою через визначені інтерфейси.

Компонентно-орієнтований підхід у Unity дозволяє будувати складні ігрові системи, де логіка поведінки об'єктів розподіляється між спеціалізованими скриптами-класами, прикріпленими до об’єктів сцени.

Реалізація алгоритму пошуку

Скрипт AStar.cs реалізує алгоритм A*, який дозволяє ігровим об’єктам (наприклад, ворогам) знаходити оптимальний маршрут від поточного положення до гравця, оминаючи перешкоди. Простір гри представлено у вигляді сітки (Grid), де кожна клітинка — це вершина або ж вузол (Node).

На рис. 3.1 знаходиться код ініціації. Він створює приватну змінну grid відповідного класу та отримує посилання на компонент Grid.

На рис. 3.2 знаходиться основна частина коду aStar, що міститься у функції FindPath(pos1, pos2), яка приймає 2 позиції на карті для знаходження шляху між ними.

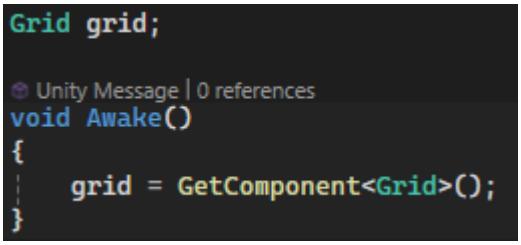

Рисунок 3.1 - Initialization code

Джерело: розроблено автором

За допомогою grid.NodeFromWorldPoint(pos) знаходимо клітини сітки, що відповідають позиціям об’єктів (ворога та гравця).

Створюємо списки вершин що вже були відвідані, та тих що ні, додаємо початковий вузол.

У циклі:

У частині for (...) if (...) node = openSet[i]; визначаємо вершину з найнижчим fScore (якщо їх декілька, то також найнижчим hScore – найближчою відстанню до кінцевої вершини).

Видаляємо вузол зі списку не відвіданих, додаємо до відвіданих.

Якщо вузол обраний зараз є кінцевою ціллю – запускається RetracePath() з вершинами початку та кінця шляху, припинення функції.

Стрічка foreach (Node neighbour in grid.GetNeighbours(node)) оброблює кожну сусідню клітину, а наступний if пропускає непрохідні вузли або вже пройдені вершини.

Кожному ж іншому сусіду обчислюється нове значення gScore за допомогою GetDistance(node, neighbour). Якщо gScore менший за минулий або ж вершини-сусіда немає у списку невідвіданих, для сусіда вираховуються нові значення gScore та hScore, також зберігається з якої клітини цього сусіда було оновлено.

На рис. 3.3 знаходиться код RetracePath(start, end) та GetDistance(nodeA, nodeB).

```
public void FindPath(Vector3 startPos, Vector3 targetPos)
{
    Node startNode = grid.NodeFromWorldPoint(startPos);
    Node targetNode = grid.NodeFromWorldPoint(targetPos);

    List<Node> openSet = new List<Node>();
    HashSet<Node> closedSet = new HashSet<Node>();
    openSet.Add(startNode);

    while (openSet.Count > 0)
    {
        Node node = openSet[0];
        for (int i = 1; i < openSet.Count; i++)
        {
            if (openSet[i].FScore < node.FScore || openSet[i].FScore == node.FScore)
            {
                if (openSet[i].hScore < node.hScore)
                    node = openSet[i];
            }
        }

        openSet.Remove(node);
        closedSet.Add(node);

        if (node == targetNode)
        {
            RetracePath(startNode, targetNode);
            return;
        }

        foreach (Node neighbour in grid.GetNeighbours(node))
        {
            if (!neighbour.walkable || closedSet.Contains(neighbour))
            {
                continue;
            }

            float newCostToNeighbour = node.gScore + GetDistance(node, neighbour);
            if (newCostToNeighbour < neighbour.gScore || !openSet.Contains(neighbour))
            {
                neighbour.gScore = newCostToNeighbour;
                neighbour.hScore = GetDistance(neighbour, targetNode);
                neighbour.cameFrom = node;

                if (!openSet.Contains(neighbour))
                    openSet.Add(neighbour);
            }
        }
    }
}
```

Рисунок 3.2 - FindPath()

Джерело: розроблено автором

RetracePath(start, end) створює новий список, призначає кінець шляху поточною клітиною і за збереженими даними node.cameFrom додає у список шлях який алгоритм пройшов раніше, щоб дістатися кінця. А Path.Reverse() змінює порядок вершин у списку.

