Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка 2D гри - платформера із використанням мови програмування Python

Виконав(ла): студент(ка) 4 курсу, групи _____
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ _____
(підпис) (прізвище та ініціали)

Керівник _____ _____
(підпис) (прізвище та ініціали)

Нормоконтроль _____ _____
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ _____
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії

(повна назва факультету)

Кафедра _____

(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)

«___»_____20 24 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр

(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

студенту _____

(прізвище, ім'я, по батькові)

1. Тема роботи Розробка 2D гри-платформера із використанням мови програмування Python

Керівник роботи _____

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20____року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи _____

4. Зміст роботи (перелік питань, які потрібно розробити)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка

Студент _____ Пановик Н.М
(підпис) (прізвище та ініціали)

Керівник роботи _____ Мудрик І.Я
(підпис) (прізвище та ініціали)

Реферат

Кваліфікаційна робота на здобуття освітнього ступеню «бакалавр» за спеціальністю 121 Інженерія програмного забезпечення. Тернопільський національний технічний університет ім. Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група ___, 2024 рік. Пояснювальна записка до кваліфікаційної роботи на здобуття освітнього ступеню «бакалавр» містить: 75 с., 46 рис.

Тема: Розробка 2D платформера із використанням PyGame.

Об'єктом дослідження є процес розробки 2D платформера за допомогою бібліотеки PyGame.

Метод дослідження – в основу роботи закладено дослідження процесу розробки 2D платформера з використанням PyGame та обґрунтування доцільності використання PyGame для створення гри.

Завданням даного проєкту є розробка 2D платформера для демонстрації можливостей бібліотеки PyGame.

Результат – гра, що повністю готова до використання.

Ключові слова: Python, PyGame, 2D.

Abstract

Qualification work for obtaining a bachelor's degree in specialty 121 Software engineering. Ternopil National Technical University named after Ivan Pulyuya, Faculty of Computer and Information Systems and Software Engineering, Department of Software Engineering, Group ___, 2024. The explanatory note to the qualification work for obtaining the bachelor's degree contains: 75 pp., 46 figures.

Topic: Development of a 2D platformer using PyGame.

The object of research is the process of developing a 2D platformer using PyGame libraries.

Research method – the basis of the work is the study of the process of developing a 2D platformer using PyGame and the justification of the feasibility of using PyGame to create a game.

The task of this project is to develop a 2D platformer to demonstrate the capabilities of the PyGame library.

The result is a game that is completely ready to use.

Keywords: Python, PyGame, 2D.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня

PyGame – набір модулів для мови програмування Python, створений для розробки відеоігор.

JSON – текстовий формат обміну даними між комп'ютерами

XML – розширювана мова розмітки

Sprite – об'єкт, який представляє будь-який візуальний елемент в грі

Tile – невеликий графічний елемент, який є частиною великої карти або рівня гри

Lua – швидка і компактна скриптова мова програмування

PNG – формат растрової графіки, який використовується для зберігання зображень з високою якістю

GIF – формат растрової графіки, який використовується для зберігання як статичних зображень, так і коротких анімацій

ВСТУП

Нині розробка ігор є актуальною темою у світі програмування. Ігрова індустрія постійно розвивається, шукає нові таланти та ідеї. Тому навички в розробці ігор можуть бути корисними для кар'єрного зростання в галузі програмування.

Розробка та використання 2D платформерів є досить поширеною серед програмістів та користувачів відповідно. Платформери, як жанр ігор, відомі своєю простотою в сприйнятті та водночас деякою складністю в розробці. Вони вимагають від розробника уваги до дрібниць, таких як фізика руху персонажа, взаємодія з об'єктами на рівні, анімація, керування тощо.

Python – це однією із найпопулярніших мов програмування, що дозволяє розробляти ігри різної складності та різних жанрів. Завдяки широкому спектру бібліотек і фреймворків, таких як Pygame, Panda3D, Pyglet та інших, розробники можуть створювати 2D та 3D ігри для різних платформ. Python приваблює своєю простотою в освоєнні, швидкістю розробки та великою кількістю ресурсів та документації, що допомагає у створенні різноманітних ігор.

Pygame, як бібліотека для розробки ігор на Python, надає зручний та потужний інструментарій для створення 2D ігор. Вона дозволяє легко керувати графікою, аудіо та взаємодією об'єктів у грі.

Метою цієї дипломної роботи дослідження та розробка 2D платформера з використанням допоміжної бібліотеки для Python – PyGame. Дослідження цієї теми дозволить краще зрозуміти можливості згаданої бібліотеки та розширить знання в галузі розробки ігор на Python.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Аналіз наявних аналогів

Аналіз наявних аналогів на тему дипломної роботи «Розробка 2D платформера із використанням PyGame».

