Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Єровенко Поліни Сергіївни*
(ПІБ)

академічної групи *122-20-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка інтелектуальної платформи для персоналізованого вивчення мови програмування з використанням технології Godot*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи				
розділів:				
спеціальний	*проф. Мещеряков Л.І.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-1* *Єровенко Поліни Сергіївни*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка інтелектуальної платформи для персоналізованого вивчення мови програмування з використанням технології Godot*

затверджена наказом ректора НТУ «ДП» від *23.05.2024* № *469-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***01.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***06.06.2024 р.***

Завдання видав _____ *проф. Мещеряков Л.І.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Єровенко П.С*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *25.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 91 с., 62 рис., 3 дод., 21 джерел.

Об’єкт розробки: інтерактивна платформа-гра для персоналізованого вивчення мови програмування – C#.

Мета кваліфікаційної роботи: розробка та впровадження інтелектуальної платформи, яка допоможе користувачам в освоєнні мови програмування C# через ігровий процес, що посприяє більш глибокому та персоналізованому розумінню матеріалу.

Вступ присвячений важливості вивчення програмування для студентів та молоді, також підіймається проблема ефективного вивчення мов програмування, уточняється постановка завдання, виконується аналіз аналогічних існуючих рішень, окрім цього аналізуються переваги використання інтерактивної навчальної платформи для вивчення C# та конкретизується мета кваліфікаційної роботи, базуючись на обґрунтуваннях актуальності теми.

У першому розділі досліджено нюанси навчання програмуванню, розроблена постановка завдання, вказано використані технології і поверхнево описано можливості користувачів-гравців.

У другому розділі описується в подробицях призначення платформи, наводиться опис алгоритму та структури програми, описується робота програми від запуску до завершення.

В економічному розділі з’ясовано рівень складності проекту, визначено трудомісткість розробленої інтелектуальної платформи, обчислено вартість виконання даного продукту, а також розраховано час на створення.

Практичне значення роботи полягає в розробці інтерактивної платформи-гри, за допомогою якої студенти та просто зацікавлені особи можуть вивчати C# через ігровий процес. Користувачі проходять завдання та тести по рівнях, що допомагає їм краще зрозуміти мову програмування та оволодіти практичними навичками, окрім цього сюжет та тематика гри заохочують користувачів проходити доходити до фіналу.

Актуальність платформи обумовлюється великим попитом молоді вивчати базове програмування не витрачаючи на це багато часу, енергії та коштів, а також необхідністю створення ефективного та захоплюючого рішення для навчання. Платформа-гра допомагає користувачам не тільки вивчити мову програмування, а і підвищити мотивацію через ігровий процес.

Список ключових слів: ІНТЕЛЕКТУАЛЬНА ПЛАТФОРМА, ІГРОВИЙ ДОДАТОК, 2D, НАВЧАННЯ, GODOT, C#, ПРОГРАМУВАННЯ, КОРИСТУВАЧ.

ABSTRACT

Explanatory note: 91 p., 62 figures, 3 app., 21 sources.

Object of development: an interactive platform-game for personalized learning of the programming language – C#.

The purpose of the qualification work: development and implementation of an intelligent platform that will help users learn the C# programming language through a game process, which will contribute to a deeper and personalized understanding of the material.

The introduction is dedicated to the importance of learning programming for students and young people, the problem of effective learning of programming languages is also raised, and the task statement is clarified, in addition, the advantages of using an interactive learning platform for learning C# are analyzed and the purpose of the qualification work is specified, based on the justifications of the relevance of the topic.

In the first section, the nuances of learning to program are studied, the formulation of the task is developed, the technologies used are indicated, and the possibilities of user-players are superficially described.

The second section describes in detail the platform's purpose, analyzes similar existing solutions, describes the algorithm and structure of the program, and describes the program's operation from launch to completion.

In the economic section, the level of complexity of the project is clarified, the labor intensity of the developed intelligent platform is determined, the cost of implementing this product is calculated, and the time for creation is also calculated.

The practical significance of the work lies in the development of an interactive game platform, with the help of which students and simply interested persons can learn C# through the game process. Users pass tasks and tests by levels, which helps them better understand the programming language and master practical skills, in addition, the story and theme of the game encourage users to pass and reach the final.

The relevance of the platform is due to the great demand of young people to learn basic programming without spending a lot of time, energy, and money on it, as well as the need to create an effective and exciting learning solution. The game platform helps users not only to learn the programming language, but also to increase motivation through the game process.

List of keywords: INTELLIGENT PLATFORM, GAMING APPLICATION, 2D, LEARNING, GODOT, C#, PROGRAMMING, USER.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

C# – це об’єктно орієнтована мова програмування, розроблена компанією Microsoft для створення різноманітних програм, що працюють на платформі .NET.

Godot – це вільний і відкритий ігровий движок, що використовується для створення 2D та 3D ігор, який підтримує різні платформи та мови програмування, включаючи C#.

Node – це основний будівельний блок у Godot, що є елементом сцени, який може містити логіку та дані.

HUD – складова графічного інтерфейсу.

GUI – графічний інтерфейс користувача.

Control – це базовий клас для всіх елементів користувальницького інтерфейсу (UI) у Godot, таких як кнопки, панелі та текстові поля.

Sprite2D – це вузол Godot, який використовується для відображення 2D-графіки, таких як спрайти.

GameDev – ігрова розробка.

IDE – інтегроване середовище розробки.

TSCN – файл сцени в Godot.

OGG – формат аудіо файлу.

GPU – графічний процесор

ЗМІСТ

ВСТУП

Розробка інтелектуальної платформи для персоналізованого вивчення мови програмування C# є актуальною задачею в сфері освіти та підготовки спеціалістів по програмуванню. В сучасних умовах ринку інформаційних технологій, що швидко розвивається, кваліфіковані програмісти стають необхідними та цінними робітниками в різних областях. Ефективне навчання програмуванню, особливо з використанням інтерактивних та ігрових методів, сприяє глибшому розумінню матеріалу, кращому засвоєнню знань та підвищує мотивацію учнів.

Метою даної кваліфікаційної роботи є створення інтерактивної платформи-гри, яка допоможе користувачам в засвоєнні мови програмування C# через ігровий процес. Навчальна гра, розроблена в рамках цієї роботи, створена для студентів, молодих спеціалістів або просто для любителів IT індустрії, які хочуть вивчити основи C# в захоплюючій та інтерактивній формі. Платформа може бути використана в освітніх закладах, таких як школи та університети, а також для самостійного вивчення програмування.

Актуальність даної теми обумовлена високою потребою в кваліфікованих спеціалістах у програмуванні та прагненням до покращення методів навчання. На мою думку, традиційні методи викладання програмування часто не враховують індивідуальні особливості та рівень підготовки учнів, що знижує ефективність навчання. Інтерактивні навчальні платформи, як та, що розроблена в рамках даної роботи, пропонують новий підхід, який поєднує теоретичне навчання з практичними завданнями в ігровій формі, сприяючи більш глибокому засвоєнню матеріалу.

Дана інтерактивна платформа-гра створена у стилі середньовіччя, оскільки провівши аналіз запитів на ігрових платформах, було з’ясовано, що останнім часом попит на ігри в ретро-стилі на тематику середньовіччя підвищується, тому було вирішено поєднати особисті вподобання з актуальним вирішенням

проблеми. Для досягнення поставленої мети кваліфікаційної роботи необхідно вирішити наступні задачі:

– розробка структури та дизайну начальної гри, яка включає в себе два рівня – два поверхи замку, на кожному з яких є кімнати, через які проходить гравець та поступово вивчає певні теми з C# за допомогою теоретичних відомостей та практичних завдань;

– реалізація інтерактивних елементів, таких як текстові та візуальні завдання, вибір правильних відповідей, заповнення пропусків в коді, введення текстових відповідей та взаємодія з теоретичними оголошеннями та елементами замку;

– забезпечення персоналізації навчання шляхом обрання рівнів: перший рівень для базового вивчення C# без обтяження важкими темами, а другий рівень для більш досвідчених знавців цієї мови програмування, тому і теми там відповідно складніші;

– впровадження музичного супроводження для більшого залучення гравця та для створення відповідної атмосфери гри;

– тестування та налагодження платформи для забезпечення її стабільної роботи та зручності користування.

В результаті виконання задач вище створена ефективна навчальна платформа, яка допоможе користувачам оволодіти мовою програмування C# та підготуватися до професійної діяльності в сфері інформаційних технологій.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

1.1.1 Загальні положення

В сучасних умовах цифрових проривів, трансформації та інформаційних технологіях, що розвиваються, програмування є одним з найбільш ключових та затребуваних навиків. Кваліфіковані програмісти необхідні в різноманітних сферах, таких як розробка програмного забезпечення, веб-розробка, ігрова індустрія, фінансові технології та багато іншого. Мова програмування C# є однією з основних інструментів для створення застосунків на платформі .NET, а також використовується в розробці ігор з використанням таких рушіїв, як Godot Engine та Unity.

На даний момент існує багато навчальних платформ та ресурсів, призначених для вивчення програмування, таких як Codecademy, Coursera, Udemy, Udacity та інші. Але більшість з них не пропонують персоналізованого підходу та достатньої інтерактивності, що знижує мотивацію учнів та сповільнює глибоке вивчення матеріалу. Традиційні методи навчання, засновані на лекціях та практичних заняттях, часто не можуть повністю задовільнити потреби студентів, які віддають перевагу більш інтерактивним та захоплюючим формам навчання.

Розробка інтерактивної навчальної платформи-гри для персоналізованого вивчення мови програмування C# направлена на вирішення цих проблем. Платформа пропонує ігровий підхід до навчання, де користувачі виконують завдання, проходячи через різні кімнати двох рівнів замку. Цей метод дозволяє покращити залученість та мотивацію студентів, пропонуючи їм можливість вчитись через практику та інтерактивні завдання.

1.1.2 Аналіз аналогів

Існує декілька платформ, які пропонують вивчення сови програмування C#. Codeacademy пропонує базові курси, але їхній інтерфейс та зміст не адаптується під рівень знань користувача [9].

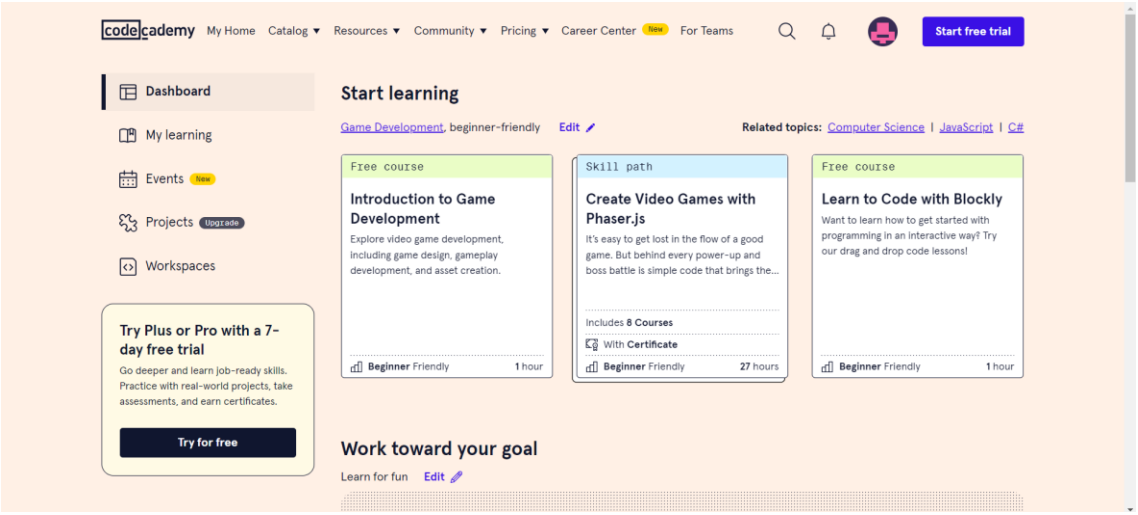

Рис. 1.1 Стартова сторінка сайту Codeacademy для GameDev

Coursera та Udacity пропонують курси, розроблені провідними університетами та компаніями, але вони також страждають від нестачі персоналізації та інтерактивності [21]. В цих платформах відсутній елемент гейміфікації, який зазвичай може значно підвищити мотивацію студентів [10].

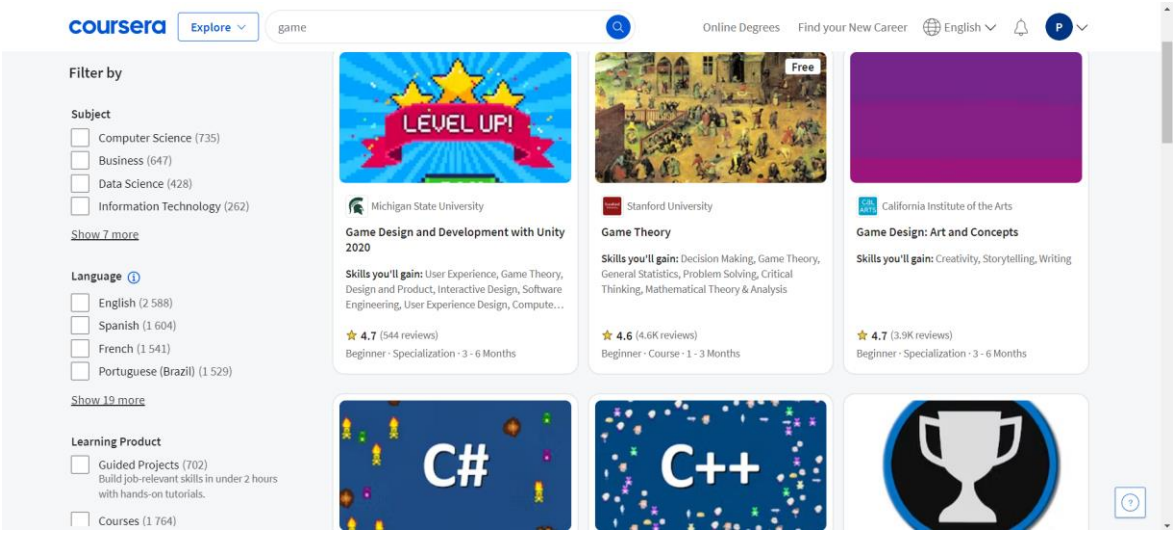

Рис. 1.2 Стартова сторінка сайту Coursera під час пошуку за запитом «Game»

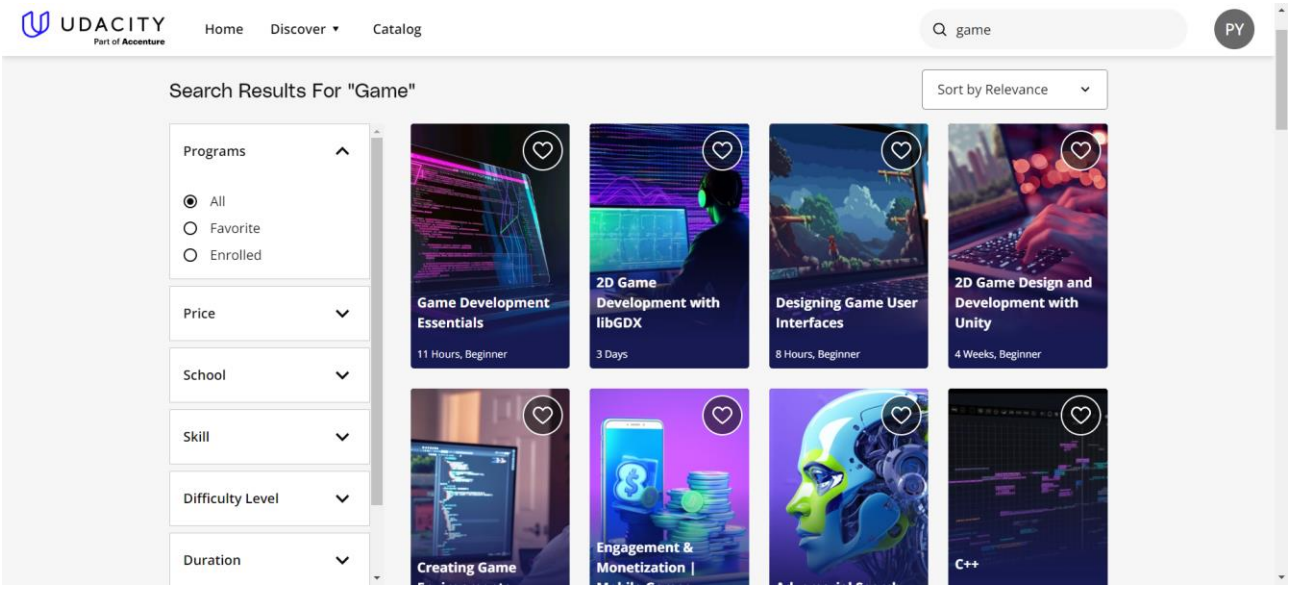

Рис. 1.3 Стартова сторінка сайту Udacity під час пошуку за запитом «Game»

Платформи, як Unity Learn, пропонують вивчення C# в контексті розробки ігор, але вони орієнтовані на користувачів, які вже мають досвід, тому не передбачають структурований підхід до вивчення основ мови програмування [20].

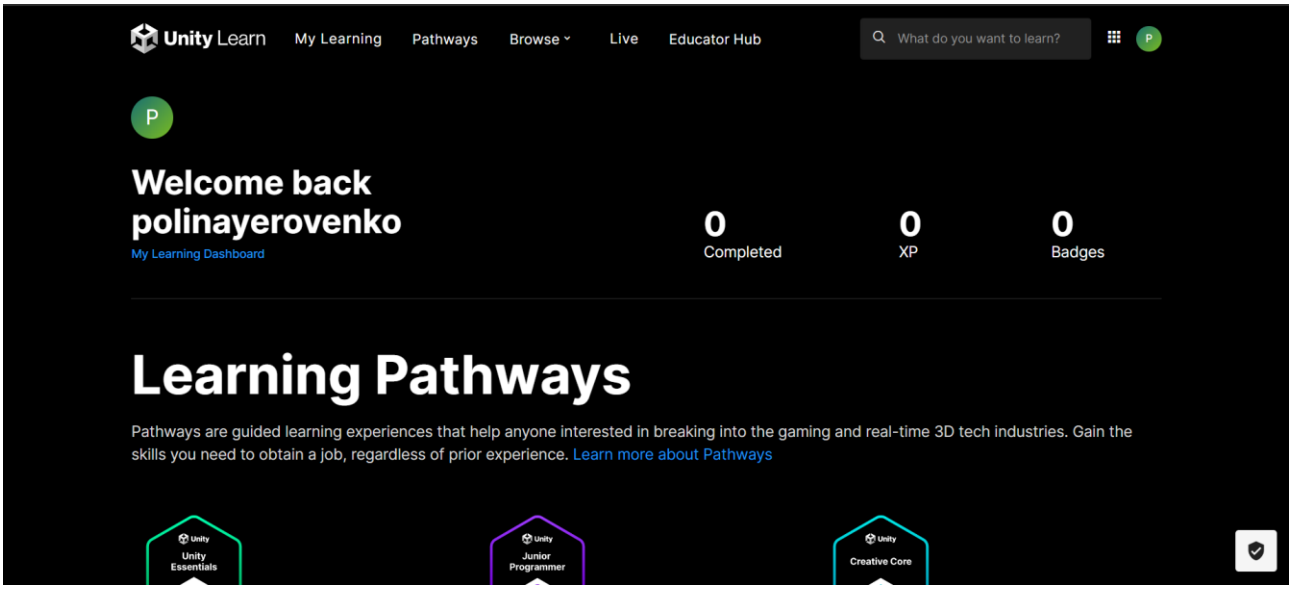

Рис. 1.4 Головна сторінка сайту Unity Learn

Таким чином, існує явна прогалина в освітніх ресурсах, які могли б запропонувати персоналізований та інтерактивний підхід до вивчення C#.

1.2. Призначення розробки та галузь застосування

Інтерактивна навчальна платформа для персоналізованого мови програмування C# являє собою систему, яка забезпечує взаємодію з користувачем в режимі реального часу; підхід до навчання, який враховує індивідуальні особливості та потреби учнів; застосування ігрових елементів в неігрових процесах для підвищення мотивації та залучення.

Основними причинами виникнення необхідності розробки є нестача мотивації та заохочення студентів в процесі традиційного навчання програмування, а також відсутність персоналізації в існуючих платформах для вивчення програмування.

Щодо області застосування, то розроблена платформа може використовуватися викладачами для інтерактивного навчання студентів основам програмування мовою C#, окрім цього будь-який бажаючий може використовувати платформу для самостійного вивчення програмування. Платформа-гра може бути включена в програми курсів та тренінгів, щоб дати учасникам можливість практичного вивчення C# [2].

1.3. Підстави для розробки

Згідно з навчальним планом та відповідно до навчальної програми та графіка навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу. Тема роботи узгоджується з її науковим керівником, кафедрою, а також затверджується наказом ректора. Таким чином, підставами для виконання кваліфікаційної роботи є:

- освітня програма 122 Комп’ютерні науки;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 469-с від 23.05.2023 р;

– завдання на кваліфікаційну роботу на тему «Розробка інтелектуальної платформи для персоналізованого вивчення мови програмування з використанням технології Godot».

1.4. Постановка завдання

Задача кваліфікаційної роботи полягає в створенні інтелектуальної платформи-гри для персоналізованого вивчення мови програмування, а саме C#, з використанням технології Godot для забезпечення ефективного та захоплюючого вивчення програмування через ігровий процес [11].

Додаток має реалізувати наступні функції:

1. Введення імені користувача.
2. Доступ до інтерактивних даних по програмуванню.
3. Виконання різних типів завдання:

1) заповнення пропусків в коді;
2) введення текстових відповідей;
3) вибір правильної відповіді серед запропонованих опцій.

4. Моніторинг прогресу користувача:

1) відстеження виконання завдань;
2) зберігання результатів виконання.

5. Наявність підказок у вигляді маркеру послідовності.
6. Інтерактивні елементи ігрового процесу:

1) переміщення лицаря по замку;
2) взаємодія з об’єктами (знаки, двері, стіни).

Для досягнення поставлених цілей необхідно:

– намалювати основне ігрове поле, де будуть відбуватись події;

– написати програмний код, розробити структуру та логіку гри роботи додатку мовою C# в середовищі Godot [16];

– наповнити додаток учбовими матеріалами, завданнями по програмуванню, музикою та різними предметами, з якими персонаж не може взаємодіяти;

– протестувати платформу для забезпечення надійної роботи.

Ці етапи дозволять створити повноцінну навчальну платформу, яка буде використовуватись для інтерактивного вивчення мови програмування C# через ігровий процес [1].

1.5. Вимоги до програми або програмного виробу

1.5.1 Вимоги до функціональних характеристик

Кінцева версія розробленої платформи має відповідати наступним вимогам:

– мати інтерактивне та інтуїтивно зрозуміле головне меню, яке дозволить користувачу ввести своє ім’я та обрати рівень для проходження;

– відображати навчальні матеріали з мови програмування C# у вигляді текстів та прикладів коду, які будуть представлені як інтерактивні знаки та питання;

– забезпечувати можливість взаємодії з об’єктами у грі, такими як оголошення з теоретичними матеріалами та питання для перевірки знань;

– реалізувати систему перевірки відповідей на питання, яка включає різні типи питань: заповнення пропусків в коді, вибір правильної відповіді та введення короткої відповіді;

– забезпечити перехід між рівнями по мірі успішного виконання завдань та відповідей на питання;

– вмикати аудіосупровід, такий як фонова музика та звукові ефекти, які активуються при ході та взаємодії з об’єктами.