GetDistance(nodeA, nodeB) вираховує дистанцію між двома клітинами, де кардинальне з'єднання рахується за 10, а діагональне за 14 (припустимо, що відстань між кардинальними вузлами є 1см, то за теоремою Піфагора діагональна відстань має бути ~1.4см).

```
void RetracePath(Node startNode, Node endNode)
{
    List<Node> path = new List<Node>();
    Node currentNode = endNode;

    while (currentNode != startNode)
    {
        path.Add(currentNode);
        currentNode = currentNode.cameFrom;
    }
    path.Reverse();

    grid.path = path;

}

2 references
int GetDistance(Node nodeA, Node nodeB)
{
    int dstX = Mathf.Abs(nodeA.gridX - nodeB.gridX);
    int dstY = Mathf.Abs(nodeA.gridY - nodeB.gridY);

    if (dstX > dstY)
        return 14 * dstY + 10 * (dstX - dstY);
    return 14 * dstX + 10 * (dstY - dstX);
}
```

Рисунок 3.3 RetracePath()

Джерело: розроблено автором

3.2 Конструювання

Діаграма компонентів проєкту зображено на рис. 3.4.

Зображені об'єкти та їх компоненти, що відповідають за:

- відображення здоров'я гравця (Healthbar);

- відстеження здоров’я гравця (Health);
- система відродження, смерті (PlayerRespawn, Health);
- бойова система (PlayerAttack, Slash);
- система A-Star для ворога кажана (Bat_AI, Grid, aStar).

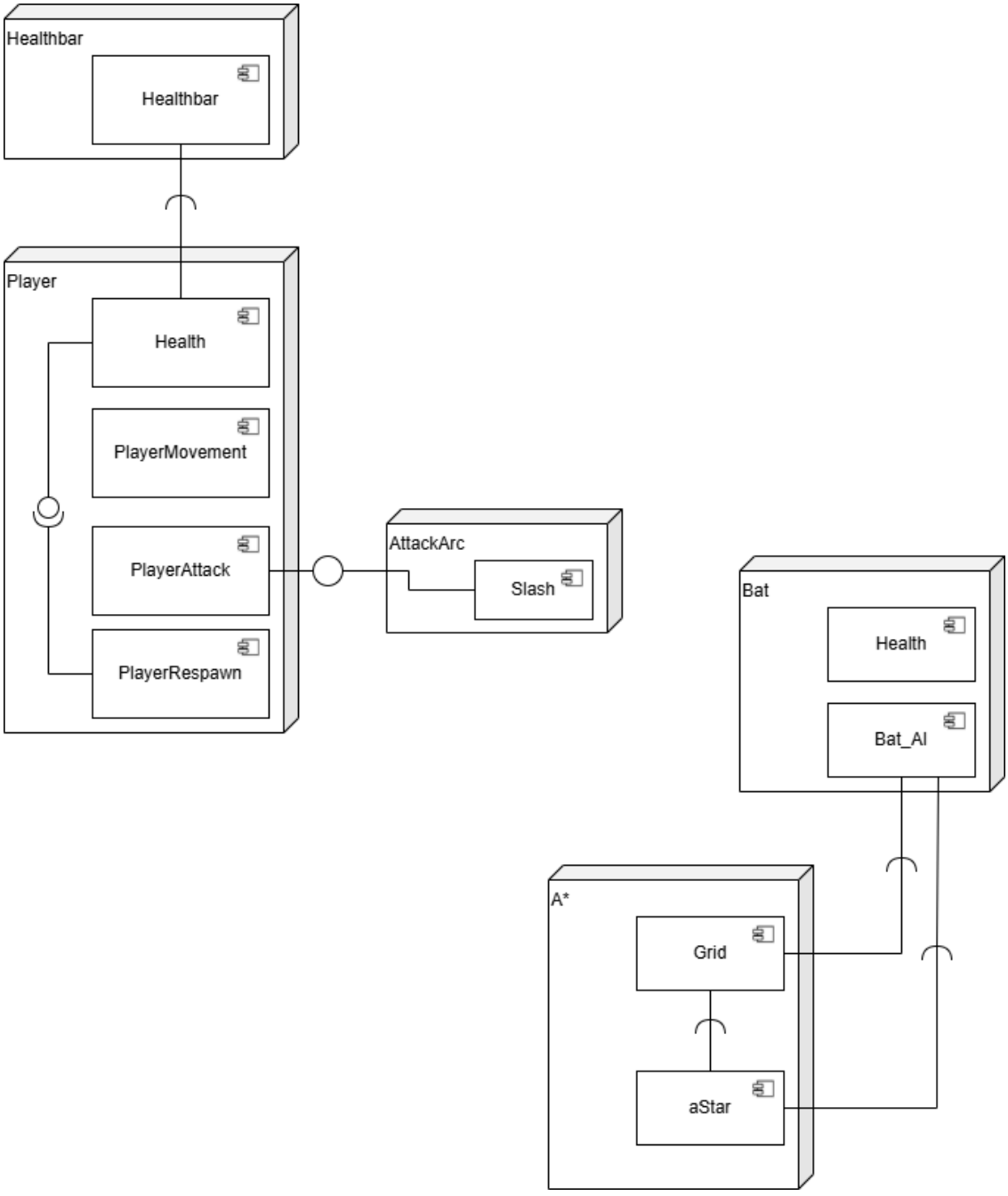

Рисунок 3.4 Component diagram

Джерело: розроблено автором

3.3 Тестування

Були проведені тестування, за яких перевірялося коректне функціонування застосунку, зокрема бойова система, система пересування, здоров'я та його відображення.

