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

Бакалаврська кваліфікаційна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ НАВЧАЛЬНОЇ ГРИ MATHQUEST

Виконав: студент 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Гавриш А. І.
(прізвище та ініціали)

Керівник: к. т. н., доцент каф. КН
Арсенюк І. Р.
(прізвище та ініціали)
«04» червня 2025 р

Рецензент: к. т. н., доцент каф. АПТ
Кабачій В.В.
(прізвище та ініціали)
«05» червня 2025 р

Допущено до захисту
Зав. кафедри Яровий А. А.
«06» червня 2025 р

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д. т .н. Яровий А. А.
«05» травня 2025 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гавришу Андрію Ігорьовичу

1. Тема роботи: Програмний модуль навчальної гри MathQuest.
керівник роботи: Арсенюк Ігор Ростиславович, к. т. н., доц.
затверджені наказом вищого навчального закладу «20» березня 2025 року №97

2. Термін подання студенткою роботи 30 травня 2025р.

3. Вихідні дані до роботи:
- розмір ігрового поля: 1024 × 768 пікселів;
- кількість можливих рівнів важкості розважальної частини гри – 3;
- рівень математичної частини гри 1 – 4 клас;
- кількість ворогів у розважальній частині гри – 4;
- мова програмування – об'єктно–орієнтована;
- операційна система – Windows.

4. Зміст текстової частини (перелік питань, які потрібно розробити): вступ; обґрунтування доцільності розробки програмного модуля навчальної гри MathQuest; розробка програмного модулю навчальної гри MathQuest; програмна реалізація модуля навчальної гри MathQuest; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): загальна структурна схема ігрового процесу MathQuest; UML-діаграма основних класів навчальної частини гри MathQuest; ієрархія класів для навчальної частини гри MathQuest; UML-діаграма класів розважальної частини гри MathQuest; ієрархія класів для розважальної частини гри MathQuest; схема алгоритму події отримання будь-якої шкоди персонажу; блок-схема алгоритму обробки ушкодження головного героя; приклад роботи програми (скріншот ключової сцени).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–3	Арсенюк І. Р., к. т. н., доц. каф. КН		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вступ. Обґрунтування доцільності розробки програмного модуля навчальної гри MathQuest	05.05.25	07.05.25	
2	Проектування програмного модуля навчальної гри MathQuest	08.05.25	11.05.25	
3	Програмна реалізація програмного модуля навчальної гри MathQuest	12.05.25	15.05.25	
4	Висновки та аналіз отриманих результатів	16.05.25	26.05.25.	
5	Оформлення додатків	27.05.25	23.05.25	
6	Розробка інструкції користувача для програми MathQuest	30.05.25	01.06.25	
7	Підготовка матеріалів до захисту бакалаврської кваліфікаційної роботи	02.06.25	04.06.25	

Студента _____ **Гавриша А.І.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Арсенюк І.Р.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 92 сторінок формату А4, на яких розміщено 28 ілюстрацій та одну таблицю; список використаних джерел налічує 31 найменування.

У роботі обґрунтовано потребу в інтерактивних інструментах для формування математичних навичок на основі аналізу існуючих освітніх проєктів та доведено доцільність розширення функціоналу програмного модуля навчальної гри MathQuest. Розроблено алгоритм роботи модуля та UML-діаграми класів для навчального та розважального режимів гри, що спрощують структурування коду та полегшують його розуміння. Програмний модуль реалізовано за допомогою кросплатформного 2D-рушія з використанням мови програмування C# та Unity. Здійснене тестування підтверджує коректність роботи програмного модуля та відповідність вимогам навчальної гри.

Ключові слова: навчальна гра, MathQuest, Unity, 2D TopDown, C#.

ABSTRACT

The bachelor's thesis consists of 92 A4 pages, which include 28 illustrations and one table; the list of references includes 31 titles.

The work substantiates the need for interactive tools for the development of mathematical skills based on an analysis of existing educational projects and proves the feasibility of expanding the functionality of the MathQuest educational game software module. The module's algorithm and UML class diagrams for the game's educational and entertainment modes have been developed, which simplify code structuring and facilitate its understanding. The software module is implemented with a cross-platform 2D engine using the C# and Unity programming languages. The conducted testing confirms the correctness of the software module and its compliance with the requirements of the educational game.

Keywords: educational game, MathQuest, Unity, 2D TopDown, C#.

ЗМІСТ

ВСТУП

Актуальність досліджень. У сучасному цифровому освітньому середовищі інтерактивні навчальні ігри відіграють ключову роль у підвищенні зацікавленості учнів і якості їхнього навчання, адже вони об’єднують розважальні елементи з навчальним контентом і дозволяють адаптувати складність завдань до індивідуальних потреб кожного користувача. З розвитком дистанційного та змішаного форматів навчання виникає нагальна потреба в рішеннях, які забезпечують не тільки візуальну привабливість, а й глибоку педагогічну цінність, а також здатні швидко реагувати на прогрес учня. MathQuest, створений на базі сучасного ігрового рушія з використанням об’єктно-орієнтованого підходу, прагне задовольнити ці вимоги, пропонуючи динамічне генерування вправ різного типу – від простих обчислень до складних логічних задач. Завдяки вбудованим алгоритмам аналізу відповіді система миттєво відображає результат виконання завдання. Крім того, MathQuest підтримує кілька рівнів складності й дає можливість користувачеві самостійно обирати потрібний рівень, що робить його придатним як для початківців, які щойно знайомляться з базовими арифметичними операціями, так і для більш досвідчених учнів, які прагнуть відточити навички алгебри чи розв’язання задач із логіки. Об’єктно-орієнтована структура забезпечує простоту розширення новими видами завдань і швидку інтеграцію в різні освітні платформи. Такий комплексний підхід до поєднання гри та науки робить MathQuest потужним інструментом для формування міцних математичних компетенцій і сприяє впровадженню інноваційних методів навчання в сучасних школах та онлайн-курсах.

Метою дослідження є розширення функціональних можливостей програмного модуля навчальної гри MathQuest шляхом впровадження у розважальну частину гри математичних квестів рівня 1 – 4 класів.

Об’єкт дослідження – процес взаємодії користувача з навчальним ігровим середовищем, спрямований на розвиток математичних навичок учня.

Предмет дослідження – програмний модуль MathQuest, що передбачає математичні завдання рівня 1 – 4 класів.

Задачі дослідження:

1. Здійснити аналіз наявних ігор та проаналізувати їх переваги та недоліки;
2. Спроєктувати загальну структуру програмного модуля та UML-діаграми класів для навчального і розважального режимів;
3. Розробити загальний алгоритм роботи програмного модуля;
4. Реалізувати програмний модуль гри MathQuest та здійснити його тестування;
5. Підготувати інструкцію користувача.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ НАВЧАЛЬНОЇ ГРИ MATHQUEST

1.1 Огляд індустрії навчальних відеоігор

Протягом останніх двох десятиліть індустрія навчальних відеоігор пройшла шлях від примітивних текстових вправ і вправ для запам’ятовування до масштабних інтерактивних середовищ з адаптивним навчанням і елементами штучного інтелекту. Перші комп’ютерні тренажери 1980-х років, зосереджені на вивченні мов та арифметики, поступилися місцем мультимедійним енциклопедіям і віртуальним лабораторіям у 1990-х, а вже в межах останнього десятиліття виникли симулятори складних процесів і кросплатформні курси з ігровими механіками. Сьогодні рішення для поєднання навчання й розваг охоплюють мобільні додатки, великі симулятори реальних систем і повноцінні освітні світи, що значно підвищує мотивацію та залученість учнів.

Навчальні ігри відрізняються від традиційних методів тим, що пропонують безпосередню взаємодію з матеріалом у цифровому середовищі. Замість письмових тестів і лекцій учень виконує завдання безпосередньо в ігровому світі, отримуючи негайний зворотний зв’язок і систему винагород. Завдяки цьому формується досвід практичного застосування знань, будь то друк тексту для виконання дій у пригодницькій грі чи побудова інженерних конструкцій із врахуванням фізичних законів.

Прикладом поєднання навчання та пригод є Epistory: Typing Chronicles [1], де всі дії героя – від стрибка до атаки – виконуються через набір тексту. Гравець мандрує фантастичним світом, змагається з ворогами й розв’язує головоломки, вводячи слова, які відповідають за ту чи іншу дію. Така механіка одночасно розвиває швидкість друку та розширює словниковий запас.

На рисунку 1.1 наведено ігровий процес Epistory: Typing Chronicles.

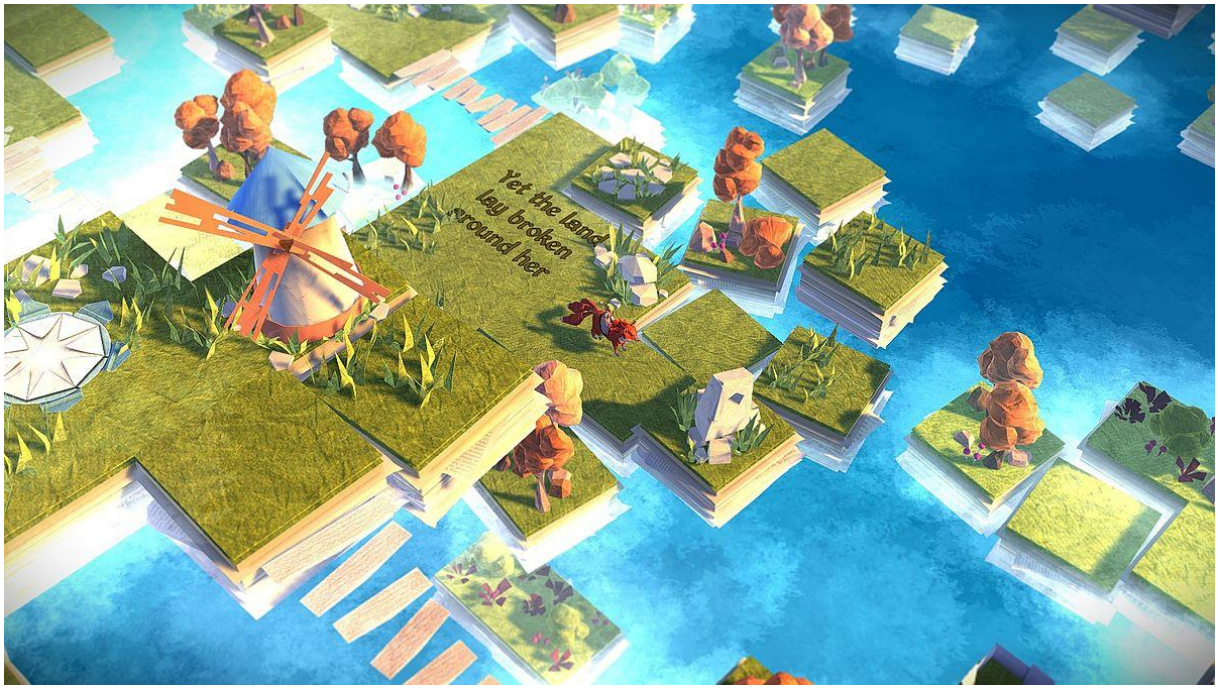

Рисунок 1.1 – Ігровий процес Epistory: Typing Chronicles

Симулятори природи, як-от Planet Zoo, пропонують учням зануритися у справжню біологію та екологію через управління повноцінним парком дикої природи [2]. Гравець проєктує кожен вольєр, обирає тип ґрунту, рослинність і температуру, щоб задовольнити фізіологічні потреби різних видів тварин – від гігантських слонів до крихітних панд. Для підтримки здоров’я мешканців необхідно своєчасно поповнювати запаси корму, очищувати воду та стежити за соціальною структурою груп: деякі види вимагають наявності партнерів або певної кількості території для розмноження.

Фінансова частина гри моделює бюджетні обмеження, змушуючи учня розраховувати витрати на будівництво, обслуговування та зарплатню персоналу (доглядачів, ветеринарів, інструкторів). Окрім цього, гравці збирають наукові дані про поведінку тварин під час спостережень і можуть використовувати їх для поліпшення умов утримання або розробки просвітницьких програм для відвідувачів. Таким чином, Planet Zoo поєднує у собі елементи екології, біології, економіки та соціальної педагогіки,

створюючи інтегроване середовище для проєктних досліджень та моделювання.

На рисунку 1.2 наведено ігровий процес Planet Zoo.

Рисунок 1.2 – Ігровий процес Planet Zoo

У Opus Magnum гравець стає алхіміком-інженером, який на шестикутній дошці розміщує маніпулятори, труби та каталізатори для трансформації вихідних атомів у складні молекули [3]. Кожен елемент машини виконує свою функцію за чітко спланованою траєкторією, що імітує програмування візуальними блоками. Основна мета – мінімізувати кількість тактів, собівартість пристроїв і зайняту площу, що розвиває навички оптимізації та системного мислення. За допомогою спеціальних інструментів гравці реалізують умовні маршрути та циклічні процеси, забезпечуючи паралельне виконання реакцій та контроль за потоком молекул. Художній стиль у дусі ретро-вікторіанської алхімії і зрозумілий інтерфейс дозволяють легко

зануритися в гру, а водночас вчать базових понять хімії й інженерії через практичний досвід.

На рисунку 1.3 наведено ігровий процес Opus Magnum.

Рисунок 1.3 – Ігровий процес Opus Magnum

Інженерний симулятор Poly Bridge відтворює основні принципи мостобудування, дозволяючи учням експериментувати з різними матеріалами та конструктивними схемами в безпечному цифровому середовищі [4]. Гравець починає з простої задачі – з’єднати два береги ріки за допомогою обмеженого бюджету й набору доступних елементів: дерев’яних балок, сталевих балок, тросів і гідравлічних важелів. Інтерфейс відображає реальний розподіл напруги на кожному елементі конструкції: червоні ділянки сигналізують про перевищення допустимого навантаження, а сині – про нестачу жорсткості. Після розміщення компонентів користувач запускає симуляцію, у якій віртуальні автомобілі різної ваги проїжджають по мосту, і гравець одразу бачить, які ділянки потребують посилення чи переналаштування.

Кожен рівень має зростаючу складність: збільшуються прольоти, додаються рухомі частини й змінюється тип навантаження (наприклад,

вантажівки чи локомотиви). При цьому учень вчиться враховувати статику – баланс сил і моментів – та динаміку, адже під час проїзду рухомих об’єктів міст піддається вібраціям і деформаціям. Завдяки ітеративному процесу «побудова – тест – корекція» закріплюються знання з механіки матеріалів, принципів розподілу навантаження та економного використання ресурсів.

На рисунку 1.4 наведено ігровий процес Poly Bridge.

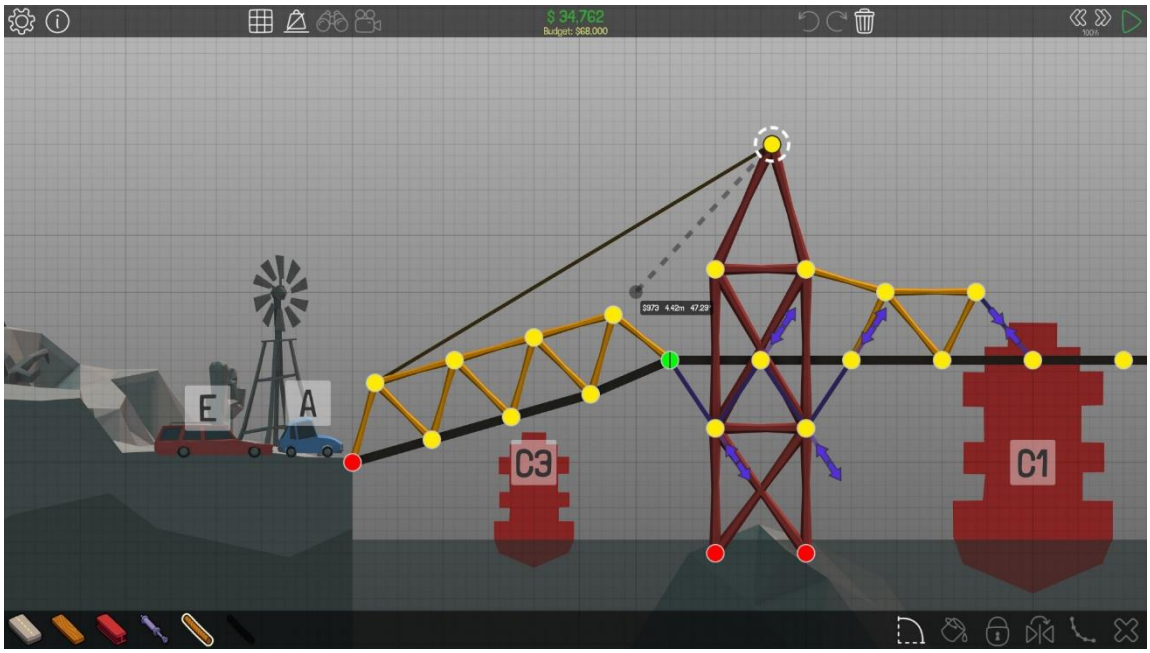

Рисунок 1.4 – Ігровий процес Poly Bridge

Особливе місце серед освітніх платформ посідає Minecraft Education Edition, адже воно поєднує відкритий тривимірний світ із потужними засобами збору даних та аналізу результатів, надаючи вчителю та учням інструменти для оперативного контролю прогресу [5]. У кабінеті геометрії учні виставляють кубічні блоки у формі різноманітних фігур – від простих кубів і прямокутних призм до правильної тетраедричної піраміди або правильного восьмигранника. Після завершення побудови достатньо активувати вбудований лічильник, і система автоматично обчислює довжини ребер, площі граней та об’єм фігури, відображаючи результати на екрані та занотовуючи їх у журналі уроку. Це дозволяє вчителю негайно перевірити правильність

розв’язку, порівняти роботи учнів у режимі реального часу та надати індивідуальні рекомендації з корекції помилок.

На рисунку 1.5 наведено приклад розрахунку об’єму кубічної конструкції в Minecraft Education Edition.

Рисунок 1.5 – Геометричне моделювання об’єму в Minecraft Education Edition

Хімічний модуль містить «Element Constructor» та «Compound Creator», де з кубічних атомів збирають молекули й спостерігають їх реакції, отримуючи інтерактивні графіки енергії реакцій і складу продуктів. Завдяки вбудованому «Code Builder» з підтримкою MakeCode і JavaScript учні програмують віртуального агента, який може будувати, копіювати структури та досліджувати світ за заданим алгоритмом. Універсальність платформи доповнюється можливістю спільної роботи над проєктами – це розвиває навички командної взаємодії, планування та управління ресурсами.

Отже, у перелічених іграх є такі недоліки: по-перше, вони важкі й часто не запускаються на старих або мобільних пристроях через високі вимоги до

«заліза»; по-друге, готові ігрові механіки не завжди підходять для математики й потребують додаткового налаштування; по-третє, більшість ігор не враховують рівень знань учнів і не налаштовують завдання під кожного, тому мотивація падає; MathQuest вирішує ці проблеми так: він працює на легкому 2D-движку, тож запускається навіть на слабких пристроях; алгоритми підлаштовують складність задач у реальному часі під рівень учня; модульна структура дозволяє швидко додавати нові теми (дроби, алгебра, логіка) без складних налаштувань; це робить MathQuest простим, доступним та ефективним для вивчення математики.

1.2 Аналіз методів гейміфікації та мотивації в навчальних іграх

Система нарахування балів і рівнів у навчальних іграх ґрунтується на принципі безперервного зворотного зв’язку: учень одразу бачить результат своїх дій і розуміє, скільки балів досвіду ще потрібно накопичити для переходу на наступний рівень. Кожне завдання супроводжується чітким числовим показником досвіду, який додається до профілю гравця. Важливо продумати криву накопичення: на початку рівні даються відносно легко (щоб учень швидко відчув успіх), а потім поступово зростають вимоги, стимулюючи його вдосконалювати навички.

Порогові значення балів досвіду для переходу між рівнями можуть визначатися за лінійною або експоненціальною моделлю зростання. У лінійній моделі кожен наступний рівень вимагає однакової кількості балів досвіду, що підходить для коротких серій вправ. Експоненціальна модель, коли обсяг необхідного досвіду зростає з кожним рівнем, краще мотивує до тривалого навчання, але потребує ретельного балансування, щоб уникнути відчуття «застою».

З набуттям нового рівня часто відкриваються додаткові можливості: доступ до просунутого контенту, нових ігрових режимів чи налаштувань персонажа. Такий підхід підтримує інтерес учня до довготривалої взаємодії та

створює відчуття постійного прогресу, що за даними досліджень є критично важливим для збереження мотивації в освітніх проєктах.

Бейджі та досягнення. Візуальні відзнаки (бейджі) слугують маркерами ключових успіхів або завершених тематичних блоків. Отримані бейджі зберігаються в профілі та можуть відображатися в загальнокласному рейтингу чи портфоліо. Ефект колекціонування стимулює учнів освоювати нові навички, адже кожен бейдж символізує подолану складність. Наприклад, за 15 правильно розв'язаних задач із геометрії учневі присвоюється бейдж «Геометричний гуру».