Щодо переліку функцій платформи, то до них відноситься:

– інтерактивне меню з можливістю вибору рівня;

– показ та приховування інформаційних панелей при взаємодії з об’єктами;

– реалізація системи переміщення персонажа по всьому ігровому полю;

– система взаємодії з навчальними знаками, які відображають текст та приклади коду;

– система питань та відповідей, яка перевіряє знання гравця по темі та дає зворотній зв’язок;

– перехід на наступний рівень після виконання всіх завдань на поточному рівні;

– аудіосупрвід для створення атмосферного ігрового досвіду.

Всі ці функціональні характеристики забезпечать інтерактивність та навчальну цінність гри, а також підтримають інтерес гравця через різноманітні ігрові механіки та елементи взаємодії.

1.5.2 Вимоги до інформаційної безпеки

Забезпечення інформаційної безпеки в розроблюваній платформі є вкрай важливим завданням, необхідно бути впевненим, що дотримані всі вимоги для захисту даних, а саме.

Ключові вимоги до інформаційної безпеки поєднують:

– захист даних користувача від несанкціонованого доступу;

– забезпечення точності та повноти даних під час їх передачі та збереження;

– гарантія того, що інформація доступна користувачам в будь-який час.

Щодо режиму роботи, немає необхідності авторизації користувачів, платформа міститься в єдиному загальному файлі, і користувач не повинен реєструватись або створювати акаунт, достатньо просто ввести ім’я, тому в цьому аспекті немає ніяких специфічних умов до інформаційної безпеки користувача.

Умовами забезпечення інформаційної безпеки є:

– автономна робота, тобто платформа функціонує без необхідності постійного мережного зв’язку, що зменшує ризики, пов’язані з мережевими атаками;

– своєчасне оновлення, а саме встановлення всіх оновлень для захисту від можливих потенційних вразливостей;

– відсутність реєстрації, тобто користувач може використовувати платформу без необхідності реєструватись та створювати особистий кабінет;

– закритий вихідний код, тобто код платформи має бути закритим для попередження вивчення та експлуатації вразливостей.

1.5.3 Вимоги до складу та параметрів технічних засобів

Для запуску та коректної роботи розроблюваної платформи необхідні наступні технічні засоби:

– наявність ноутбуку або комп’ютеру;

– клавіатура та мишка для керування персонажем та взаємодії з об’єктами у грі;

– операційна система Windows 7/8/10/11 (64-bit)

– процесор (CPU) Intel Core i3 або краще з частотою 1 ГГц чи вище;

– оперативна пам’ять (RAM) 4 GB чи краще;

– відеокарта NVIDIA з 1 GB пам’яті чи краще;

– жорсткий диск HDD або SSD об’ємом 10 GB чи більше для установки та збереження файлів гри.

1.5.4 Вимоги до інформаційної та програмної сумісності

Щоб розроблена платформа-гра працювала ефективно нам необхідно, щоб програмне забезпечення було сумісним з різноманітними пристроями та операційними системами, а також відповідало наступним вимогам:

Інформаційна сумісність:

– операційні системи Windows 7 чи вище (64-bit), Linux, MacOS;

– ноутбук чи комп’ютер;

– жорсткий диск HDD або SSD об’ємом 10 GB чи більше для установки та збереження файлів гри.

Програмна сумісність:

– платформа-гра розроблена на основі ігрового рушія Godot;

– основною мовою програмування обраний C#;

– використовуються вбудовані можливості Godot для створення користувальницького інтерфейсу;

– підтримка форматів аудіофайлів для використання фонової музики для звукових ефектів (я використовувала формат .ogg);

– програма має бути встановлена на систему за допомогою інсталятора, тому оновлення відбувається через перевстановлення програми.

Ці вимоги забезпечують сумісність гри з різноманітними операційними системами та пристроями, а також дозволять користувачам легко встановлювати та оновлювати додаток.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

В результаті виконання даної кваліфікаційної роботи була розроблена інтерактивна навчальна платформа-гра, напрямлена на вивчення мови програмування C#. Платформу створено з використанням ігрового рушія Godot та призначено для використання на персональних комп'ютерах чи ноутбуках. В ході роботи додатку підтримується освітній процес, а також забезпечується зрозумілий та захоплюючий ігровий процес для користувачів. Платформа-гра сприяє покращенню знань в програмуванні та розвитку логічного мислення у користувачів [12].

Основні цілі розробленого додатку:

- платформа-гра пропонує користувачам можливість вивчити основи програмування мовою C# через інтерактивні завдання та теоретичні матеріали;
- забезпечується цікавий та зрозумілий ігровий процес в стилі середньовіччя, який мотивує користувачів проходити завдання та поглиблювати свої знання;
- платформа направлена на розвиток аналітичного та логічного мислення, що є важливим аспектом навчання програмуванню.

Основні функції ігрового додатку:

- керування персонажем, тобто користувач може переміщувати головного героя по намальованому ігровому полю, використовуючи клавіатуру;
- в процесі гри користувачі взаємодіють з різними об'єктами, такими як навчальні оголошення та питання, отримуючи доступ до теоретичних матеріалів та практичним завданням;
- платформа-гра включає різноманітні типи питань з програмування, такі як заповнення пропусків у коді, вибір правильної відповіді з декількох

варіантів і введення короткої відповіді. Відповіді перевіряються на правильність, а користувачу надається зворотній зв’язок;

- користувач проходить два рівні гри, кожен рівень – це поверх, поверхи умовно поділені на кімнати-секції, і в кожній кімнаті вивчається певна тема для вивчення C#. Якщо користувач відповідає на всі питання з певної кімнати правильно, то йому надається доступ до наступної кімнати, це реалізовано за допомогою дверей, а якщо ж користувач відповідає неправильно, то двері до наступної секції залишаються закритими. Після того, як на першому поверсі користувач відкрив всі кімнати, то він автоматично переходить на наступний поверх, тобто на наступний рівень, і після проходження він повертається в головне меню.
- гра супроводжується фоновою музикою та звуковими ефектами, які створюють атмосферу та занурюють користувача в ігровий процес.

Ці функції та операції забезпечують захоплююче та ефективне навчання мові програмування C#, підтримуючи інтерес користувачів та допомагаючи їм оволодіти важливими навичками програмування через ігрову форму.

2.2. Опис застосованих математичних методів

При створенні даної платформи-гри було застосовано різноманітні математичні методи, які забезпечують функціональність та взаємодію різних компонентів додатку. Ці методи допомагають при створенні логіки гри, обробці користувальницького вводу та управлінні ігровим процесом.

1. Математичні методи для керування рухом персонажа

Для керування рухом головного героя – лицаря, було використано базові математичні операції з векторами. В файлі movement.cs представлено наступні формули для руху персонажа:

```
float speed = Input.IsKeyPressed(Key.Shift) ? 4 : 2;
```

```
Velocity = new Vector2(0, 0);

if (CanMove)
{
   if (Input.IsKeyPressed(Key.D))
      Velocity += new Vector2(50 * speed, 0);
   if (Input.IsKeyPressed(Key.A))
      Velocity += new Vector2(-50 * speed, 0);
   if (Input.IsKeyPressed(Key.W))
      Velocity += new Vector2(0, -50 * speed);
   if (Input.IsKeyPressed(Key.S))
      Velocity += new Vector2(0, 50 * speed);
}
```

Тобто змінна Velocity задає поточну швидкість персонажа, і якщо персонаж може рухатись (CanMove), то швидкість змінюється в залежності від натиснутих клавіш. Якщо розбирати детальніше, то на початку кожної ітерації швидкість Velocity ініціалізується значенням (0,0), персонаж може рухатись тільки якщо змінна CanMove встановлена в true. Якщо натиснуто клавішу D (рух вправо), то збільшується компонент швидкості по осі X на (50 * speed), значення 50 тут представляє базову швидкість, помножену на змінну speed, яка може змінюватись при натисканні на клавішу Shift для прискорення. За таким самим алгоритмом відбувається і рух вліво, вперед і назад, при натисканні на клавіші A, W, S відповідно.

2. Математичні методи для перевірки колізій.

Важливою складовою гри є взаємодія персонажа з об’єктам ігрового простору, такими як знаки та двері. Для цього використано методи перевірки колізій, в файлі sign.cs застосовано формулу для перевірки відстані між персонажем та об’єктом взаємодії.

var close = Movement.Player.GlobalPosition.DistanceTo(GlobalPosition) <= 50;

Тобто функція DistanceTo рахує евклідову відстань між двома точками в ігровому просторі, такими як знаки та двері, це допомагає зрозуміти коли персонаж знаходиться досить близько до об’єкту, щоб взаємодіяти з ним. Змінна close встановлюється в true, якщо персонаж знаходиться на відстані менше або дорівнює 50 одиниць від об’єкту, що дозволяє йому взаємодіяти з потрібними ігровими об’єктами.

Отже ці приклади показують як математичні формули та логічні перевірки використовуються в ігровому коді для руху персонажа та перевірки взаємодії з об’єктами. Ці методи та формули грають ключову роль в створенні захоплюючого та пізнавального ігрового досвіду, сприяючи вивченню мови програмування C# через ігровий процес.

2.3. Опис використаних технологій та мов програмування

Для створення проекту був вибраний ігровий рушій Godot та мова програмування C# [5]. Даний вибір обумовлений наступними факторами:

- Godot – популярний ігровий рушій, який підтримує розробку як 2D, так і 3D ігор. Він пропонує потужний набір інструментів для створення ігор, включно з системою анімації, фізики, мережевих можливостей та багато іншого. Godot також підтримує різноманітні мови програмування, такі як GDScript, VisualScript та C#. Для цього проекту було обрано мову C#, оскільки вона пропонує більш високу продуктивність та можливість використання потужних бібліотек та інструментів .NET [7].

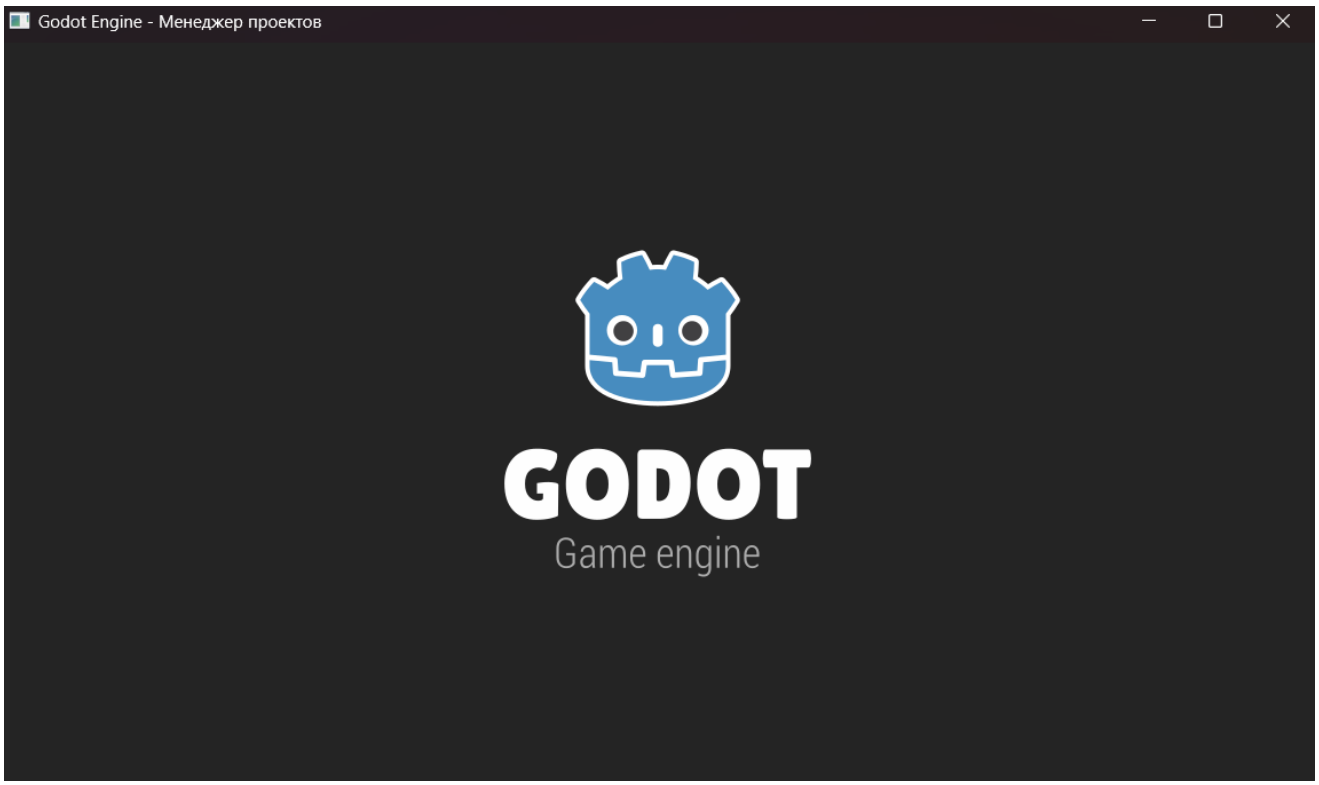

Рис. 2.1 Логотип програми Godot

- C# – мова програмування, розроблена компанією Microsoft. Вона є потужною, типізованою мовою, яка підтримує об’єктно-орієнтоване програмування. В контексті Godot, C# використовується для написання логіки гри, створення ігрових об’єктів та їх взаємодій [3].

Щодо середовища розробки, то було обрано вбудоване середовище, яке пропонує рушій Godot. Godot IDE надає всі необхідні інструменти для написання, налагодження та тестування коду, а також інтеграцію з різноманітними системами контролю версій та розширеннями, які роблять процес розробки більш продуктивним та зручним.

- Godot IDE – інтегроване середовище розробки, вбудоване в рушій Godot. Воно підтримує написання коду на різних мовах, включно C# і GDScript, та надає інструменти для візуального редагування сцен, налаштування, профілювання та управління проектом.

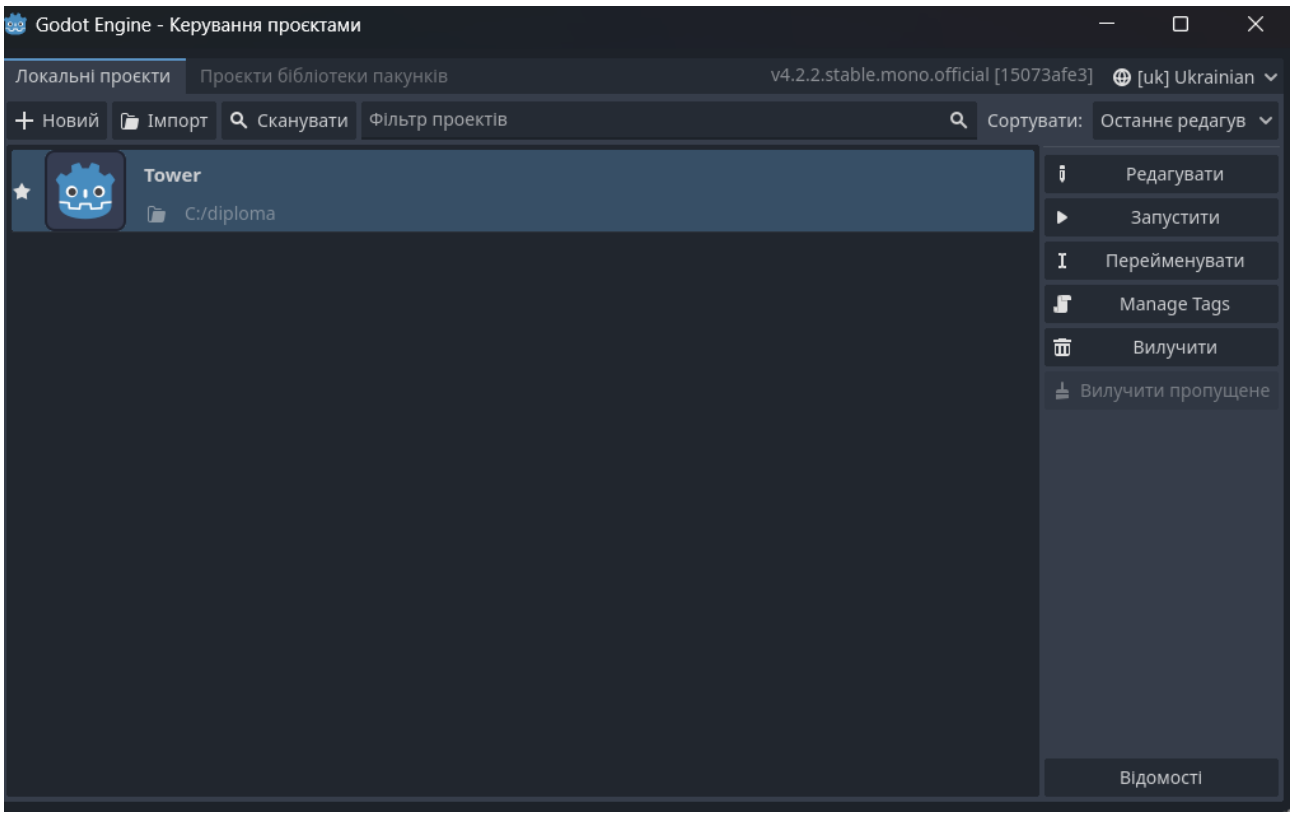

Рис. 2.2 Стартова сторінка після запуску Godot

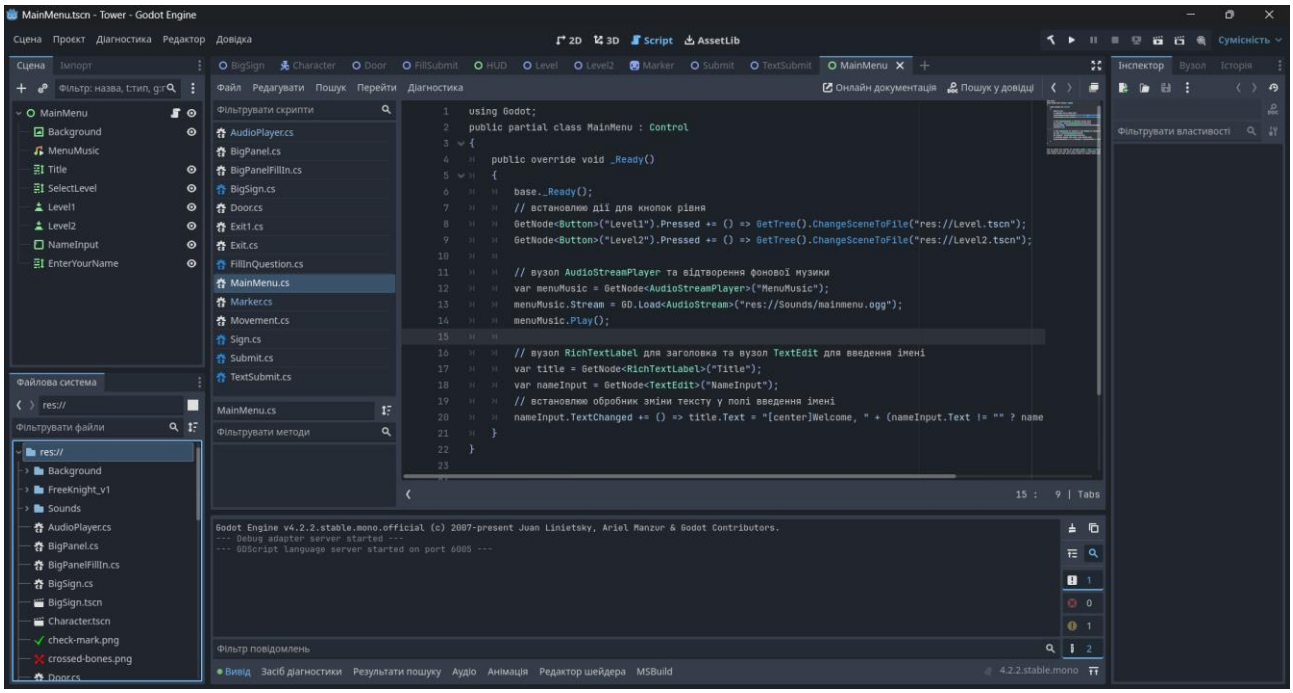

Рис. 2.3 Інтерфейс Godot при виборі редактора Script для головного меню

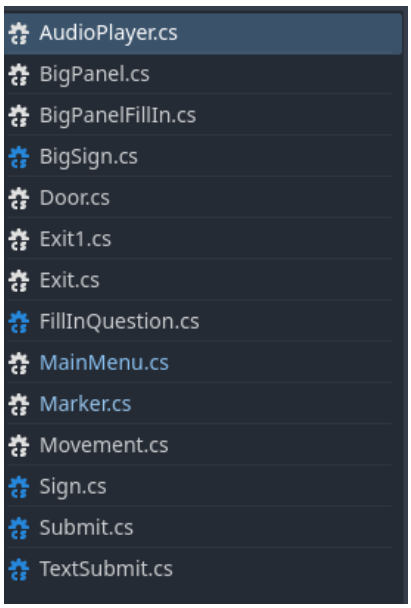

Рис. 2.4 Кількість наявних файлів в проекті

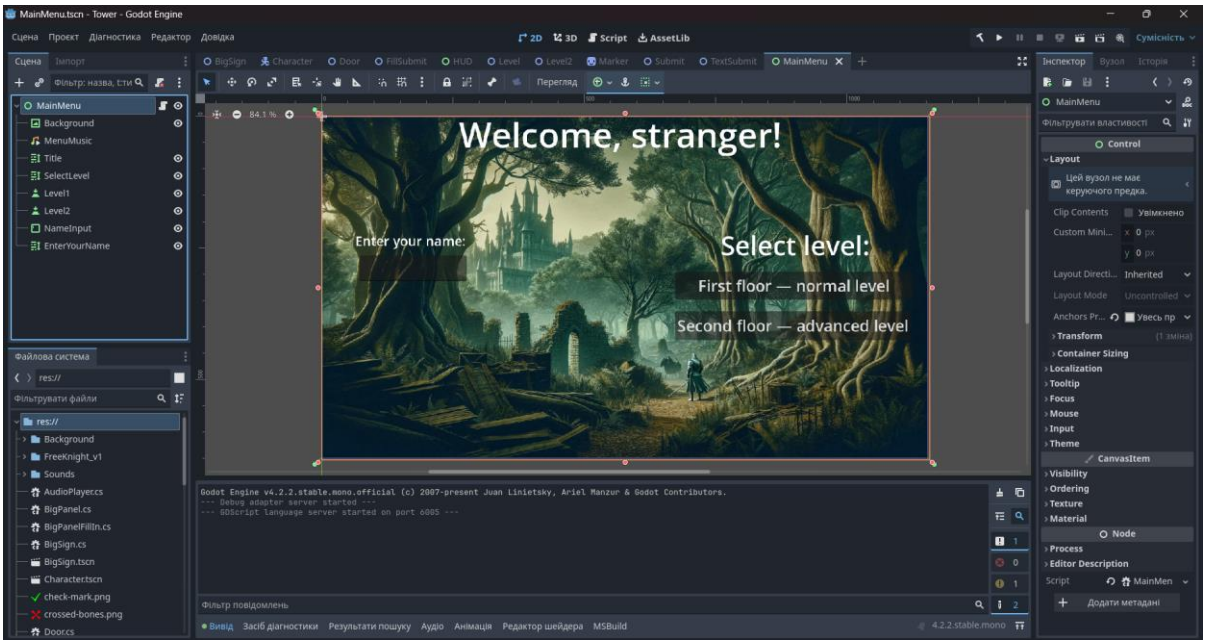

Рис. 2.5 Інтерфейс Godot при виборі редактора 2D для головного меню

Використання Godot та C# для розробки платформи-гри дозволило створити потужний та гнучкий додаток, який може бути легко модифіковано та розширено. Вибір вбудованого середовища розробки Godot IDE забезпечив зручність та ефективність написання коду.