Також була протестована система навігації ворогів. На рис. 3.5 за допомогою методу OnDrawGizmos, що відповідає за генерацію візуальних позначок, було намальовано сітку рівня, де:

- клітини вільні для пересування позначаються білим;
- клітини недоступні для пересування позначаються червоним;
- клітини які входять до визначеного шляху неігрового персонажа позначаються чорним.

```
void OnDrawGizmos()
{
    Gizmos.DrawWireCube(transform.position, new Vector3(gridWorldSize.x, gridWorldSize.y, 1));


    if (grid != null)
    {
        //Node playerNode = NodeFromWorldPoint(player.position);
        foreach (Node n in grid)
        {
            Gizmos.color = (n.walkable) ? Color.white : Color.red;
            if (path != null && path.Contains(n))
                Gizmos.color = Color.black;
            Gizmos.DrawCube(n.worldPosition, Vector3.one * (nodeDiameter - .1f));
        }
    }
}
```

Рисунок 3.5 фрагмент коду візуалізації сітки та шляху

Джерело: розроблено автором

Як результат отримуємо сітку як на рис 3.6, яка динамічно оновлюється зі шляхом неігрового персонажа в залежності від позиції гравця. Таким чином переконуємося що побудований алгоритм визначає шлях з урахуванням перешкод та намагається уникнути їх.