У межах дипломної роботи проведено аналіз ринку і вибрано декілька прикладів аналогів, які розглянуті нижче.

«Super Mario Bros.» від Nintendo є прикладом 2D-платформера. Гра має лінійний геймплей з міцно пов'язаними рівнями і простим управлінням персонажем. Одною з переваг цієї гри є те, що нові види ворогів і перешкод вводяться поступово, щоб було цікаво гравцю грати. Секретні проходи і бонусні рівні додають заохочення до повторного проходження. Проте, до її недоліків можна віднести відносно просту графіку і обмежені можливості проходження рівнів, що може зробити її менш привабливою для гравців.

Рисунок 1.1.1 – Платформер «Super Mario Bros»

«Celeste», розроблений Maddy Makes Games, виділяється складним геймплеєм з акцентом на точність рухів. Історія розкривається через геймплей і персонажів, що робить гру цікавою. Використання механік, що змінюють фізику гри, додає цікавоті геймплею. Рівні з альтернативними шляхами також є великим

плюсом. Але висока складність може бути мінусом для деяких гравців, які можуть втратити інтерес до гри через часті помилки та невдачі.

Рисунок 1.1.2 – Платформер «Celeste»

Створений Team Cherry, "Hollow Knight" є грою з великим світом і нелінійним макетом рівнів. У грі є дуже добрий баланс платформи і бойових елементів, які дозволяють гравцеві покращувати свої навички, здібності. Атмосферний дизайн є головним аспектом, який занурює гравця у гру. Але така складна структура може бути недоліком для початківців, оскільки важко орієнтуватися у великому світі з багатьма шляхами [1].

Рисунок 1.1.3 – Платформер «Hollow Knight»

1.2 Обґрунтування вибору напрямку дослідження

Для обґрунтування вибору розробки 2D платформера із використанням Pygame потрібно врахувати декілька ключових аспектів, які охоплюють технічні, освітні та ринкові фактори.

Перш за все, вибір розробки 2D платформера є обґрунтованим з точки зору популярності жанру. Платформери займають важливе місце в історії відеоігор та залишаються актуальними і сьогодні завдяки своїй простоті в освоєнні та можливості створення цікавого ігрового процесу. Вивчення принципів створення такого роду ігор дозволяє ознайомитися з основами ігрової механіки, дизайном рівнів та керуванням персонажем, що є важливими аспектами для будь-якого розробника ігор.

Вибір Pygame як інструменту для розробки є обґрунтованим через його простоту використання і наявність великої кількості ресурсів для навчання. Pygame є однією з найпоширеніших бібліотек для створення двовимірних ігор на Python, що надає всі необхідні засоби для розробки: від обробки подій і керування графікою до роботи зі звуком. Це чудовий вибір для новачків завдяки доступності та великій кількості навчальних матеріалів і прикладів.

З точки зору освітнього процесу розробка 2D платформера з використанням Pygame дозволяє не лише застосувати на практиці теоретичні знання з програмування, але й розвинути навички проєктного мислення, роботи в команді та вирішення комплексних задач. Процес створення гри включає в себе різноманітні етапи, такі як проєктування, програмування, тестування та налагодження, що сприяє всебічному розвитку.

Розробка 2D платформера також має велике значення з точки зору карєрних можливостей. Індустрія відеоігор продовжує скоро зростати, і 2D ігри займають велику частину цього ринку. Створення власного проєкту дозволяє отримати практичний досвід, який може бути корисним для подальшого працевлаштування в цій галузі. Крім того, успішний проєкт може стати основою для власного бізнесу.

1.3 Постановка завдання для розробки 2D платформера

На основі аналізу прототипів з вибраної області можна виконати параметризацію завдання і спроєктувати основні функції ресурсу.

Основними завданнями створення 2D платформера є:

− проаналізувати і систематизувати характеристики 2D платформерів різних розробників;

− здійснити аналіз програмного забезпечення для вибору оптимального способу реалізації проєкту;

− розробити графічний інтерфейс;

− розробити інструменти для створення і відображення ігрових елементів;

− провести тестування розробленої гри.

Все це базується на детальному аналізі і дослідженні поточного стану 2D платформерів, а також – аналізі використовуваних інструментів для розробки гри із використанням PyGame.