2.4. Опис структури системи та алгоритмів її функціонування

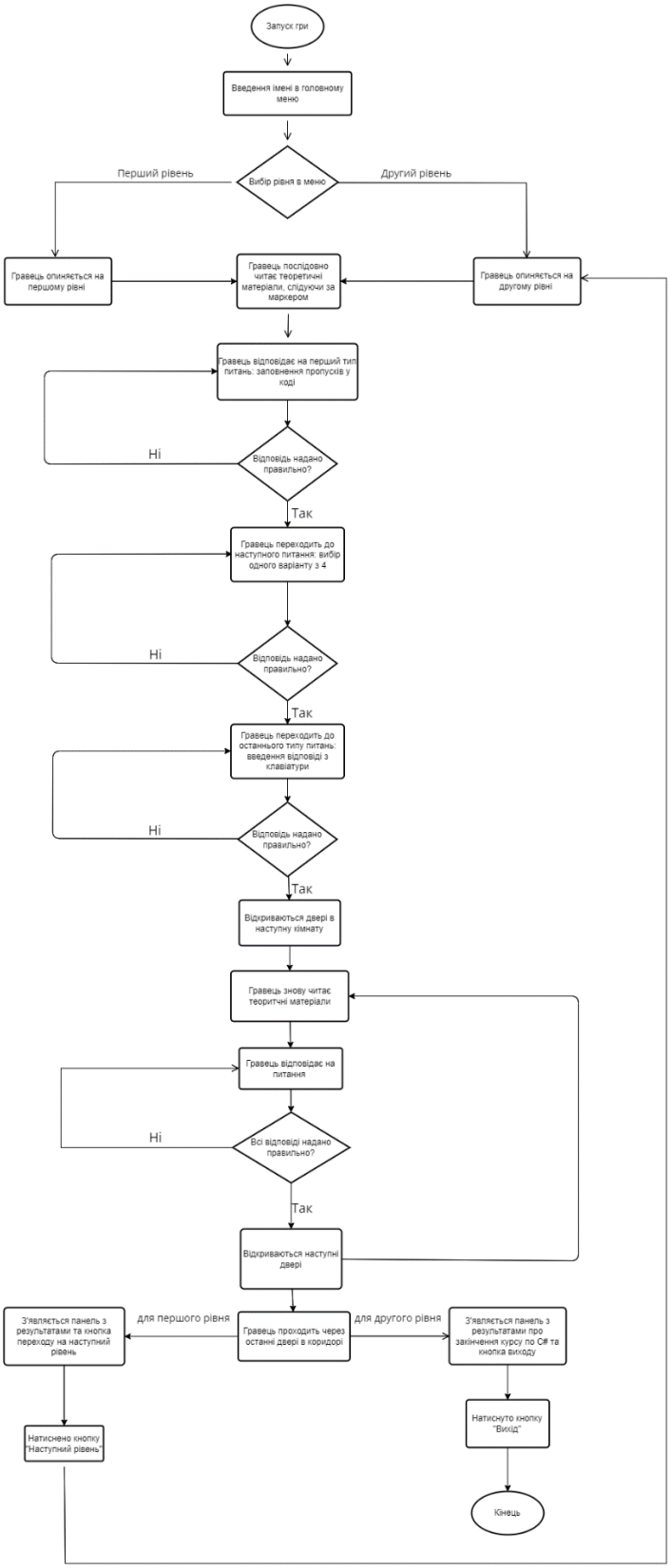

Рис. 2.6 Алгоритм роботи програми для гравця

Інтелектуальна платформа-гра розроблена на основі ігрового рушія Godot з використанням мови програмування C#, і система складається з декількох компонентів, кожен з яких виконує свою роль в забезпеченні функціональності гри [15].

1) Головне меню (MainMenu), тобто інтерфейс, що дозволяє користувачу почати гру, обрати рівні та ввести своє ім’я [8]. Компонентами головного меню є дві кнопки (Button) для вибору першого та другого рівня відповідно, а також панель (TextEdit) для введення свого ім’я.

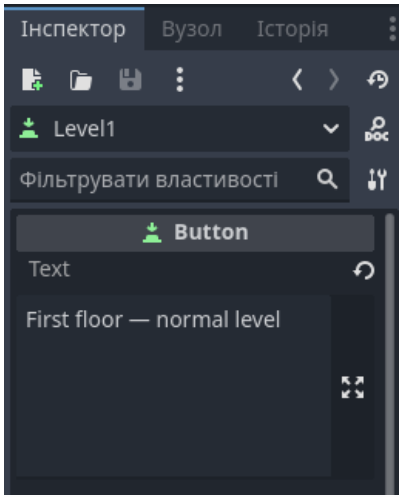

Рис. 2.7 Ознаки та назва кнопки Button для першого рівня

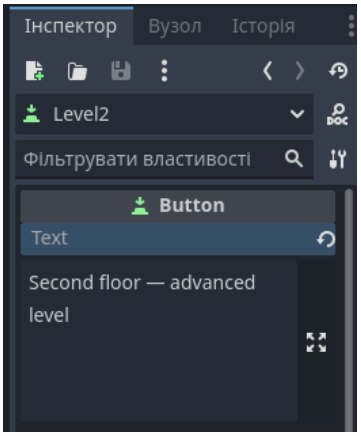

Рис. 2.8 Ознаки та назва кнопки Button для другого рівня

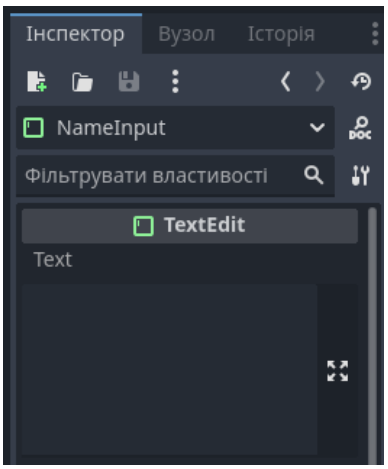

Рис. 2.9 Ознаки та назва панелі TextEdit для введення імені

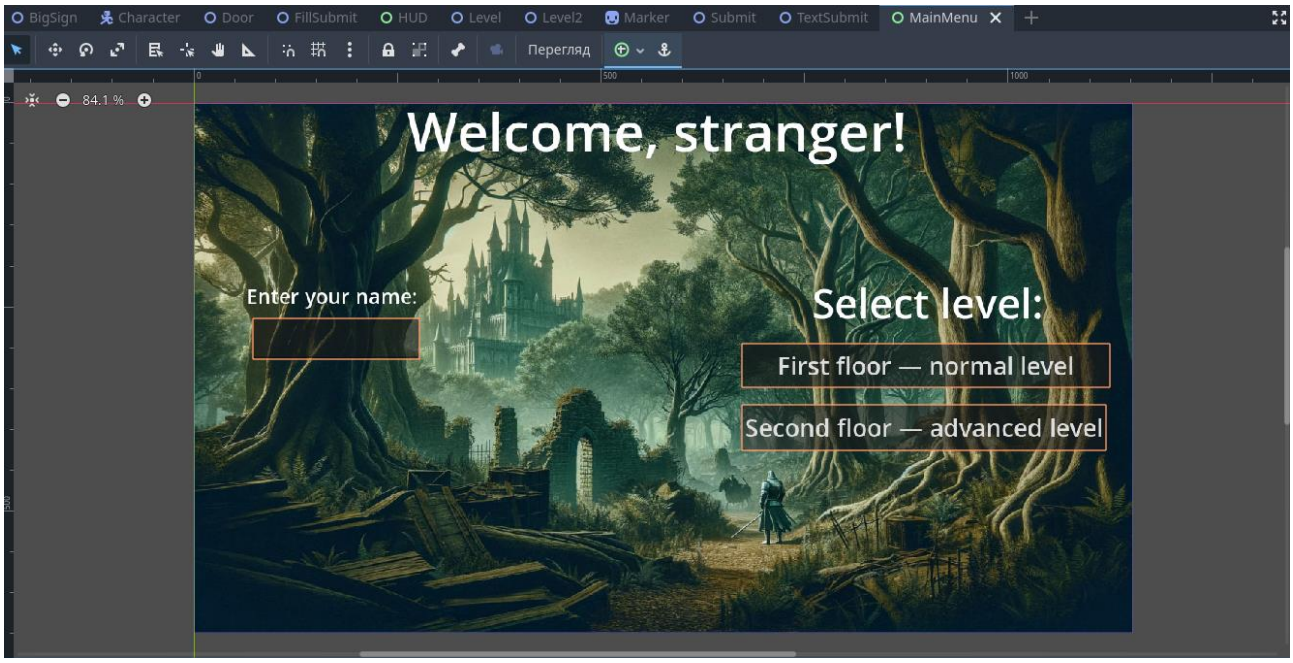

Рис. 2.10 Три виділені основні компоненти, з якими взаємодіє користувач

2) Ігрові рівні, оскільки головна ідея полягає в тому, що наш персонаж-лицар замкнений в замку, і для того, щоб йому вибратись, йому треба пройти два поверхи, тому кожен поверх представляє з себе рівень. Другий поверх є першим рівнем, тобто рівнем з базовими темами для вивчення C#, після проходження цього поверху лицар спускається на перший поверх, тобто на другий рівень, а відповідно на важчий [17]. Після проходження цього рівня відчиняються двері замку і герой може вибратись. На кожному поверсі є кімнати з теоретичними

матеріалами та питаннями для перевірки знань. Компонентами рівнів є ігровий персонаж – лицар, кімнати з теоретичними матеріалами та інтерактивні об’єкти, такі як знаки, оголошення та двері.

Для створення рівнів та взаємодії гравця з об’єктами виконано таку послідовність дій:

- додано кореневий вузол CharacterBody2D, який називається Player, щоб мати базову робочу фізику
- додано трьох «дітей»: Sprite2D, CollisionShape2D і Camera2D. Camera2D встановлюється як дочірня складова Player таким чином, щоб вона рухалась разом з гравцем. CollisionShape2D не дозволяє гравцю проходити крізь стіни, а Sprite2D обробляє зображення, було знайдено кілька безкоштовних спрайтів в інтернеті для використання;

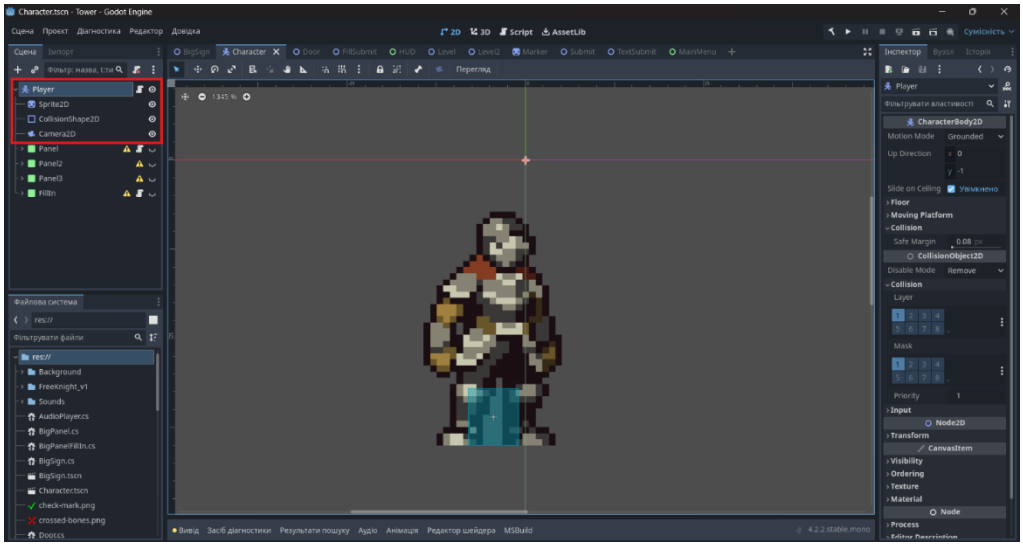

Рис. 2.11 Зовнішній вигляд Player та демонстрація залежностей вузлів в панелі зліва

- до Player додано новий скрипт – Movement.cs, в цьому скрипті визначається дві функції: _Ready і _Process. Функція _Ready виконується на початку гри, а _Process – безперервно. Створено public static змінну, яка посилається на гравця, і встановлено її в _Ready, щоб можна було отримати

доступ до вузла гравця з будь-яких скриптів без необхідності його шукати. У функції _Process перевіряється керування гравцем, тобто натискання клавіш W, A, S, D, Shift викликає відповідне переміщення вузла;

- наступним кроком додано вузол Tilemap до головної карти, це дає можливість малювати стіни та підлогу , щоб створювати структури, які хочеться [4]. Для цього також було знайдено спрайти з інтернету. Далі було створено два шари: підлогу (Floors) та стіни (Walls), щоб надати стінам форму колізії їх поміщено на окремий шар, якому надано фізику;

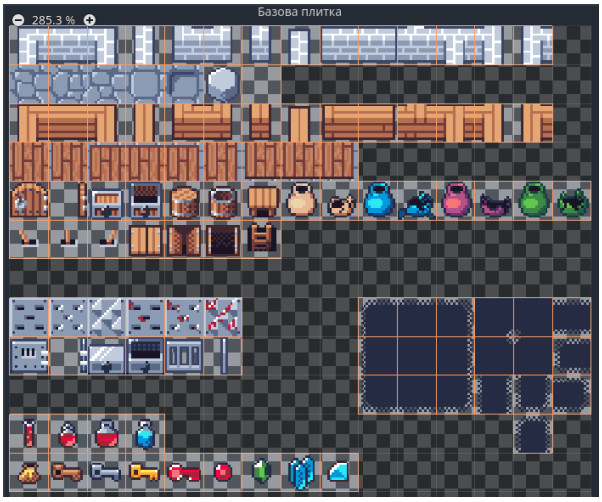

Рис. 2.12 Атлас плиток, за допомогою яких створювався ігровий простір

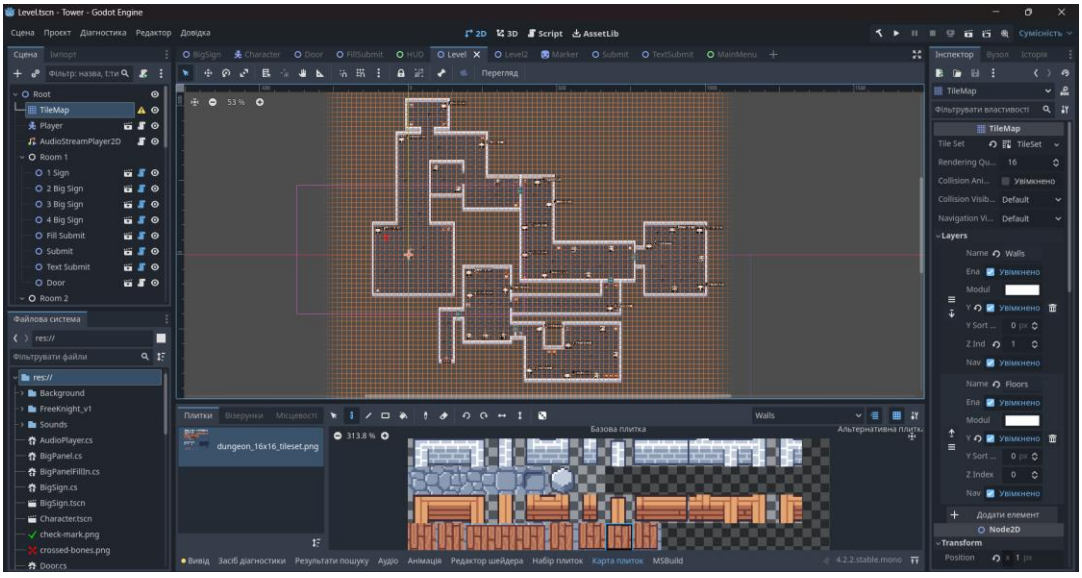

Рис. 2.13 Ось так виглядає ігрове поле в моменті редагування

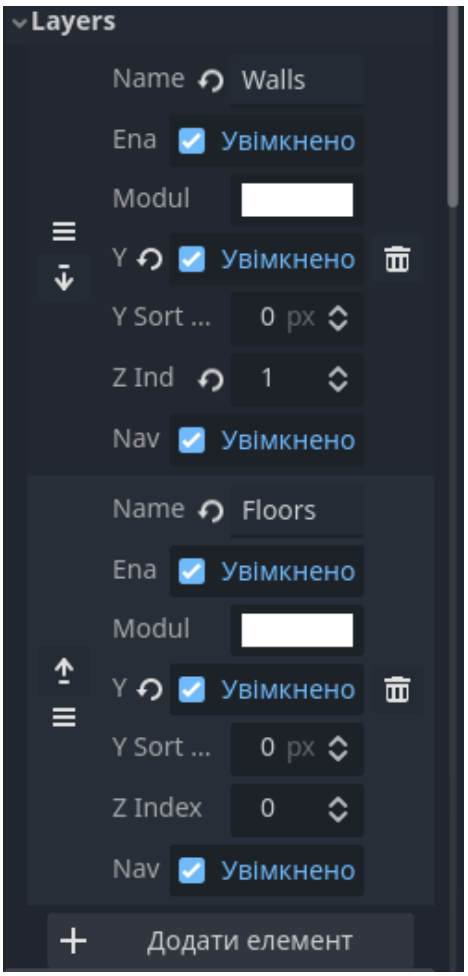

Рис. 2.14 Два шари – підлога та стіни

- далі пропрацьовано читабельність знаків, які стоять на початку кожної секції (кімнати) та повідомляють назву теми, яка буде вивчатись. Знаки мають звичайний спрайт та деякі UI елементи для підказки «Press E to read». Базовий вузол має сценарій Sign.cs. В цьому сценарії вказано public string text змінну з [Export] атрибутом, крім того є властивість bool Active, і в її налаштуваннях отримано доступ до відповідних дочірніх вузлів і зроблено їх видимими/невидимими відповідно до заданого значення. Далі у функції _Process постійно перевіряється, чи близько гравець до знаку, і якщо так, то UI стає видимим, і якщо користувач натисне клавішу E, то властивість Active перемикається, внаслідок чого з’являється текст знаку і стає видимим [19].

Зберігається поточне значення Input.IsKeyPressed у змінній lastPressed, щоб переконатись, що це саме перший фрейм, який натискається, а не наступні;

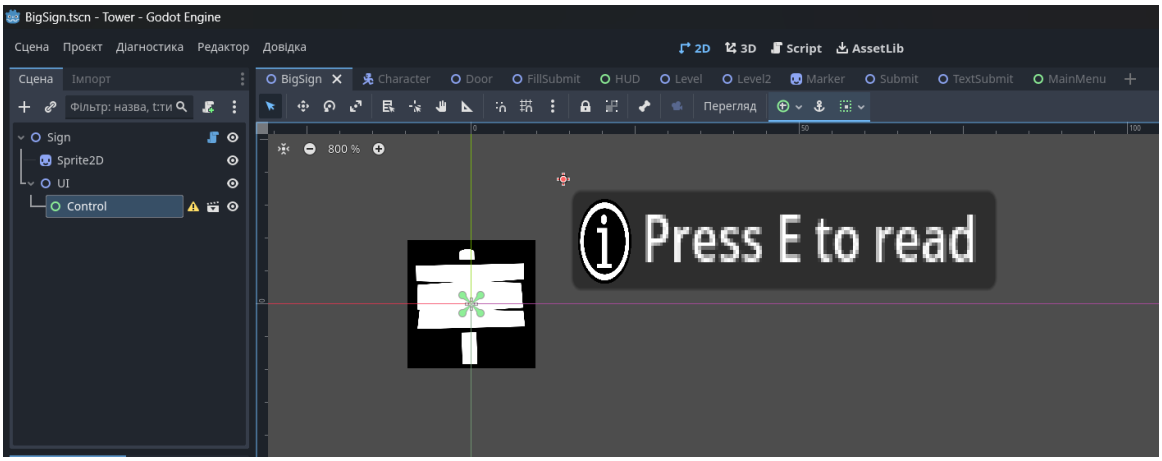

Рис. 2.15 Знак, який містить в собі Sprite2D (звичайне зображення) та UI елемент

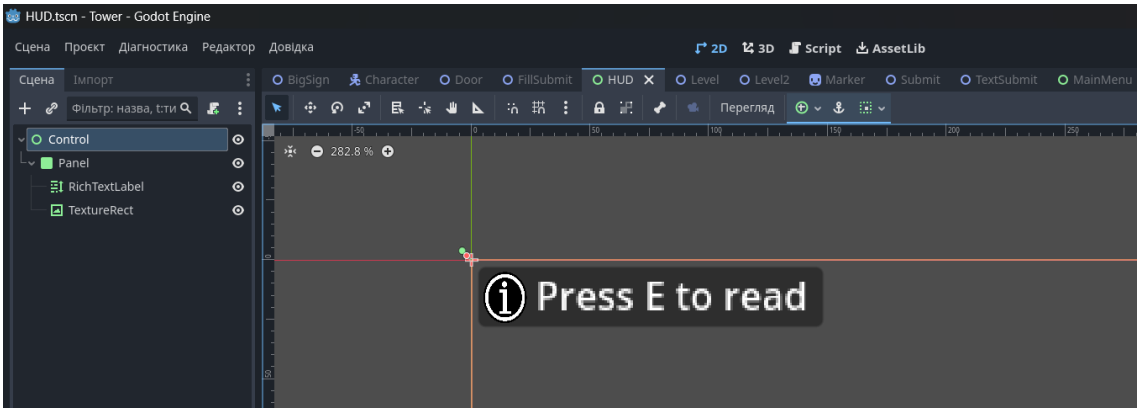

Рис. 2.16 Той самий дочірній вузол UI елемента, який відображає текстову панель та зображення

- наступним кроком пропрацювано знаки, які показують велику текстову мітку. Розміщено панель в програвачі таким чином, щоб вона була в просторі екрану, потім поміщено на неї RichTextLabel, щоб відображати текст, і кнопку X зверху справа, щоб закрити панель. До цієї панелі прикріпюється скрипт під назвою BigPanel.cs. В цьому скрипті підписано на подію кнопки X: Pressed(GetNode<Button>("X").Pressed += () => Visible = false;) – щоб закрити

панель, коли її натиснуто. Окрім цього створено static void ShowText(string what) функцію, яку можна викликати звідусіль, щоб показати певний і більш специфічний текст, таким же чином створено static void Clear() функцію, щоб вимкнути панель. Тепер для нового варіанту створено новий BigSign.cs скрипт, який працює так само, як і попередній Sign.cs, за винятком того, що він використовує велику панель;

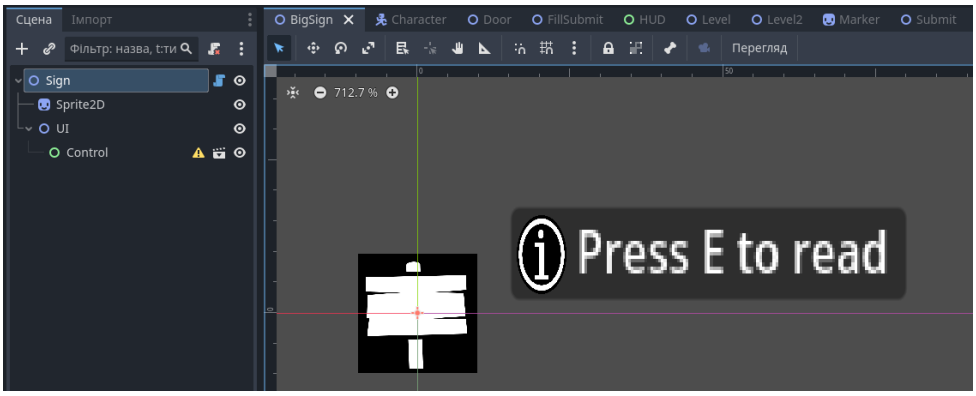

Рис. 2.17 Знак, після наближення до якого та натискання клавіші E виводиться велика панель

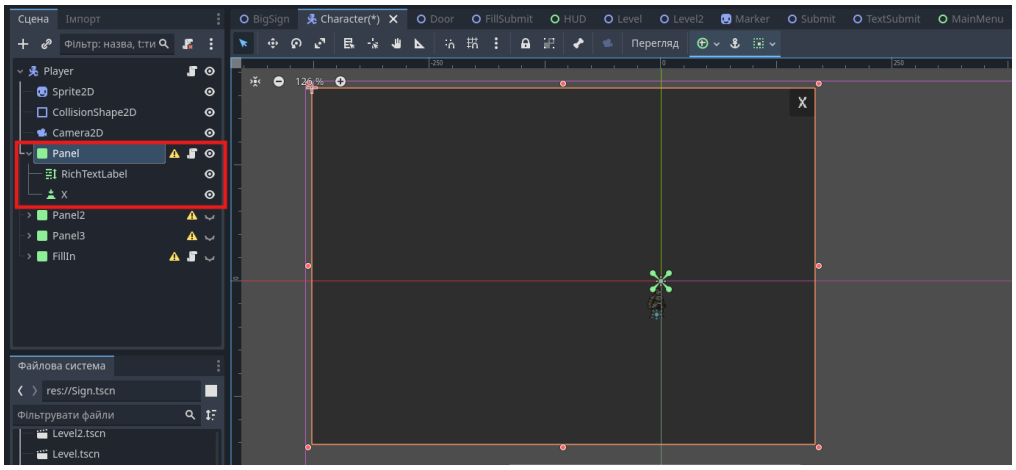

Рис. 2.18 Велика панель з кнопкою X, на якій буде відображатись теоретична частина гри

- далі додано три різні типи питань, почато з опису питань для яких треба вводити вручну відповідь, яка складається з одного або рідше двох слів.