На рисунку 1.6 наведено приклад системи рівнів і накопичення досвіду.

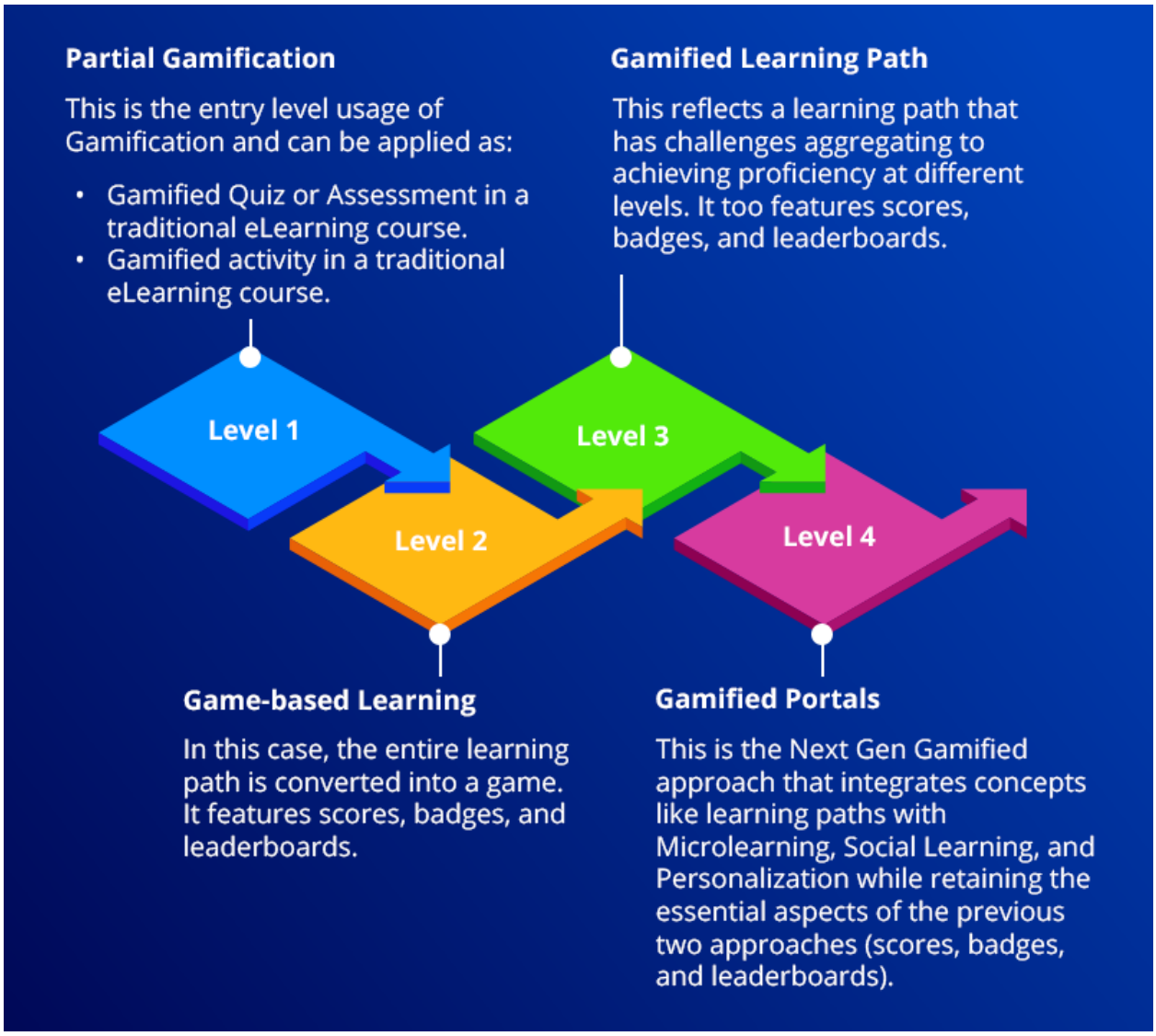

Рисунок 1.6 – Схема системи рівнів і накопичення досвіду

Бейджі та досягнення. Візуальні відзнаки (бейджі) служать маркерами ключових успіхів або завершення тематичних блоків. Отримані учнем бейджі зберігаються в його профілі та можуть відображатися в загальнокласному рейтингу або портфоліо. Психологічний ефект колекціонування стимулює працювати над опануванням нових навичок, адже кожний бейдж символізує подолану складність. Наприклад, за 15 правильно розв’язаних задач із геометрії учневі присвоюється бейдж «Геометричний гуру».

На рисунку 1.7 наведено приклад бейджа «Геометричний гуру».

Рисунок 1.7 – Бейдж «Геометричний гуру» за 15 правильно розв’язаних задач із геометрії

Виклики та квести. Запровадження структурованих завдань «квестів» дозволяє поєднати навчальні цілі із захопливим сюжетом. Квест складається з послідовності завдань, об’єднаних єдиною тематикою або історією. Завершення кожного кроку квесту відкриває наступний, що гарантує відчуття напряму й цілісності. Наприклад, щоб «рятувати замок», необхідно послідовно вирішити чотири математичні вправи: спочатку на арифметику, потім на логіку, далі на геометрію й у фіналі комплексну задачу.

На рисунку 1.8 подано структуру стандартного квесту в навчальних іграх.

Рисунок 1.8 – Приклад структури квесту з чотирьох етапів

Соціальна взаємодія та змагальні елементи. Лідерборди (таблиці лідерів) та рейтингові таблиці стимулюють учнів змагатися між собою або в командах. Змагання може бути індивідуальним за найвищий бал у вправі або груповим за сумарний прогрес команди. Додавання соціального аспекту активізує в учнів прагнення показати свої вміння, а також спонукає до співпраці: учасники можуть ділитися порадами, допомагати слабшим і разом долати складні завдання.

На рисунку 1.9 продемонстровано приклад лідерборда з відображенням балів учнів.

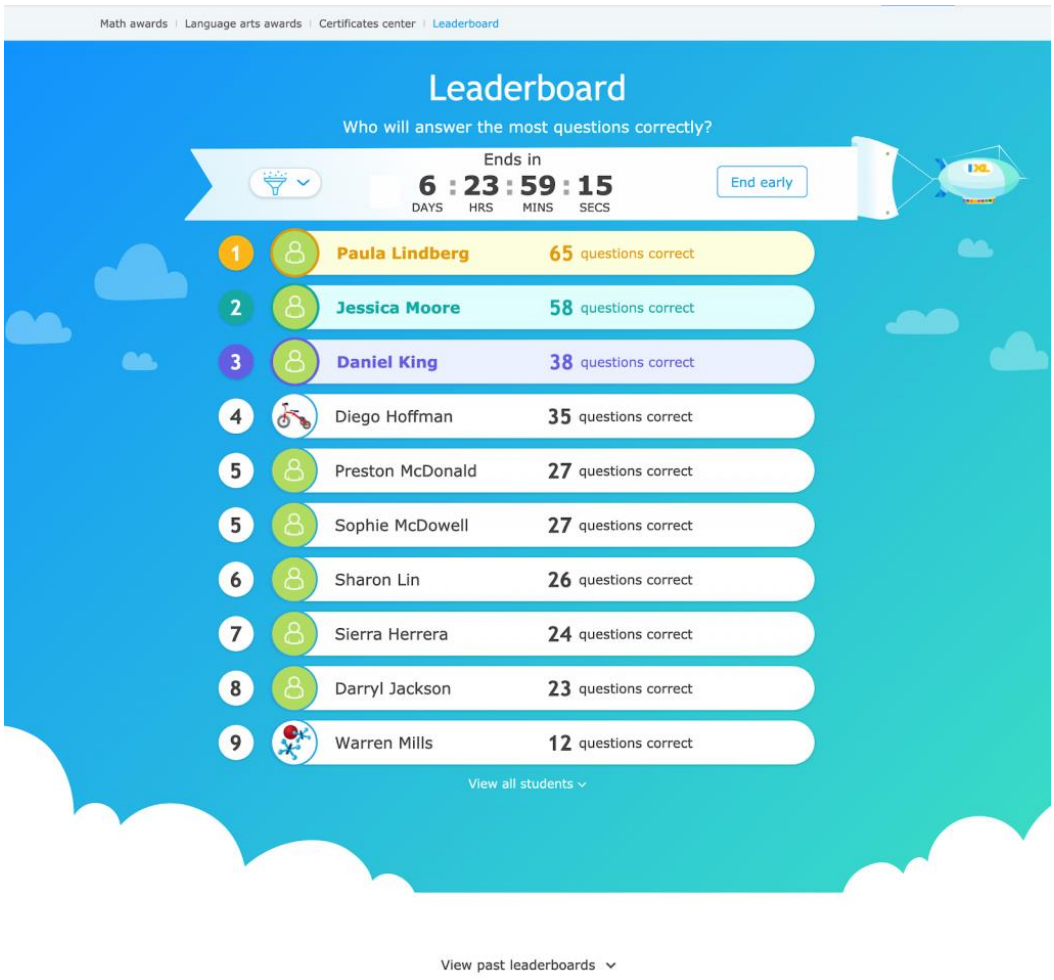

Рисунок 1.9 – Зразок рейтингової таблиці для класу

Негайний зворотний зв’язок. Однією з ключових переваг інтерактивних ігор над традиційними вправами є миттєве підтвердження або корекція дії

учня. Після введення відповіді гра негайно показує правильність рішення, підсвічує помилки та пояснює причини невдачі. Подібний підхід запобігає укоріненню неправильних алгоритмів мислення та сприяє швидшому засвоєнню матеріалу.

На рисунку 1.10 ілюстровано екран з миттєвим зворотним зв’язком після виконання вправи.

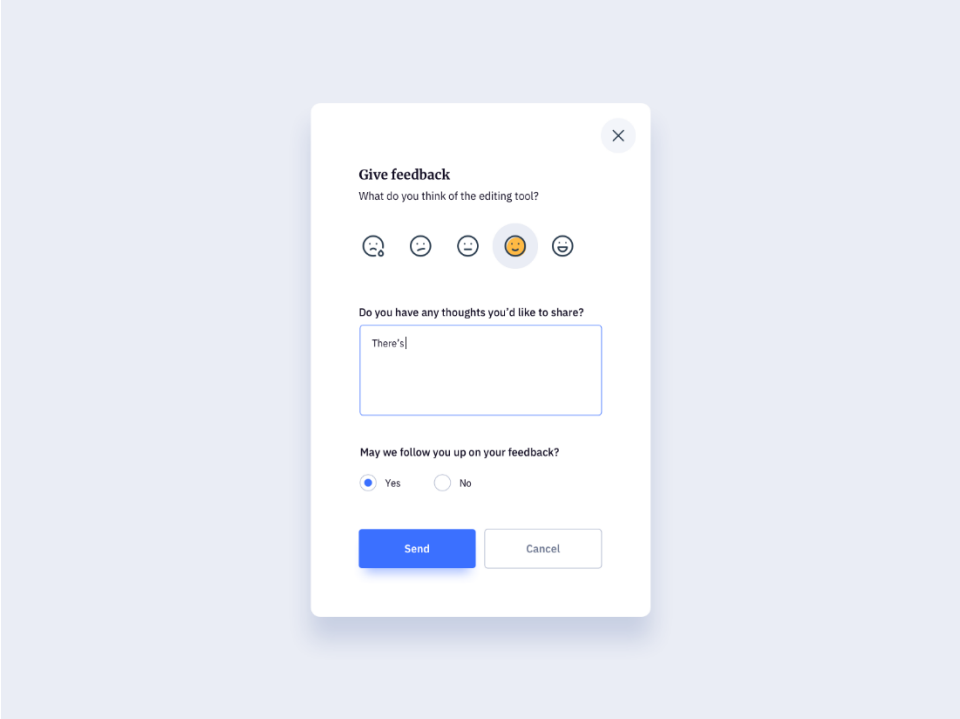

Рисунок 1.10 – Приклад інтерфейсу негайного зворотного зв’язку

Використання сюжетних елементів і наративу в навчальних іграх підсилює емоційну залученість учнів: замість сухих вправ вони стають учасниками історії. Наприклад, завдання можна побудувати навколо героя, якому потрібно відшукати втрачені символи формул – кожне правильно розв’язане рівняння відкриває нову ділянку карти або анімацію відновлення рукопису.

Сюжет поділяє великий обсяг матеріалу на короткі «епізоди»: спочатку учень рятує віртуальне місто від обвалу мосту, застосувавши знання про площі та об’єми, а потім допомагає алхіміку збирати інгредієнти за формулами.

Персонажі з унікальними рисами (мудрий наставник, веселий помічник) дають контекст підказкам, що підсилює внутрішню мотивацію.

Навіть мінімальні елементи наративу – короткі діалоги або прості анімації успіху – формують асоціації та сприяють глибшому засвоєнню матеріалу.

На рисунку 1.11 наведено концептуальну карту сюжету та персонажів.

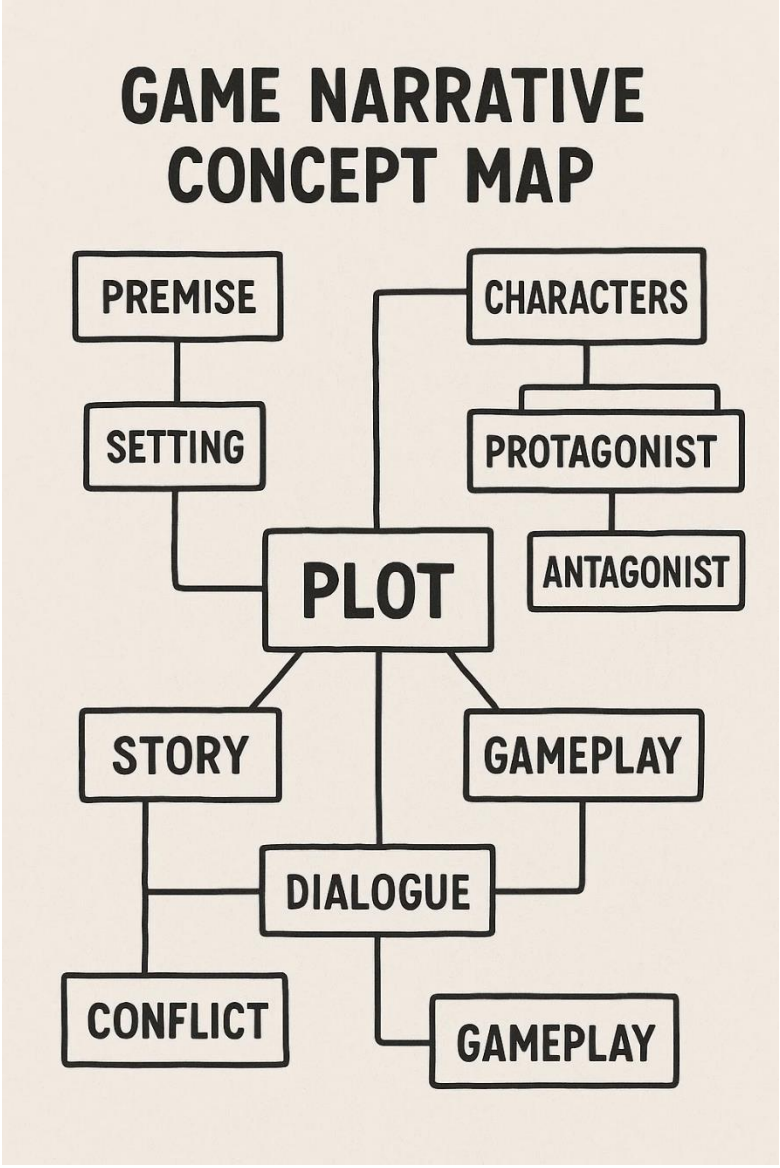

Рисунок 1.11 – Концептуальна карта сюжетних елементів

Елементи несподіванки та випадковості. Додавання рандомізованих подій або «лут-боксів» (цифрових скриньок із випадковим вмістом) може

підтримувати зацікавленість. Наприклад, після п’яти правильних відповідей система випадково видає «скриньку нагород», де учень може знайти бонусні очки, новий аватар чи невелике унікальне завдання. Такий елемент несподіванки викликає дофаміновий «сплеск» у мозку та підвищує бажання повертатися до гри.

На рисунку 1.12 подано приклад інтерфейсу скриньки нагород.

Рисунок 1.12 – Приклад «лут-боксу» з випадковими винагородами

Нагляд за прогресом і можливість самостійного вибору траєкторії навчання значно підвищують відчуття відповідальності та залученості учня. Детальна панель прогресу відображає не лише загальну кількість виконаних вправ, а й розбивку за темами – наприклад, відсоток опанованих понять з дробів, геометрії чи алгебри. Завдяки інтерактивним графікам можна відстежити динаміку успіхів у часі: учень бачить, які теми даються легше, а над якими варто попрацювати додатково.

Функція «власного шляху» дозволяє кожному учню встановлювати пріоритетність завдань відповідно до своїх індивідуальних потреб – відразу

розпочинати теми, які викликають інтерес або, навпаки, до тих, що потребують закріплення. Учень може ставити короткострокові цілі (наприклад, “завершити три вправи на додавання дробів до кінця дня”) і отримувати сповіщення про їхнє виконання. Такий підхід розвиває вміння самостійно планувати навчальний процес, усвідомлювати власний прогрес і підтримувати внутрішню мотивацію до подальшого вдосконалення.

На рисунку 1.13 показано макет персональної панелі прогресу учня.

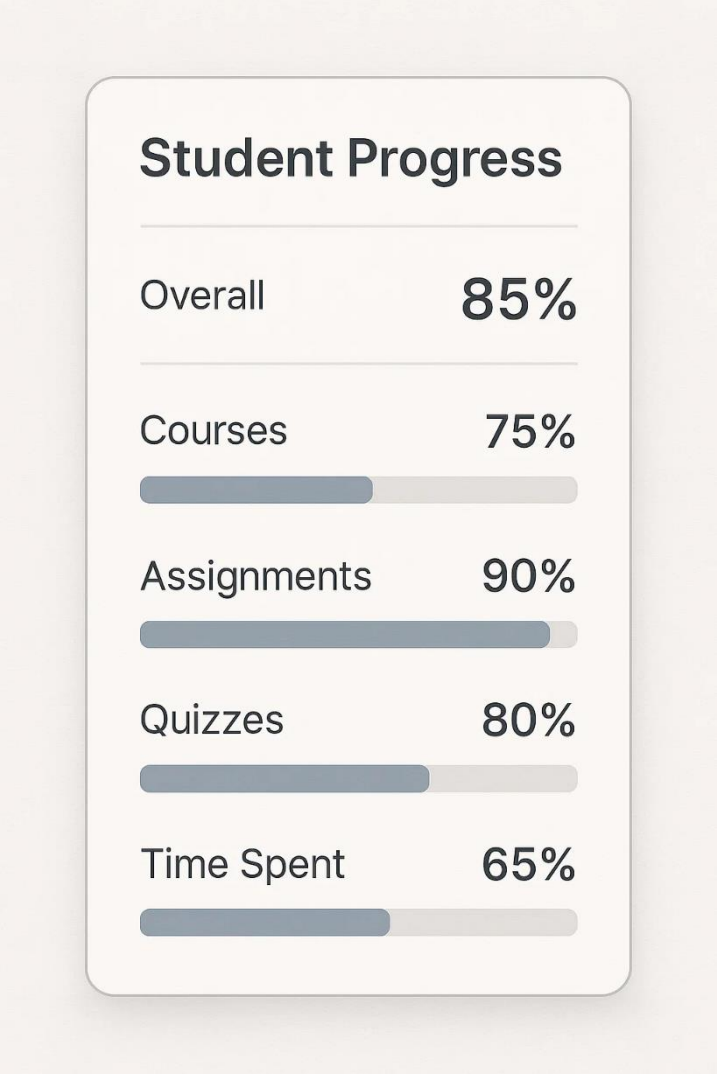

Рисунок 1.13 – Панель відстеження індивідуального прогресу

Методи гейміфікації та мотивації в навчальних іграх охоплюють широкий спектр інструментів: від системи балів та рівнів до сюжетних ліній, соціальних змагань і рандомізованих винагород. Правильне поєднання цих

механік забезпечує високу залученість учнів, стимулює самостійне навчання та розвиває як когнітивні, так і емоційні компоненти мотивації. Інтеграція гейміфікації у навчальні продукти створює динамічне та адаптивне середовище, яке відповідає запитам сучасних учнів та сприяє ефективному засвоєнню матеріалу [6].

1.3 Обґрунтування вибору цільової платформи розробки програмного модуля навчальної гри MathQuest

Коли ми плануємо випустити гру, потрібно зосередитися на тому, де й як гравці будуть у неї грати. Різні платформи диктують свої правила: на ПК користувачі розраховують на гладку графіку й чіткий контроль через клавіатуру з мишкою; на смартфонах і планшетах – на зручність торкань і миттєвий доступ без складних налаштувань; на консолях – на стабільну продуктивність та уніфіковану систему управління геймпадом. Кожна з цих платформ відкриває свої можливості, але й ставить свої виклики у розробці, оптимізації та тестуванні.