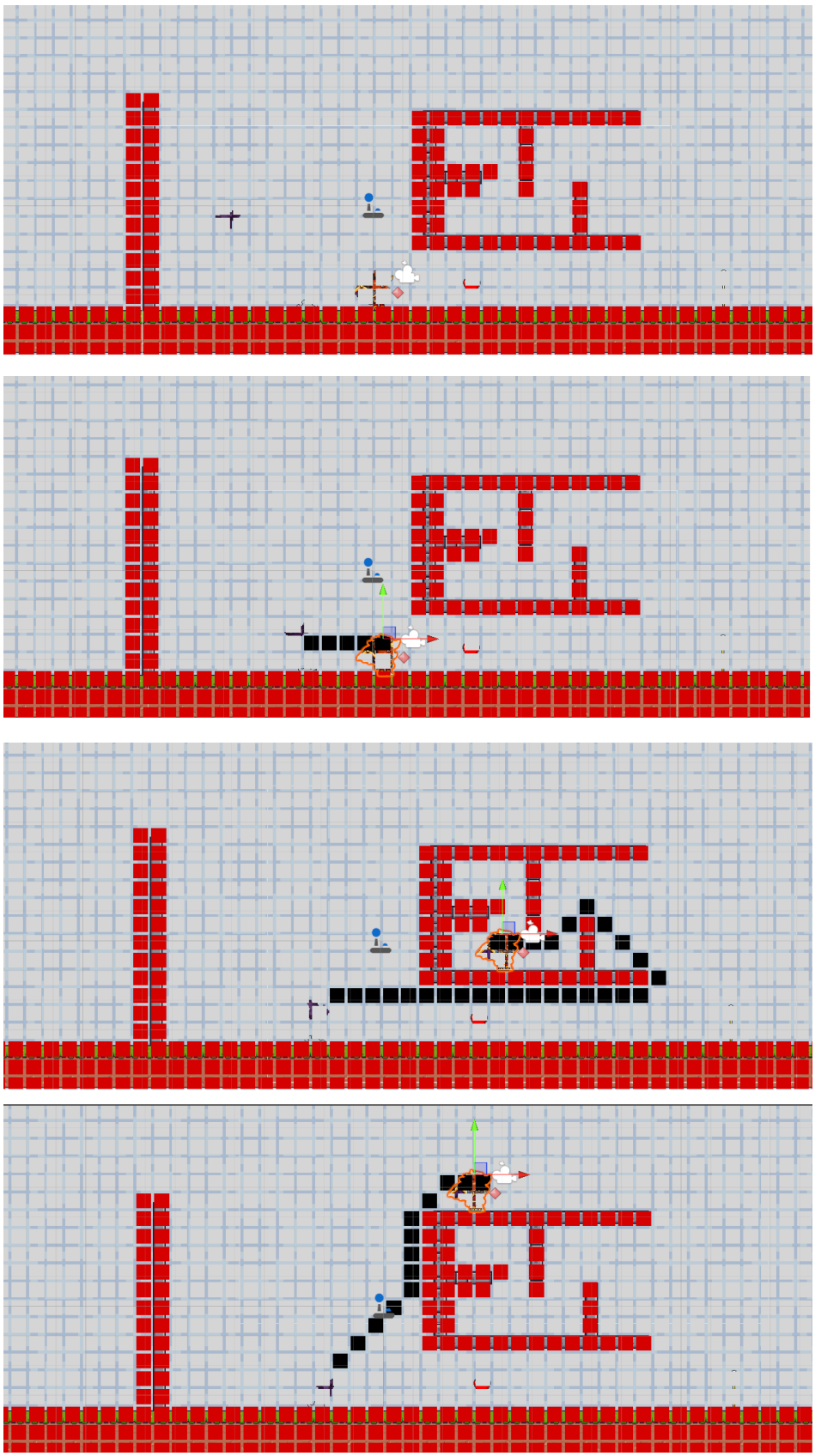

Рисунок 3.6 Демонстрація та тест функціонування A-star у ПЗ

Джерело: розроблено автором

3.4 Використання

На рис. 3.7 зображений інтерфейс та навколишнє середовище при початку гри.

Рисунок 3.7 Загальний вигляд гри

Джерело: розроблено автором

На рис. 3.8 зображено функціонування системи здоров’я та хітбоксів, де гравець отримує пошкодження від елементу навколишнього середовища. Навігація персонажа по горизонталі відбувається за допомогою стрілок на клавіатурі. Стрибок на клавішу Z, атака на клавішу, вибір цих клавіш був зроблений з урахуванням схеми управління класичних платформерів, сучасними прикладами яких є ігри Celeste та Hollow Knight.

На рисунку також можна побачити систему кадрів невразливості, під час дії яких гравець не може отримати повторні пошкодження. Окрім того що це унеможливлює нанесення пошкоджень гравцю протягом кожного кадру, це також є важливою системою для підвищення комфорту при грі.

Рисунок 3.8 Демонстрація функціонування хітбоксів та здоров'я

Джерело: розроблено автором

На рис. 3.9 зображено функціонування системи поповнення здоров'я.

На рис. 3.10 зображено активацію чекпойнту, точки з якої відбудеться відродження гравця у випадку досягнення рівня здоров'я до нуля.