Створено вузол під назвою Submission3, який складається зі звичайного Sprite2D, UI елемента, який в свою чергу складається з панелі Question, яка складається з трьох дочірніх елементів: RichTextLabel, TextEdit та TextureButton. Окрім цього у вузлі Submission3 є ще два спрайти – Correct та Incorrect, які містять в собі картинки зеленої галочки та червоного хрестика відповідно. Отже, є спільна панель Question, зверху якої накладені панелі RichTextLabel та TextEdit, більше площі займає панель RichTextLabel, адже саме в цій панелі задається питання, і нижче неї розміщено панель TextEdit, яка є трохи темнішою, і в яку користувач буде вводити відповідь, на цій же нижній панелі розміщено BaseButton під назвою TextureButton, ця кнопка розташована справа і представляє собою зелену галочку, яку користувач натискає після введення відповіді для перевірки правильності. До вузла Submission3 прикріплено скрипт TextSubmit.cs, і у функції запуску додатково підписано на події FocusEntered та FocusExited. Окрім цього керуються перемикачі змінної CanMove, яка належить скриту Movement.cs – це не дозволяє гравцю рухатись, поки він вводить відповідь. Далі підписано на TextureButton кнопку, яка викликає isCorrect делегат, і коли ShowText викликається, то він приймає список наданих відповідей, і цей делегат налаштовується на функцію, яка перевіряє, чи збігається текст з правильними відповідями. Якщо так, то відображається спрайт зеленої галочки, лунає аудіофайл kaching.ogg, а панель вимикається, якщо ж відповідь введено неправильно, то не відбувається нічого, щоб гравець міг повторити спробу.

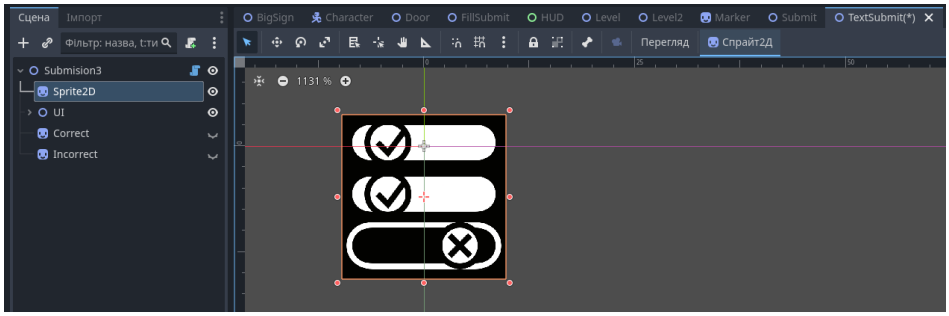

Рис. 2.19 Спрайт для вузла Submission3, саме так виглядають всі знаки з питаннями

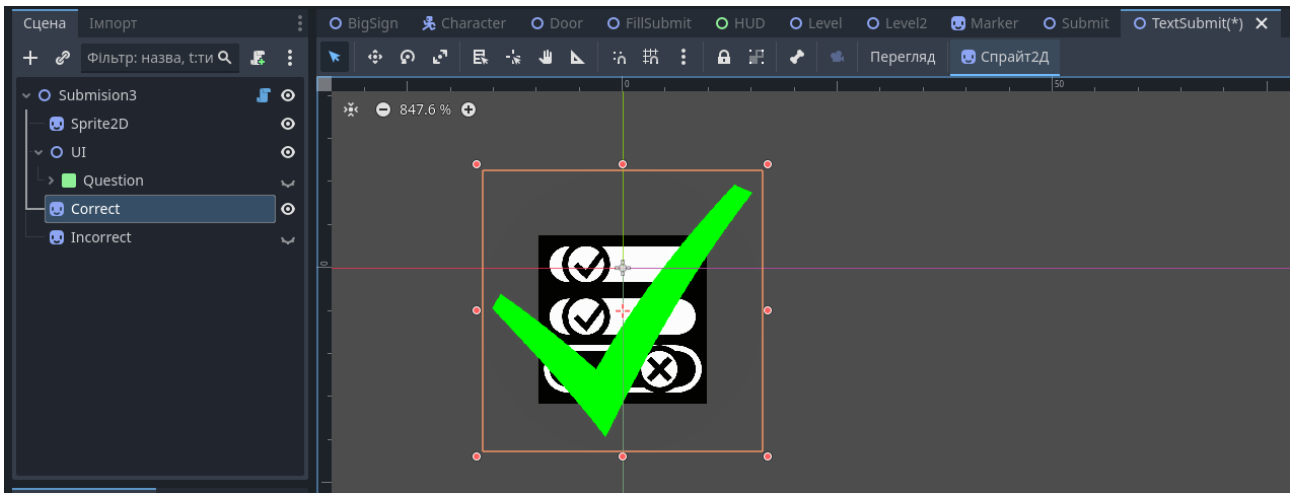

Рис. 2.20 Спрайт Correct, який з’являється зверху основного знаку після правильної відповіді

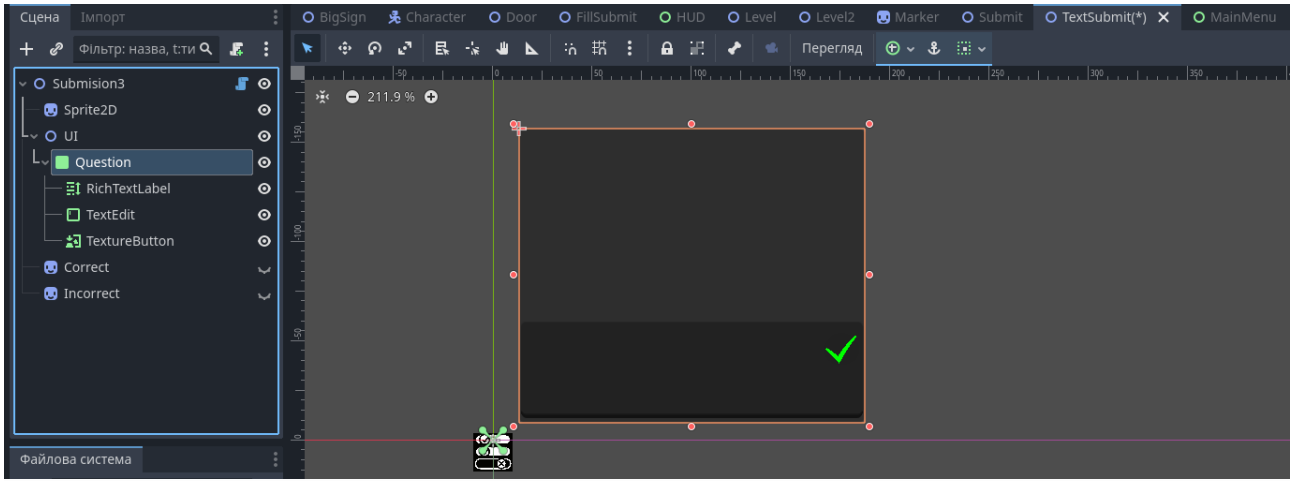

Рис. 2.21 Панель Question, яка з’являється, коли гравець підходить близько до знаку

- далі створено наступний тип питання – заповнення пропуску в коді. Знову створюється новий вузол, який має назву FillSubmit, він складається зі спрайту, який зображено на рис. 2.17, UI елементу з рис. 2.14 та спрайту з рис. 2.18. До вузла FillSubmit прикріплюється скрипт FillInQuestion.cs, який відповідає за те, щоб виводилась панель з питаннями та щоб вона закривалась, якщо питання неактивне. В той же час цей в цьому рядку BigPanelFillIn.ShowText(text, answers.Split(',').ToList(), this); відбувається

посилання на скрипт BigPanelFillIn.cs, який керує великою панеллю для відображення тексту та перевірки відповідей, і в свою чергу цей скрипт прикріплений до FillIn панелі, яка складається з таких компонентів: TextEdit, кнопки X та кнопки Confirm. Тобто створено знак з написом «Press E to read», після натискання E з'являється велика панель з кодом, справа зверху створено кнопку X, яка закриває панель, а справа знизу додано кнопку >, яка виконує функцію підтвердження відповіді. Тепер як саме реалізується перевірка відповіді: if (i >= 2 && correct[i] == '?' && correct[i - 1] == '?' && correct[i - 2] == '?'), тобто створено код, і замість певних функцій стоять три знаки питання ???, і замість них користувач має ввести правильні функції. Якщо користувач відповідає правильно, то знову відображається спрайт зеленої галочки, лунає аудіофайл kaching.ogg, а панель вимикається, якщо ж відповідь введено неправильно, то не відбувається нічого, щоб гравець міг повторити спробу.

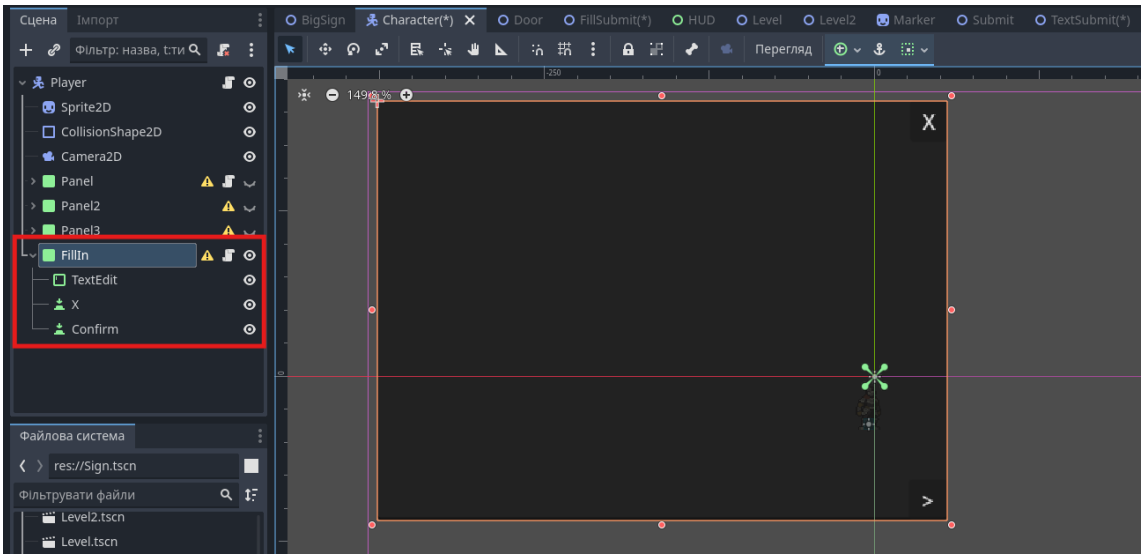

Рис. 2.22 Панель FillIn, яка з'являється, коли гравець підходить близько до знаку та натискає кнопку E

- останнім кроком створено третій тип запитань – питання з варіантами відповіді: додано вузол під назвою Submit, який складається вже з попередньо відомого спрайту з рис. 2.17, панелі Question, яка складається з

кнопок A, B, C, D та RichTextLabel, окрім цього це знову є спрайт з рис. 2.18 і ще до цього додається спрайт Incorrect, який є зображенням червоного хрестика, що з'являється після вибору неправильної відповіді. До вузла Submit прикріплено скрипт Submit.cs, який керує питаннями, перевіряє відповіді та оновлює стан гри, залежно від правильності відповідей. GetNode<Node2D>(_.Name == CorrectAnswer ? "Correct" : "Incorrect").Visible = true; тобто якщо відповідь неправильна, то на знаку з'являється спрайт червоного хрестика, але користувач все ще може переобрати відповідь, і тоді з'являється зелена галочка і лунає аудіофайл kaching.ogg, після чого панель зникає. За виглядом інтерфейс схожий на перший тип питань, тобто отримано велику панель, більшу верхню половину панелі займає панель RichTextLabel, а знизу розташовані 4 кнопки;

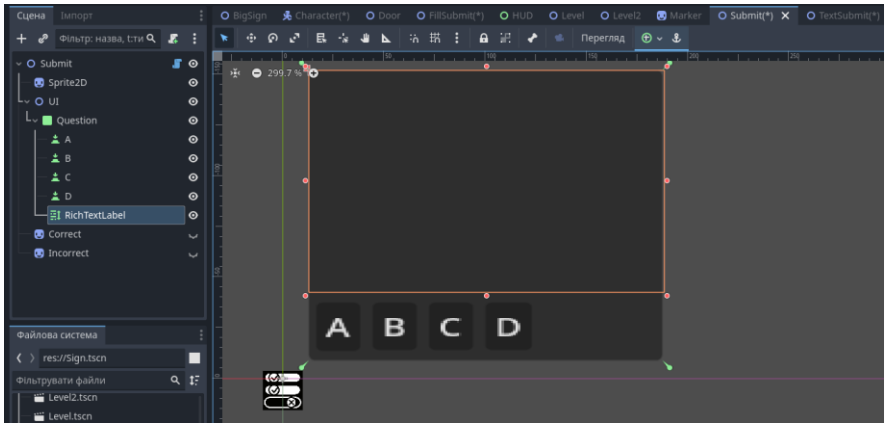

Рис. 2.23 Панель Question, яка з'являється, коли гравець підходить близько до знаку

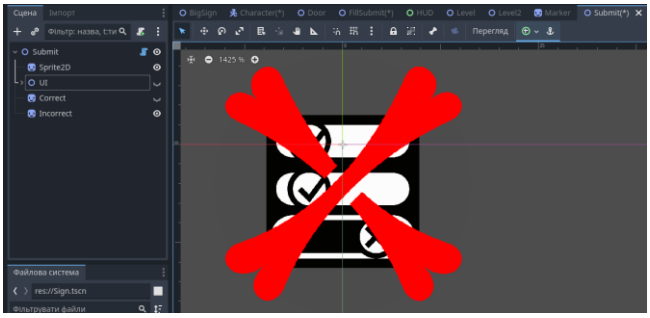

Рис. 2.24 Спрайт Incorrect, який з'являється зверху основного знаку після неправильної відповіді

- найважливішою частиною гри є система шляхових точок, яка була створена наступним пунктом, щоб гра була менш заплутаною та більш спрощеною. Було вирішено, що має бути певний індикатор, який показуватиме куди гравець має йти далі, і навіть більше – він має запобігти пропуску гравця вперед, а також він має відкривати двері лише тоді, коли гравець відповів на всі питання в кімнати, в якій він перебуває.

```
public interface Prompt
{
public Func<bool> BecameNext { get; set; }
public Action Interacted { get; set; }
public Prompt Next { get; }
public bool IsActive { get; set; }
}
```

Передбачається, що інтерфейс успадковується кожною взаємодіючою річчю в грі, такими як знаки, запитання та двері. Він має Action для того, щоб коли з цільовим предметом відбувалась взаємодія, наприклад, в скрипті Sign.cs, то викликалась Interacted(), коли гравець натискав E біля знаку, щоб прочитати його. А для питань з варіантами відповідей викликається цей інтерфейс тільки тоді, коли гравець відповів правильно на питання. Важливо, що маркер-індикатор не викликається, коли на питання надано неправильну відповідь, тому це головна опція – правильно відповісти.

Ось початкова функція скрипту Marker.cs:

```
public Node Current

{

    get => _current;

    set
```

```
        {
            // поки value не дорівнює null і наступний вузол став активним
            while (value != null && (value as Prompt).BecameNext())
                value = ((value as Prompt).Next as Node);
            // якщо value не дорівнює null, встановлюю IsActive в true
            if (value != null)
                (value as Prompt).IsActive = true;
            // оновлюю поточне значення та позицію
            _current = value;
            GlobalPosition = ((Node2D)value).GlobalPosition + new Vector2(0,-10);
        }
    }
```

Спершу викликається функція BecameNext, і якщо повертається true, то підказку пропускається і замість цього відбувається перехід до наступної – це повторювальний процес. Цю властивість застосовано для дверей, оскільки на меті є функція, щоб двері пропускали маркер підказки як тільки стають активними. Наступним встановлено нову IsActive властивість на true, що значить, що скрипт вузла підказки може робити щось з цим (робити себе активним для взаємодії), і після цього налаштовано позицію маркера знову. Ось і все, властивість IsActive для кожного інтерактивного елемента (знаки та питання) гарантує те, що з цим елементом неможливо взаємодіяти, коли він

неактивний. Окрім цього для властивості IsActive не встановлено попередніх запитів значення false, тому що метою є, щоб раніше відвідані інтерактивні елементи залишались взаємодіючими, коли гравець вже пройшов їх;

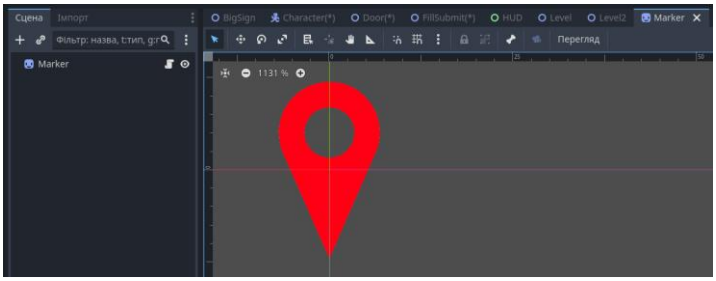

Рис. 2.25 Спрайт Marker, який вказує послідовність питань та знаків

- окремо на рівнях створено двері, тобто дадається новий вузол під назвою Door, який складається зі Sprite2D та вузла під назвою StaticBody2D, що має дочірній вузол з фізикою CollisionShape2D, який не дозволяє гравцю просто проходити крізь двері. До вузла Door прикріплено скрипт Door.cs, і у скрипті ключовою задачею є успадкування від інтерфейсу Prompt. Для дверей зберігаються всі значення за замовчуванням, за винятком того, що на початку перевіряється BecameNext, щоб видалити двері та повернути true, тому вони миттєво пропускаються, коли їх відкривають. Тобто на початку гри всі двері залишаються зачиненими, і як тільки маркер підказки переходить з однієї кімнати до іншої, то двері автоматично зникають, що значить, що вони пропускають гравця вперед в наступну кімнату;

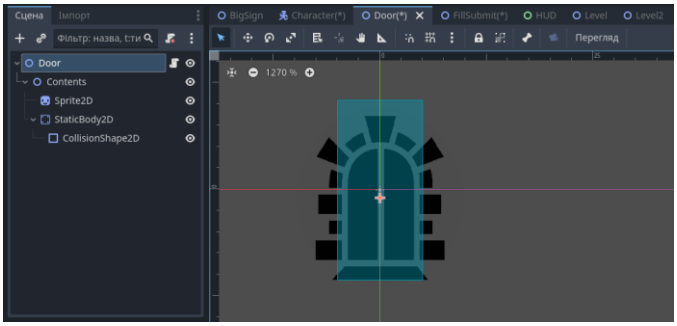

Рис. 2.26 Спрайт Door з фізикою, який розмежовує кімнати

- далі розглядаються два скрипти: Exit1.cs та Exit.cs, які відповідають за перехід між рівнями та вихід з гри [6]. Отже скрипт Exit1.cs застосовується для першого рівня, після проходження всіх завдань в кінці коридору створено Node2D, до якого і прикріплено скрипт Exit1.cs. В моменті, коли гравець стикається з цим вузлом, то з'являється панель, на якій виведено його результати та на якій є кнопка, яка автоматично переносить гравця на наступний рівень. Щодо Exit.cs, то він застосовується для другого рівня, там так само після проходження рівня створено вузол, після зіткнення з яким виводиться вже інша панель, теж з результатами, але кнопкою повернення в головне меню;

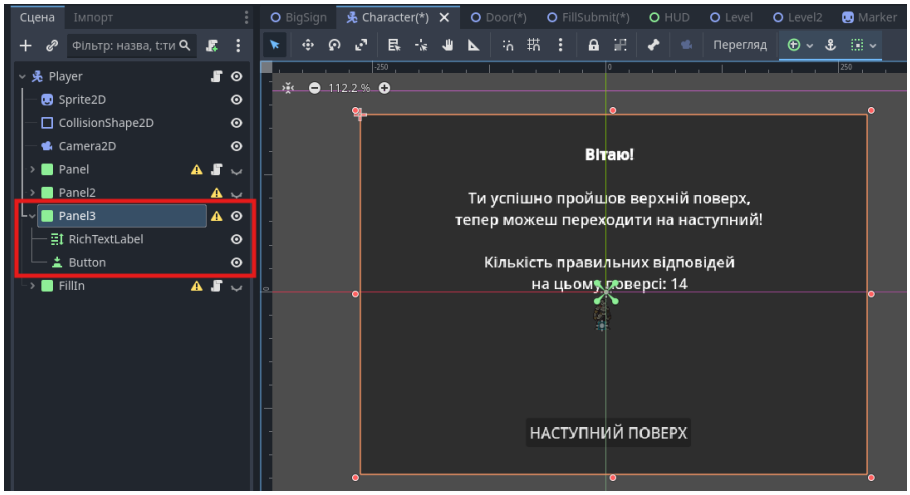

Рис. 2.27 Панель для фіналу першого рівня

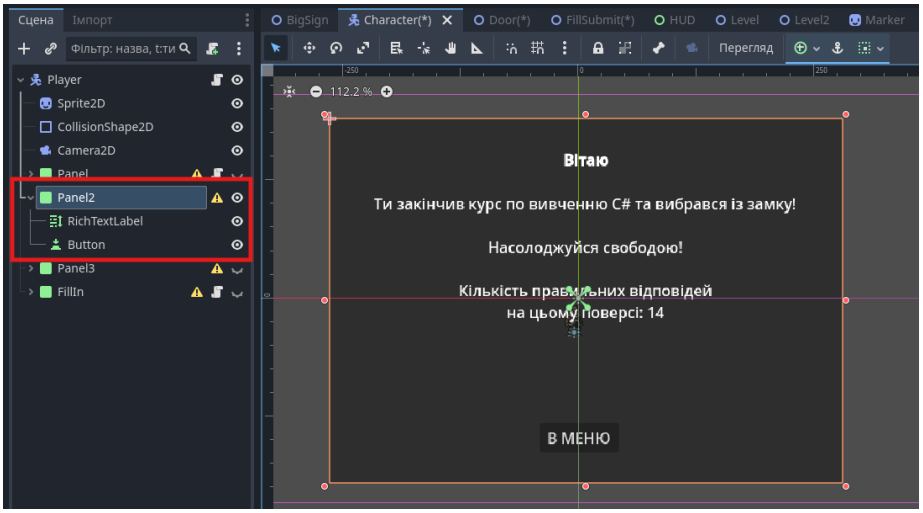

Рис. 2.28 Панель для фіналу другого рівня

- наостанок розглядається скрипт AudioPlayer.cs, який прикріплено до вузла AudioStreamPlayer2D. Оскільки маємо 5 аудіофайлів зі звуками кроків, цей скрипт реалізовує звук випадковим чином, обираючи різні файли кожної секунди.