ПК-версія гри дозволяє повною мірою використовувати потужність сучасних комп’ютерів: швидкі процесори й відеокарти забезпечують плавну картинку й реалістичний рух об’єктів, а велика оперативна пам’ять дає змогу відкривати великі й детальні ігрові світи; керування за допомогою клавіатури та миші забезпечує високу точність у навігації та діях; також можна підключати додаткові модулі для роботи з мережею чи зберігання даних; платформи на кшталт Steam або Epic Games Store спрощують випуск нових версій гри й обмін результатами між гравцями. Проте різні конфігурації комп’ютерів ускладнюють забезпечення однакового рівня роботи: потрібно перевіряти гру на різних процесорах, відеокартах і об’ємах пам’яті, підбирати налаштування графіки під середні та слабкі комп’ютери і стежити, щоб гра не гальмувала.

На рисунку 1.14 наведено приклад типової ігрової станції ПК з основними компонентами.

Рисунок 1.14 – Типове апаратне забезпечення для PC-версії гри

Мобільний геймінг охоплює сотні мільйонів користувачів та дозволяє швидко залучати нову аудиторію через App Store і Google Play. Сенсорний екран відкриває унікальні механіки – доторки, свайпи, жестові рухи та навіть управління нахилом пристрою через акселерометр. Однак смартфони й планшети мають обмежену оперативну пам’ять і ресурс акумулятора, тому гру потрібно оптимізувати: зменшити розмір текстур, підняти частоту кадрів без надмірного споживання енергії та обмежити фонові процеси. Крім того, доводиться тестувати гру на різних моделях пристроїв із різними розмірами екранів і співвідношеннями сторін – це допоможе переконатися, що інтерфейс

не «розїжджається» і кнопки лишаються під рукою. Публікація в App Store чи Google Play вимагає додаткових кроків: підготовки пакетів залежно від версії ОС, заповнення метаданих і проходження перевірки політик платформи. І тільки після цього користувачі зможуть швидко завантажити й запустити гру за кілька секунд.

На рисунку 1.15 наведено приклад мобільної ігрової станції з основними компонентами.

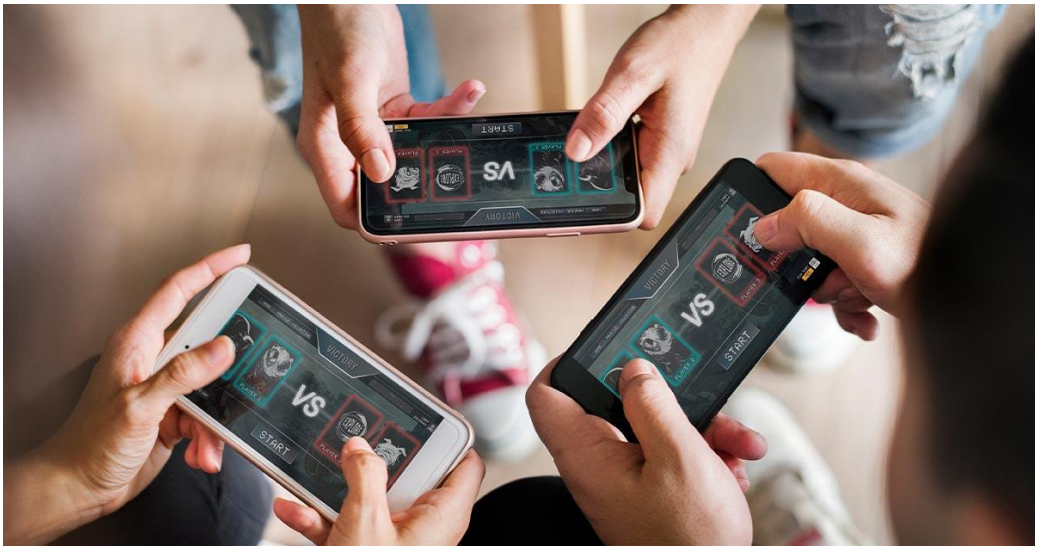

Рисунок 1.15 – Типове обладнання для мобільного геймінгу

Консолі пропонують єдине апаратне середовище для всіх користувачів, що значно полегшує роботу над продуктивністю: розробникам не потрібно підлаштовувати гру під десятки різних конфігурацій процесорів і відеокарт, а оптимізація “під одну залізку” гарантує стабільну частоту кадрів та відсутність зависань. Єдиний контролер – геймпад – забезпечує однаковий досвід для всіх гравців, а підключення до телевізора створює комфортний великий екран.

Разом з тим, розробка під консолі пов’язана з рядом адміністративних вимог: необхідно отримати ліцензію від виробника (Sony, Microsoft чи Nintendo), адаптувати гру під їхні SDK та пройти сертифікацію на відповідність технічним та правовим стандартам. Це включає перевірку пакета гри, ігрового контенту, правильної роботи оновлень і відповідність вимогам

платформи щодо захисту даних. Після виходу, оновлення поширюються виключно через офіційні канали консолі та можуть виходити з затримкою, а також обмежені за частотою випуску, що слід враховувати при плануванні випуску нових версій.

На рисунку 1.16 наведено вигляд консолей.

Рисунок 1.16 – Типові ігрові консолі

Підтримка гри одночасно на ПК, мобільних пристроях і консолях відкриває доступ до набагато більшої аудиторії та дозволяє гравцям переходити з одного пристрою на інший, зберігаючи свій прогрес і придбані досягнення. Проте це накладає додаткові зобов’язання на команду розробки:

– Управління різними пристроями вводу. На ПК гравці звикли до клавіатури й миші, на консолях – до геймпада, а на смартфонах та планшетах – до дотиків, жестів і датчиків руху. Для кожного способу взаємодії потрібно створити власний інтерфейс, налаштувати чуйність елементів керування й забезпечити інтуїтивні підказки.

– Графічні налаштування та продуктивність. Комп’ютери можуть витримати високі рівні деталізації, тіней і ефектів частинок, тоді як мобільні

пристрої потребують спрощених моделей і зменшених текстур, щоб не «гальмувати» й економити акумулятор. Консолі ж мають стабільну потужність, але все одно вимагають окремих оптимізацій під конкретні моделі.

– Розміщення в цифрових магазинах. Кожна платформа має власний магазин: Steam, Epic Games Store, App Store, Google Play, PlayStation Store, Xbox Marketplace тощо. Для кожного з них необхідно готувати окремі збірки, заповнювати метадані, проходити перевірку контенту й дотримуватися їхніх правил оновлень.

– Системи монетизації та аналітики. Якщо на ПК і консолях часто використовують одноразову покупку або сезонні абонементи, то в мобільних додатках звичні вбудовані покупки й реклама. Це означає, що вам доведеться вбудувати кілька варіантів оплати й інтегрувати різні аналітичні сервіси для відстеження поведінки користувачів.

– Підтримка й тестування. Щоб гарантувати якісну гру на всіх пристроях, слід регулярно запускати тести на максимально різних комбінаціях «заліза» та версіях операційних систем. Це значно збільшує навантаження на відділ тестування і вимагає більше часу й ресурсів на виправлення помилок.

З одного боку, мультиплатформність дає гравцям свободу вибору – можна почати на телефоні у дорозі, продовжити на ПК вдома й навіть закінчити на консолі з комфортного дивана. З іншого – вона вимагає ретельного планування, додаткових зусиль з адаптації під різні умови та збільшує витрати на підтримку й тестування.

Враховуючи технічні вимоги, потреби цільової аудиторії та бюджет розробки, ми зупинилися на ПК як основній платформі; це рішення дозволяє використати повний потенціал рушія для високої якості графіки й складних механік; спростити процес тестування завдяки стандартизованому «залізу»; забезпечити зручну систему оновлень через популярні дистрибутиви; забезпечити точне керування за допомогою клавіатури та миші; таким чином,

обравши ПК як головну платформу, ми гарантуємо максимальну продуктивність, стабільність і простоту підтримки продукту.

1.4 Постановка задачі дослідження

Необхідно реалізувати гру, яка поєднує навчальний ігровий процес у простому та доступному форматі. Математичні вправи будуть подані у вигляді коротких мінізавдань, що інтегровані в геймплей та виконуються за допомогою зручного інтерфейсу. Рішення задач з арифметики й геометрії здійснюється безпосередньо під час гри з миттєвим відображенням результатів.

Графіка матиме спокійну кольорову палітру з чіткими силуетами, щоб мінімізувати візуальне навантаження та зберігати фокус на завданнях. Плавна анімація та живе середовище створюють приємний фон без перевантаження деталей.

Ігровий процес реалізується на сучасному 2D-рушії з Top-Down перспективою. Основна логіка проєкту буде побудована з використанням об’єктно-орієнтованих принципів, що дозволить забезпечити масштабованість та зручність підтримки коду.

Рух персонажа, обробка фізики, а також система колізій налаштовуються через стандартні засоби рушія. Карта створюється за допомогою інтегрованого tilemap-редактора, а спрайти, кнопки й інші графічні елементи беруться з відкритих бібліотек та адаптуються до стилістики проєкту. Для зменшення навантаження використовується ретро-піксель-арт графіка без тривимірної геометрії.

Анімацію персонажа та візуальні ефекти реалізовано через систему кадрів (frame-based animation), що забезпечує плавні переходи між станами (рух, взаємодія, підбір предметів).

Меню, інтерфейс оцінювання результатів та відображення балів будуть створені на базі стандартних UI-компонентів рушія. Потрібно забезпечити збереження прогресу та налаштувань у локальному сховищі. Статистика учня відображатиметься в реальному часі за допомогою що спливає вікнами й HUD-елементів, що активуються при зміні стану (наприклад, правильне розв’язання задачі чи завершення рівня).

Потрібно забезпечити збереження прогресу та налаштувань у локальному сховищі, а також забезпечити відображення статистики учня в реальному часі за рахунок спливаючих вікон.

Основні завдання для реалізації базового модуля MathQuest у двомірному Top-Down форматі:

- Створити новий проєкт у рушії з шаблоном двомірної гри та налаштувати структуру папок для графічних файлів, рівнів і скриптів.
- Забезпечити переміщення головного героя та його обертання, використавши фізичні компоненти руху та стандартну систему обробки вводу від клавіатури чи сенсорного екрана.
- Додати індикатор здоров’я героя з короткою безпечністю після ураження та відобразити його на панелі інтерфейсу.
- Реалізувати збір предметів на рівні: при підборі запускатиметься математичне завдання, і герой відновлюватиме здоров’я тільки за правильної відповіді.
- Додати зони-перешкоди, які зменшуватимуть здоров’я героя при контакті з ними та застосовуватимуть період тимчасової невразливості.
- Спроєктувати базову поведінку ворогів з патрулюванням, нанесенням шкоди при зіткненні та можливістю «вирішення проблеми» через стрільбу.
- Створити шаблон кидка снаряда від героя з його запуском, рухом та видаленням після попадання або виходу за межі екрану.
- Побудувати простий інтерфейс вікторини з питанням, кількома варіантами відповіді, можливістю призупинити гру й відобразити результати.

- Перевірити коректність руху, анімацій, взаємодій, оновлення здоров’я та роботу вікторин у різних сценаріях гри.

1.5 Висновок

На основі аналізу поточного стану освітніх відеоігор було виявлено ключові тенденції еволюції жанру – від простих інтерактивних тренажерів до масштабних ігрових світів, персонажами та розгалуженими системами прогресу. Ефективність таких рішень безпосередньо залежить від впроваджених гейміфікованих механік. Зокрема, система балів, рівнів і адаптивних завдань підтримує мотивацію, забезпечує баланс між складністю й цікавістю, а також утримує гравця в навчальному процесі.

У межах дослідження було здійснено порівняльний аналіз основних платформ – персональних комп’ютерів, мобільних пристроїв та ігрових консолей. ПК забезпечує максимальну технічну гнучкість, підтримку складних обчислень і масштабовану графіку. Смартфони – це висока доступність, інтуїтивна взаємодія та швидке розгортання. Консолі гарантують єдину апаратну архітектуру, що спрощує тестування й оптимізацію. Було обрано персональний комп’ютер як основну платформу, оскільки він забезпечує максимальну гнучкість, підтримку складної графіки та зручність у розробці. На відміну від мобільних пристроїв і консолей, ПК дає більше свободи для реалізації ідеї.

Здійснено постановку задачі дослідження, де чітко сформульовано функціональні вимоги до майбутнього продукту: реалізація гри з адаптивною системою складності, підтримкою персоналізованого навчання, стабільною роботою на обраній платформі та можливістю швидкої модифікації контенту. Інтерфейс має автоматично підлаштовуватись під тип керування – клавіатура, сенсорне введення або геймпад – із додатковими підказками в процесі навчання.

2 РОЗРОБКА ПРОГРАМНОГО МОДУЛЮ НАВЧАЛЬНОЇ ГРИ MATHQUEST

2.1 Розробка загальної структурної схеми програмного модуля MathQuest

У багатьох сучасних навчальних іграх поєднання дослідження просторів із тематичними завданнями перетворює навчання на захопливу пригоду: у Minecraft Education Edition учні моделюють геометричні фігури в тривимірному світі й одразу перевіряють свої обчислення, DragonBox Algebra вводить поняття рівнянь через перестановку блоків без формул, Prodigy Math поєднує бої з фантастичними істотами й завдання на арифметику з винагородою новими здібностями, а Zoombinis тренує логіку через дедуктивні головоломки, відкриваючи фрагменти сюжету за успішні рішення. Проте більшість із них не враховує одночасно гнучку адаптацію складності, різноманіття режимів і детальний зворотний зв’язок на кожен крок учня, часто обмежуючись лише одним типом вправ або загальною візуальною привабливістю без глибокого аналізу помилок. На відміну від цього MathQuest динамічно генерує завдання різного рівня – від простих обчислень до складних логічних. [7 – 9].

У MathQuest застосовано формат 2D Top-Down: гравець досліджує карту, а при підборі спеціального об’єкта активується «зона виклику», яка відображає одне математичне запитання з трьома варіантами відповіді [10].

Правильна відповідь супроводжується миттєвими звуковими й візуальними підказками, а в разі помилки гравець має можливість повторити спробу. MathQuest поєднує навчальні завдання з елементами виживання: запас здоров’я зменшується не лише через неправильні відповіді, а й при контакті з ворогами або пастками (наприклад, шипами на карті). Індикатор HealthBar одразу відображає втрати здоров’я [11]. Щоб відновити «життя», гравець має

знайти на рівні спеціальний об’єкт – наприклад, полуницю – та відповісти на коротке запитання. У разі правильної відповіді запас здоров’я поповнюється, а також нараховуються додаткові очки.

На рисунку 2.1 показано основні етапи ігрового циклу MathQuest: від дослідження карти й активації викликів до відновлення здоров’я та легкого екшену через стрільбу снарядами по ворогах.

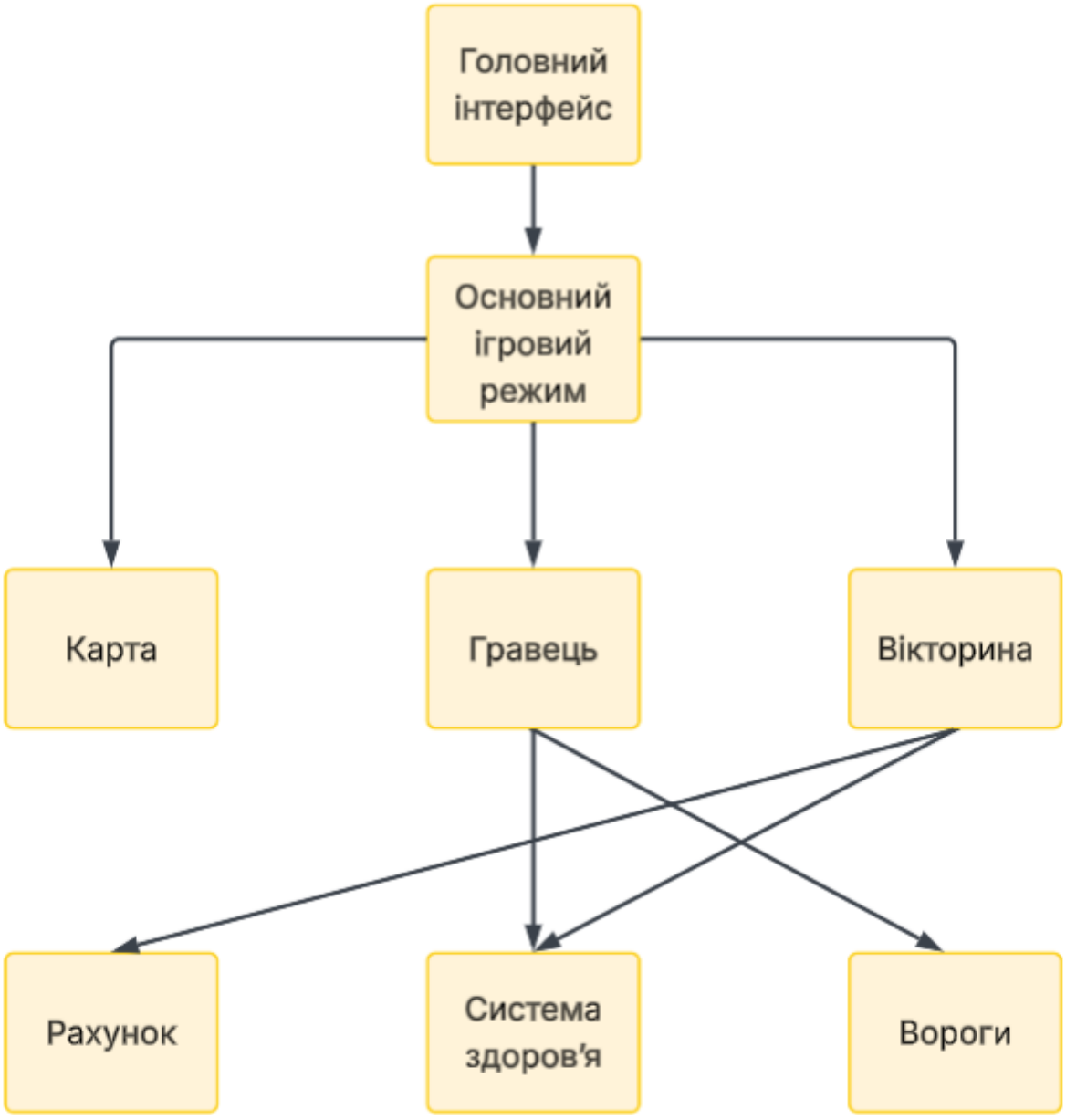

Рисунок 2.1 – Загальна структурна схема ігрового процесу MathQuest

Нарешті, усі навчальні та ігрові механіки об’єднані фіксованою системою нарахування балів: за кожну правильну відповідь гравець отримує певну кількість очок, яка відображається цифровим лічильником у HUD і стимулює прагнути кращих результатів у наступних проходженнях. Цей набір механік створює динамічний, зрозумілий та мотиваційний ігровий досвід, у який хочеться повертатися знову й знову [12].

2.2 Розробка UML-діаграм програмного модуля MathQuest

Розподіл функціональності на класи у процесі розробки відеоігор дає змогу чітко відокремити кожен компонент системи та визначити його роль. Так, у навчальному проєкті класи можуть відповідати за керування гравцем, поведінку ворогів, взаємодію з об’єктами та організацію вікторин. Кожен із цих класів містить власні властивості – наприклад, позицію в просторі, рівень здоров’я або складність завдань – і методи, які реалізують відповідну логіку: рух, опрацювання вводу чи перевірку відповіді.

Механізми успадкування та поліморфізму дають змогу визначити загальну поведінку в базових класах і потім спеціалізувати її для різних типів об’єктів [13]. У контексті навчальних ігор це означає, що одна й та сама система перевірки відповідей може обслуговувати як математичні, так і логічні вправи, а єдина модель відображення результатів підходить до будь-якого виду вікторин. Такий підхід значно полегшує модифікацію окремих частин програми – наприклад, оновлення алгоритму генерації запитань – без загрози внести помилки в інші модулі.

Для документування та візуалізації побудованої структури використовують UML-діаграми класів. Ці схеми наочно показують, які класи входять до складу системи, які атрибути та методи вони містять і як встановлені зв’язки між ними. Завдяки цьому команда розробників може

швидко орієнтуватися в структурі проєкту, спрощуючи передачу знань і прискорюючи процес інтеграції нових учасників.

На рисунку 2.2 зображено UML-діаграму основних класів персонажів навчальної частини гри MathQuest.

QuizManager – це клас, що відповідає за відображення запитань та обробку відповідей у навчальному режимі. Коли настає час поставити запитання, він відкриває панель із текстом запитання та кнопками з варіантами відповідей. Після вибору гравцем відповіді, клас тимчасово підсвічує результат (правильно чи неправильно) та передає відповідну інформацію назад об’єкту, який ініціював навчальний режим [14].