Рисунок 3.10 Система чекпойнтів

Джерело: розроблено автором

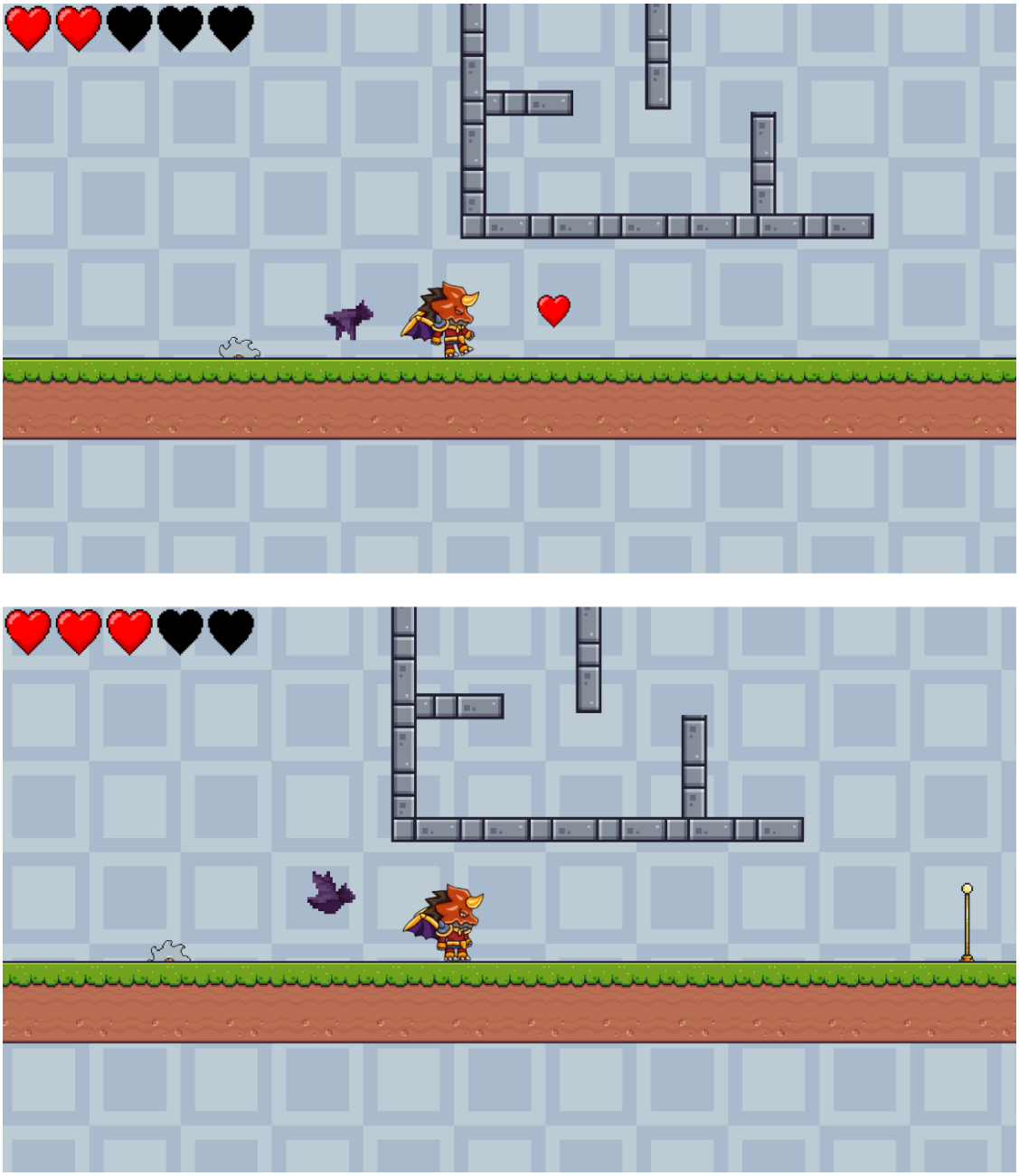

Рисунок 3.9 Система поповнення здоров’я

Джерело: розроблено автором

На рис. 3.11 продемонстровано функціонування системи чекпойнів та хітбоксу ворога. Після анімації смерті головного героя він відроджується на позиції останнього активованого чекпойнту з повним здоров’ям.