Отже, говорячи про ієрархію всіх цих файлів, то важливо зазначити, що вузли зі знаками та питаннями є дочірніми для вузлів загальних кімнат, тобто для двох рівнів існує 5 кімнат, які є вузлами, і для кожної кімнати є певна кількість дочірніх вузлів – знаків з теорією та завданнями, і вузли дверей.

Тобто послідовність вузлів та файлів наступна:

1. Вузол Root, який є основним вузлом рівня
    1) TileMap
    2) Player
    3) AudioStreamPlayer2D
    4) Room 1
    5) Room 2
    6) Room 2+
    7) Room 3
    8) Room 4
    9) Marker
    10) Node2D

Таку саму структуру має і другий рівень.

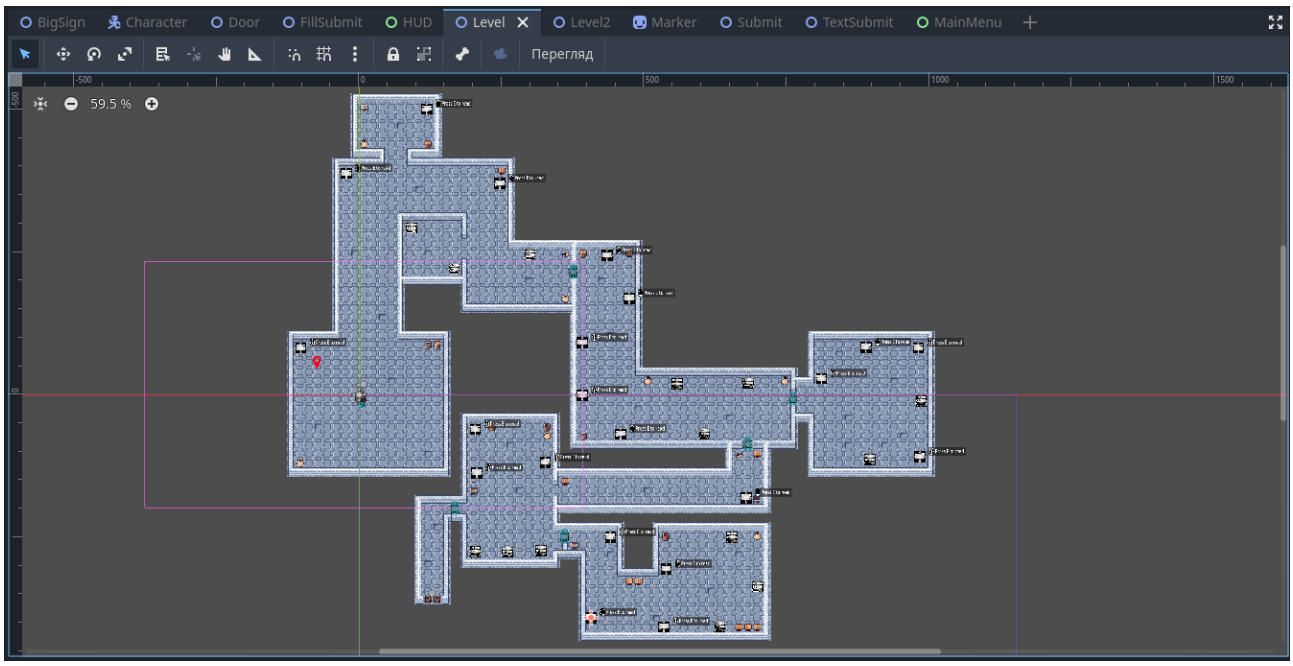

Рис. 2.29 Зовнішній вигляд першого поверху в редакторі 2D

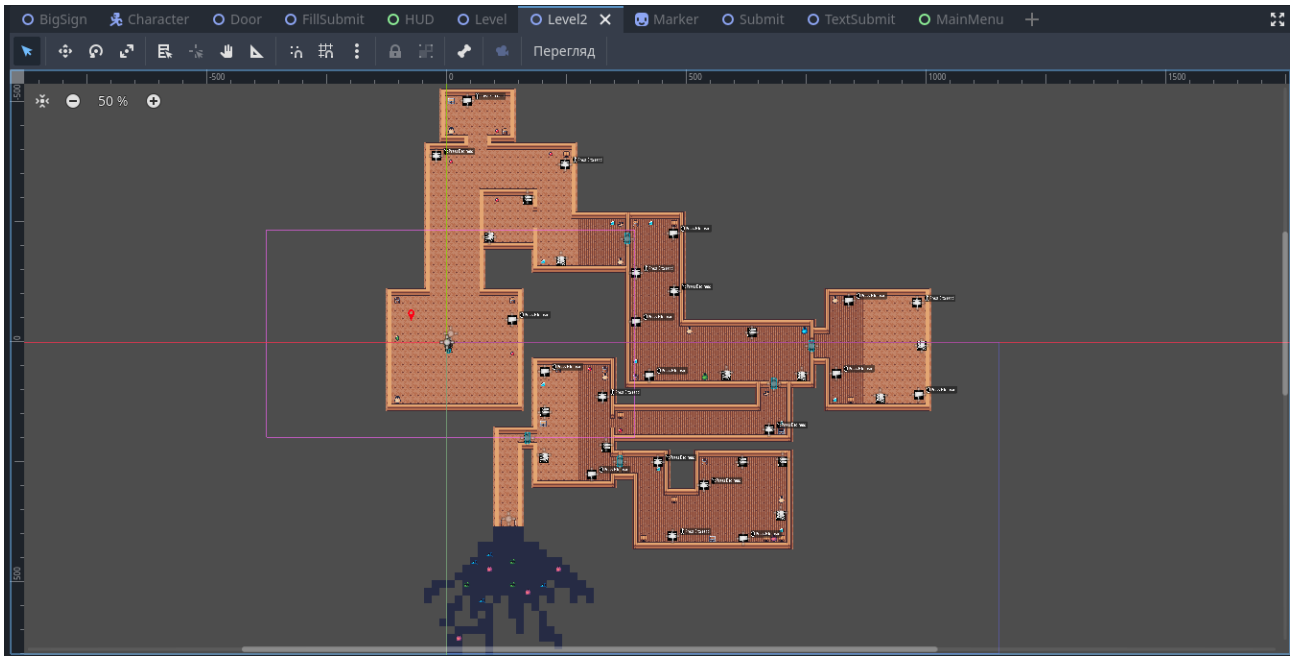

Рис. 2.30 Зовнішній вигляд другого поверху в редакторі 2D

Опис структури системи та алгоритмів її функціонування показує як саме різні компоненти гри взаємодіють один з одним для забезпечення цілісного та захоплюючого ігрового процесу. Важливо наголосити, що структура коду була організована таким чином, щоб полегшити подальше розширення функціоналу

та підтримку гри. Результати показали, що створена структура головного меню та рівнів дозволяє ефективно керувати ігровим процесом, забезпечуючи користувачу інтуїтивно зрозумілий інтерфейс та плавну взаємодію з грою. Подальший розвиток проекту може включати в себе додавання новий функцій, покращення візуальних ефектів та розширення можливостей штучного інтелекту для створення більш захоплюючого ігрового досвіду. Таким чином, розроблені алгоритми та структура системи закладає міцну основу для подальшого розвитку ігрового проекту, забезпечуючи високу продуктивність та зручність використання.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідні дані в даній платформі-грі представляють собою взаємодію користувача з інтерфейсом та ігровими елементами. Основними типами вхідних даних є:

- введення з клавіатури, яке використовується для керування персонажем та взаємодії з елементами ігрового інтерфейсу, а саме: клавіші WASD використовуються для переміщення персонажа по ігровому полю, оскільки саме цей спосіб є стандартним та інтуїтивно зрозумілим для більшості гравців, а також клавіша E використовується для взаємодії з об’єктами, такими як знаки та питання, бо це дозволяє гравцю швидко та зручно виконувати дії, не відволікаючись від ігрового процесу;
- введення з мишки для навігації по меню та вибору опцій забезпечує зручний та точний спосіб взаємодії з інтерфейсом.

Щодо обробки вхідних даних, то для них у грі використовуються влаштовані функції ігрового рушія Godot, такі як _Process та _Input. Ці функції дозволяють відслідковувати стан клавіш та переміщення миші, а також оброблювати відповідні дії.

Вихідними даними нашої платформи-гри є візуальна та звукова інформація, яка відображається користувачу у відповідь на його дії. Основні типи вихідних даних включають:

- графічні елементи, які відображають меню, а саме: панелі станів персонажа та підказок за допомогою стандартних елементів інтерфейсу Godot, таких як Button, TextureRect, RichTextLabel та Panel, що дозволяє створити інтуїтивно зрозумілий та візуально приємний інтерфейс; анімація персонажу, для якої використовуються спрайти та анімаційні фрейми, що забезпечує плавні та реалістичні рухи;
- звукові ефекти, а саме фонова музика та звукові ефекти, оскільки на моїй платформі реалізовано фонову музику в головному меню, звуки кроків та правильних відповідей, то цей аспект є одним з ключових, тобто застосування звукових файлів формату .ogg дозволяє створити насичене звукове середовище, яке доповнює ігровий процес та покращує занурення гравця у гру.

Для організації вихідних даних використовуються вузли та ресурси Godot, такі як Sprite2D, AudioStreamPlayer, RichTextLabel та Panel, що дозволяє гнучко керувати візуальними та звуковими елементами гри.

Організація вхідних та вихідних даних в розробленій платформі-грі забезпечує зручну та інтуїтивно зрозумілу взаємодію користувача з ігровим процесом. Вибір методів вводу та виводу даних заснований на найкращих практиках розробки ігор і дозволяє створити високоякісний ігровий досвід для користувачів.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Для розробки, тестування та налагодження платформи-гри було задіяно технічний засіб (ноутбук) з такими характеристиками [13]:

- ноутбук HP Laptop 15-fd0038ua;

- операційна система: Windows 11;
- процесор: Intel Core i3-1315U (2.4 ГГц) [14];
- відеокарта: Intel UHD Graphics;
- накопичувач: SSD 512 ГБ;
- оперативна пам’ять: 16 ГБ;
- монітор: розширення 1920x1080.

2.6.2. Використані програмні засоби

Проект створено на основі Godot Engine, який є відкритим ігровим рушієм з підтримкою як 2D, так і 3D графіки. Godot представляє потужний редактор сцен та об’єктів, систему сигналів для зручної взаємодії між об’єктами, а також вбудований фізичний рушій.

Основні використані функції Godot Engine:

– редактор сцен та об’єктів, який спрощує створення та редагування ігровий сцен, додавання та налаштування об’єктів;

– скриптинг та C#, оскільки Godot підтримує написання скриптів мовою C#, котру я і використала для реалізації логіки гри;

– фізичний рушій, що забезпечує реалістичну поведінку об’єктів у грі.

Для керування звуком у грі використано вбудовані можливості Godot Engine, оскільки звукові ефекти та музика обробляються за допомогою вузлів AudioStreamPlayer та AudioStreamPlayer2D, які допомагають програвати звукові файли в різноманітних форматах та керувати їх параметрами.

Мовою програмування було обрано C#, який використовується для розробки логіки гри, керування ігровими об’єктами та взаємодії з користувачами.

Використання Godot Engine та мови C# дозволило створити функціональну та інтерактивну платформу-гру, в якій лицар навчається програмуванню на C#.

Вбудовані в Godot можливості керування звуком та фізикою допомогли реалізувати якісний ігровий процес.

2.6.3. Виклик та завантаження програми

Оскільки для платформи-гри створено файл формату .exe, то завантаження не вимагається, достатньо просто запустити вихідний файл. Назва та ярлик цього файлу показано на малюнку знизу, і отже користувач може просто завантажити архів з усіма файлами, розпакувати його та запустити файл .exe.

Рис. 2.31 Ярлик для запуску гри

2.6.4. Опис інтерфейсу користувача

Запуск гри розпочинається з головного меню, яке пропонує кілька опцій. Коли користувач запускає файл, що виконується (.exe), то відбувається ініціалізація всіх компонентів гри, включаючи завантаження текстур, звуків, анімацій та інших ресурсів. При цьому відображається головне меню, в якому користувач може вибрати подальші дії. Основні елементи коду, такі як MainMenu.cs, відповідають за відображення головного меню та обробку натискань на кнопки, такі як Level1 та Level2, що дозволяють розпочати нову гру з певного рівня. Далі, після вибору рівня, система завантажує відповідні ігрові сцени, використовуючи функції, реалізовані в Exit.cs та Exit1.cs, для керування

переходами між рівнями та відображенням ігрових об'єктів. Основні вузли та функції, такі як Movement.cs, забезпечують управління персонажем та його взаємодію з ігровим світом, тоді як інші компоненти, такі як AudioPlayer.cs та BigPanel.cs, відповідають за аудіо та інтерфейсні елементи [18]. Весь процес завантаження та виклику програмних модулів здійснюється синхронно, що дозволяє гравцю плавно розпочати гру та поринути у ігровий процес.

Як вже було вказано, кожна кімната представляє собою певну тему для вивчення C#. Обрані теми для першого рівня:

1. Основна інформація про C#
2. Основний синтаксис
3. Змінні та типи даних
4. Оператори та вирази
5. Умовні конструкції (if, else)

Теми для другого рівня:

1. Цикли (for, while)
2. Функції та методи
3. Класи та об’єкти
4. Робота з файлами та базами даних
5. Фінальна перевірка загальних знань.

В кожній кімнаті міститься по 3-5 теоретичних оголошень з прикладами коду, а також по 3 питання різних типів: заповнити пропуски в коді, ввести відповідь вручну, обрати один варіант з чотирьох.