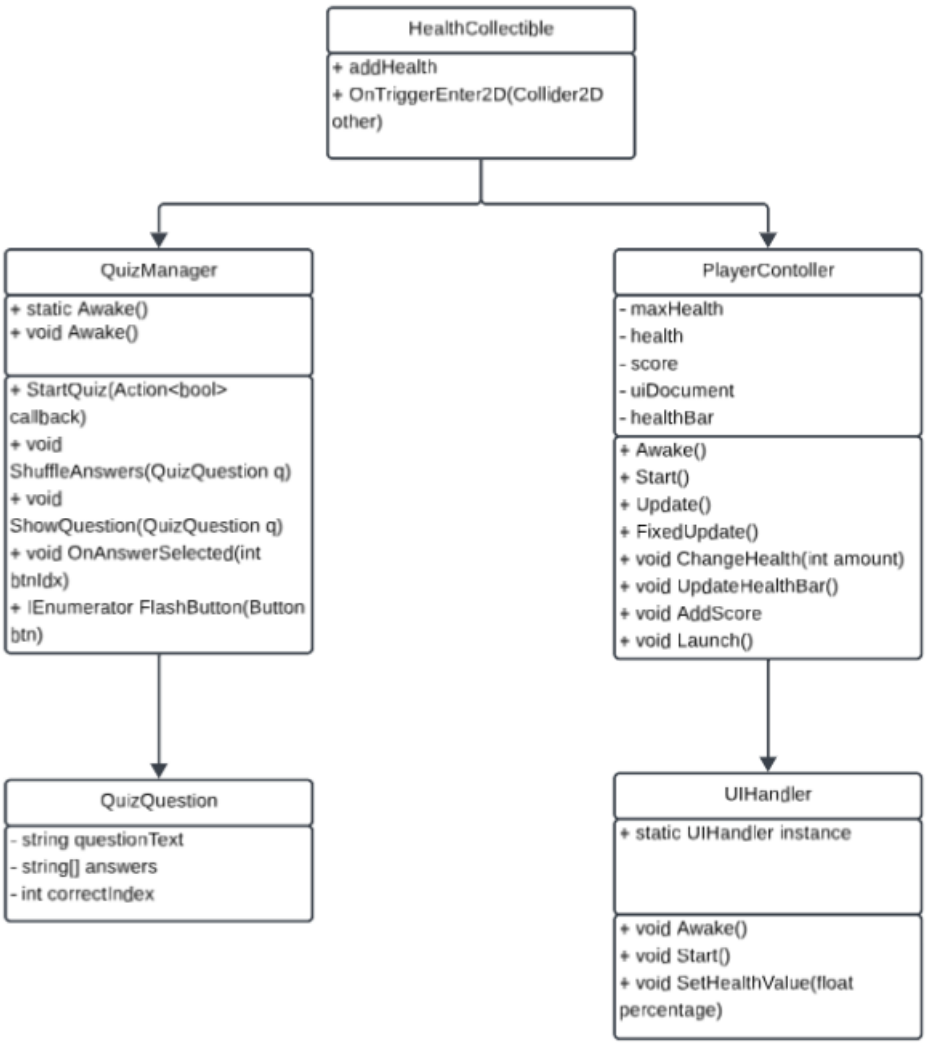

Рисунок 2.2 – UML-діаграма основних класів навчальної частини гри MathQuest

QuizQuestion – це проста структура, я1ка зберігає одне запитання, кілька варіантів відповіді та зазначає правильний серед них. Клас, що відповідає за відображення запитань, лише зчитує ці дані й показує їх гравцеві.

HealthCollectible – клас, прикріплений до об'єкта на карті, що надає гравцеві додаткове здоров’я та очки. Коли гравець наближається до об’єкта, запускається показ запитання через QuizManager. У разі правильної відповіді гравець отримує бонус до здоров’я та бали; якщо ж відповідь неправильна – об’єкт просто зникає.

PlayerController основний клас, що керує рухом персонажа, його здоров’ям і рахунком. Він слідкує за натисканнями клавіш, оновлює індикатор здоров’я та показує поточний рахунок. Інші об’єкти (наприклад, ті, що дають здоров’я) звертаються до цього класу, щоб збільшити життя або додати бали гравцю.

На рисунку 2.3 показано рекомендовану ієрархію класів для навчальної частини гри MathQuest

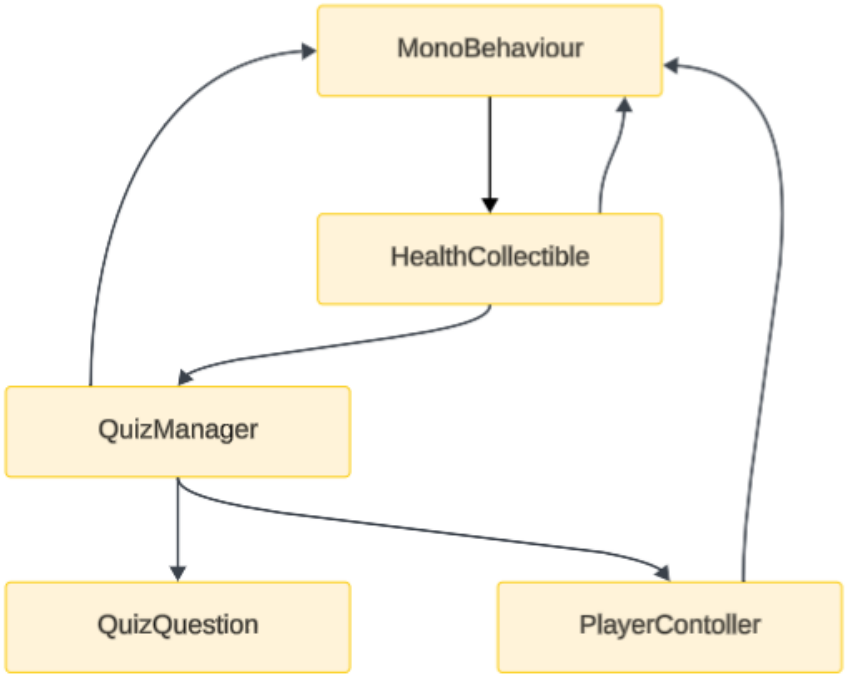

Рисунок 2.3 – Ієрархія класів для навчальної частини гри MathQuest

На рисунку 2.3 показано, що MonoBehaviour є основним класом, від якого походять усі ключові компоненти навчальної частини [15]. Клас QuizManager відповідає за завантаження й демонстрацію запитань. Він зберігає їх у масиві QuizQuestion і запускає процедуру показу вікторини гравцеві. Завдяки цьому центральному контролеру можна легко змінювати або додавати нові запитання без втручання в інші частини коду.

Клас HealthCollectible служить тригером для початку вікторини. Коли гравець торкається об’єкта, метод OnTriggerEnter2D звертається до QuizManager через його одиничний екземпляр і дає команду розпочати показ запитань. Таким чином сама взаємодія з колекційним об’єктом залишається простою та чіткою.

Після того як гравець відповідає на запитання, QuizManager обробляє відповідь і передає результат класу PlayerController. Цей клас змінює здоров’я персонажа відповідно до успіху чи помилки у вікторині. Якщо відповідь правильна, здоров’я відновлюється на певну кількість одиниць, якщо ні – гравець може спробувати ще раз або отримати меншу винагороду.

Клас QuizQuestion – це проста структура даних із трьома полями: текст запитання, масив варіантів відповідей і індекс правильної відповіді. Він не наслідує MonoBehaviour і існує лише для зберігання даних. Така структура гарантує, що кожен клас виконує лише одну роль і легко піддається розширенню: можна додавати нові типи колекторів чи змінювати поведінку вікторини, не порушуючи загальну структуру проєкту.

На рисунку 2.4 представлено UML-діаграму класів розважальної частини гри MathQuest.

PlayerContoller займається управлінням гравцем у просторі та опрацюванням усіх його дій. Під час запуску гри цей клас ініціалізує необхідні налаштування, а в кожному кадрі обробляє введення від клавіатури, переміщує персонажа і слідкує за його здоров’ям і набраними очками. Саме тут формується взаємодія з елементами інтерфейсу: від оновлення шкали здоров’я до відображення рахунку. Крім цього, коли гравець вирішує вистрелити,

PlayerController створює снаряд (Projectile) і відправляє його в обраному напрямку.

EnemyController відповідає за логіку ворога: він сам рухається по сцені відповідно до налаштованих параметрів, реагує на зіткнення з перешкодами й зоною пошкодження, а у разі попадання снаряда від гравця може “полагодитися” – тобто перейти в інший стан взаємодії. Життєвий цикл ворога настроюється таким чином, щоби він міг завдавати шкоди гравцю при контакті й одночасно стати ціллю для Projectile.

DamageZone – це область на карті, яка постійно віднімає здоров’я гравця, доки він у ній перебуває. Клас, прив’язаний до цієї зони, перевіряє кожен кадр, чи знаходиться персонаж всередині, і передає виклик PlayerController для зменшення значення здоров’я.

GameEnder стежить за тим, чи досяг гравець фінішної точки або чи вичерпався його ресурс здоров’я. Як тільки виконуються умови завершення (перемога чи програш), цей клас активує спеціальну панель із підсумком гри та відповідним повідомленням, блокуючи подальшу взаємодію ігрових компонентів.

Всі взаємодії між цими класами позначені стрілками на UML-діаграмі: відкриті стрілки вказують на спадкування від MonoBehaviour, а суцільні лінії з підписами “uses” або “calls” показують, які класи викликають методи інших. Така структура робить код зрозумілим і гнучким – можна додавати нові елементи або змінювати поведінку кожного класу, не порушуючи загальну структуру.

Клас PlayerContrller відповідає за відстеження руху гравця, обробку команд вводу, підтримку показників здоров’я і рахунку, а також за створення снарядів. Саме тут при кожному кадрі перевіряється стан клавіш або джойстика, оновлюється положення персонажа на екрані та передається команда для запуску Projectile у потрібному напрямку. Крім того, PlayerController отримує виклики від DamageZone і HealthCollectible, щоб зменшувати або відновлювати здоров’я відповідно до подій на карті.

EnemyController керує поведінкою ворогів: він рухає їх за заданими параметрами, стежить за зіткненнями з іншими об’єктами і змінює стан залежно від того, чи потрапив у нього снаряд гравця. Коли Projectile контактує з ворогом, EnemyController може перейти в стан “виправлено” або активувати власну анімацію отримання пошкодження, а в разі зіткнення з гравцем – завдати йому шкоди.

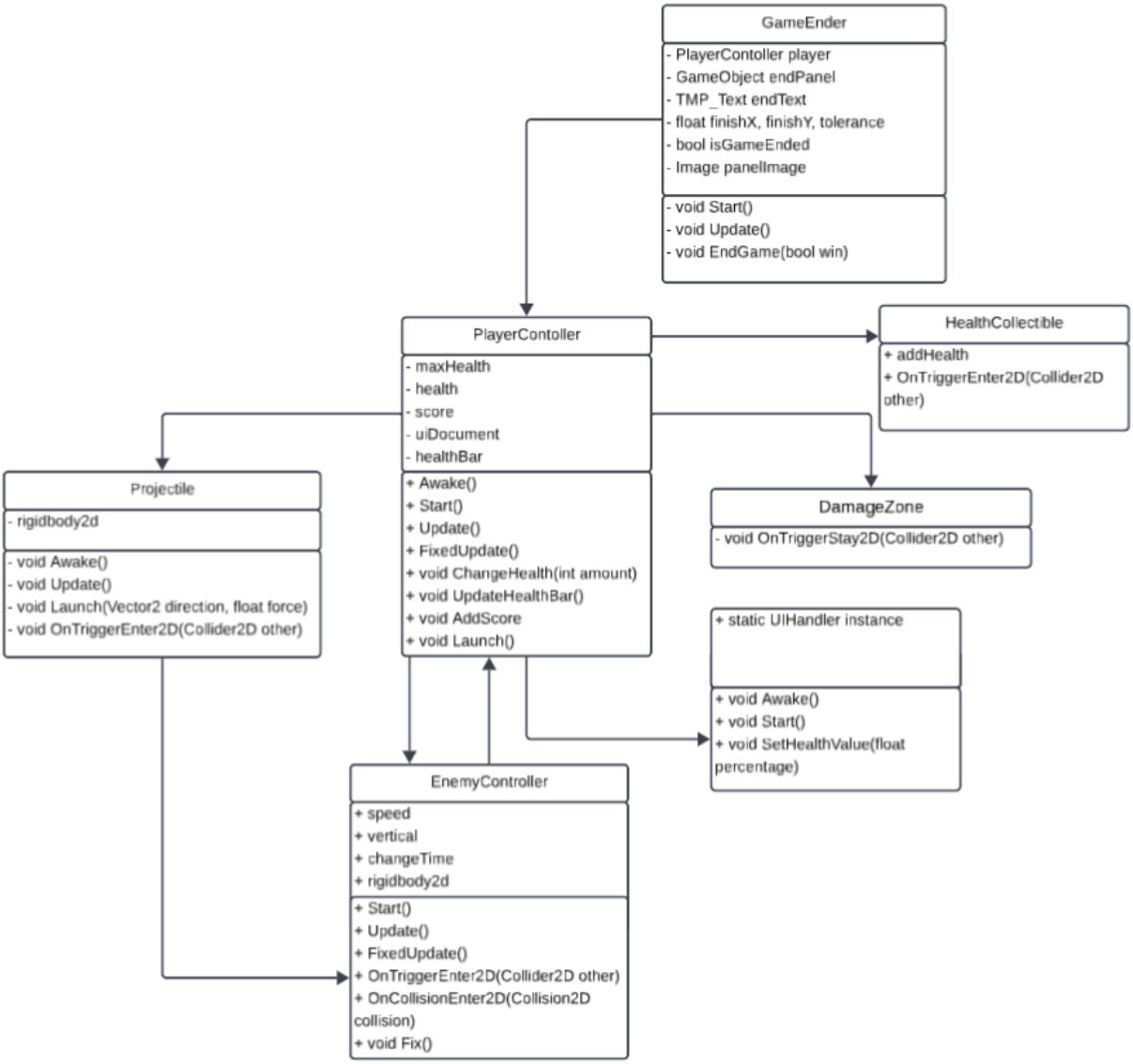

Рисунок 2.4 – UML-діаграма класів розважальної частини гри MathQuest

DamageZone – спеціальна область на карті, яка постійно перевіряє наявність у ній гравця й у разі контакту викликає метод зміни здоров'я в PlayerController. HealthCollectible, навпаки, концентрується на відновленні – при торканні до такого об'єкта ігрова логіка починає "навчальний" сценарій, де гравець може пройти випробування у вигляді короткої вікторини, а потім отримати бонус до здоров'я.

Клас GameEnder стежить за умовами завершення розважальної частини: перевіряє, чи досяг гравець заданої точки або вичерпав ресурс здоров'я, і вмикає відповідну панель із повідомленням про перемогу чи поразку. Поряд із цим, UIHandler відповідає за оновлення інтерфейсу, наприклад за відображення шкали здоров'я або кількості зібраних бонусів [16].

Завдяки такій організації всі класи легко розширювати та підтримувати: кожен з них виконує чітко відокремлену роль у випробувальному режимі, а зміни в одному місці не впливають на роботу інших.

На рисунку 2.5 представлено ієрархію класів для розважальної частини гри MathQuest

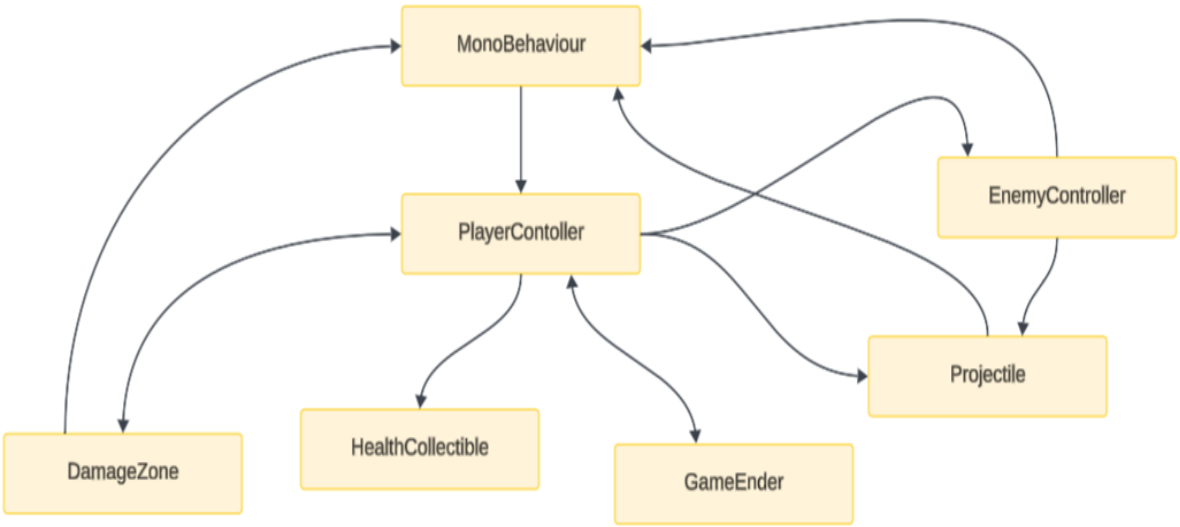

Рисунок 2.5 – Ієрархія класів для розважальної частини гри MathQuest

2.3 Розробка загального алгоритму роботи програмного модуля MathQuest

Алгоритм можна уявити як чітко прописану інструкцію, що складається з послідовних кроків для досягнення конкретного результату. У звичайному житті алгоритми супроводжують нас на кожному кроці: рецепт улюбленої страви детально пояснює, які інгредієнти й коли додавати, інструкція зі складання меблів покроково показує, як поєднати деталі, а ранковий ритуал водія – заправити паливо, повернути ключ у замку та рушити в дорогу – теж є своїм різновидом алгоритму [17].

У розробці програм алгоритми забезпечують порядок і передбачуваність: комп’ютер виконує виключно ті дії, які вказані в коді, і саме алгоритмічна структура керує цим процесом. Одні алгоритми виконуються лінійно, крок за кроком, без жодних умов – наприклад, спочатку завантажити ресурси, потім ініціалізувати інтерфейс, а вже після цього надати користувачеві можливість взаємодіяти з меню. Інші містять умовні гілки: якщо гравець має достатньо ресурсів – відкрити новий рівень, інакше – показати підказку або екран з інструкцією. Також широко застосовуються циклічні конструкції, коли певні дії повторюються, поки не буде виконано умову – наприклад, очікувати, поки користувач не введе коректні дані, або оновлювати фізику руху персонажа до завершення рівня.

На рисунку 2.6 наведено загальний алгоритм обробки події отримання ушкодження в MathQuest.

Крок 1. Перевірка можливості нанесення шкоди. Переконуються, що персонаж не мертвий і не перебуває в стані тимчасової невразливості. Якщо він може отримати ушкодження, то перехід на крок 2.

Крок 2. Зменшення показника здоров’я. Викликається метод ChangeHealth(-amount) компонента здоров’я, який коректно оновлює рівень HP персонажа.

Крок 3. Візуалізація пошкодження. Запускається ефект мерехтіння чи зміни кольору спрайта, що чітко сигналізує гравцю про отримане ушкодження.

Крок 4. Виклик додаткової логіки обробки шкоди. Активується розширений під алгоритм, який може містити відтворення звукових ефектів, анімацій реакції чи інших спеціальних дій.

Крок 5. Установка стану тимчасової невразливості. Персонаж отримує короткочасний буфер, протягом якого подальші атаки ігноруються, даючи час на відновлення та ухилення.

Рисунок 2.6 – Схема алгоритму події отримання будь-якої шкоди персонажу

Рисунок 2.7 ілюструє алгоритм обробки ушкоджень головного героя.

Крок 1. Визначення типу об’єкта. Система дізнається, з чим саме зіткнувся герой: із зоною ушкодження, ворогом чи корисним предметом.

Крок 2. Ушкодження від небезпечної зони. Якщо це зона, яка автоматично шкодить, втрачається одне «життя», і герою відображається візуальний ефект пошкодження.

Крок 3. Ушкодження від ворога. Якщо зіткнення сталося з ворогом, втрачається дві одиниці здоров’я, після чого запускається ефект трясіння або мерехтіння персонажа.

Крок 4. Підбір лікувального предмета. Якщо ж герой підібрав аптечку або інший колекційний предмет, починається коротка вікторина.

Крок 5. Проведення вікторини. На екрані з’являється запитання з кількома варіантами відповідей, і гравець обирає найбільш правильний.

Крок 6. Наслідки вікторини

– У разі вірної відповіді повертається одне «життя», а предмет знищується.

– Якщо відповідь неправильна, герой втрачає одне «життя», але предмет залишається для повторного підбору.