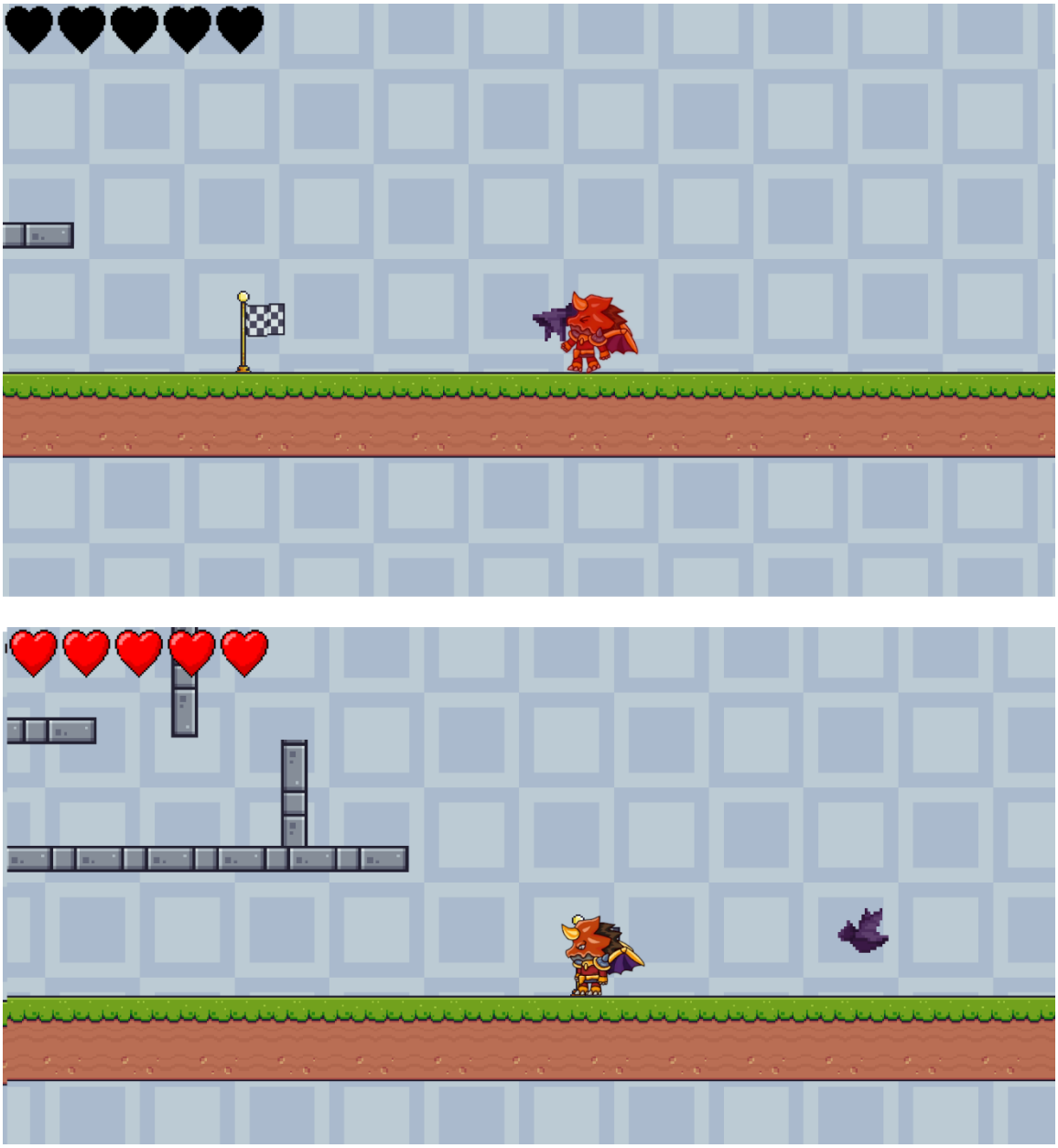

Рисунок 3.11 Відродження головного героя

Джерело: розроблено автором

На рис. 3.12 продемонстровано результат роботи алгоритму A-Star. Неігровий персонаж успішно визначає шлях від своєї позиції до позиції гравця з урахуванням перешкод.

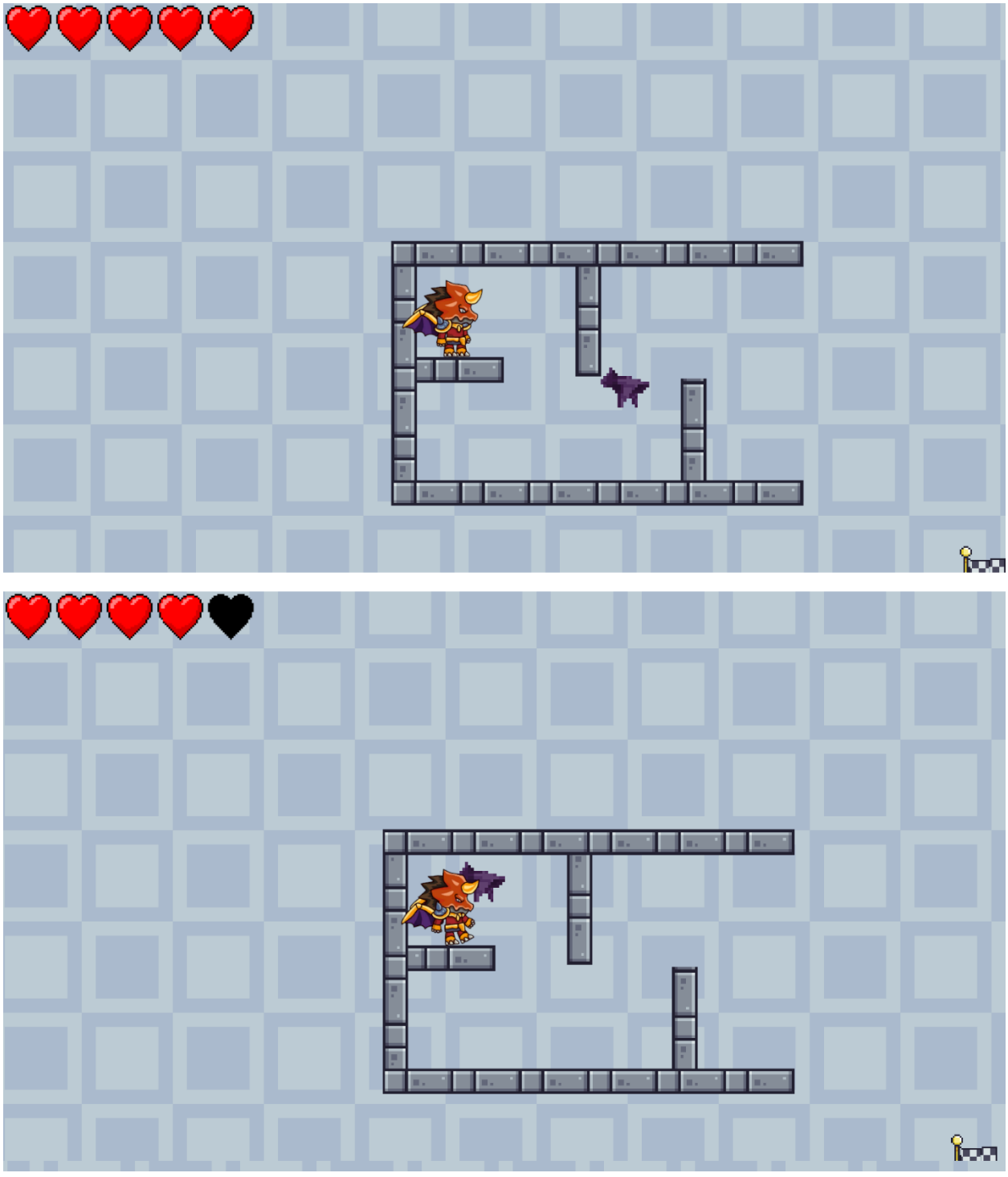

Рисунок 3.12 Кажан з алгоритмом пошуку шляху A-Star

Джерело: розроблено автором

На рис. 3.13 зображено бойову систему. Можливість гравця атакувати ворога, реєстрація попадання, відстеження здоров’я та анімація смерті.