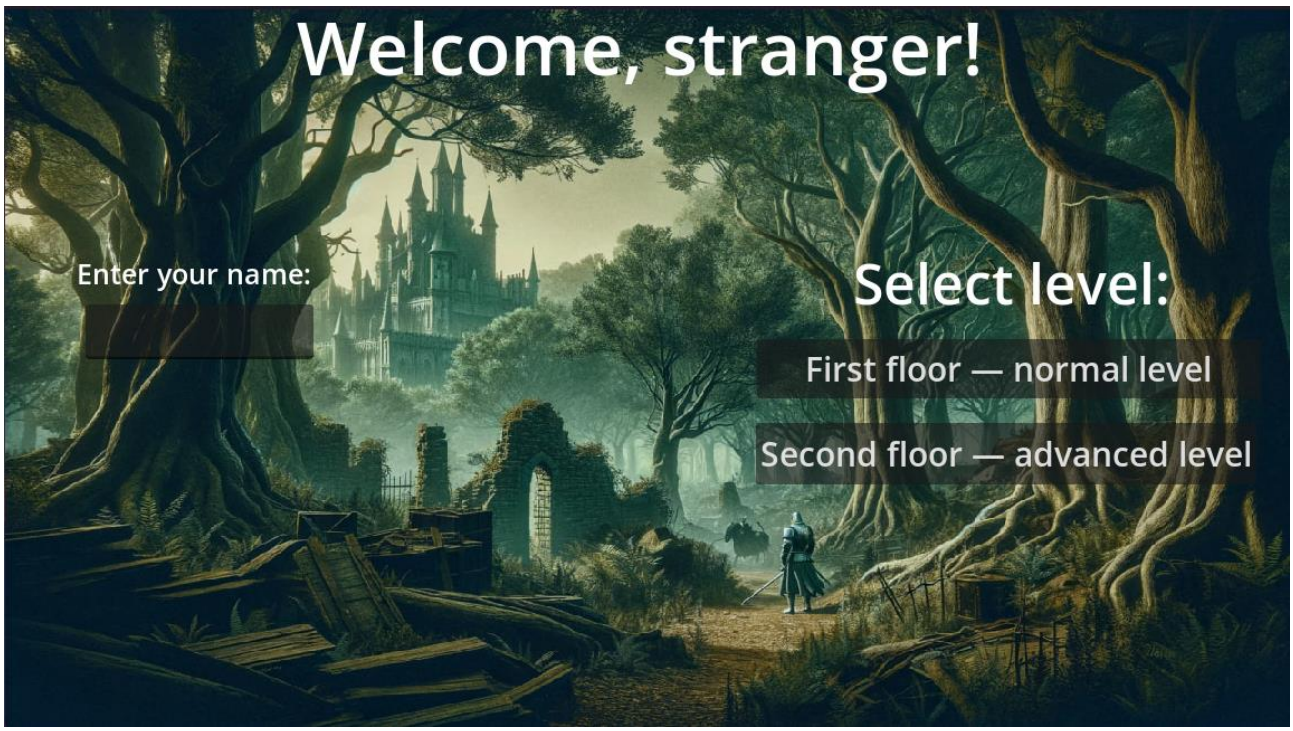

Рис. 2.32 Головне меню

Рис. 2.33 Головне меню після введення імені

Рис. 2.34 Стартове положення гравця після вибору першого рівня

Рис. 2.35 Гравець підходить до першого знаку з маркером

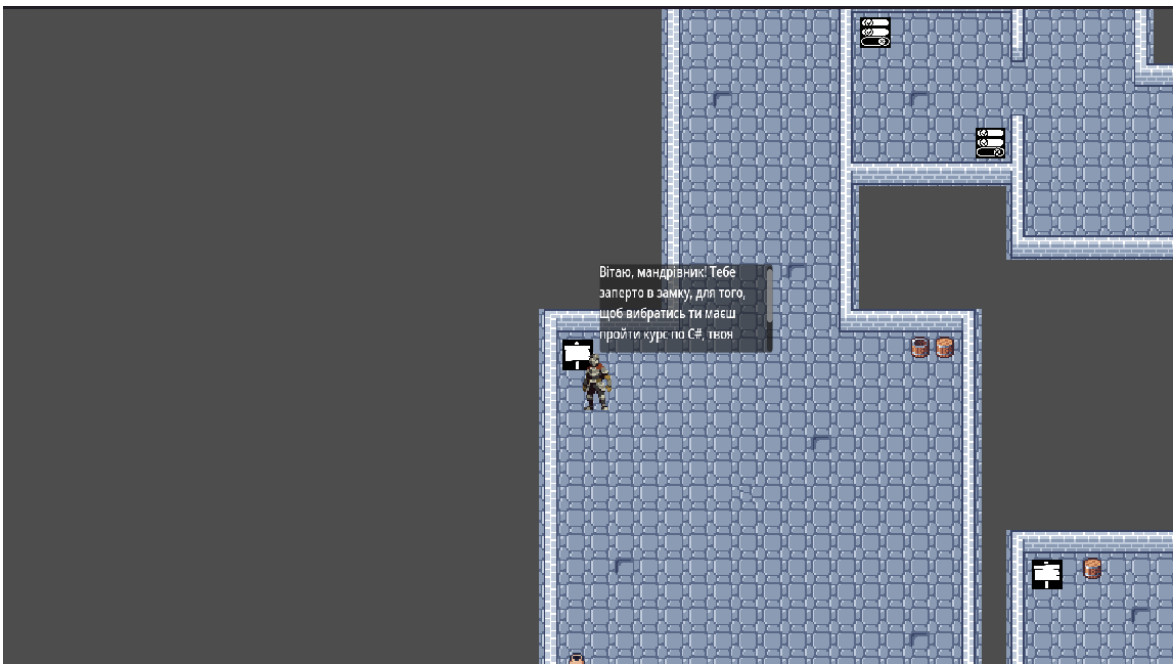

Рис. 2.36 Користувач натискає E і читає інформацію

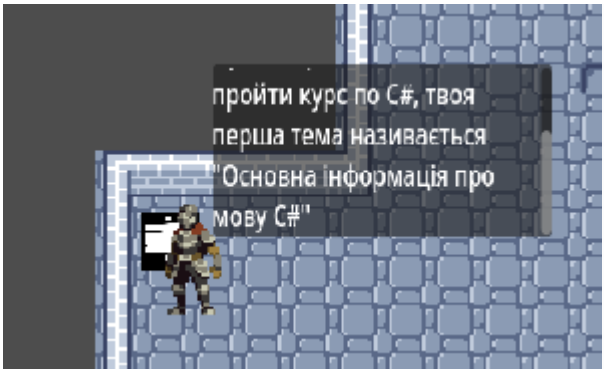

Рис. 2.37 Користувач прокручує повзунок і дочитує оголошення

Рис. 2.38 Переходимо до наступного знаку та натискаємо E

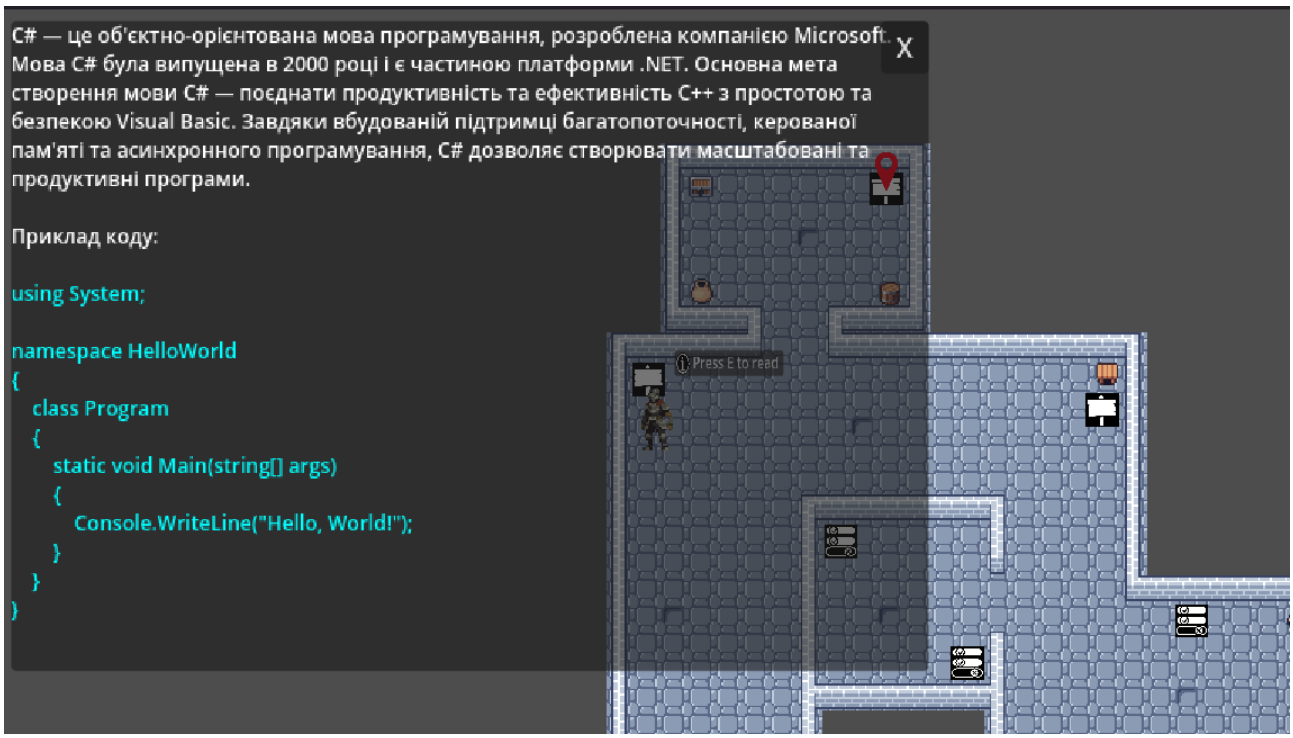

Рис. 2.39 Читаємо теоретичні матеріали

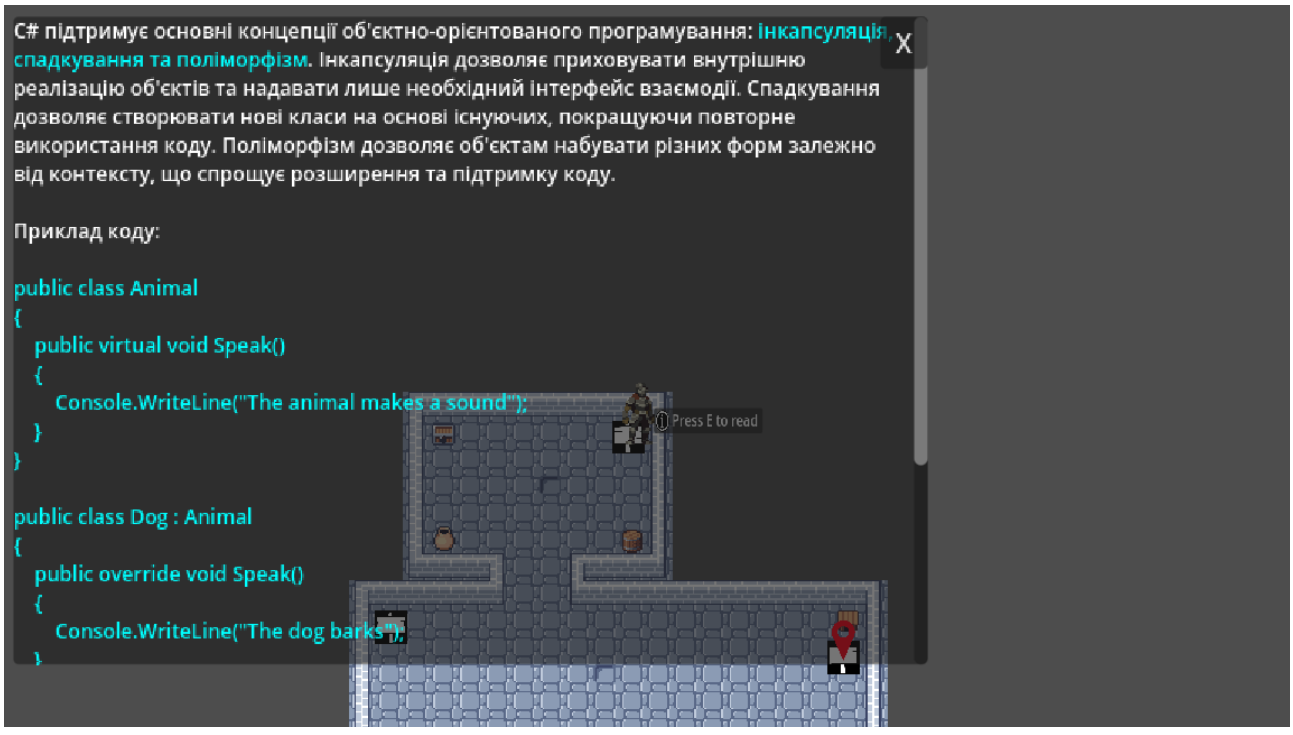

Рис. 2.40 Читаємо матеріали наступного знаку

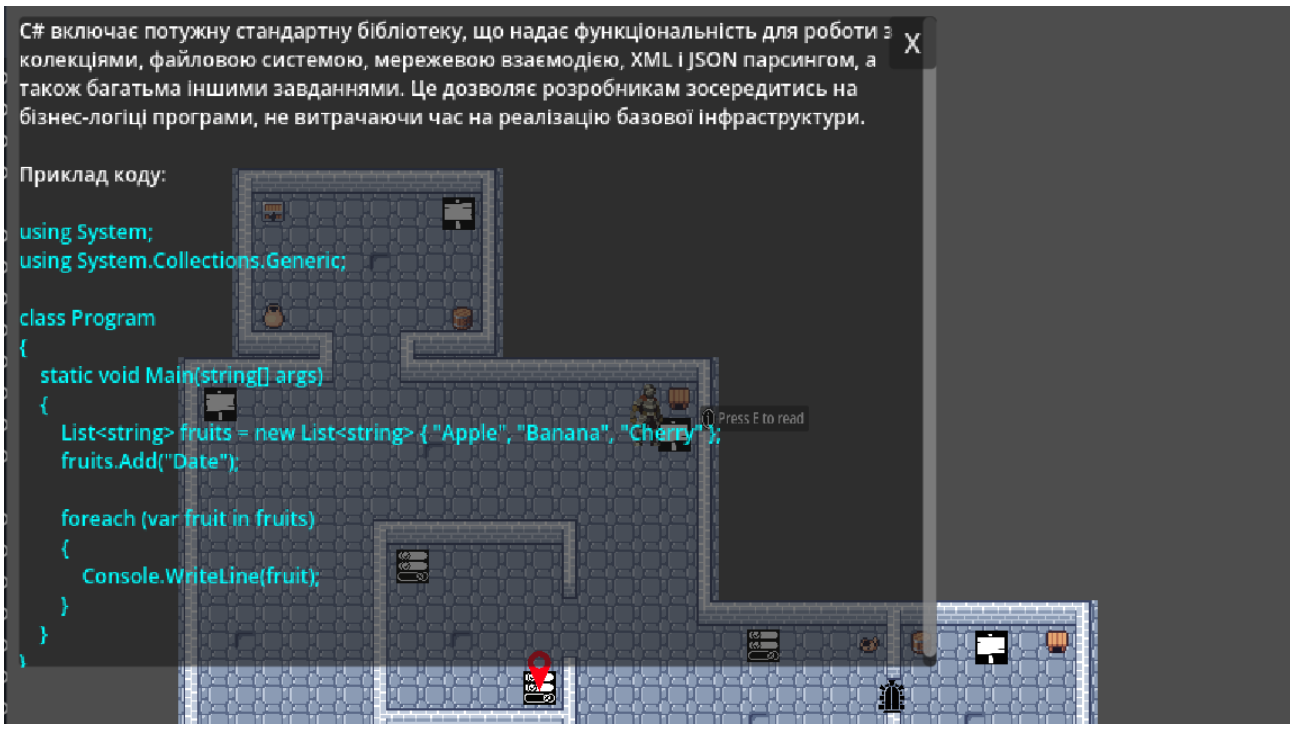

Рис. 2.41 Читаємо інформацію на третьому знаку

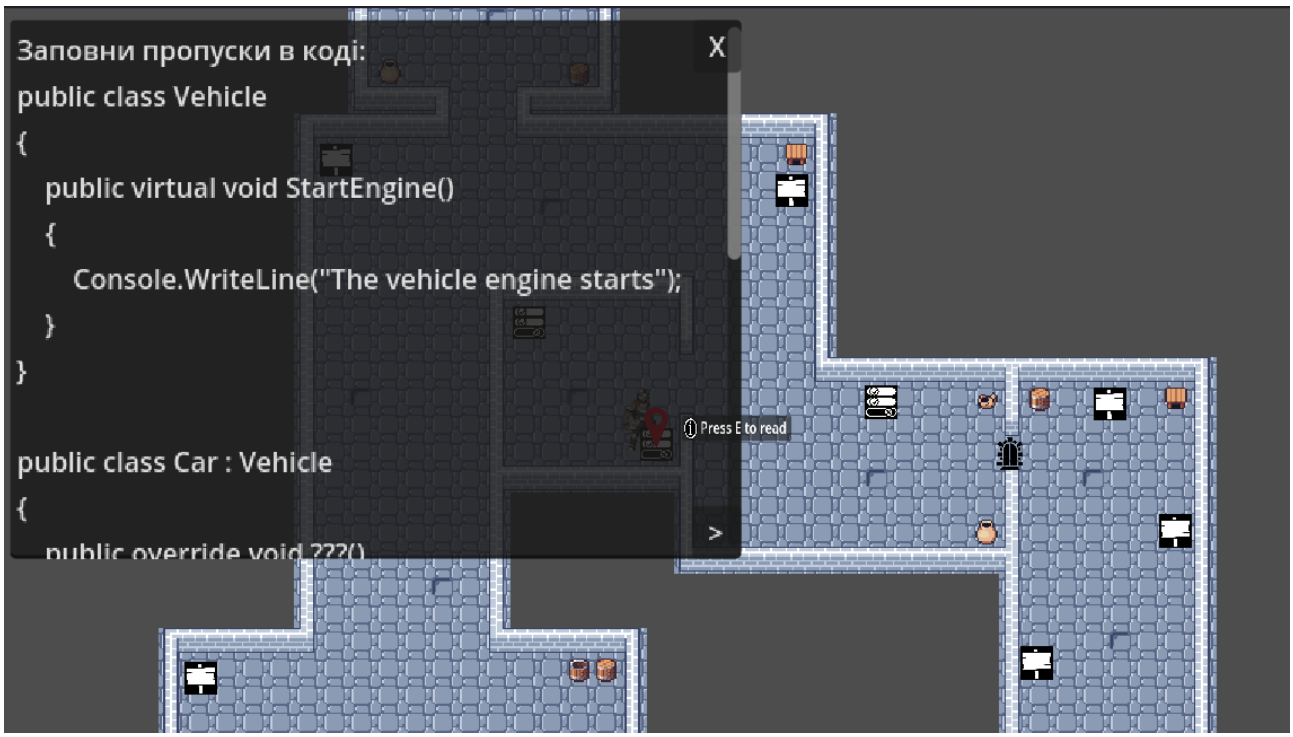

Рис. 2.42 Переходимо до питань, читаємо умову першого запитання

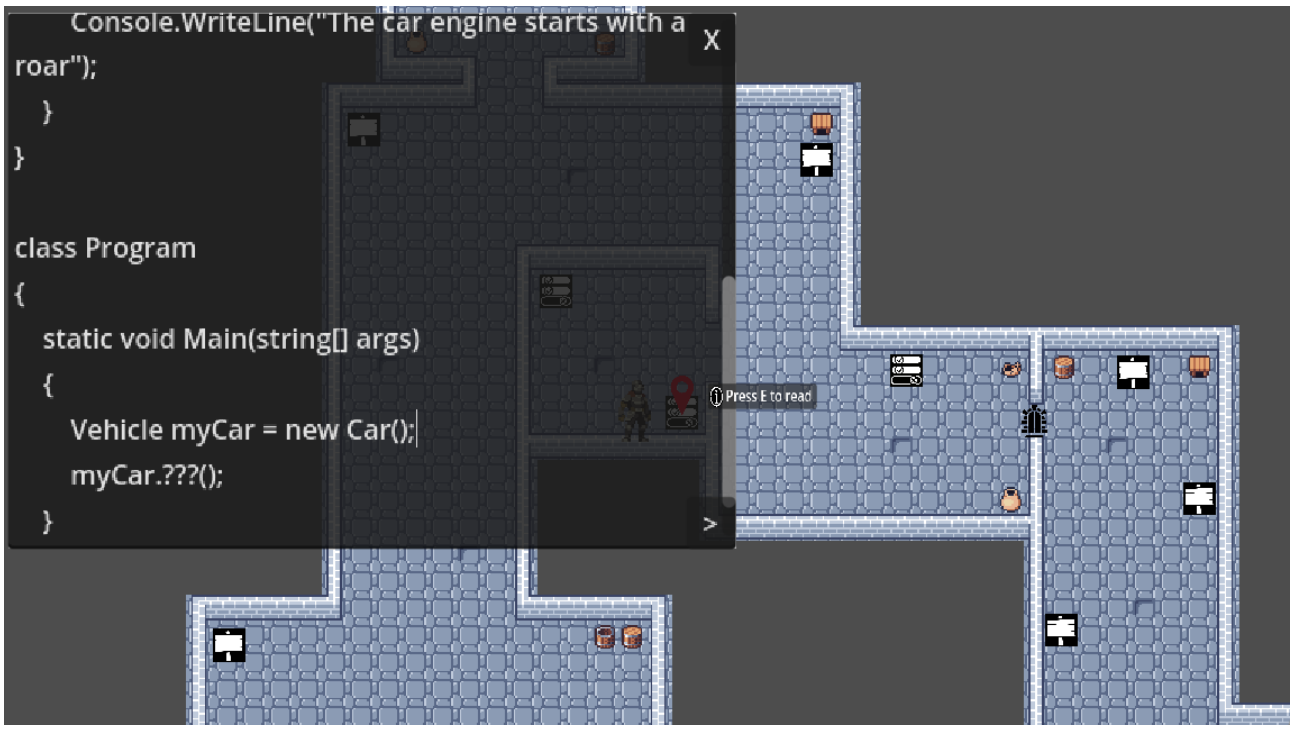

Рис. 2.43 Прокручуємо повзунок до кінця питання

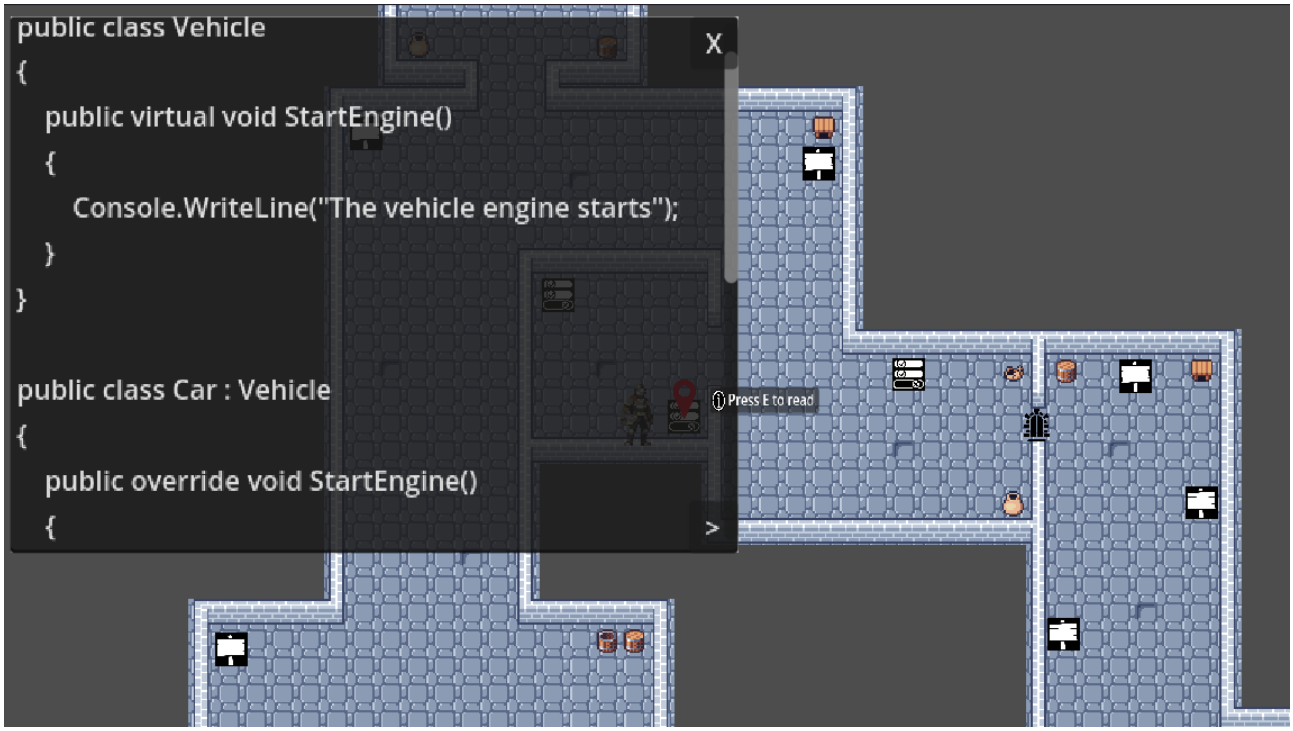

Рис. 2.44 Вводимо відповідь замість знаків питання

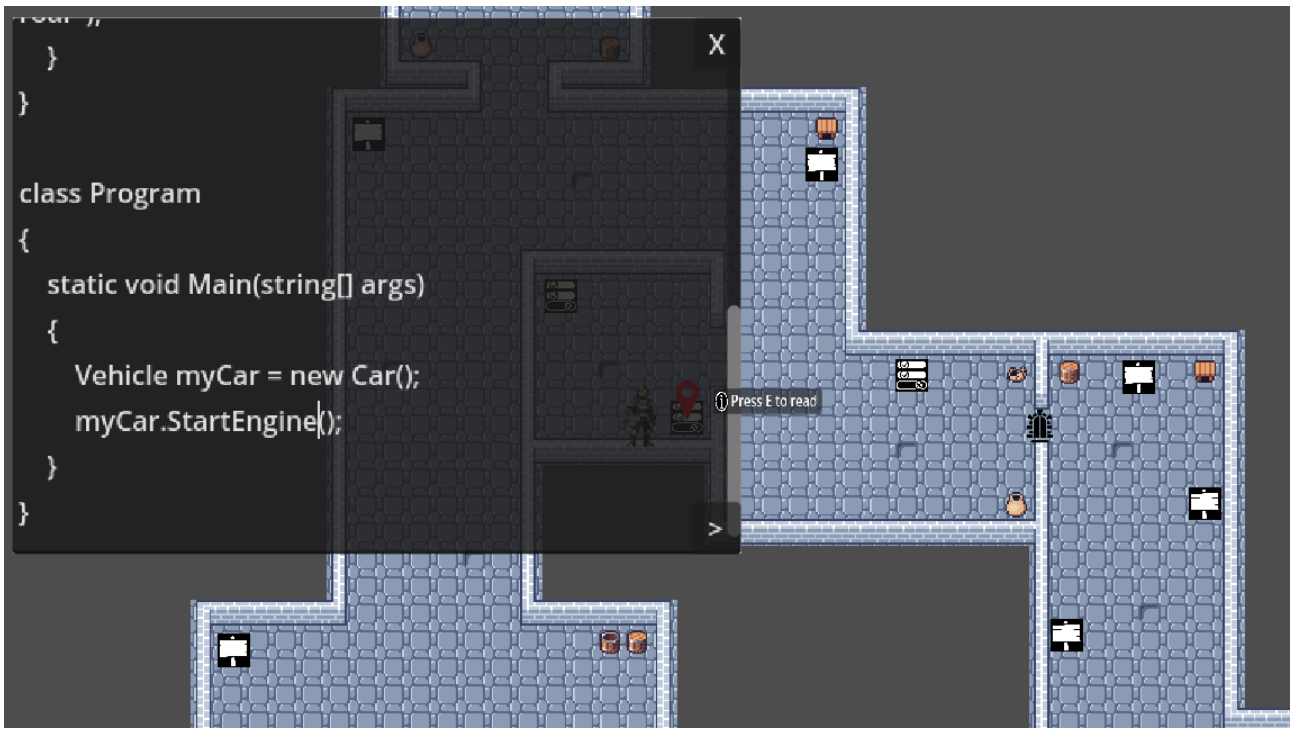

Рис. 2.45 Прокручуємо повзунок та вводимо відповідь замість знаків питання

Рис. 2.46 Оскільки відповідь на попереднє питання правильна, переходимо до наступного

Рис. 2.47 Обираємо неправильну відповідь, бачимо, що маркер все ще на тому самому питанні та воно закреслено хрестиком

Рис. 2.48 Тепер відповідаємо правильно, бачимо, що маркер перемістився, і з’явилась зелена галочка

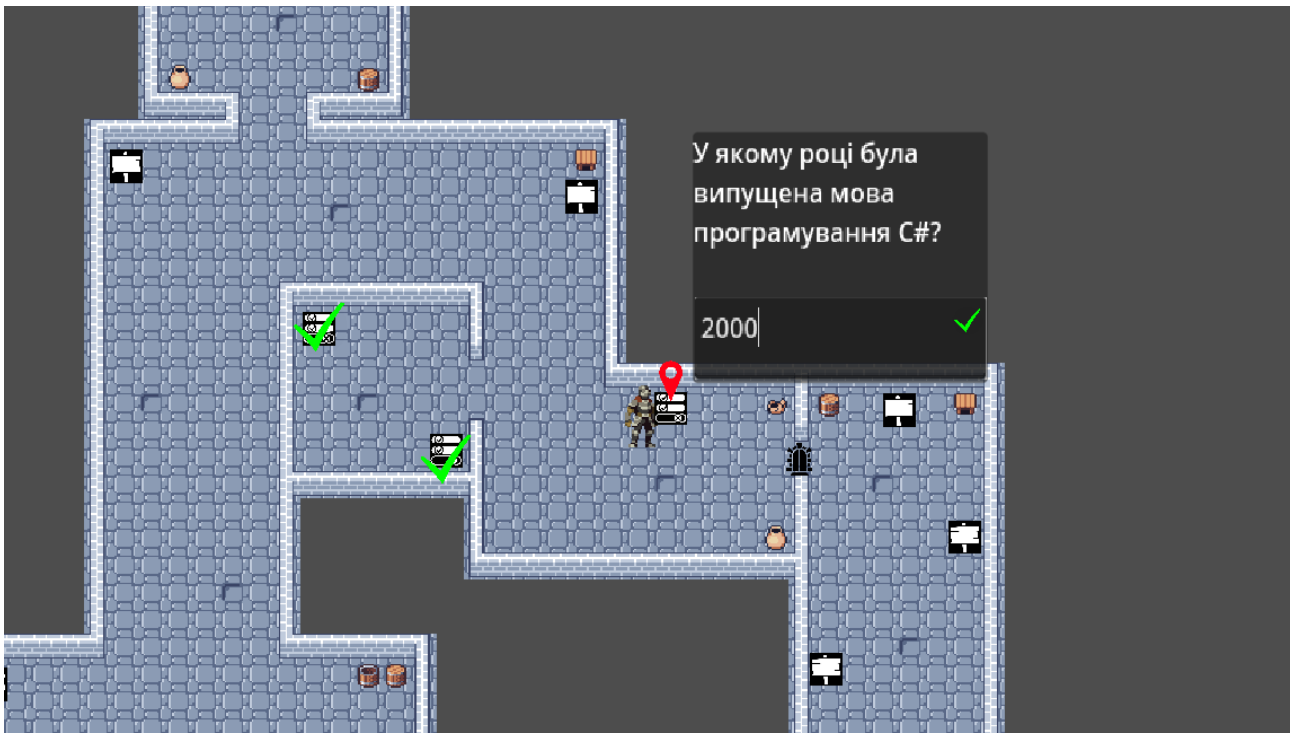

Рис. 2.49 Вводимо відповідь на останнє питання в кімнаті

Рис. 2.50 Відповідь правильна, тому маркер перемістився та відчинились двері в наступну кімнату

За таким алгоритмом працюють всі кімнати поверху, тому одразу перейдемо до останньої.