Крок 7. Оновлення стану здоров’я. Після будь-якого зміни здоров’я на екрані миттєво оновлюється індикатор «життів», щоб гравець бачив свій поточний рівень.

Крок 8. Перевірка вичерпання «життів». Система перевіряє, чи не стало здоров’я нульовим або від’ємним.

Крок 9. Показ екрана поразки. Якщо «життів» не залишилось, з’являється екран поразки з відповідним повідомленням.

Крок 10. Продовження гри. Якщо ж ще є запас здоров’я, на персонажу програється короткий ефект мерехтіння, і гра повертається до нормального режиму.

Крок 11. Завершення обробки події. Після всіх цих дій алгоритм повертає управління основному циклу гри, готовий реагувати на наступні зіткнення.

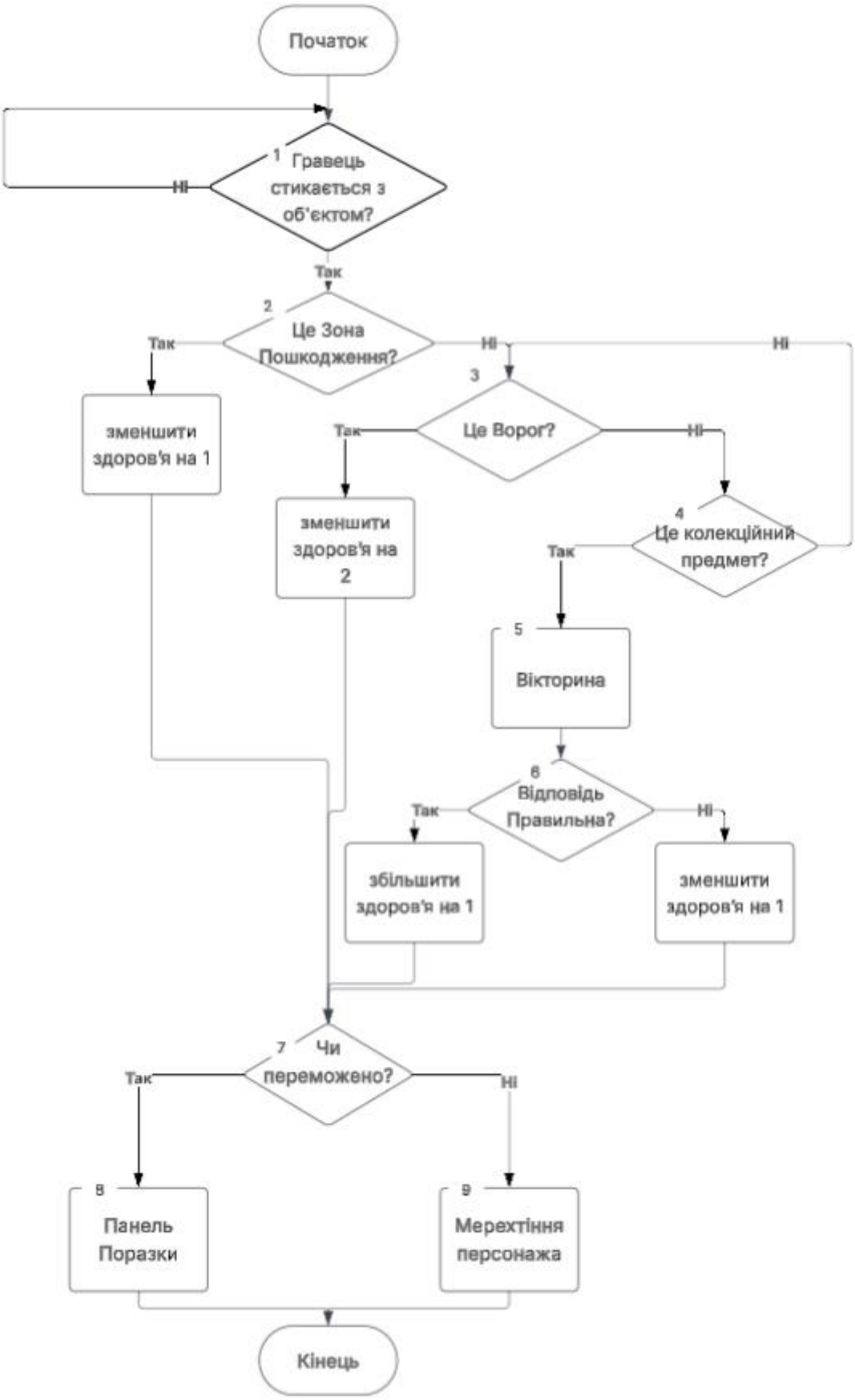

Рисунок 2.7 – Алгоритм обробки ушкодження головного героя

2.4 Висновок

Розроблено загальну структурну схему програмного модуля MathQuest. Дана схема відрізняється тим, що в ній виокремлено блок взаємодії з ігровим середовищем та блок «Вікторини», що відповідає за генерацію та перевірку математичних завдань. Це дає змогу краще і швидше навчатися, адже чітке розділення ігрової та навчальної частин дозволяє зосереджено виконувати завдання без зайвих перемикань і отримувати високий рівень мотивації через інтерактивний формат.

Для програмного модуля MathQuest створено дві UML-діаграми, що окремо описують навчальну та розважальну частини. Кожна з них містить спеціалізовані класи для відповідних режимів роботи, що значно спрощує орієнтацію в структурі модуля. Навчальна діаграма ілюструє процес ініціалізації вікторини, зберігання питань, відображення інтерфейсу та оновлення стану гравця, тоді як розважальна показує, як класи керують рухом персонажа, обробляють зіткнення з ворогами та зонами ушкодження, а також визначають умови завершення рівня й відображення результатів.

Розроблено алгоритми роботи модуля MathQuest, які забезпечують чітку обробку події нанесення шкоди: загальний варіант контролює можливість атаки, змінює рівень здоров’я, відтворює візуальні та звукові ефекти й активує тимчасову невразливість, а спеціалізований для головного героя додатково розрізняє тип об’єкта (зона, ворог або лікувальний предмет), застосовує різні величини втрат, проводить інтерактивну вікторину з наслідками та оновлює стан «життів». Відмінність полягає в рівні деталізації та гнучкості: базова схема гарантує передбачуваність і простоту, а розширена — динаміку й залучення гравця. Це спростило підтримку коду, зробило поведінку модуля стабільною та покращило ігровий досвід завдяки зрозумілим індикаціям і взаємодії.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ НАВЧАЛЬНОЇ ГРИ MATHQUEST

3.1 Обґрунтування вибору рушія та мови програмування

Для розробки модуля MathQuest було обрано ігровий рушій Unity у поєднанні з мовою програмування C#. Такий вибір ґрунтується на ряді вагомих переваг, що роблять даний технологічний стек оптимальним для 2D Top-Down ігор із навчальною частиною [18 – 20].

По-перше, Unity пропонує потужний та водночас інтуїтивний редактор сцен із вбудованими інструментами для створення 2D-ігор: Tilemap дозволяє швидко малювати ігрові рівні з тайлів, а Sprite Renderer спрощує роботу з анімаціями спрайтів. Для MathQuest, де карта складається з величезної кількості повторюваних елементів, можливість малювати рівень у редакторі замість ручного розміщення кожного тайла значно прискорює процес дизайну [21 – 23].

По-друге, Unity має Asset Store – величезну онлайн-бібліотеку готових пакетів і ресурсів. Це дозволяє заощадити час на розробку базових механік і UI-елементів, а також швидко інтегрувати готові плагіни для ефектів, фізики чи аудіо. Для MathQuest обходитися без такої економії ресурсів було б надто дорого та довго, адже необхідна швидка ітерація над навчальними викликами й інтерфейсом.

По-третє, рушій підтримує крос-платформеність: розроблений модуль можна зібрати під Windows, macOS, Linux, Android, iOS та веб-платформи без суттєвих змін у коді. Це важливо для освітнього продукту, що може використовуватися як у шкільних комп’ютерних класах, так і на персональних ґаджетах учнів [24].

Щодо C#, ця мова відрізняється суворою типізацією та об’єктно-орієнтованою парадигмою, що сприяє зрозумілому й підтримуваному коду. У

Unity всі скрипти – це класи, успадковані від MonoBehaviour, тому C# з його синтаксисом дозволяє створювати чітку структуру класів, інкапсулювати механіки та легко реалізовувати патерни проєктування (Singleton, Observer тощо).

Крім того, C# добре працює з .NET – це означає, що можна легко підключати готові бібліотеки для роботи з JSON, базами даних (наприклад, SQLite) або для тестування коду (наприклад, NUnit). Це робить програму більш надійною та гнучкою. А такі програми, як Visual Studio та Rider, допомагають розробнику: вони показують помилки, підказують код і мають зручні інструменти для його покращення, що пришвидшує роботу та полегшує пошук проблем.

Отже, рішення використати Unity із C# обґрунтоване їхньою доступністю, широким функціоналом для 2D-ігрових механік, ефективною підтримкою спільноти та можливістю масштабувати продукт на різні платформи. Це забезпечує швидкий цикл розробки, чисту структуру коду та зручність подальшого супроводу MathQuest.

3.2 Вибір програмного середовища

Під час вибору середовища розробки для проекту на C# з використанням Unity було розглянуто кілька популярних рішень, що дозволяють працювати з великою кількістю скриптів та складними взаємодіями. Першим кандидатом став JetBrains Rider, який відомий своїми розвиненими можливостями автодоповнення й оптимізації коду [25]. Інструменти аналізу коду в Rider миттєво виявляють потенційні помилки й застарілі виклики API, а глибока інтеграція з Unity дозволяє швидко переходити від інспектора рушія до відповідних рядків скриптів. Однак для комерційного використання необхідна платна ліцензія, а під час роботи з великими сценами спостерігається помітне

навантаження на оперативну пам’ять, що може уповільнювати інші інструменти розробника

Як альтернативу було протестовано Visual Studio Code [26]. Це легке та безплатне рішення має широку бібліотеку розширень для C# (OmniSharp), Unity Snippets і налагодження за допомогою плагіна Debugger for Unity. Незважаючи на мінімальні системні вимоги та швидкий старт, VS Code виявився недостатньо зручним для комплексного проєкту: автодоповнення працює на базовому рівні, а налаштування відлагодження потребує ручного втручання та детальної конфігурації. Крім того, в редакторі відсутні вбудовані інструменти профілювання коду та глибокого аналізу продуктивності.

MonoDevelop або його оновлений аналог Visual Studio for Mac надали можливість одразу почати писати C#-скрипти на macOS без додаткових установок [27]. Інтерфейс здається знайомим для користувачів Unity, але відсутність частих оновлень та нестабільна робота автодоповнення створюють труднощі під час роботи з новими версіями C# та Unity. Система рефакторингу тут обмежена базовими командами, а налагодження не підтримує “гаряче” оновлення коду під час виконання гри.

Оптимальним виявилося використання Visual Studio Community від Microsoft. Це середовище розробки забезпечує автоматичне генерування проєктних файлів для Unity, коректне підключення всіх бібліотек та налаштувань IntelliSense для класів рушія. Вбудований налагоджувач підтримує встановлення точок зупину прямо в редакторі, перегляд значень змінних у реальному часі та “гаряче” перезавантаження коду без повного перезапуску гри. Інструменти профілювання продуктивності та аналізу коду дозволяють працювати з найскладнішими сценаріями, своєчасно виявляючи “вузькі місця” в алгоритмах. Наявність розвиненої підтримки Git, інтеграція з Azure DevOps і GitHub Actions полегшують організацію безперервної інтеграції та тестування збірок.

Оскільки Visual Studio Community безплатно доступна для індивідуальних розробників і невеликих команд, а також отримує регулярні

оновлення від Microsoft із підтримкою нових стандартів C# і плагінів для Unity, було вирішено саме її використовувати. Це забезпечує збалансованість між зручністю налагодження, потужністю інструментів аналізу та стабільністю роботи над великим ігровим проєктом.

3.3 Розробка розважальної частини гри MathQuest

Розважальний режим у MathQuest побудовано як окремий сценарій, у якому гравець одночасно долає перешкоди, б’ється з ворогами та проходить короткі освітні випробування. У цьому розділі наведено огляд ключових класів, їхню взаємодію та фрагменти коду, які демонструють реалізацію основних механік.

Клас PlayerContoller є головним елементом: у методі Awake() встановлюється початкове здоров’я, а в кожному кадрі обробляється ввід гравця, рух персонажа та запуски снарядів. Метод ChangeHealth(int amount) змінює поточне значення здоров’я, гарантуючи його в межах від 0 до максимуму, і одразу оновлює індикатор в UI. У наведеному прикладі:

```
public class PlayerContoller : MonoBehaviour
{
   public int maxHealth = 5;
   public int health => currentHealth;
   int currentHealth;
   int score;

   void Awake()
   {
      currentHealth = maxHealth;
   }
```

```
    public void ChangeHealth(int amount)
    {
        currentHealth = Mathf.Clamp(currentHealth + amount, 0, maxHealth);
        UIHandler.instance.SetHealthValue(currentHealth / (float)maxHealth);
    }

    public void AddScore(int amount)
    {
        score += amount;
        UIHandler.instance.SetScoreValue(score);
    }
}
```

У класі EnemyController взаємодія з гравцем і снарядами побудована на двох подіях колізії. Перший сценарій спрацьовує в методі OnTriggerEnter2D(Collider2D other): коли фізичний тригер ворога виявляє інший об’єкт, Unity передає його в змінну other. Ми намагаємося отримати з цього об’єкта компонент PlayerController–і якщо такий є, викликаємо player.ChangeHealth(-2). Таким чином, як тільки гравець заходить у зону ворога, у його скрипті зменшується запас здоров’я на дві одиниці, а UI одразу оновлює значення шкали життя.

Другий сценарій активується ззовні, наприклад, коли по ворогу влучає снаряд і його скрипт викликає публічний метод Fix(). У цьому методі змінна broken встановлюється в false, що зупиняє подальший рух ворога (у FixedUpdate() є перевірка, яка припиняє будь-які кроки патрулювання, якщо broken == false). Далі відключається фізична симуляція через rigidbody2d.simulated = false, щоб ворог більше не реагував на зіткнення й грав гравцем фізичний світ його ігнорував. І нарешті спрацьовує анімаційний тригер animator.SetTrigger("Fixed"), який заводить відповідну анімацію

«ремонту» – показує, що ворог відновив бойову здатність і більше не становить загрози. Завдяки такому розділенню логіки можна легко контролювати два різні стани ворога: активний і небезпечний, а також відремонтований і безпечний для гравця.

```
void OnTriggerEnter2D(Collider2D other)
{
   PlayerController player = other.GetComponent<PlayerController>();
   if (player != null)
      player.ChangeHealth(-2);
}

public void Fix()
{
   broken = false;
   rigidbody2d.simulated = false;
   animator.SetTrigger("Fixed");
}
```

У класі Projectile поєднано просту, але ефективну фізичну модель польоту з обробкою колізій у Unity. Коли гравець натискає кнопку стрільби, у PlayerController створюється новий об’єкт Projectile, і викликається його метод Launch(direction, force). Усередині цього методу rigidbody2d.AddForce(direction * force) додає імпульс у вказаному напрямку та з заданою силою, після чого снаряд самостійно рухається згідно з налаштуваннями гравітації, маси та опору в компоненті Rigidbody2D.

Як тільки снаряд перетинає область іншого колайдера з прапорцем Is Trigger, Unity викликає метод OnTriggerEnter2D(Collider2D other). У ньому ми пробуємо отримати з об’єкта other компонент EnemyController. Якщо такий знайдено, це означає, що снаряд влучив у ворога, тому ми одразу викликаємо

enemy.Fix(). Цей виклик переводить ворога в безпечний «відремонтований» стан: зупиняє його рух, відключає фізичну симуляцію й активує анімацію ремонту.

Після обробки колізії снаряд уже не потрібен, і щоб уникнути зайвих фізичних обчислень та звільнити пам’ять, ми викликаємо Destroy(gameObject). Таким чином, Projectile забезпечує лаконічний цикл життя: від створення та польоту до взаємодії з ворогом і самознищення одразу після виконання своєї функції.

```
public void Launch(Vector2 direction, float force)
{
   rigidbody2d.AddForce(direction * force);
}

void OnTriggerEnter2D(Collider2D other)
{
   EnemyController enemy = other.GetComponent<EnemyController>();
   if (enemy != null)
      enemy.Fix();
   Destroy(gameObject);
}
```

Небезпечні зони DamageZone – це області геймплею, у яких гра автоматично зменшує поточні очки здоров’я (HP) персонажа. Коли гравець потрапляє у таку зону, запускається процес поступової втрати здоров’я: поки персонаж перебуває в зоні колізії з DamageZone, щосекунди з його показника вилучається певна кількість HP.

```
void OnTriggerStay2D(Collider2D other)
{
```

```
    PlayerController controller = other.GetComponent<PlayerController>();
    if (controller != null)
        controller.ChangeHealth(-1);
}
```

Щоб урівноважити випробування та додати освітній елемент, на рівні розміщені об’єкти HealthCollectible, що відновлюють життя лише після правильного проходження вікторини. При зіткненні запускається QuizManager.Instance.StartQuiz, і в колбеку, якщо відповідь коректна, гравець отримує здоров’я й додаткові бали:

```
void OnTriggerEnter2D(Collider2D other)
{
    var player = other.GetComponent<PlayerController>();
    if (player != null && player.health < player.maxHealth)
    {
        QuizManager.Instance.StartQuiz(correct =>
        {
            if (correct)
            {
                player.ChangeHealth(addHealth);
                player.AddScore(1);
                Destroy(gameObject);
            }
        });
    }
}
```

Клас GameEnder у методі Update() постійно перевіряє, чи гравець втратив усі життя або досягнув фінішної точки. Якщо Lives стають рівними

нулю, викликається EndGame(false) та відображається екран поразки, а подальший ігровий процес блокується. Якщо ж гравець торкається об’єкта фінішу, викликається EndGame(true), і з’являється екран перемоги.

Метод EndGame вимикає керування персонажем і відтворює відповідні звукові ефекти, а також активує потрібну UI-панель (Win або Lose) і зупиняє гру (наприклад, через Time.timeScale = 0). Таким чином, поки не встановиться прапорець завершення (isGameEnded), Update() реагує лише на два ключові стани повна втрата життів або проходження рівня і зумовлює показ відповідного підсумкового екрана.

```
void Update()
{
   if (player.health <= 0)
   {
      EndGame(false);
      return;
   }

   Vector2 pos = player.transform.position;
   if (Mathf.Abs(pos.x - finishX) <= tolerance &&
      Mathf.Abs(pos.y - finishY) <= tolerance)
   {
      EndGame(true);
   }
}
```

Завдяки чітко розділеним ролям кожного класу та використанню подій Unity для обробки колізій, випробувальний режим MathQuest виявився гнучким і легко розширюваним. Для оформлення візуальної частини гри були використані безплатні асети з Unity Asset Store [30].

На рисунку 3.1 наведено вигляд вікна гри MathQuest.

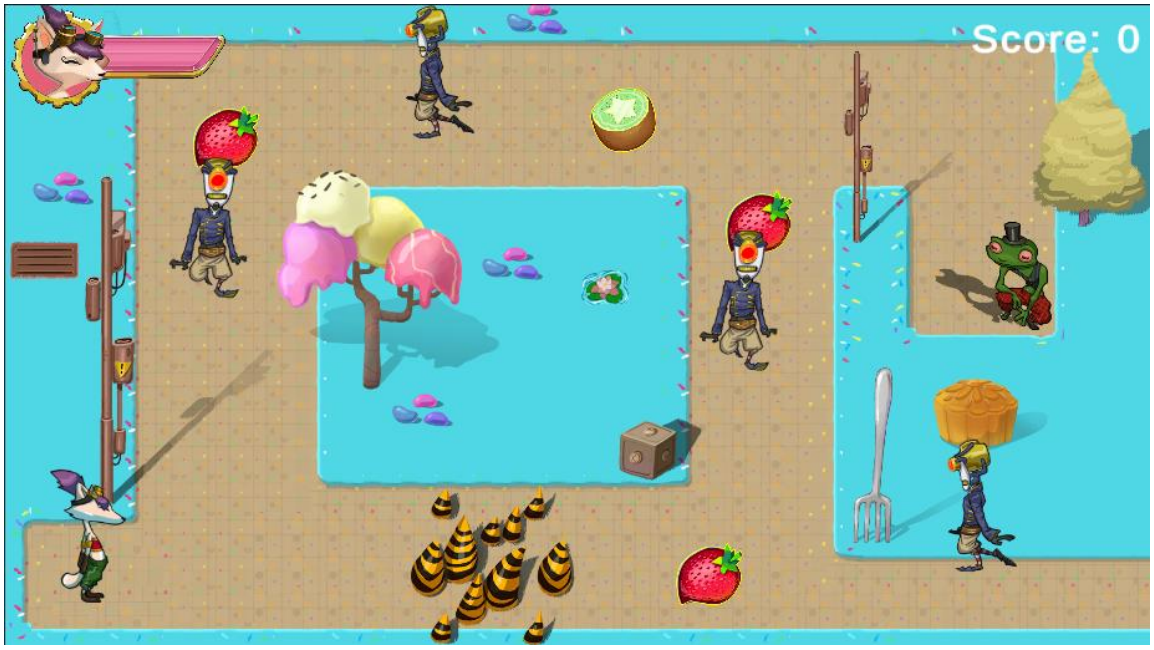

Рисунок 3.1 – Вигляд вікна гри MathQuest

3.4 Розробка навчальної частини гри MathQuest

Навчальний режим у MathQuest створено так, щоб ігровий процес органічно поєднувався з розвитком математичних навичок. Гравець досліджує рівень і збирає спеціальні об’єкти, але щоб отримати бонус – відновити здоров’я або заробити додаткові бали – він проходить коротку вікторину. Це перетворює кожен рівень на інтерактивне завдання, де правильна відповідь відкриває нові можливості, а помилка мотивує до повторної спроби.