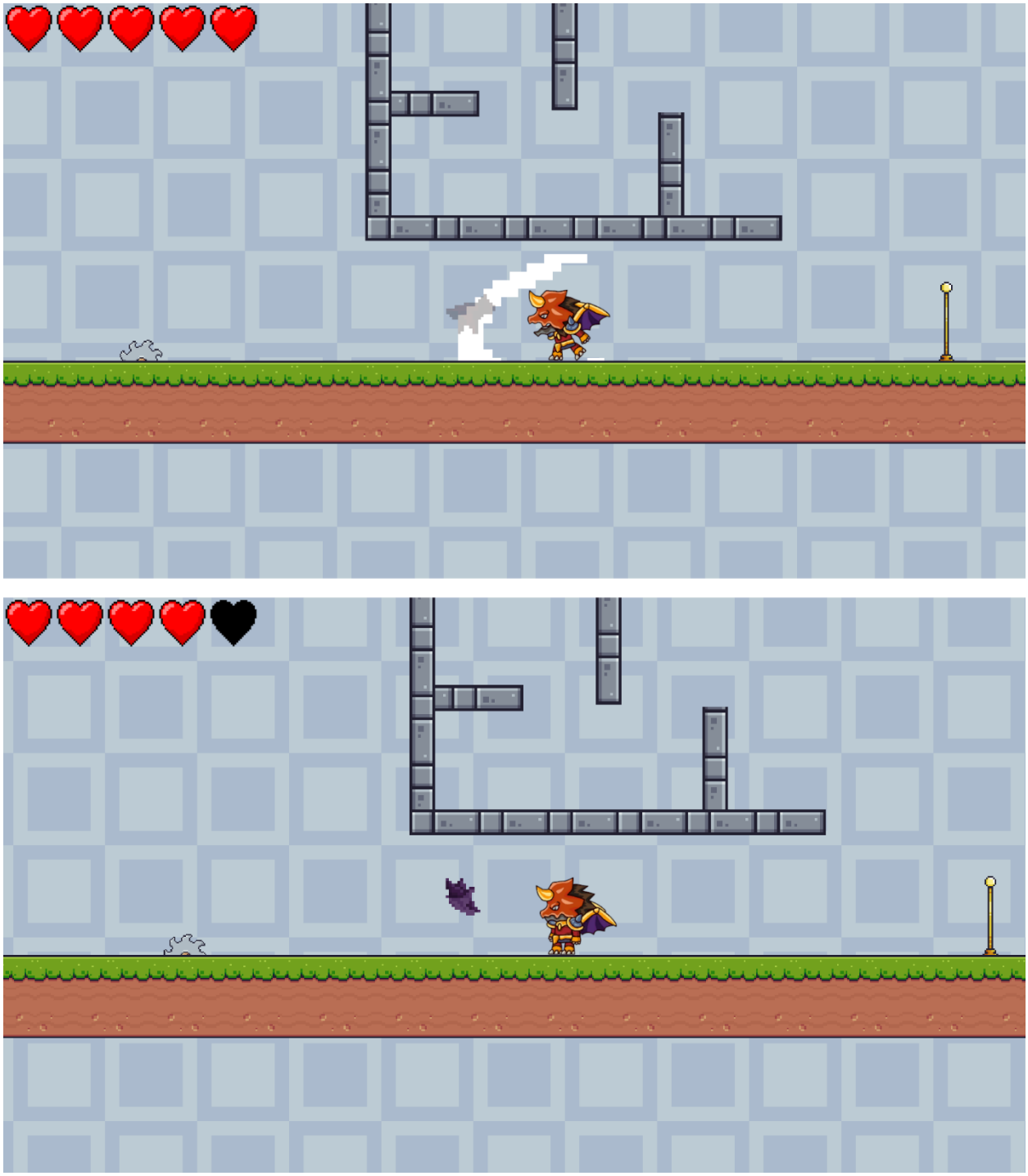

Рисунок 3.13 Бойова система та анімація смерті ворога

Джерело: розроблено автором

Висновки до розділу 3

Розглянуто та обґрунтовано причини обрання певного ПЗ для реалізації проєкту.

Визначена архітектура ПЗ та створено діаграму компонентів.

Було проведено успішне тестування реалізації алгоритму теорії графів для пошуку шляху. Реалізовано заплановані функції та продемонстрована їх робота.