Рис. 2.51 Читаємо перший знак останньої кімнати

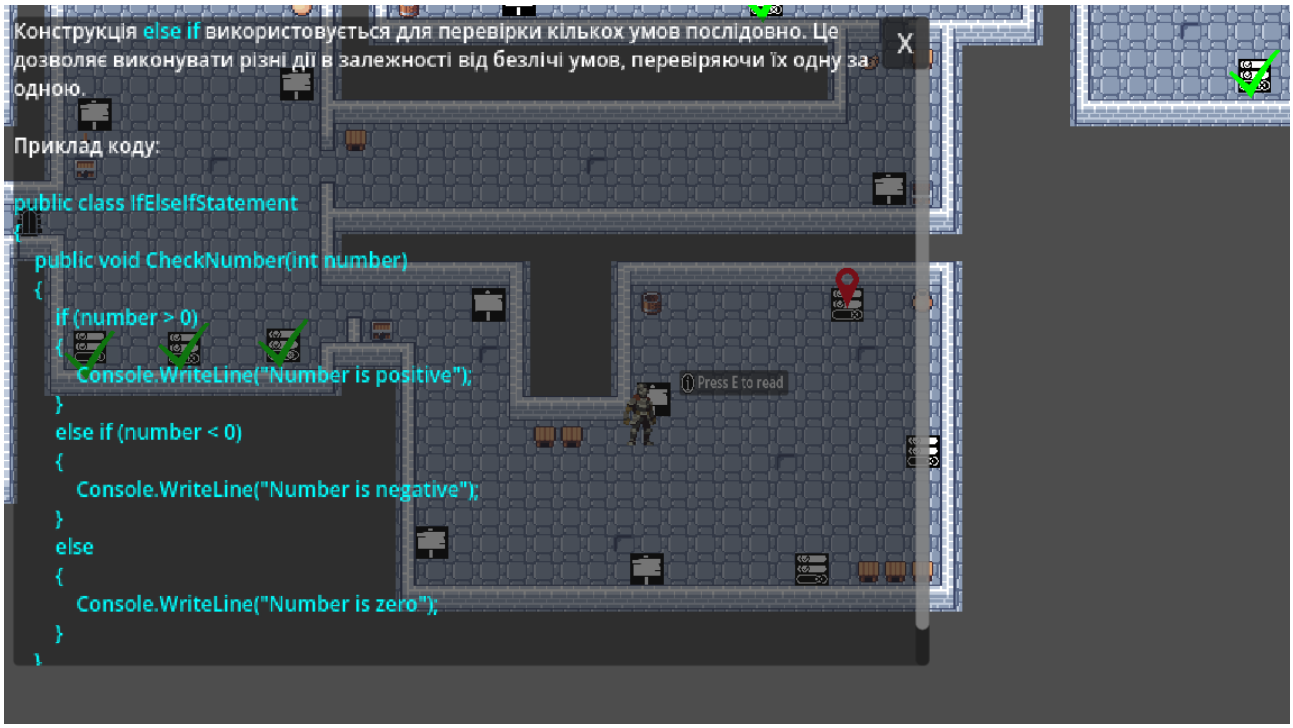

Рис. 2.52 Читаємо теоретичну інформацію

Рис. 2.53 Переходимо до запитань

Рис. 2.54 Заповнюємо пропуски в коді

Рис. 2.55 Відповідаємо правильно на останнє питання, бачимо, що маркер залишився на місці, що значить, що відчинились останні двері

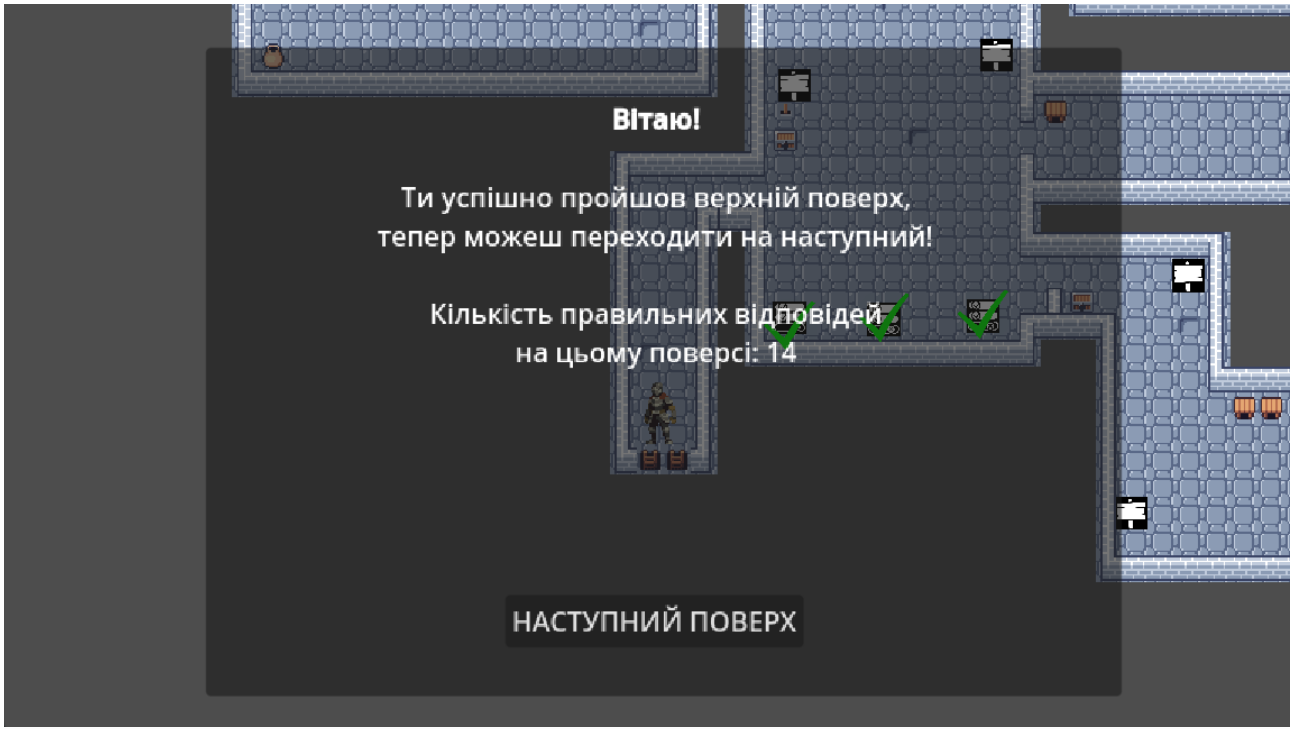

Рис. 2.56 Проходимо крізь відчинені двері та отримуємо таке повідомлення

Рис. 2.57 Після того як натиснули на кнопку «Наступний поверх» опинились на наступному рівні

Рис. 2.58 Читаємо перший знак на поверсі

Рис. 2.59 Оскільки алгоритм такий самий як і на першому поверсі, то показую одразу останню кімнату, де ми читаємо оголошення

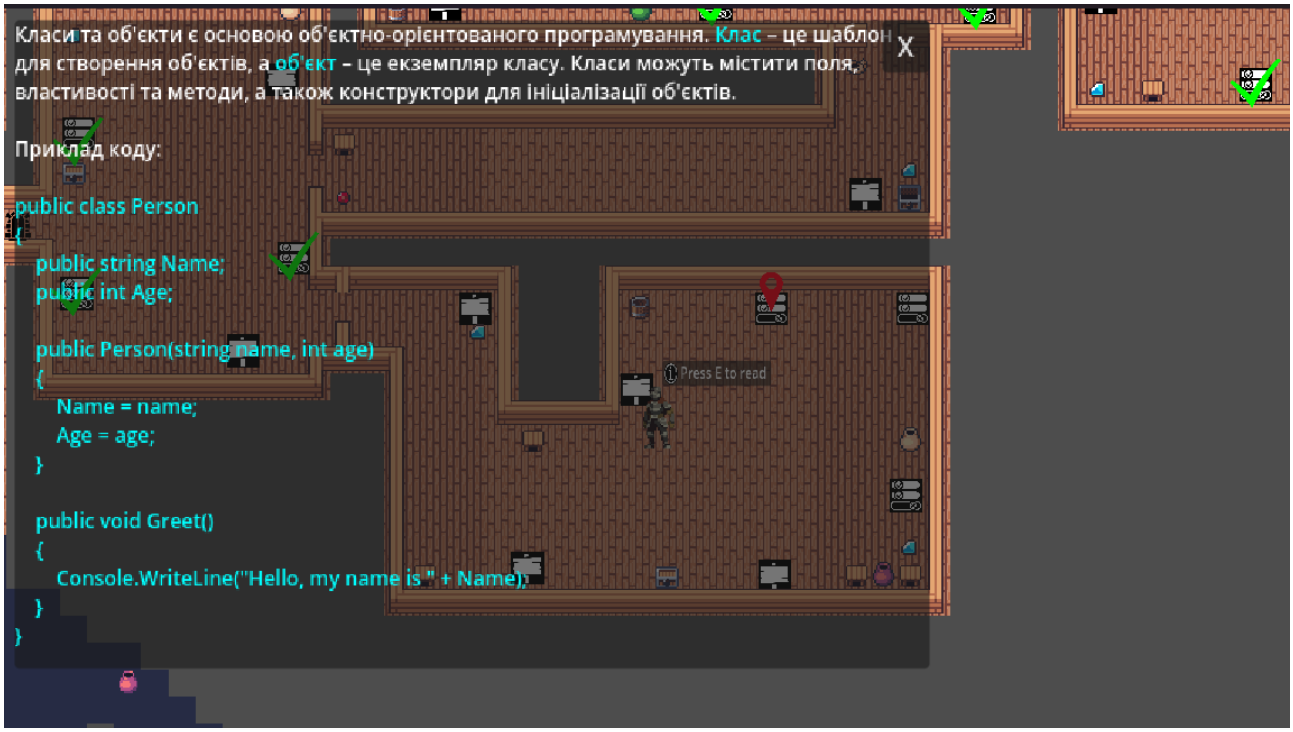

Рис. 2.60 Читаємо теоретичні матеріли

Рис. 2.61 Відповідаємо на фінальне запитання

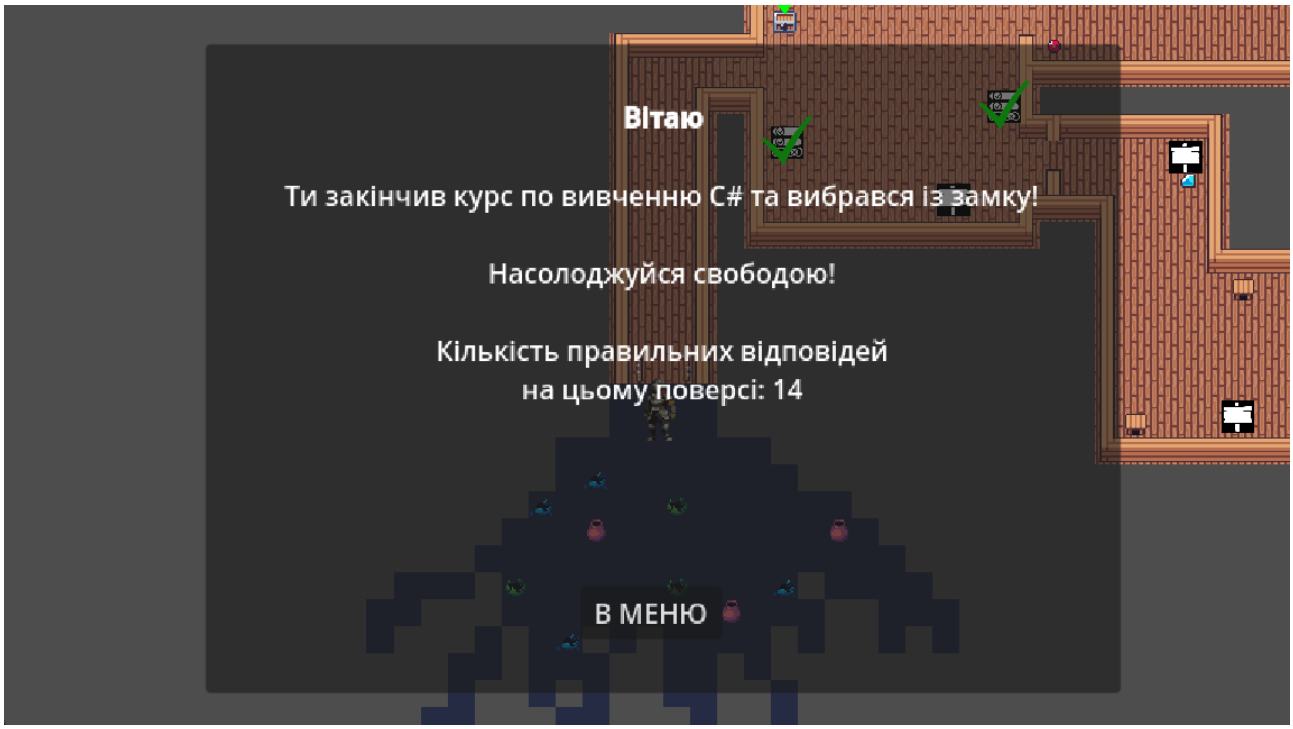

Рис. 2.62 Проходимо крізь останні відчинені двері та отримуємо таке повідомлення, після чого можемо повернутись в меню

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості розробки програмного забезпечення

Вхідні дані:

1. передбачуване число операторів (q) – 780;
2. коефіцієнт складності програми (C) – 1,9;
3. коефіцієнт корекції програми в ході її розробки (p) – 0,09;
4. годинна заробітна плата програміста, грн / год – 149, 95 грн/год;

За допомогою фільтрів на сайті «GameDev DOU – Українська спільнота геймдев спеціалістів» (https://gamedev.dou.ua/) було з'ясовано, що студент, зі стажем менше як 1 рік – 2 роки у сфері Game Development отримує приблизно 681 доларів США на місяць. Ця інформація актуальна на грудень 2023, але заробітна плата для Junior Software Engineer в GameDev залишається незмінною станом на червень 2024. Враховуючи офіційний курс валют Національного банку України на період розробки програми та на грудень 2023, який становить 37,98, отримуємо, що середня місячна заробітна плата складає 681 * 37,98 = 25, 864 гривень. Оскільки середня кількість робочих днів складає 22 дні, а кількість робочих годин – 8 годин на день, то загальна кількість робочих годин на місяць – 176 годин, отже середня заробітна плата програміста за годину складає 25, 864 / 176 = 149, 95 гривні.

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (B) – 1,4;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності (k) – 0,8;
7. вартість машино-години ЕОМ – 0,68 грн.

Оскільки мій ноутбук, назва моделі якого – HP Laptop 15, витрачає 41 Вт·год = 0,041 кВт·год, згідно характеристикам, а вартість електроенергії на період написання програми, з 01.09.2023 по 31.05.2024, за постановою Кабінету

Міністрів України становить 2,64 грн/кВт·год, то витрати електроенергії за годину складають 0,041 * 2,64 = 0,19 грн.

Щомісячно я сплачую тарифну плату за домашній інтернет від провайдера «Київстар» у розмірі 350 гривень, враховуючи, що в період з 01.09.2023 по 31.05.2024, середня тривалість місяця складала 30 днів, то отримуємо загальну кількість годин на місяць – 720. Базуючись на даному аналізі, маємо, що вартість користування інтернетом на годину складає 350 грн / 720 год = 0,49 грн/год. З цих результатів маємо, що вартість машино-години ЕОМ складає 0,19 + 0,49 = 0, 68 грн.

Враховуючи творчий характер роботи програміста, нормування праці в процесі створення ПЗ стало значно складнішим. Тому трудомісткість розробки ПЗ може бути розрахована за допомогою системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_0 + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \quad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \quad (3.2)$$

де q– передбачуване число операторів – 780;

C – коефіцієнт складності програми – 1,9;

p – коефіцієнт кореляції програми в ході її розробки – 0,09.

За формулою (3.2) умовне число операторів складає:

$$Q = 780 \cdot 1{,}9 \cdot (1 + 0{,}09) = 1615{,}38$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста за формулою (3.3) складають:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k} \text{ людино-годин,} \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,4;

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8.

Результат:

$$t_u = \frac{1615{,}38 \cdot 1{,}4}{80 \cdot 0{,}8} = 35{,}34 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі за формулою (3.4) складають:

$$t_a = \frac{Q}{(20..25) \cdot k}, \text{ людино-годин,} \qquad (3.4)$$

де Q – умовне число операторів – 1615,38;

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8.

Результат:

$$t_a = \frac{1615{,}38}{20 \cdot 0{,}8} = 100{,}7 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі за формулою (3.5) складають:

$$t_n = \frac{Q}{(20..25) \cdot k}, \text{ людино-годин,} \qquad (3.5)$$

де Q – умовне число операторів – 1615,38;

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8.

Результат:

$$t_n = \frac{1615{,}38}{25 \cdot 0{,}8} = 80{,}77 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання за формулою (3.6) складають:

$$t_{отл} = \frac{Q}{(4..5) \cdot k}, \text{ людино-годин,} \qquad (3.6)$$

Результат:

$$t_{отл} = \frac{1615{,}38}{5 \cdot 0.8} = 403{,}84 \text{ людино-годин.}$$

– за умови комплексного налагодження завдання за формулою (3.7) складають:

$$t_{отл}^{k} = 1{,}5 \cdot t_{отл}, \text{ людино-годин,} \qquad (3.7)$$

Результат:

$$t_{отл}^{k} = 1{,}5 \cdot 403{,}84 = 605{,}76 \text{ людино-годин.}$$

Витрати праці на підготовку документації за формулою (3.8) складають:

$$t_{д} = t_{др} + t_{до}, \text{ людино-годин,} \qquad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису;

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації.

Трудомісткість підготовки матеріалів і рукопису ($t_{др}$) визначаємо за формулою (3.9)

$$t_{др} = \frac{Q}{(15..20) \cdot k}, \text{ людино-годин,} \qquad (3.9)$$

Результат:

$$t_{др} = \frac{1615{,}38}{20 \cdot 0{,}8} = 100{,}96 \text{ людино-годин.}$$

Трудомісткість підготовки матеріалів і рукопису ($t_{до}$) визначаємо за формулою (3.10)

$$t_{до} = 0{,}75 \cdot t_{др}, \text{ людино-годин,} \qquad (3.10)$$

Результат:

$$t_{до} = 0{,}75 \cdot 100{,}96 = 75{,}72 \text{ людино-годин.}$$

Повертаємось до розрахунку витрати праці на підготовку документації за формулою (3.8)

Результат:

$$t_д = 100{,}96 + 75{,}72 = 176{,}68 \text{ людино-годин.}$$

Розрахувавши всі показники, за формулою (3.1) отримаємо трудомісткість розробки програмного забезпечення:

$t = 50 + 35{,}34 + 100{,}7 + 80{,}77 + 403{,}84 + 176{,}68 = 847{,}33,$ людино-годин.

3.2 Розрахунок витрат на створення програми

Витрати на створення ПЗ складаються із заробітної плати розробника та вартості машинного часу, який використовується для налагодження програми на ЕОМ та розраховується за формулою (3.11):

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \qquad (3.11)$$

де $З_{ЗП}$ – витрати на заробітну плату програміста;

$З_{МВ}$ – витрати машинного часу.

Заробітна плата програміста визначається за формулою (3.12):

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн ,} \qquad (3.12)$$

де t – загальна трудомісткість – 847,33 людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста – 149,95 грн/год.

Результат:

$$З_{ЗП} = 847{,}33 \cdot 149{,}95 = 127\,057{,}13 \text{ грн}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ розраховується за формулою (3.13):

$$З_{мв} = t_{отл} \cdot С_{мч}, \text{ грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ (403,84год);

$С_{мч}$ – вартість машино-години ЕОМ (0,68 грн/год).

Результат:

$$З_{мв} = 403{,}84 \cdot 0{,}68 = 274{,}61 \text{ грн.}$$

Повертаємось до розрахунку витрат на створення ПЗ за формулою (3.11):

$$К_{ПО} = 127\,057{,}13 + 274{,}61 = 127331{,}74 \text{ грн.}$$

Очікуваний період створення ПЗ визначаємо за формулою (3.14)

$$T = \frac{t}{B_k \cdot Fp}, \text{ міс,} \qquad (3.13)$$

де t – загальна трудомісткість (847,33 людино-годин);

B_k – число виконавців (1);

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Результат:

$$T = \frac{847{,}33}{1 \cdot 176} = 4{,}81 \text{ міс}$$

Висновок: для розробки інтелектуальної платформи-гри знадобиться витратити 847,33 людино-години, враховуючи всі вхідні дані маємо, що для створення продукту одним розробником-початківцем, який працює 5 днів на тиждень по 8 годин, знадобиться майже 5 місяців. Вартість даного застосунку складає 127 331,74 гривень. Розрахунки включають витрати на заробітну плату програміста та машинний час, враховуючи всі необхідні коефіцієнти та вартість ресурсів. Це дозволяє оцінити не лише трудові, а й фінансові аспекти розробки програмного забезпечення.

ВИСНОВКИ

Метою даної кваліфікаційної роботи було створення інтерактивної платформи для вивчення мови програмування C# через ігрову програму з використанням Godot Engine. У ході роботи було реалізовано гру, в якій користувачі вивчають основи C# за допомогою виконання ігрових завдань та взаємодії з різними об’єктами.

У процесі розробки були застосовані сучасні методи та інструменти, що забезпечують високу якість та продуктивність програми. Гра орієнтована операційну систему Windows, що робить її доступною широкої аудиторії користувачів. Програмування було виконано мовою C#, що дозволило реалізувати всі необхідні функції та забезпечити стабільність роботи програми.

Основне призначення розробленої ігрової програми – навчання основ програмування мовою C#. Воно сприяє розвитку логічного мислення, уваги та навичок вирішення завдань у користувачів. У грі реалізовані такі функції:

– рух головного героя;
– взаємодія із об’єктами;
– виконання завдань із програмування;
– перехід між рівнем.

Економічний аналіз показав, що проект є життєздатним та економічно доцільним. Розробка подібного додатку потребує значних ресурсів та часу, але кінцевий продукт має велике освітнє значення та потенціал для подальшого розвитку та інтеграції з онлайн-платформами.