Клас QuizManager – серце навчальної частини. Як єдиний екземпляр (Singleton), він зберігає масив об’єктів QuizQuestion, випадково обирає питання, зупиняє основний цикл гри (Time.timeScale = 0f) і відображає вікно з текстом запитання та кнопками відповідей. Після вибору гравцем одного з варіантів перевіряє відповідь, відтворює короткий ефект правильності чи помилки та викликає передану йому функцію, яка має виконатися після завершення цього процесу.

```
public void StartQuiz(Action<bool> callback)
{
   onQuizComplete = callback;
   var q = defaultQuestions[Random.Range(0, defaultQuestions.Length)];
   ShuffleAnswers(q);
   ShowQuestion(q);
   Time.timeScale = 0f;
}
```

Клас QuizQuestion служить простою структурою для кожного запитання. Його поля – текст питання, масив варіантів і індекс правильної відповіді – дозволяють легко додавати нові завдання без переробки логіки керування вікториною:

```
[Serializable]
public class QuizQuestion
{
   public string questionText;
   public string[] answers;
   public int correctIndex;
}
```

Об’єкт HealthCollectible поєднує ігрову механіку відновлення здоров’я з навчанням. Коли гравець торкається такого предмета, скрипт перевіряє, чи є в нього запас для поповнення, і тільки потім запускає вікторину:

```
void OnTriggerEnter2D(Collider2D other)
{
   var player = other.GetComponent<PlayerController>();
```

```
    if (player != null && player.health < player.maxHealth)
    {
        QuizManager.Instance.StartQuiz(correct =>
        {
            if (correct)
            {
                player.ChangeHealth(addHealth);
                player.AddScore(1);
                Destroy(gameObject);
            }
        });
    }
}
```

Якщо відповідь правильна, HealthCollectible передає гравцю бонусні одиниці здоров’я та додаткові бали, а потім зникає з рівня. У разі помилки предмет залишається – гравець може спробувати ще раз.

Клас PlayerContoller відповідає за стан героя: переміщення, здоров’я, рахунок і стрільбу [28]. Методи ChangeHealth(int amount) та AddScore(int amount) оновлюють відповідні показники та миттєво відображають зміни в UI:

```
public void ChangeHealth(int amount)
{
    currentHealth = Mathf.Clamp(currentHealth + amount, 0, maxHealth);
    UIHandler.instance.SetHealthValue(currentHealth / (float)maxHealth);
}

public void AddScore(int amount)
{
    score += amount;
```

```
    UIHandler.instance.SetScoreValue(score);
}
```

Гравець наближається до HealthCollectible, після чого запускається метод StartQuiz у QuizManager. Питання з’являється на екрані, і гра зупиняється, доки не буде обрана відповідь. Якщо гравець відповідає вірно, у PlayerContoller викликаються ChangeHealth і AddScore, що відновлює життя й додає бали. Якщо ж ні – предмет лишається, а гра відновлюється без бонусу [29].

Такий підхід забезпечує тісний зв’язок між ігровими механіками та навчальним контентом: кожен об’єкт, з яким взаємодіє герой, може стати приводом для невеликого освітнього виклику, що робить процес пізнання цікавим і динамічним. Завдяки чіткій розподіленій структурні додавання нових питань, видів бонусів чи складніших завдань не потребує змін у самому рушії – достатньо розширити масив QuizQuestion або реалізувати новий тип коллекційного об’єкта.

На рисунку 3.2 наведено приклад вікна для розв’язування задачок у навчальному режимі.

Рисунок 3.2 – Вигляд вікна гри під час розв’язування задачі

3.5 Тестування роботи програми та аналіз результатів

Завершальним етапом розробки гри MathQuest стало комплексне тестування всіх ігрових механік та аналіз отриманих результатів. Протягом розробки було проведено понад 50 тестових запусків, щоб переконатися у стабільності роботи кожного елемента, а також у зручності та зрозумілості геймплею для користувача.

Після запуску гри гравець одразу потрапляє у навчальний режим, де знайомиться з основними елементами керування. Для переміщення персонажа використовується класична схема WASD. Всі дії гравця, такі як рух, атака, підбір бонусів чи взаємодія з об’єктами, були ретельно перевірені.

Особливу увагу під час тестування було приділено освітній складовій гри. Наприклад, при підборі бонусу для здоров’я гравець отримує математичне питання. Якщо відповідь правильна – персонаж отримує додаткове здоров’я та очки. Цей механізм було протестовано на різних рівнях складності питань, щоб переконатися у коректній роботі логіки перевірки відповідей та нарахування бонусів.

Під час тестування також перевірялась робота ворогів та небезпечних зон. Вороги реагують на появу гравця, можуть завдати шкоди при зіткненні, а також "ремонтуються" після влучання снарядом. Зони шкоди поступово зменшують здоров’я персонажа, якщо той затримується у них надто довго. Всі ці механіки були протестовані на різних рівнях, щоб уникнути неочікуваних багів та забезпечити чесний баланс складності.

Під час тестування також перевірялась робота ворогів та небезпечних зон. Вороги реагують на появу гравця, можуть завдати шкоди при зіткненні, а також "ремонтуються" після влучання снарядом. Зони шкоди поступово зменшують здоров’я персонажа, якщо той затримується у них надто довго. Всі ці механіки були протестовані на різних рівнях, щоб уникнути неочікуваних багів та забезпечити чесний баланс складності.

На рисунку 3.3 показано вікно гри, де гравець дає неправильну відповідь на навчальне запитання і невірний варіант підсвічується червоним.

Рисунок 3.3 – Відповідь на питання у навчальному режимі

Якщо під час проходження рівня шкала здоров’я повністю спустошується, на екрані автоматично з’являється вікно поразки. Воно перекриває ігровий сцени, повідомляє про поразку. Це дозволяє гравцеві швидко повернутися до гри та заново спробувати пройти рівень, не відволікаючись на додаткові меню чи налаштування.

На рисунку 3.4 продемонстровано, як виглядає вікно поразки.

У випадку успішного подолання всіх викликів з’являється вікно перемоги. Воно вітає гравця з завершенням рівня і переходом на наступний рівень. Такий підхід підтримує інтерес і заохочує продовжувати гру.

На рисунку 3.5 продемонстровано, як виглядає вікно перемоги

Під час тестування було виявлено та усунуто кілька незначних помилок, пов’язаних із некоректною роботою колізій, відображенням UI на різних пристроях та обробкою геймпадів. Після виправлення цих недоліків гра стала стабільною та зручною для користувача.

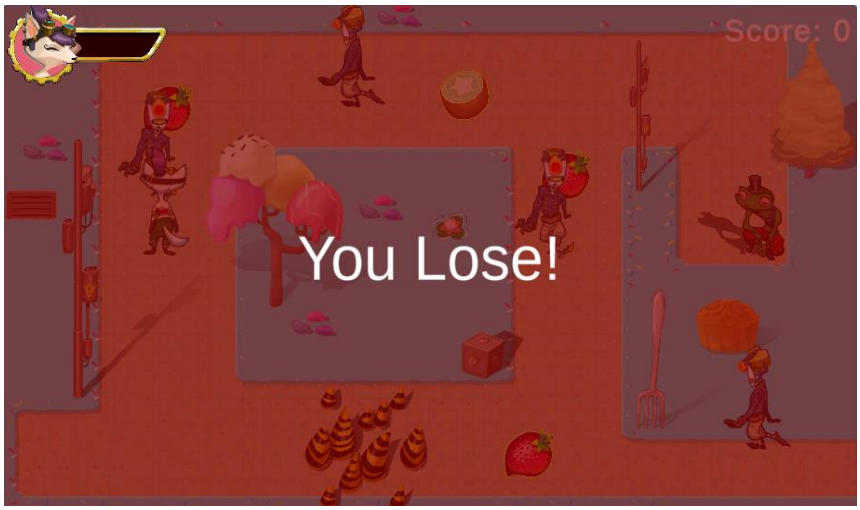

Рисунок 3.4 – Вигляд вікна завершення гри у випадку поразки

Для порівняння результатів тестування було проведено аналіз аналогічних інді-ігор. MathQuest вигідно вирізняється поєднанням освітньої та розважальної складової, простотою керування, відсутністю критичних багів та оптимізацією для слабких ПК. Гра підтримує як клавіатуру, так і геймпади, що робить її доступною для ширшої аудиторії, включаючи гравців з особливими потребами.

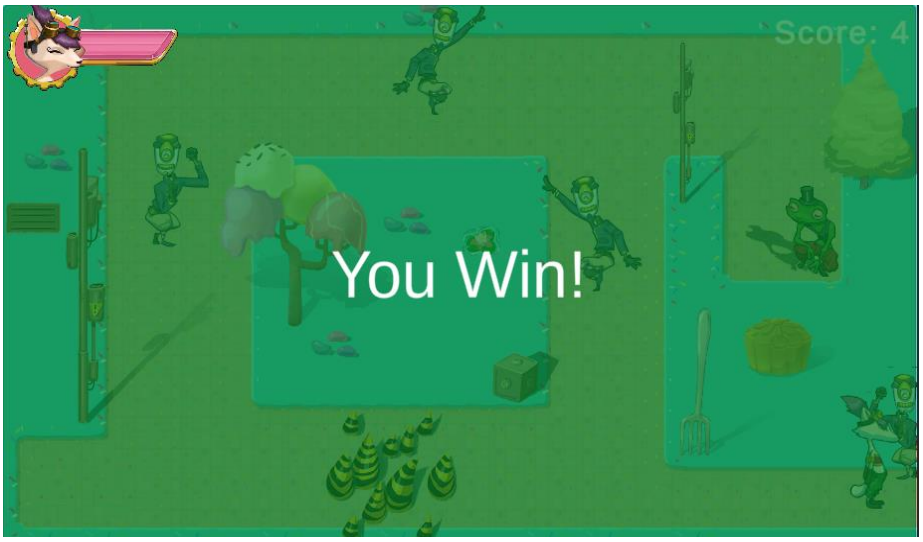

Рисунок 3.5 – Вигляд вікна завершення гри у випадку перемоги

У таблиці 3.1 наведено порівняння основних характеристик MathQuest та аналогічних ігор.

Таблиця 3.1 – Порівняння результатів тестування MathQuest та Minecraft Education

	Poly Bridge	Minecraft Education	MathQuest
Поєднання навчання і геймплею	+	+	+
Безкоштовна	-	–	+
Відсутність критичних багів	-	+	+
Оптимізація для слабких ПК	+	–	+

Підсумовуючи, можна сказати, що розроблений програмний продукт повністю відповідає поставленим вимогам. Всі основні ігрові механіки працюють стабільно, а освітній елемент гармонійно інтегрований у геймплей. MathQuest є прикладом того, як можна поєднати цікаву гру з навчанням, забезпечивши при цьому якісний користувацький досвід.

3.6 Висновок

У цьому розділі здійснено програмну реалізацію модуля навчальної гри MathQuest. Обґрунтовано використання рушія Unity та мови програмування C#, що дозволило досить швидко створити 2D-гру з інтегрованою навчальною частиною. На відміну від інших платформ, Unity забезпечує зручний візуальний редактор, підтримку анімацій, фізики та кросплатформеність, а C# дає змогу будувати логічно зрозумілу, підтримувану архітектуру коду. Це дало змогу реалізувати складні ігрові сценарії без надмірних витрат часу та ресурсів, а також гарантувати масштабованість і стабільність проєкту.

У ході роботи було проведено аналіз програмних середовищ для розробки на C#. Серед розглянутих рішень найкращим виявилося Visual Studio Community, що забезпечує глибоку інтеграцію з Unity, інструменти для

налагодження, підтримку Git і регулярні оновлення. На відміну від Rider або VS Code, які мають обмеження (платність або недостатня стабільність), обране середовище є безкоштовним і повноцінно підтримує всі необхідні функції.

Особливістю проєкту стало створення розважального ігрового режиму, що поєднує керування персонажем, бої з ворогами, взаємодію з об’єктами та зони шкоди. На відміну від типових навчальних ігор, де освітній контент представлений ізольовано, в MathQuest геймплей інтегрує освітню логіку без втрати динаміки гри. Це дасть змогу утримувати інтерес користувачів, зробити процес пізнання природною частиною гри, а не зовнішнім доповненням.

У рамках освітнього режиму було реалізовано систему вікторин, керовану окремим класом QuizManager, що забезпечує гнучке керування питаннями та обробку відповідей. Завдяки цьому рішення стало можливим легко розширювати базу завдань, додавати нові види взаємодії та гейміфікувати навчання.

Під час тестування було проведено понад 50 запусків із перевіркою усіх сценаріїв: руху, боїв, використання бонусів, вікторин, роботи UI, логіки завершення гри. MathQuest вирізняється від аналогів поєднанням навчальної та розважальної частин гри та доступністю для слабких ПК.

ВИСНОВКИ

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

– здійснено аналіз освітніх ігор серед основних недоліків – високі апаратні вимоги, через які вони часто не запускаються на старих чи мобільних пристроях, та ігрові механіки, що не враховують різний рівень підготовки учнів.

– спроєктовано загальну структуру програмного модуля, а також розроблено UML-діаграми класів для обох режимів – навчального та розважального. Це забезпечило чітке уявлення про структуру системи та взаємодію її компонентів;

– розроблено алгоритм роботи програмного модуля, що об’єднує навчальний контент, геймплей і їхню взаємодію. Архітектура з незалежних блоків і чітким розділенням функцій дозволяє додавати або змінювати компоненти без впливу на інші частини. Це скорочує час налагодження та підвищує надійність модуля;

– реалізовано програмний модуль MathQuest, що містить функціонал для взаємодії гравця з ігровим середовищем та блоками математичної вікторини, що відрізняється наявністю ігрової та навчальної частин гри. Здійснено тестування яке підтвердило правильність тестування програмного модуля;

– підготовлено інструкцію користувача, що містить опис функціоналу гри, правила взаємодії та основні рекомендації, що забезпечує зручність використання продукту кінцевими користувачами.

В ході виконання роботи було поєднано перевірені педагогічні підходи з захопливими ігровими механіками та створили наочні схеми, що пояснюють структуру й алгоритми системи. Готовий продукт побудовано на модульній структурі, що полегшує його масштабування й підтримку, а адаптивна складність і миттєвий зворотний зв’язок роблять інтерфейс зручним і зрозумілим.

Мету розширення функціональних можливостей було досягнуто шляхом інтеграції навчального контенту з елементами розваги в єдиному 2D Top-Down середовищі та використання об’єктно-орієнтованого підходу для гнучкого й масштабованого проєктування. Оптимізація під слабкі ПК і всебічне тестування забезпечили стабільну, коректну роботу застосунку та високу залученість користувачів.

Робота виконана у повному обсязі відповідно до поставлених цілей, вимог та методичних рекомендацій щодо підготовки бакалаврських кваліфікаційних робіт. Усі етапи були реалізовані належним чином, що підтверджує якість, завершеність і практичну цінність розробленого рішення [31].

Отже, розроблений програмний модуль авторської гри MathQuest повністю відповідає встановленим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Epistory: Typing Chronicles. URL: https://epistorygame.com (дата звернення: 7.05.2025)

2. Planet Zoo. URL: https://www.planetzoo.com (дата звернення: 7.05.2025)

3. Opus Magnum. URL: https://store.steampowered.com/app/562880/Opus_Magnum (дата звернення: 8.05.2025)

4. Poly Bridge. URL: https://store.steampowered.com/app/367450/Poly_Bridge (дата звернення: 8.05.2025)

5. Minecraft Education Edition. URL: https://education.minecraft.net (дата звернення: 9.05.2025)

6. DragonBox Algebra. URL: https://dragonbox.com/products/dragonbox-algebra (дата звернення: 10.05.2025)

7. Вдовиченко А.І., Арсенюк І.Р. Підхід щодо гейміфікації процесу тестування під час навчання. URL: https://repository.example.edu.ua/vdovychenko_arseniuk_gamification (дата зверення: 10.05.2025)

8. Prodigy Math. URL: https://www.prodigygame.com (дата звернення: 10.05.2025)

9. Zoombinis. URL: https://www.zoombinis.com (дата звернення: 10.05.2025)

10. More Mountains TopDown Engine – The TopDown Engine lets you develop top-down games, from 2D to 3D, to get you inspired. URL: https://topdown-engine.moremountains.com/#:~:text=The%20TopDown%20Engine%20lets%20you,3D%2C%20to%20get%20you%20inspired. (дата звернення: 11.05.2025)

11. Beginning Game Development: Health Bar – стаття про створення індикатора здоров’я в іграх. URL: https://medium.com/@lemapp09/beginning-game-development-health-bar-a0c9104bcd14 (дата звернення: 11.05.2025)

12. MyGames Dev – Heads-Up: What Is HUD and Why Is It So Important in Video Games. URL: https://www.artstation.com/blogs/mygames_dev/mEQ2/heads-up-what-is-hud-and-why-is-it-so-important-in-video-games (дата звернення: 28.05.2025)

13. SoftServe Inc. – What Is Object-Oriented Programming (OOP)? Explaining Four Major Principles. URL: https://career.softserveinc.com/uk-ua/stories/what-is-object-oriented-programming-oop-explaining-four-major-principles (дата звернення: 12.05.2025)

14. Foxminded – Класи в програмуванні. URL: https://foxminded.ua/klasy-v-prohramuvanni/ (дата звернення: 13.05.2025)

15. MonoBehaviour (Unity Scripting API, версія 6000.1) – опис основного базового класу для скриптів у Unity. URL: https://docs.unity3d.com/6000.1/Documentation/ScriptReference/MonoBehaviour.html (дата звернення: 13.05.2025)

16. RetroStyleGames – UI Game Meaning: Значення інтерфейсу користувача в іграх. URL: https://retrostylegames.com/blog/ui-game-meaning/ (дата звернення: 14.05.2025)

17. ITStep Cloud – Building and Understanding Algorithms: A Step-by-Step Guide for Beginners. URL: https://cloud.itstep.org/blog/building-and-understanding-algorithms-a-step-by-step-guide-for-beginners (дата звернення: 15.05.2025)

18. W3Schools – C# Tutorial. URL: https://www.w3schools.com/cs/index.php (дата звернення: 16.05.2025)

19. Unity Official Site – Головна сторінка ігрового рушія Unity. URL: https://unity.com (дата звернення: 16.05.2025)

20. Unity Manual – Документація Unity2D. URL: https://docs.unity3d.com/6000.1/Documentation/Manual/Unity2D.html (дата звернення: 17.05.2025)

21. Unity Manual (версія 2019.4) – Tilemap Class. URL: https://docs.unity3d.com/ru/2019.4/Manual/class-Tilemap.html (дата звернення: 18.05.2025)

22. Unity Manual – Sprite Renderer. URL: https://docs.unity3d.com/6000.1/Documentation/Manual/sprite/renderer/renderer-landing.html (дата звернення: 20.05.2025)

23. Brian David – 2D Sprite Animation in Unity: How to Animate a Sprite Sheet. URL: https://medium.com/@Brian_David/2d-sprite-animation-in-unity-how-to-animate-a-sprite-sheet-b72a5afab83a (дата звернення: 19.05.2025)

24. Yubico – Cross-Platform: Glossary entry on cross-platform development. URL: https://www.yubico.com/resources/glossary/cross-platform/ (дата звернення: 20.05.2025)

25. JetBrains Rider – кросплатформене IDE для .NET від JetBrains. URL: https://www.jetbrains.com/rider/ (дата звернення: 21.05.2025)

26. Visual Studio Code – редактор коду від Microsoft. URL: https://code.visualstudio.com/ (дата звернення: 21.05.2025)

27. Visual Studio – інтегроване середовище розробки (IDE) від Microsoft. URL: https://visualstudio.microsoft.com/ (дата звернення: 21.05.2025)

28. 2D Sugar World Asset Pack URP – набір 2D-активів для середовища. URL: https://assetstore.unity.com/packages/2d/environments/2d-sugar-world-asset-pack- urp-256585 (дата звернення: 22.05.2025)

29. CharacterController.Move (Unity Scripting API, версія 6000.1) – опис методу переміщення персонажа в Unity. URL: http://docs.unity3d.com/6000.1/Documentation/ScriptReference/CharacterController.Move.html (дата звернення: 22.05.2025)

30. 2D Sugar World Asset Pack URP – набір 2D-активів для середовища (Unity Asset Store). URL: https://assetstore.unity.com/packages/2d/environments/2d-sugar-world-asset-pack-urp-256585 (дата звернення: 22.05.2025)

31. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп’ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. Вінниця : ВНТУ, 2023. (PDF, 54 с.).