ВИСНОВКИ

Виконана кваліфікаційна робота щодо розробки гри «Intra» з використанням алгоритмів теорії графів. У процесі виконання було досліджено стан ігрової індустрії, роль алгоритмів теорії графів та проаналізовано конкурентів для визначення вимог.

Було проведено аналіз декількох алгоритмів пошуку (A*, Дейкстри, Хвильовий) та обрано оптимальний для реалізації у проєкті. Спроектовано діаграми структури ПЗ.

Розроблено гру «Intra» з можливістю навігації гравця, бойовою системою, відстеженням стану здоров’я, інтерфейсом та реалізацією навігації неігрових персонажів через алгоритм A-Star.

ДОДАТОК А

ФРАГМЕНТИ ЛІСТИНГУ

PlayerAttack.cs

```
public class PlayerAttack : MonoBehaviour
{
    [SerializeField] private float attackCooldown;
    [SerializeField] private Transform attackPoint;
    private Animator anim;
    private PlayerMovement1 PlayerMovement1;
    private float cooldownTimer = Mathf.Infinity;
    public GameObject AttackArc;

    // Start is called once before the first execution of Update
after the MonoBehaviour is created
    void Start()
    {
        anim = GetComponent<Animator>();
        PlayerMovement1 = GetComponent<PlayerMovement1>();
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKey(KeyCode.X) && cooldownTimer >
attackCooldown && PlayerMovement1.canAttack()) {
            Attack();
        }
        cooldownTimer += Time.deltaTime;

    }
    private void Attack()
    {
        GameObject slash = Instantiate(AttackArc,
attackPoint.position, attackPoint.rotation, transform);
        anim.SetTrigger("Attack");
        cooldownTimer = 0;
    }

     }
```

Slash.cs

```
public class Slash : MonoBehaviour
{
    private BoxCollider2D boxCollider;
    private float lifetime = 0.1f;
    [SerializeField] private float damage = 1;
    // Start is called once before the first execution of Update
after the MonoBehaviour is created
    void Start()
    {
        boxCollider = GetComponent<BoxCollider2D>();

    }

    // Update is called once per frame
    void Update()
    {
        Destroy(gameObject, lifetime);
    }

    public void SetDirection(float direction)
    {
        transform.localScale = new Vector3(direction,
transform.localScale.y, 1);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Enemy")
        {
            collision.GetComponent<Health>().TakeDamage(damage);
        }
    }
     }
```

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Gaming is bigger than music and movies combined // mediacat.uk URL: https://mediacat.uk/dentsu-gaming-is-bigger-than-music-and-movies-combined/
2. ANIMAL WELL // Steam Store Page URL: https://store.steampowered.com/app/813230/ANIMAL_WELL/?l=ukrainian
3. Hollow Knight // Steam Store Page URL: https://store.steampowered.com/app/367520/Hollow_Knight/
4. The Binding of Isaac Rebirth // Steam Store Page URL: https://store.steampowered.com/app/250900/The_Binding_of_Isaac_Rebirth/
5. Steam games rating in Roguelike genre // Steam Store URL: https://store.steampowered.com/tags/en/Roguelike/?flavor=contenthub_toprated
6. Researchgate «Experience with the CADAPPLETS project» // Researchgate URL: https://www.researchgate.net/figure/Lee-algorithm-maze-routing-animation-expansion-phase-Gridpoints-are-labeled-in_fig5_3052969
7. Dijkstra Algorithm // Analytics Steps URL: https://www.analyticssteps.com/blogs/dijkstras-algorithm-shortest-path-algorithm
8. A-Star Pathfinding explanation // Youtube URL: https://www.youtube.com/watch?v=-L-WgKMFuhE
9. Демонстрація алгоритму A-Star Class Meeting 08: Path Finding // uchicago.edu URL: https://klasses.cs.uchicago.edu/archive/2022/spring/20600-1/class_meeting_08.html
10. Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem. Dian Rachmawati and Lysander Gustin 2020 J. Phys.: Conf. Ser. 1566 012061. // IOPScience URL: https://iopscience.iop.org/article/10.1088/1742-6596/1566/1/012061
11. Steam games that use Unity as their game engine // Streamdb URL: https://steamdb.info/tech/Engine/Unity/

12. Main Character sprites – Dragon Warrior // Unity Asset Store URL: https://assetstore.unity.com/packages/2d/characters/dragon-warrior-free-93896
13. Environment sprites - Pixel Adventure 1 // Unity Asset Store URL: https://assetstore.unity.com/packages/2d/characters/pixel-adventure-1-155360
14. Enemy sprites - Enemy Galore 1 - Pixel Art // Unity Asset Store URL: https://assetstore.unity.com/packages/2d/characters/enemy-galore-1-pixel-art-208921