Проект може бути розширений додаванням нових рівнів, завдань та функціоналу, таких як штучний інтелект для взаємодії з гравцем, онлайн-змагання та додаткові навчальні модулі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development by Mark J. Price
2. C# Programming Yellow Book by Rob Miles
3. Clean Code: A Handbook of Agile Software Craftsmanship by Robert C. Martin
4. Design Patterns: Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
5. Game Programming Patterns by Robert Nystrom
6. Head First Design Patterns by Eric Freeman, Bert Bates, Kathy Sierra, and Elisabeth Robson
7. Sams Teach Yourself C# in 24 Hours by Scott J. Dorman
8. The C# Player's Guide by RB Whitaker
9. Codecademy Programming Courses URL: https://www.codecademy.com/
10. Coursera Programming Courses URL: https://www.coursera.org/
11. Godot Engine Documentation URL: https://docs.godotengine.org/en/stable/
12. Godot Game Development Projects URL: https://www.packtpub.com/product/godot-game-development-projects/9781788831505
13. HP Laptop Specifications URL: https://www.hp.com/
14. Intel Processor Information URL: https://www.intel.com/
15. Introduction to Algorithms URL: https://mitpress.mit.edu/books/introduction-algorithms
16. Microsoft C# Documentation URL: https://docs.microsoft.com/en-us/dotnet/csharp/
17. Official Godot Tutorials URL: https://godotengine.org/learn

18. Sams Teach Yourself Unreal Engine 4 Game Development in 24 Hours by Aram Cookson

19. The C++ Programming Language by Bjarne Stroustrup URL: https://docs.unrealengine.com/

20. Unity Learn Platform URL: https://learn.unity.com/

21. Udacity Programming Courses URL: https://www.udacity.com/

ДОДАТОК А

ВИХІДНИЙ КОД КОМП’ЮТЕРНОЇ ПРОГРАМИ

Лістинг AudioPlayer.cs

```
using System.Collections.Generic;
using Godot;

public partial class AudioPlayer : AudioStreamPlayer2D
{
// статичний лінк на екземпляр класу AudioPlayer
public static AudioPlayer refer;
public override void _Ready()
{
	base._Ready();
	// встановлюю лінк на теперішній екземпляр класу AudioPlayer
	refer = this;
}

//метод для відтворення аудио по вказаному шляху
public static void Play(string path)
{
	// завантажую аудіопоток з ресурсу
	refer.Stream = ResourceLoader.Load<AudioStream>(path);
	refer.Play();
}
// метод для випадкового відтворення звуку кроків
public static void PlayRandomStepSound()
{
	// створюю список шляхів до звуків кроків та відтворюю
	Play(new List<string>
	{
		"res://Sounds/step1.ogg",
		"res://Sounds/step2.ogg",
		"res://Sounds/step3.ogg",
		"res://Sounds/step4.ogg",
		"res://Sounds/step5.ogg",
	//генерую випадковий індекс для вибору випадкового звуку зі списку
	}[new RandomNumberGenerator().RandiRange(0, 4)]);
}
}
```

Лістинг BigPanel.cs

```
using Godot;

public partial class BigPanel : Panel
{
// статичний лінк на екземпляр класу BigPanel
```

```
public static BigPanel refer;
public override void _Ready()
{
	base._Ready();
	// встановлюю посилання на поточний екземпляр класу BigPanel
	refer = this;
	// додаю обробник події натискання кнопки X
	GetNode<Button>("X").Pressed += () => Visible = false;

}
// метод для відображення тексту на панелі
public static void ShowText(string what)
{
	GD.Print("Showing text..");
	// встановлюю текст у RichTextLabel і включаю можливість вибору тексту
	refer.GetNode<RichTextLabel>("RichTextLabel").Text = what;
	refer.GetNode<RichTextLabel>("RichTextLabel").SelectionEnabled = true;
	// роблю панель видимою
	refer.Visible = true;
}
// метод очищення панелі
public static void Clear()
{
	if (refer == null) return;
	GD.Print("Clearing big panel");
	// роблю панель невидимою
	refer.Visible = false;
}
}
```

Лістинг BigPanelFillIn.cs

```
using System;
using System.Collections.Generic;
using Godot;

public partial class BigPanelFillIn : Panel
{
// статичне посилання на екземпляр класу BigPanelFillIn
public static BigPanelFillIn refer;

public override void _Ready()
{
	base._Ready();
	// встановлюю лінк на поточний екземпляр класу BigPanelFillIn
	refer = this;
	// додаю обробник події натискання кнопки X
	GetNode<Button>("X").Pressed += () =>
	{
		Visible = false;
```

```
                Movement.CanMove = true;
            };
            // додаю обробник події натискання кнопки Confirm
            refer.GetNode<Button>("Confirm").Pressed += () =>
            {
                submission();
            };

            var texEdit = GetNode<TextEdit>("TextEdit");
            // відключаємо можливість руху під час фокусування на текстовому полі
            texEdit.FocusEntered += () =>
            {
                Movement.CanMove = false;
            };
            texEdit.FocusExited += () =>
            {
                Movement.CanMove = true;
            };
        }
        // статична дія для обробки відправки
        static Action submission = () => { GD.Print("Attempting"); };

        // метод для відображення тексту та перевірки відповідей
        public static void ShowText(string what, List<string> answers, Node2D us)
        {
            refer.Visible = true;

            string correct = what;
            // замінюю маркери "???" у тексті на відповіді
            while (answers.Count > 0)
            {
                for (int i = 0; i < correct.Length; i++)
                {
                    if (i >= 2 && correct[i] == '?' && correct[i - 1] == '?' && correct[i - 2]
== '?')
                    {
                        i -= 2;
                        correct = correct.Remove(i, 3);
                        correct = correct.Insert(i, answers[0]);
                        answers.RemoveAt(0);
                        break;
                    }
                }
            }

            var edit = refer.GetNode<TextEdit>("TextEdit");
            edit.Text = what;

            submission = () =>
            {
```

```
                // перевірка правильності відповіді
                if (edit.Text == correct)
                {
                        us.GetNode<Sprite2D>("Correct").Visible = true;
                        us.GetNode<Node2D>("UI").Visible = false;
                        AudioPlayer.Play("res://Sounds/kaching.ogg");
                        refer.Visible = false;
                        Movement.CanMove = true;
                        us.SetProcess(false);
                        (us as Prompt).Interacted();
                }
        };
    }
    // метод очищення панелі
    public static void Clear()
    {
        if (refer == null) return;
        GD.Print("Clearing text..");
        refer.Visible = false;
    }
    }
```

Лістинг BigSign.cs

```
    using System;
    using Godot;

    [Tool]
    public partial class BigSign : Node2D, Prompt
    {
    // експортоване текстове поле для відображення тексту на знаку
    [Export(PropertyHint.MultilineText)] public string text;
    // властивість керувати активністю знака
    bool Active
    {
        get => _active;
        set
        {
            if (value == _active)
                return;

            if (value)
                BigPanel.ShowText(text); // показую текст на панелі, якщо знак активний
            else
                BigPanel.Clear(); // очищую панель, якщо знак неактивний
            _active = value;
        }
    } bool _active;
```

```
	bool lastPressed = false;
	public override void _Process(double delta)
	{
		base._Process(delta);
		if (Engine.IsEditorHint())
			return;

		// якщо гравець близько, показуємо UI
		var close = Movement.Player.GlobalPosition.DistanceTo(GlobalPosition) <= 50 &&
IsActive;
		GetNode<Node2D>("UI").Visible = close;
		if (!close)
			Active = false;

		var pressed = Input.IsKeyPressed(Key.E);
		if (close && !lastPressed && pressed)
		{
			GD.Print("Showing sign " + Name);
			lastPressed = true;
			Active = !Active;
			Interacted();
		}
		lastPressed = pressed;
	}
	// експортований вузол для наступного кроку
	[Export]
	public Node next;
	// функція, що визначає, чи це стало наступним кроком
	public Func<bool> BecameNext { get; set; } = () => false;
	// дія, що виконується при взаємодії
	public Action Interacted { get; set; } = () => { };
	// інтерфейс для наступного кроку
	public Prompt Next => (Prompt)next;
	// властивість керувати активністю
	public bool IsActive { get; set; } = false;
	}
```

Лістинг Door.cs

```
	using System;
	using Godot;

	public partial class Door : Node2D, Prompt
	{
	// еспортований вузол, що вказує на наступний крок або вузол
	[Export]
	public Node next;
	// функція, що визначає, чи це стало наступним кроком
	public Func<bool> BecameNext { get; set; } = () => false;
	// дія, що виконується при взаємодії
```

```
public Action Interacted { get; set; } = () => { };
// інтерфейс для наступного кроку
public Prompt Next => (Prompt)next;
// властивість керувати активністю
public bool IsActive { get; set; } = false;

public override void _Ready()
{
	base._Ready();
	// визначаємо поведінку під час переходу до наступного кроку
	BecameNext = () =>
	{
		// переміщуємо двері на нову позицію по осі Y
		GlobalPosition = new Vector2(GlobalPosition.X, 999);
		return true;
	};
}
}
```

Лістинг Exit1.cs

```
using Godot;

public partial class Exit1 : Node2D
{

public override void _Process(double delta)
{

	// перевіряю, чи гравець близько до виходу (менше 50 одиниць)
	if (Movement.Player.GlobalPosition.DistanceTo(GlobalPosition) < 50f)
	{
		// зупиняю обробку цього вузла
		SetProcess(false);
		Movement.Player.GetNode<Control>("Panel3").Visible = true;
		// додаю обробник натискання кнопки для переходу на наступний рівень
		Movement.Player.GetNode<Button>("Panel3/Button").Pressed += () =>
		{
			GetTree().ChangeSceneToFile("res://Level2.tscn"); // перехід на
другий рівень
		};
	}
}
}
```

Лістинг Exit.cs

```
using Godot;

public partial class Exit : Node2D
```

```
{

public override void _Process(double delta)
{
	// перевіряю, чи гравець близько до виходу (менше 50 одиниць)
	if (Movement.Player.GlobalPosition.DistanceTo(GlobalPosition) < 50f)
	{
		// зупиняю обробку цього вузла
		SetProcess(false);
		// показую панель із результатами
		Movement.Player.GetNode<Control>("Panel2").Visible = true;
		// додаю обробник натискання кнопки для переходу на головний екран
		Movement.Player.GetNode<Button>("Panel2/Button").Pressed += () =>
		{
			GetTree().ChangeSceneToFile("res://MainMenu.tscn"); // Переход к
MainMenu.tscn
		};
	}
}
}
```

Лістинг FillInQuestion.cs

```
using Godot;
using System;
using System.Linq;

[Tool]
public partial class FillInQuestion : Node2D, Prompt
{
// експортоване текстове поле для відображення питання
[Export(PropertyHint.MultilineText)] public string text;
// експортоване текстове поле для зберігання відповідей, розділених комами
[Export(PropertyHint.MultilineText)] public string answers;
// властивість керувати активністю питання
bool Active
{
	get => _active;
	set
	{
		if (value == _active)
			return;

		if (value)
		{
			GD.Print("Showing 0");
			// показати текст питання та можливі відповіді на панелі
			BigPanelFillIn.ShowText(text, answers.Split(',').ToList(), this);
		}
		else
```

```
                        BigPanel.Clear(); // очистити панель, якщо питання неактивне
                    _active = value;
                }
        } bool _active;

        bool lastPressed = false;
        public override void _Process(double delta)
        {
                base._Process(delta);
                if (Engine.IsEditorHint())
                        return;

                // якщо гравець знаходиться близько до питання, показати UI
                var close = Movement.Player.GlobalPosition.DistanceTo(GlobalPosition) <= 50 &&
IsActive;
                GetNode<Node2D>("UI").Visible = close;
                if (!close)
                        Active = false;

                var pressed = Input.IsKeyPressed(Key.E);
                if (close && !lastPressed && pressed && !Active)
                {
                        lastPressed = true;
                        Active = !Active;
                }
                lastPressed = pressed;
        }
        // експортований вузол для наступного кроку
        [Export]
        public Node next;
        // функція, що визначає, чи це стало наступним кроком
        public Func<bool> BecameNext { get; set; } = () => false;
        // дія, що виконується при взаємодії
        public Action Interacted { get; set; } = () => { };
        // інтерфейс для наступного кроку
        public Prompt Next => (Prompt)next;
        // властивість керувати активністю
        public bool IsActive { get; set; } = false;
        }
```

Лістинг MainMenu.cs

```
using Godot;
public partial class MainMenu : Control
{
public override void _Ready()
{
        base._Ready();
        // встановлюю дії для кнопок рівня
```

```
		GetNode<Button>("Level1").Pressed += () => GetTree().ChangeSceneToFile("res://Level.tscn");
		GetNode<Button>("Level2").Pressed += () => GetTree().ChangeSceneToFile("res://Level2.tscn");

		// вузол AudioStreamPlayer та відтворення фонової музики
		var menuMusic = GetNode<AudioStreamPlayer>("MenuMusic");
		menuMusic.Stream = GD.Load<AudioStream>("res://Sounds/mainmenu.ogg");
		menuMusic.Play();

		// вузол RichTextLabel для заголовка та вузол TextEdit для введення імені
		var title = GetNode<RichTextLabel>("Title");
		var nameInput = GetNode<TextEdit>("NameInput");
		// встановлюю обробник зміни тексту у полі введення імені
		nameInput.TextChanged += () => title.Text = "[center]Welcome, " + (nameInput.Text != "" ? nameInput.Text : "stranger") + "!";
	}
}
```

Лістинг Marker.cs

```
using System;
using Godot;

// інтерфейс Prompt для вузлів, які можуть бути частиною послідовності
public interface Prompt
{
public Func<bool> BecameNext { get; set; }
public Action Interacted { get; set; }
public Prompt Next { get; }
public bool IsActive { get; set; }
}
// клас Marker, який успадковує Sprite2D для керування послідовністю вузлів
public partial class Marker : Sprite2D
{
// властивість Current для управління поточним вузлом у послідовності
public Node Current
{
	get => _current;
	set
	{
		// поки value не дорівнює null і наступний вузол став активним
		while (value != null && (value as Prompt).BecameNext())
			value = ((value as Prompt).Next as Node);
		// якщо value не дорівнює null, встановлюю IsActive в true
		if (value != null)
			(value as Prompt).IsActive = true;
		// оновлюю поточне значення та позицію
		_current = value;
		GlobalPosition = ((Node2D)value).GlobalPosition + new Vector2(0,-10);
```

```
        }
}
// експортоване поле для зберігання поточного вузла
[Export]
Node _current = null;

public override void _Ready()
{
        // встановлюю початкову позицію маркера
        GlobalPosition = ((Node2D)Current).GlobalPosition + new Vector2(0,-10);
        // підписуюсь на подію Interacted поточного вузла
        ((Prompt)Current).Interacted += a;
        // локальна функція для обробки взаємодії з поточним вузлом
        void a()
        {
                // відписуюсь від попереднього вузла і переходимо до наступного вузла
                ((Prompt)Current).Interacted -= a;
                Current = (Node)((Prompt)Current).Next;
                ((Prompt)Current).Interacted += a;
        }
}
}
```

Лістинг Movements.cs

```
using Godot;

public partial class Movement : CharacterBody2D
{
// статична змінна для керування можливістю руху
public static bool CanMove = true;
// статичний лінк гравця
public static Node2D Player;
// властивість отримання спрайту гравця
Sprite2D sprite => GetNode<Sprite2D>("Sprite2D");

// час останньої анімації та останнього звуку кроку
float timeOfLastAnimation;
float timeOfLastStepSound;

public override void _Ready()
{
        base._Ready();
        // встановлюю посилання на гравця за готовності вузла
        Player = this;
}

// властивість для керування станом руху
bool Moving
{
```

```
                get => _moving;
                set
                {
                        if (value == _moving)
                                return;

                        // змінюю текстуру спрайту в залежності від стану руху
                        sprite.Texture = ResourceLoader.Load<Texture2D>(value
                                ?
"res://FreeKnight_v1/Colour1/NoOutline/120x80_PNGSheets/_Run.png"
                                :
"res://FreeKnight_v1/Colour1/NoOutline/120x80_PNGSheets/_Idle.png");

                        _moving = value;
                }
        } bool _moving;

        public override void _Process(double delta)
        {
                base._Process(delta);

                // встановлюю швидкість руху (збільшується при натисканні Shift)
                float speed = Input.IsKeyPressed(Key.Shift) ? 4 : 2;

                // ініціалізую вектор швидкості
                Velocity = new Vector2(0, 0);

                if (CanMove)
                {
                        // перевіряю натискання клавіш WASD для керування рухом
                        if (Input.IsKeyPressed(Key.D))
                                Velocity += new Vector2(50 * speed, 0);

                        if (Input.IsKeyPressed(Key.A))
                                Velocity += new Vector2(-50 * speed, 0);
                        if (Input.IsKeyPressed(Key.W))
                                Velocity += new Vector2(0, -50 * speed);
                        if (Input.IsKeyPressed(Key.S))
                                Velocity += new Vector2(0, 50 * speed);
                }

                // якщо швидкість не нульова, керуємо анімацією та звуками кроків
                if (Velocity != Vector2.Zero)
                {
                        // змінюю напрямок спрайту залежно від напрямку руху
                        if (Velocity.X != 0)
                                sprite.Scale = Scale with { X = Velocity.X > 0 ? 1 : -1 };

                        Moving = true;
                        // керую анімацією
```

```
                if (Time.GetTicksMsec() - timeOfLastAnimation > 100)
                {
                        timeOfLastAnimation = Time.GetTicksMsec();
                        if (sprite.Frame == 9)
                                sprite.Frame = 0;
                        else
                                sprite.Frame++;
                }
                // керую звуками кроків
                if (Time.GetTicksMsec() - timeOfLastStepSound > 250)
                {
                        AudioPlayer.PlayRandomStepSound();
                        timeOfLastStepSound = Time.GetTicksMsec();
                }

        }
        else
                Moving = false;
        // рухаю персонажа з урахуванням колізій
        MoveAndSlide();
}
}
```

Лістинг Sign.cs

```
using System;
using Godot;

[Tool]
public partial class Sign : Node2D, Prompt
{
// експортована текстова властивість для відображення тексту на знаку
[Export]
public string text
{
        get
        {
                // перевіряю наявність вузла RichTextLabel та повертаю його текст
                if (!HasNode("UI/Text/Panel/RichTextLabel")) return "";
                return GetNode<RichTextLabel>("UI/Text/Panel/RichTextLabel").Text;
        }
        set
        {
                // перевіряю наявність вузла RichTextLabel та встановлюю його текст
                if (!HasNode("UI/Text/Panel/RichTextLabel")) return;
                GetNode<RichTextLabel>("UI/Text/Panel/RichTextLabel").Text = value;
        }
}
// властивість керувати активністю знака
bool Active
```

```
{
	get => _active;
	set
	{
		// керую видимістю елементів UI залежно від активності
		GetNode<Control>(new("UI/Control")).Visible = !_active;
		GetNode<Control>(new("UI/Text")).Visible = _active;
		_active = value;
	}
} bool _active;

bool lastPressed = false;
public override void _Process(double delta)
{
	base._Process(delta);
	if (Engine.IsEditorHint())
		return;

	// якщо гравець знаходиться близько до знаку, показую UI
	var close = Movement.Player.GlobalPosition.DistanceTo(GlobalPosition) <= 50;
	GetNode<Node2D>("UI").Visible = close;
	if (!close)
		Active = false;

	// перевіряю натискання клавіші 'E' для взаємодії зі знаком
	var pressed = Input.IsKeyPressed(Key.E);
	if (close && !lastPressed && pressed)
	{
		lastPressed = true;
		Active = !Active;
		Interacted();
	}
	lastPressed = pressed;
}

// експортований вузол для наступного кроку
[Export]
public Node next;
// функція, що визначає, чи це стало наступним кроком
public Func<bool> BecameNext { get; set; } = () => false;
// дія, що виконується при взаємодії
public Action Interacted { get; set; } = () => { };
// інтерфейс для наступного кроку
public Prompt Next => (Prompt)next;
// властивість для керування активністю
public bool IsActive { get; set; } = false;
}
```

Лістинг Submit.cs

```
using System;
using System.Collections.Generic;
using Godot;

[Tool]
public partial class Submit : Node2D, Prompt
{
// експортований вузол, що вказує на наступний крок або вузол
[Export]
public Node next;
// функція, що визначає, чи це стало наступним кроком
public Func<bool> BecameNext { get; set; } = () => false;
// дія, що виконується при взаємодії
public Action Interacted { get; set; } = () => { };
// інтерфейс для наступного кроку
public Prompt Next => (Prompt)next;
// властивість керувати активністю
public bool IsActive { get; set; } = false;

// експортована правильна відповідь
[Export] public string CorrectAnswer;

// експортоване текстове поле для відображення тексту питання
[Export(PropertyHint.MultilineText)]
public string QuestionText
{
	get
	{
		if (!HasNode("UI/Question/RichTextLabel")) return "";
		return GetNode<RichTextLabel>("UI/Question/RichTextLabel").Text;
	}
	set
	{
		if (!HasNode("UI/Question/RichTextLabel")) return;
		GetNode<RichTextLabel>("UI/Question/RichTextLabel").Text = value;
	}
}

public override void _Ready()
{
	base._Ready();
	if (Engine.IsEditorHint())
		return;

	// отримую всі кнопки вибору відповіді
	List<Button> buttons = new()
	{
		GetNode<Button>("UI/Question/A"),
		GetNode<Button>("UI/Question/B"),
		GetNode<Button>("UI/Question/C"),
```

```
                GetNode<Button>("UI/Question/D"),
            };

            foreach (var _ in buttons)
            {
                // обробник натискання кнопки
                _.Pressed += () =>
                {
                    GD.Print("Button clicked");

                    // перевіряю, чи є відповідь правильною
                    GetNode<Node2D>(_.Name == CorrectAnswer ? "Correct" :
"Incorrect").Visible = true;
                    if (_.Name == CorrectAnswer)
                        GetNode<Sprite2D>("Incorrect").Visible = false;

                    if (_.Name == CorrectAnswer)
                    {
                        // програю звук
                        AudioPlayer.Play("res://Sounds/kaching.ogg");
                        Interacted();
                        SetProcess(false);
                    }

                    // відключаю видимість UI
                    GetNode<Node2D>("UI").Visible = false;
                };
            }
        }

        public override void _Process(double delta)
        {
            base._Process(delta);
            // закінчую виконання логіки у редакторі
            if (Engine.IsEditorHint())
                return;
            if (!IsActive)
                return;
            // відображаю UI, якщо гравець знаходиться близько
            GetNode<Node2D>("UI").Visible =
Movement.Player.GlobalPosition.DistanceTo(GlobalPosition) < 50;
        }
        }
```

Лістинг TextSubmit.cs

```
        using System;
        using Godot;

        [Tool]
```

```
public partial class TextSubmit : Node2D, Prompt
{
// експортований вузол, що вказує на наступний крок або вузол
[Export]
public Node next;
// функція, що визначає, чи це стало наступним кроком
public Func<bool> BecameNext { get; set; } = () => false;
// дія, що виконується при взаємодії
public Action Interacted { get; set; } = () => { };
// інтерфейс для наступного кроку
public Prompt Next => (Prompt)next;
// властивість керувати активністю
public bool IsActive { get; set; } = false;

// експортована правильна відповідь
[Export] public string CorrectAnswer;

// експортоване текстове поле для відображення тексту питання
[Export]
public string QuestionText
{
	get
	{
		if (!HasNode("UI/Question/RichTextLabel")) return "";
		return GetNode<RichTextLabel>("UI/Question/RichTextLabel").Text;
	}
	set
	{
		if (!HasNode("UI/Question/RichTextLabel")) return;
		GetNode<RichTextLabel>("UI/Question/RichTextLabel").Text = value;
	}
}

public override void _Ready()
{
	base._Ready();
	if (Engine.IsEditorHint())
		return;

	// забороняю рух гравцю, доки він редагує текст
	var textEdit = GetNode<TextEdit>("UI/Question/TextEdit");
	textEdit.FocusEntered += () =>
	{
		Movement.CanMove = false;
	};
	textEdit.FocusExited += () =>
	{
		Movement.CanMove = true;
	};
```

```
            // додаю подію натискання кнопки
            GetNode<TextureButton>("UI/Question/TextureButton").Pressed += () =>
            {
                // приховую UI
                GetNode<Node2D>("UI").Visible = false;

                // перевіряю введений текст
                var entered = GetNode<TextEdit>("UI/Question/TextEdit").Text;
                bool isCorrect = entered.ToLower() == CorrectAnswer.ToLower();
                if (isCorrect)
                {
                    AudioPlayer.Play("res://Sounds/kaching.ogg");
                    Interacted();
                    SetProcess(false);
                }
                GetNode<Sprite2D>(isCorrect ? "Correct" : "Incorrect").Visible = true;
                if (isCorrect)
                    GetNode<Sprite2D>("Incorrect").Visible = false;

                GD.Print("Submitted. The answer was " + (isCorrect ? "correct" : "incorrect"));

                // знову включаю можливість руху
                GetNode<Node2D>("UI").Visible = false;
                Movement.CanMove = true;
            };
        }

        public override void _Process(double delta)
        {
            base._Process(delta);
            if (Engine.IsEditorHint())
                return;

            if (!IsActive)
                return;
            // роблю UI видимим, якщо гравець знаходиться поблизу
            GetNode<Node2D>("UI").Visible                                                    =
Movement.Player.GlobalPosition.DistanceTo(GlobalPosition) <= 50;
        }
        }
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра на тему:

«Розробка інтелектуальної платформи для персоналізованого вивчення мови програмування з використанням технології Godot»

студентки групи 122-20-1

Єровенко Поліни Сергіївни

Керівник економічного розділу доц. **Л.В. Касьяненко**

каф. ПЕП та ПУ, к.е.н

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Єровенко П.С.диплом.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Єровенко П.С.диплом.pdf	Пояснювальна записка до кваліфікаційної роботи у форматі PDF
Програма	
Єровенко П.С.диплом.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Єровенко П.С.диплом.ppt	Презентація кваліфікаційної роботи