ДОДАТКИ

ДОДАТОК А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль навчальної гри MathQuest

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,14 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Арсенюк І.Р.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Гавриш А.І.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (обов'язковий)

Лістинг програми

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

public class QuizManager : MonoBehaviour
{
   public static QuizManager Instance;

   [Header("UI Elements")]
   public GameObject quizPanel;       // Panel in Canvas (initially Inactive)
   public TMP_Text questionText;      // Where to display the question
   public Button[] answerButtons;     // Array of buttons (Size = 3)

   // Internal
   private int[] answerOrder;
   private int correctButtonIndex;
   private Action<bool> onQuizComplete;
   private Image panelImage;

   // Default questions
   private readonly QuizQuestion[] defaultQuestions = new QuizQuestion[]
   {
   // 4th Grade – slightly more difficult tasks
   new QuizQuestion
   {
      questionText = "У книжці 240 сторінок. Якщо учень прочитав \n" +
```

```
                "⅓ книжки в понеділок і ще ⅖ залишку у вівторок,\n" +
                "скільки сторінок залишилося прочитати?",
        answers = new[] { "88", "96", "104" },
        correctIndex = 2
    },
    new QuizQuestion
    {
        questionText = "Обчисліть: (125 − 47) + (63 ÷ 7) = ?",
        answers = new[] { "141", "139", "137" },
        correctIndex = 1
    },
    new QuizQuestion
    {
        questionText = "Прямокутник має довжину 15 см і ширину 8 см.\n" +
                "На кожну сторону зробили рамку шириною 2 см.\n" +
                "Знайдіть площу рамки (різницю між площами великого\n" +
                "і початкового прямокутника)",
        answers = new[] { "94 см²", "79 см²", "86 см²" },
        correctIndex = 0
    },
    new QuizQuestion
    {
        questionText = "У кіоску продали 72 шоколадки за 4 дні.\n" +
                "Щодня продавалося на 5 шоколадок більше, ніж попереднього дня.\n" +
                "Скільки шоколадок продали в найдень з найменшим продажем?",
        answers = new[] { "14", "16", "18" },
        correctIndex = 1
    },
```

```
    new QuizQuestion
    {
        questionText = "Скільки вольтів буде, якщо 3 батарейки по 1,5 В\n" +
                       "з'єднати послідовно, а ще одну батарейку 1,5 В під'єднати\n" +
                       "так, щоб вони були в паралелі з уже послідовно з'єднаними трьома?",
        answers = new[] { "6 В", "4,5 В", "7,5 В" },
        correctIndex = 1
    },
    new QuizQuestion
    {
        questionText = "Знайдіть суму дробів: 3/5 + 2/3 = ?",
        answers = new[] { "19/15", "17/15", "11/15" },
        correctIndex = 1
    },
    new QuizQuestion
    {
        questionText = "Якщо коробка важить 3,6 кг, а у ній 12 однакових штук іграшок,\n" +
                       "а вага самої коробки без іграшок – 0,8 кг, то скільки важить одна іграшка?",
        answers = new[] { "0,22 кг", "0,25 кг", "0,30 кг" },
        correctIndex = 1
    },
    new QuizQuestion
    {
        questionText = "У класі 28 учнів. Вони розбилися на 7 рядків по одному стільцю.\n" +
                       "Якщо переставити учнів у 4 ряди по  ? (скільки в кожному ряду?),\n" +
```

```
            "щоб всі місця були зайняті, скільки учнів у кожному ряду?",
        answers = new[] { "7", "6", "8" },
        correctIndex = 0
    }
    };



    void Awake()
    {
        // Singleton
        if (Instance == null) Instance = this;
        else { Destroy(gameObject); return; }

        // Check bindings
        if (quizPanel == null || questionText == null || answerButtons == null)
        {
            Debug.LogError("QuizManager: one of the UI elements is not assigned!");
            return;
        }

        // 1) Configure panel background
        panelImage = quizPanel.GetComponent<Image>();
        if (panelImage != null)
        {
            panelImage.color = new Color(0.9f, 0.95f, 1f, 0.85f);  // gentle blue with
transparency
            panelImage.type = Image.Type.Sliced;                // for 9-slice sprite
        }
```

```
// 2) Reduce question text size
questionText.fontSize = 18;
questionText.color = new Color32(20, 20, 20, 255);
questionText.enableWordWrapping = true;
questionText.alignment = TextAlignmentOptions.MidlineLeft;
// Add slight outline for readability
var outline = questionText.gameObject.AddComponent<Outline>();
outline.effectColor = new Color32(255, 255, 255, 128);
outline.effectDistance = new Vector2(1, -1);

// 3) Style buttons: small radius, light shadow, colored border
foreach (var btn in answerButtons)
{
    var img = btn.GetComponent<Image>();
    if (img != null)
    {
        img.color = new Color32(255, 255, 255, 220);
        img.type = Image.Type.Sliced;
        img.pixelsPerUnitMultiplier = 1;

        // apply corner radius via MaskableGraphic
        btn.targetGraphic = img;
    }

    // add Outline component to button
    var btnOutline = btn.gameObject.AddComponent<Outline>();
    btnOutline.effectColor = new Color32(0, 100, 200, 150);
    btnOutline.effectDistance = new Vector2(1, -1);

    // add slight shadow
```

```
        var btnShadow = btn.gameObject.AddComponent<Shadow>();
        btnShadow.effectColor = new Color32(0, 0, 0, 100);
        btnShadow.effectDistance = new Vector2(2, -2);

        var lbl = btn.GetComponentInChildren<TMP_Text>();
        if (lbl != null)
        {
            lbl.fontSize = 16;
            lbl.color = new Color32(30, 30, 30, 255);
            lbl.alignment = TextAlignmentOptions.Center;
        }
    }

    quizPanel.SetActive(false);
}

public void StartQuiz(Action<bool> callback)
{
    onQuizComplete = callback;
    var list = defaultQuestions;
    if (list.Length == 0) { Debug.LogError("no questions available"); return; }

    var q = list[UnityEngine.Random.Range(0, list.Length)];
    ShuffleAnswers(q);
    ShowQuestion(q);
    Time.timeScale = 0f;
}

private void ShuffleAnswers(QuizQuestion q)
{
```

```
        int n = q.answers.Length;
        answerOrder = new int[n];
        for (int i = 0; i < n; i++) answerOrder[i] = i;
        for (int i = n - 1; i > 0; i--)
        {
            int j = UnityEngine.Random.Range(0, i + 1);
            (answerOrder[i], answerOrder[j]) = (answerOrder[j], answerOrder[i]);
        }
        for (int i = 0; i < n; i++)
            if (answerOrder[i] == q.correctIndex)
                correctButtonIndex = i;
    }

    private void ShowQuestion(QuizQuestion q)
    {
        quizPanel.SetActive(true);
        questionText.text = q.questionText;

        // --- Auto reduce font size for long questions ---
        if (q.questionText.Length > 120) questionText.fontSize = 15;
        else questionText.fontSize = 18;

        // Update layout for text

LayoutRebuilder.ForceRebuildLayoutImmediate(questionText.rectTransform);

        // Adjust panel size based on text
        var panelRect = quizPanel.GetComponent<RectTransform>();
        var textRect = questionText.GetComponent<RectTransform>();
        float paddingWidth = 80f; // Slightly larger horizontal padding
```

```
        float paddingHeight = 40f; // Slightly smaller vertical padding
        float minWidth = 350f, minHeight = 160f, maxWidth = 700f, maxHeight =
250f;
        float newWidth = Mathf.Clamp(textRect.rect.width + paddingWidth,
minWidth, maxWidth);
        float newHeight = Mathf.Clamp(textRect.rect.height + paddingHeight +
answerButtons.Length * 45f, minHeight, maxHeight);
        panelRect.SetSizeWithCurrentAnchors(RectTransform.Axis.Horizontal,
newWidth);
        panelRect.SetSizeWithCurrentAnchors(RectTransform.Axis.Vertical,
newHeight);
        // --- End additions ---

        for (int i = 0; i < answerButtons.Length; i++)
        {
            if (i < answerOrder.Length)
            {
                answerButtons[i].gameObject.SetActive(true);
                var lbl = answerButtons[i].GetComponentInChildren<TMP_Text>();
                lbl.text = q.answers[answerOrder[i]];
                answerButtons[i].onClick.RemoveAllListeners();
                int idx = i;
                answerButtons[i].onClick.AddListener(() => OnAnswerSelected(idx));
            }
            else answerButtons[i].gameObject.SetActive(false);
        }
    }

    private void OnAnswerSelected(int btnIdx)
    {
```

```
        bool correct = btnIdx == correctButtonIndex;
        if (correct)
        {
            quizPanel.SetActive(false);
            Time.timeScale = 1f;
            onQuizComplete?.Invoke(true);
            onQuizComplete = null;
        }
        else StartCoroutine(FlashButton(answerButtons[btnIdx]));
    }

    private IEnumerator FlashButton(Button btn)
    {
        var img = btn.GetComponent<Image>();
        if (img == null) yield break;
        var orig = img.color;
        img.color = new Color32(255, 150, 150, 255);
        yield return new WaitForSecondsRealtime(0.4f);
        img.color = orig;
    }
}
```

ДОДАТОК В (обов'язковий)

ГРАФІЧНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ НАВЧАЛЬНОЇ ГРИ MATHQUEST»

Виконала: студентка 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Гавриш А. І.
(прізвище та ініціали)

Керівник: к. т. н., доцент каф. КН
Арсенюк І. Р.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

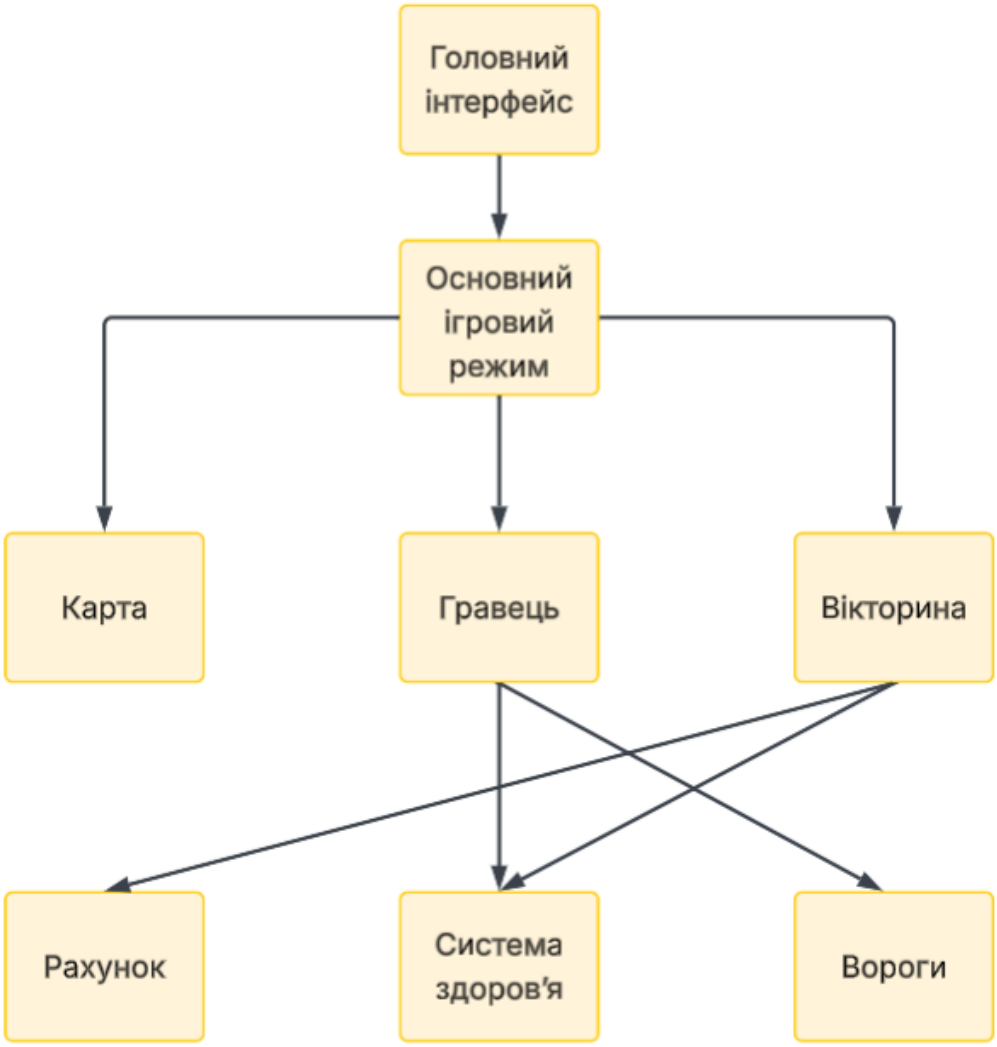

Рисунок В.1 – Загальна структурна схема ігрового процесу MathQuest

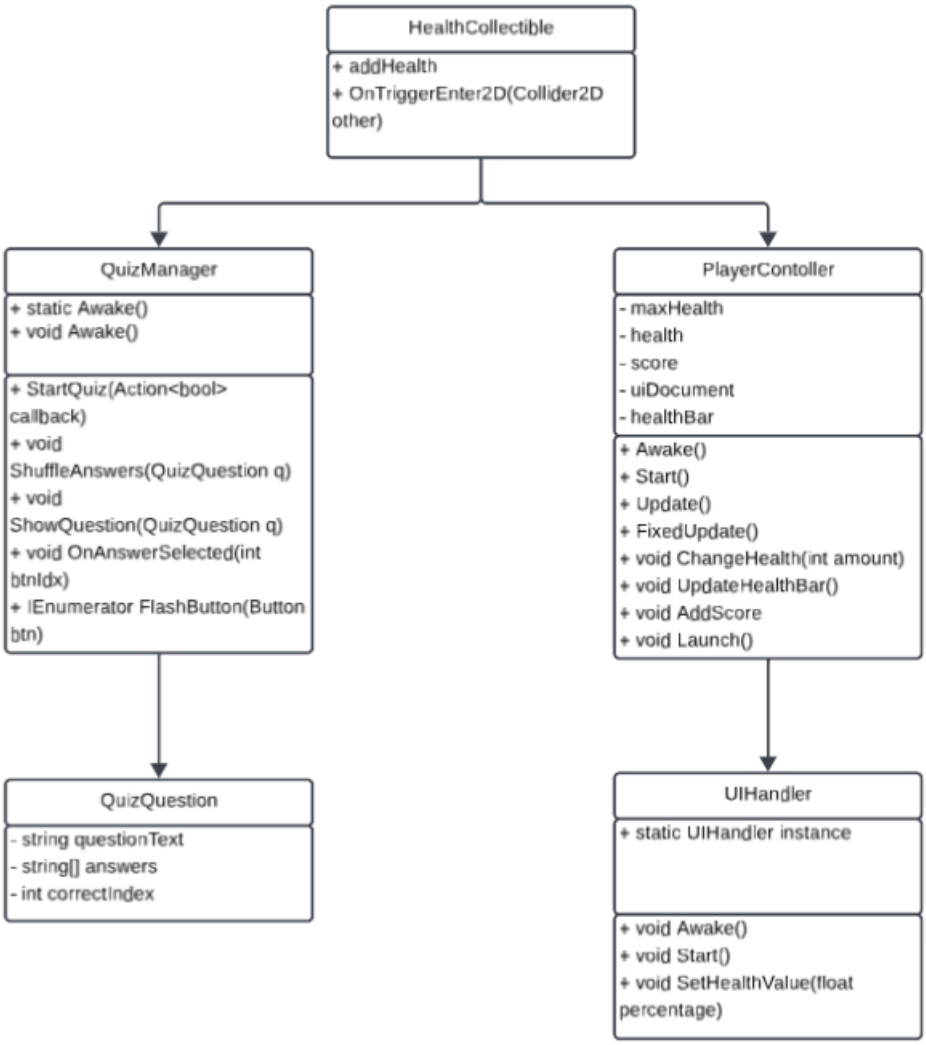

Рисунок В.2 – UML-діаграма основних класів навчальної частини гри MathQuest.

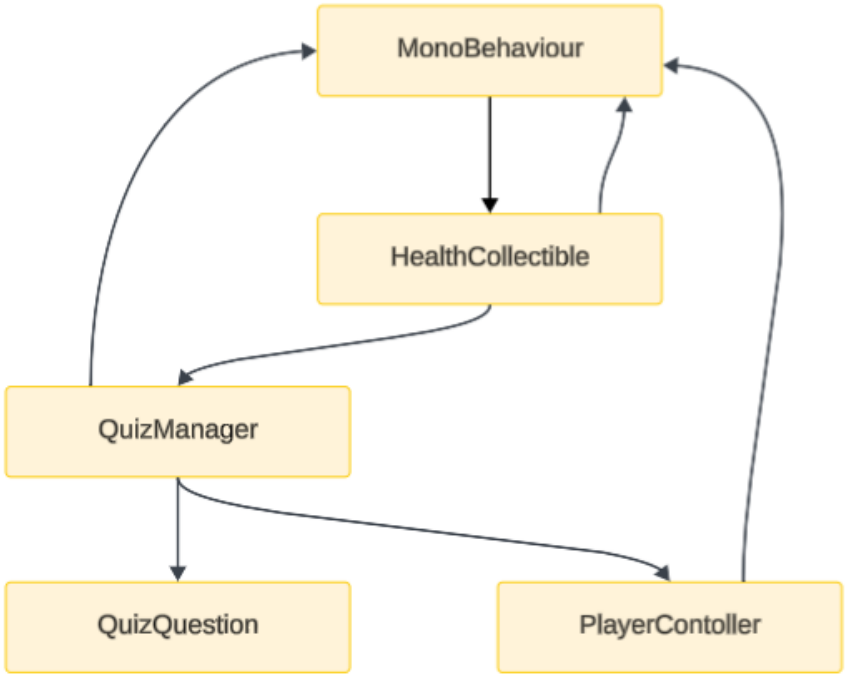

Рисунок В.3 – Ієрархія класів для навчальної частини гри MathQuest

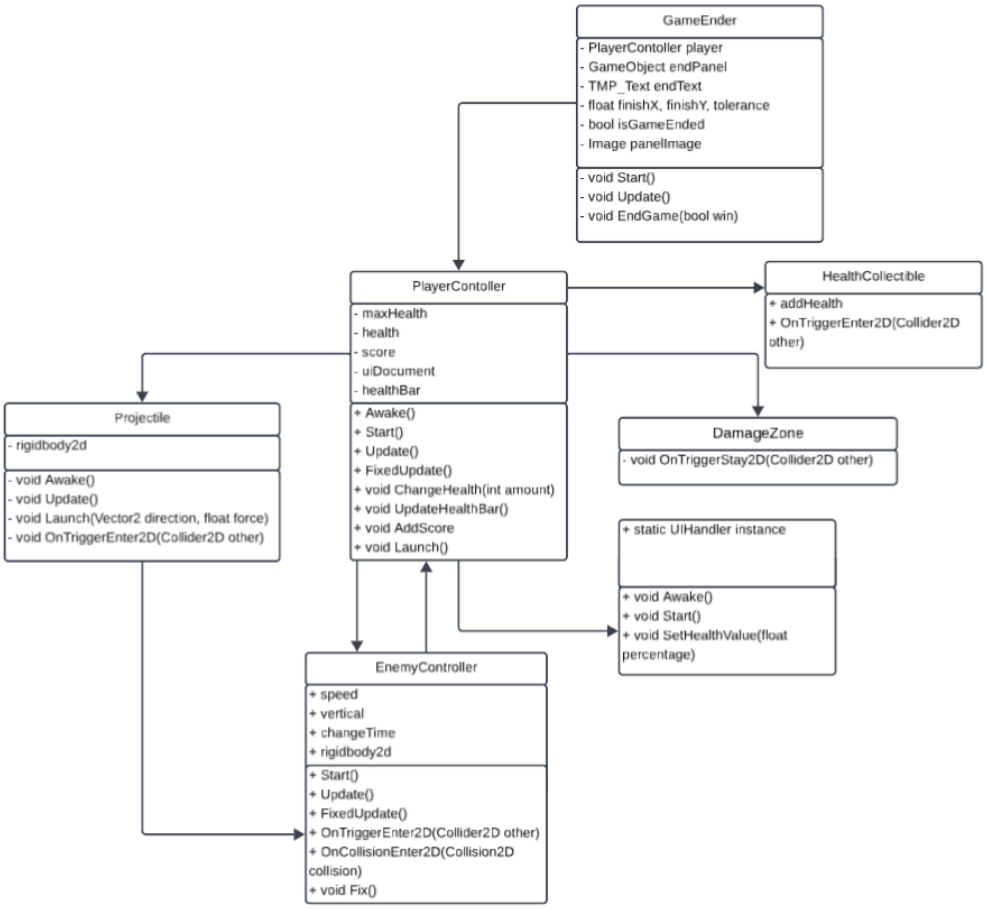

Рисунок В.4 – UML-діаграма класів розважальної частини гри MathQuest

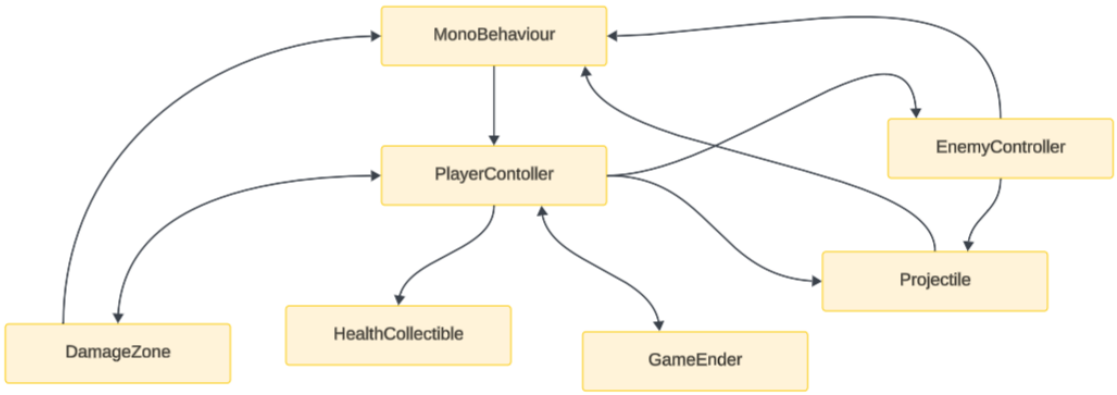

Рисунок В.5 – Ієрархія класів для розважальної частини гри MathQuest

Рисунок В.6 – Схема алгоритму події отримання будь-якої шкоди

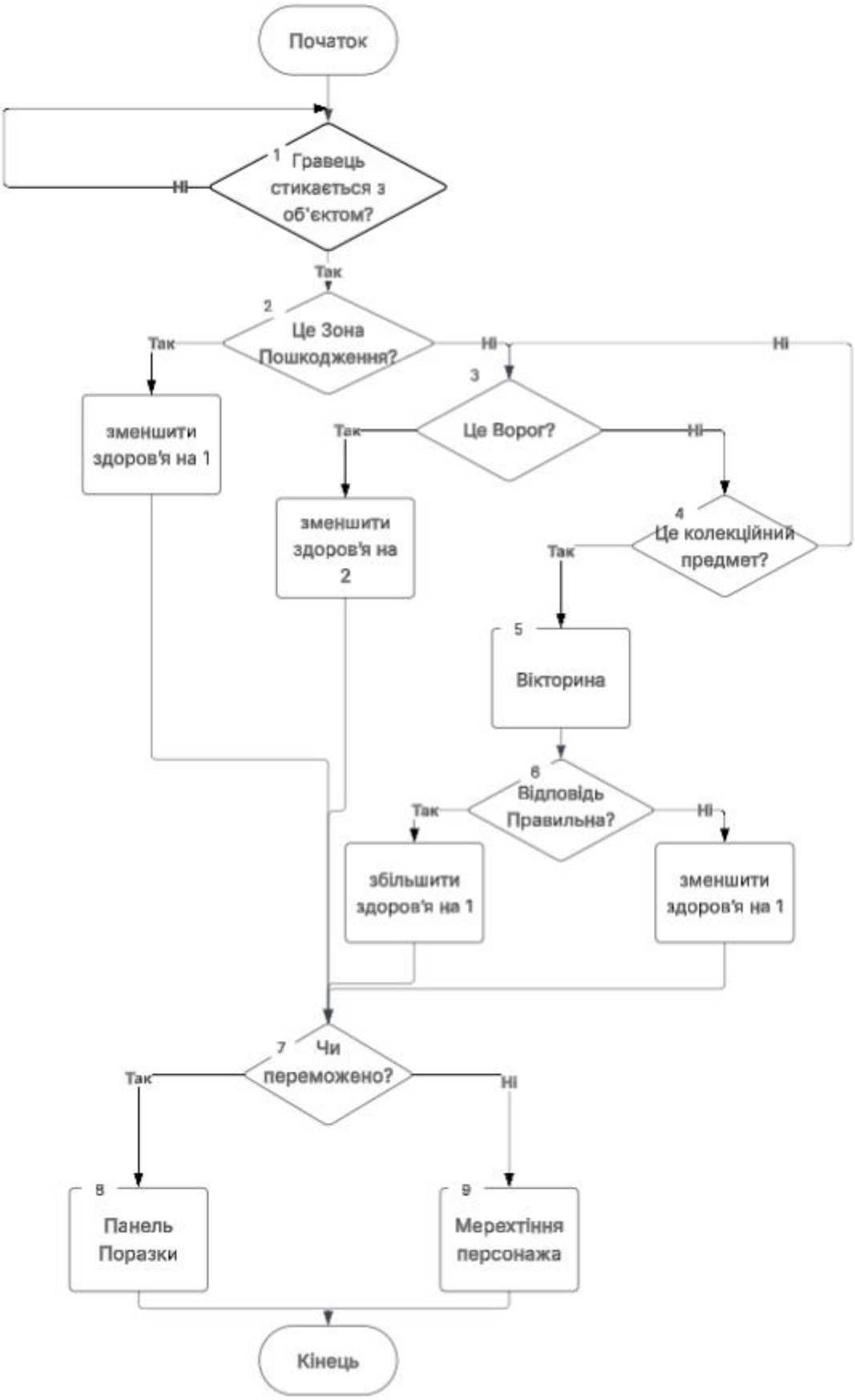

Рисунок В.7 – Алгоритм обробки ушкодження головного героя

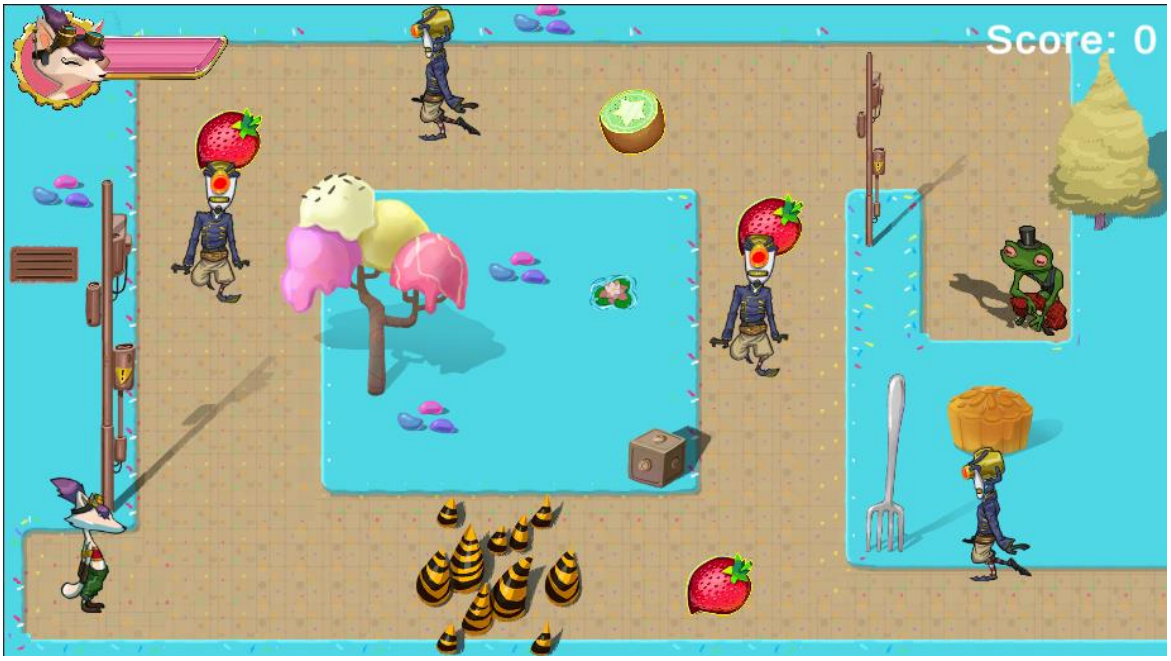

Рисунок В.8 – Вигляд вікна гри MathQuest

Рисунок В.9 – Вигляд вікна гри під час розв’язування задачі

Рисунок В.10 – Відповідь на питання у навчальному режимі

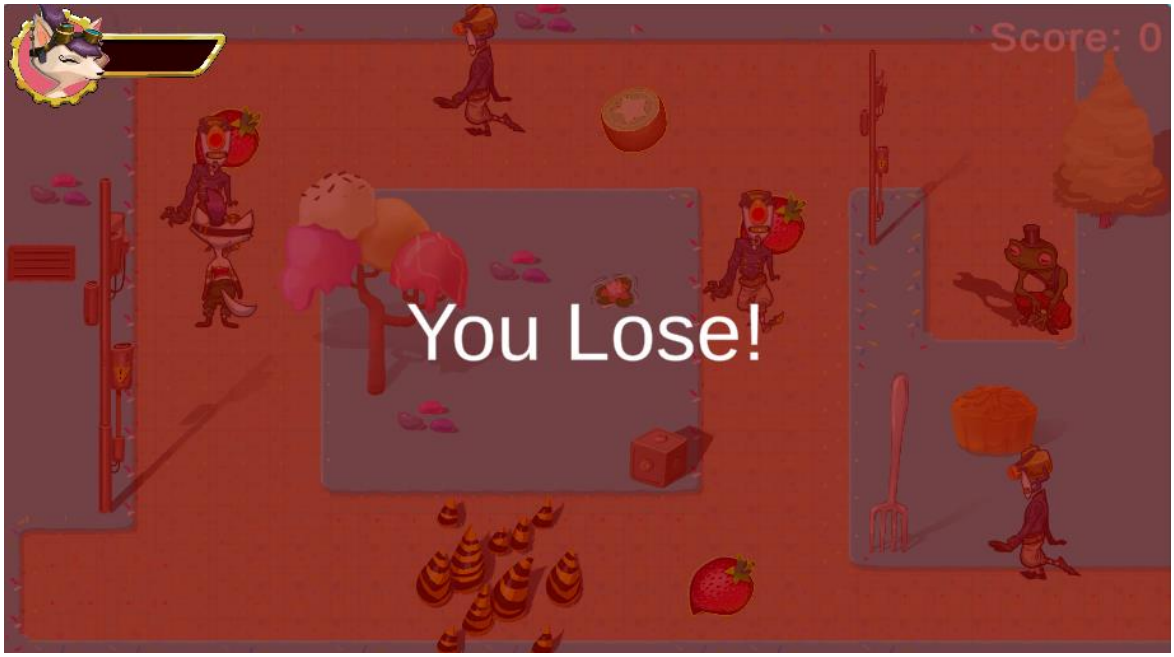

Рисунок В.11 – Вигляд вікна завершення гри у випадку поразки

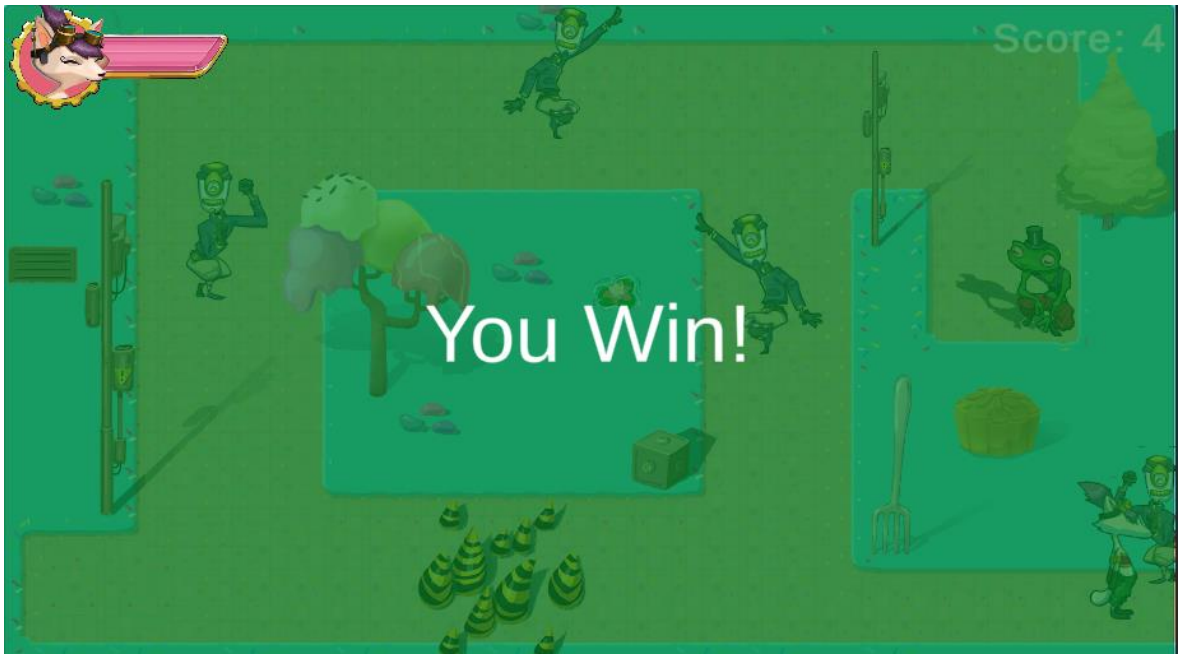

Рисунок В.12 – Вигляд вікна завершення гри у випадку перемоги

ДОДАТОК Г (довідниковий)

Інструкція користувача

Щоб відкрити гру потрібно відкрити файл MathQuest.unity у теці файлів гри (рис. Г.1).

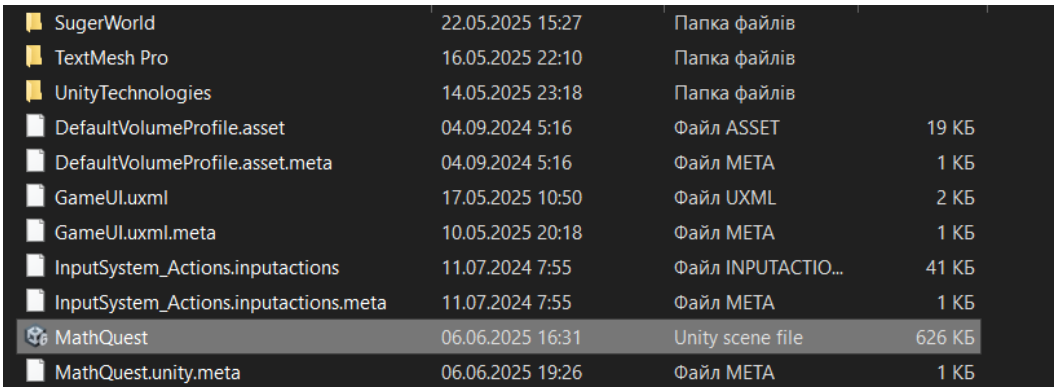

Рисунок Г.1 – Тека файла гри

Після запуску гри спочатку з’являється вікно вибору рівня складності: оберіть «Easy», «Medium» або «Hard», після чого починається сам геймплей (рис.Г.2).

Рисунок Г.2 – Взаємодія з вікном вибору складності

Керування персонажем можливе за допомогою клавіш WASD, вистріл – клавіша C (рис. Г.3).

Рисунок Г.3 – Вигляд вікна гри під час вистрілу персонажу

Коли ви підходите достатньо близько до колекційного предмету, автоматично з’явиться діалогове вікно завдання (див. рис. Г.4).

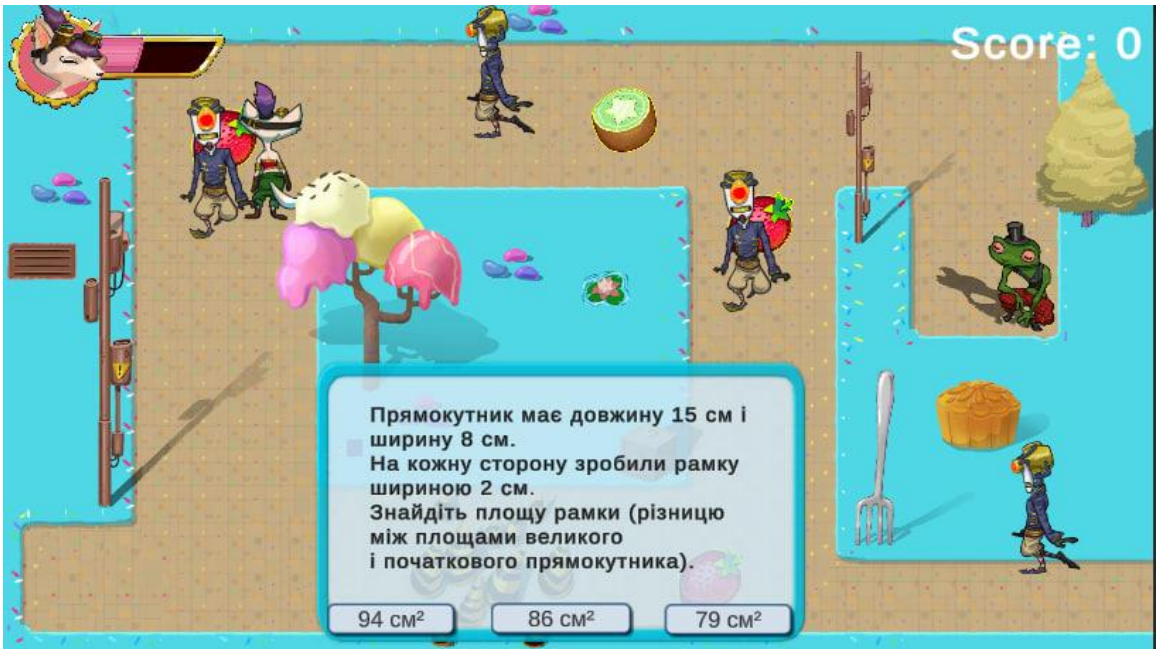

Рисунок Г.4 – Вигляд вікна гри під час розв’язування задачі

Ваша мета – дійти до фінальної точки та зібрати якомога більше очок(див. рис. Г.5).

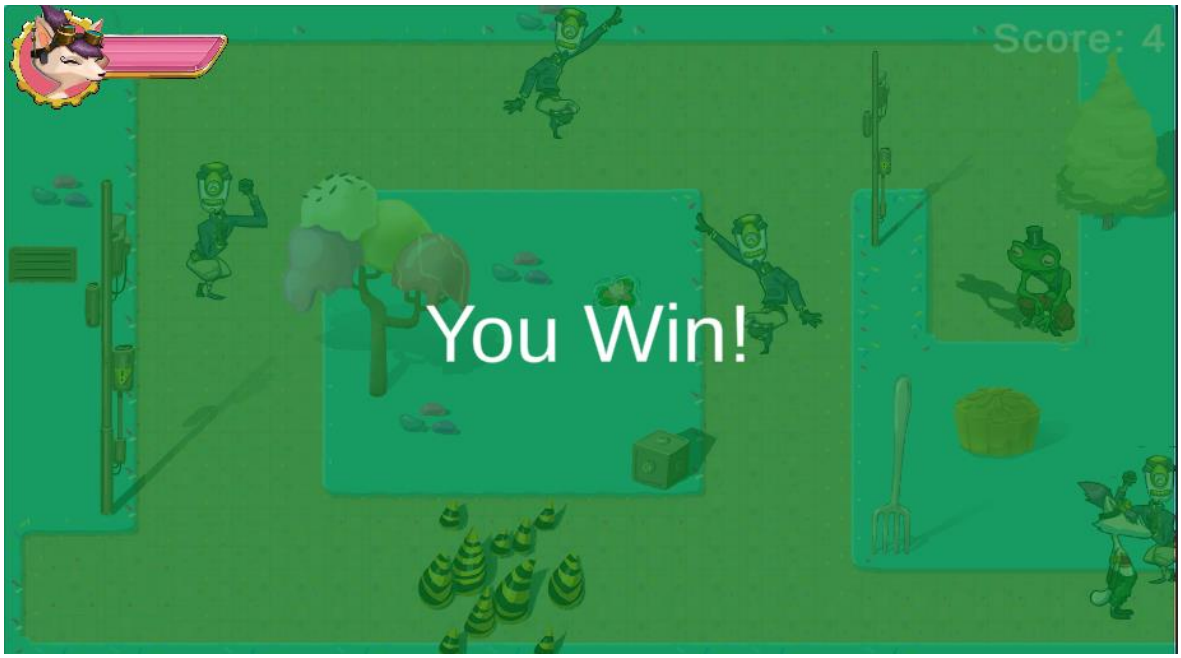

Рисунок Г.5 – Вигляд вікна завершення гри у випадку перемоги