2 ПРОЄКТУВАННЯ ГРИ

2.1 Формалізація вимог до системи

Система повинна забезпечувати всі необхідні функції і механіки для створення цікавого 2D платформера. Вимоги до системи включають наступні пункти:

1. Ігрова логіка:
    - Реалізація стрибків, бігу і інших дій персонажа.
    - Обробка зіткнень між персонажем і об'єктами оточення.
    - Керування станами персонажа (рух, стояння, стрибки).
2. Графіка і анімація:
    - Відображення персонажів, фону і інших об'єктів.
    - Реалізація анімацій для різних дій персонажа (біг, стрибки, смерті).
    - Використання спрайтів і тайлів для створення ігрового світу.
3. Інтерфейс користувача:
    - Створення основного меню гри з можливістю запуску гри, налаштувань і виходу.
    - Відображення ігрової статистики (рахунок) під час гри.
    - Екран паузи і завершення гри.
4. Аудіо:
    - Додавання музики і звукових ефектів для різних дій (стрибки, збір очків, смерті, фонова музика).
    - Налаштування гучності і можливість відключення звуку через налаштування.
5. Ігрові карти:
    - Створення карт з різними платформами, перешкодами і ворогами.

Для цього використовуються методи функціонального моделювання і графічного опису процесу lucidchart.com. Результати функціонального моделювання у вигляді діаграми роботи гри показано на рисунку 2.1.1

Основні функціональні можливості:

- Гра: основний компонент, який координує взаємодію між іншими елементами гри.
- Головне меню: забезпечує користувачеві доступ до налаштувань гри, можливість виходу з гри, а також налаштування гучності музики і звуків.
- Вихід: дозволяє користувачеві завершити гру.
- Налаштування: включає налаштування гучності музики і звуків.
- Платформи: елементи, на яких може стояти гравець і по яких він може пересуватися.
- Гравець: основний персонаж, яким керує гравець. Може рухатися вліво і вправо, стрибати, збирати предмети.
- Вороги: противники, які можуть пересуватися і взаємодіяти з гравцем.
- Збиральні предмети: об'єкти, які гравець може збирати для отримання бонусів.
- Стрибки: механіка гри, що дозволяє гравцеві пересуватися вертикально, перестрибуючи перешкоди.

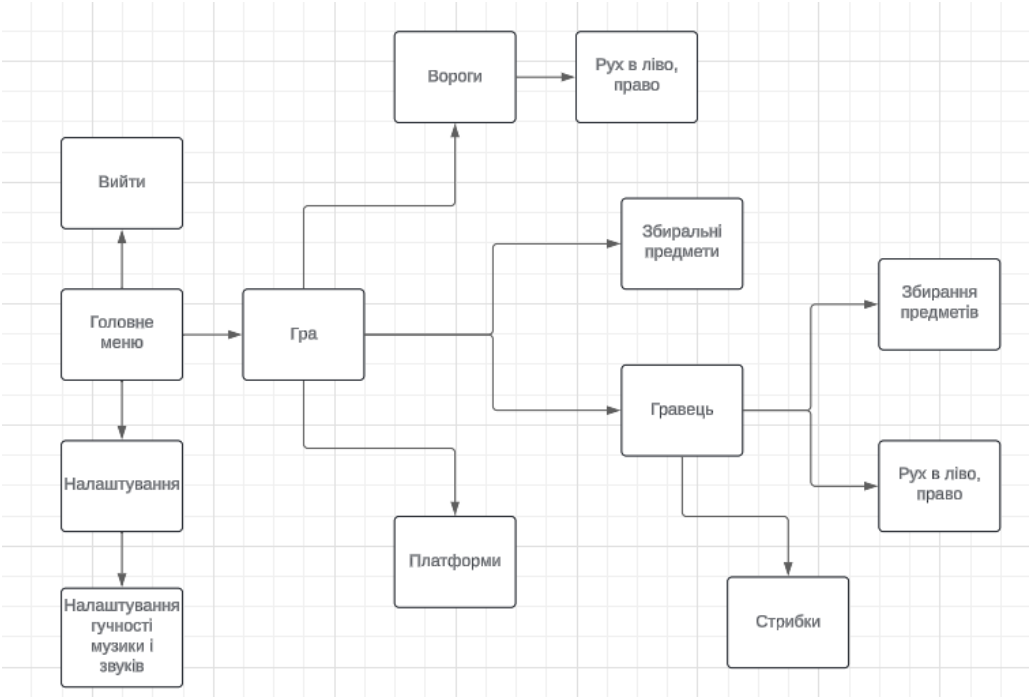

Рисунок 2.1.1 – Логіка роботи гри

Також для створення гри потрібно спроєктувати її деталі ігрового процесу (рисунок 2.1.2).

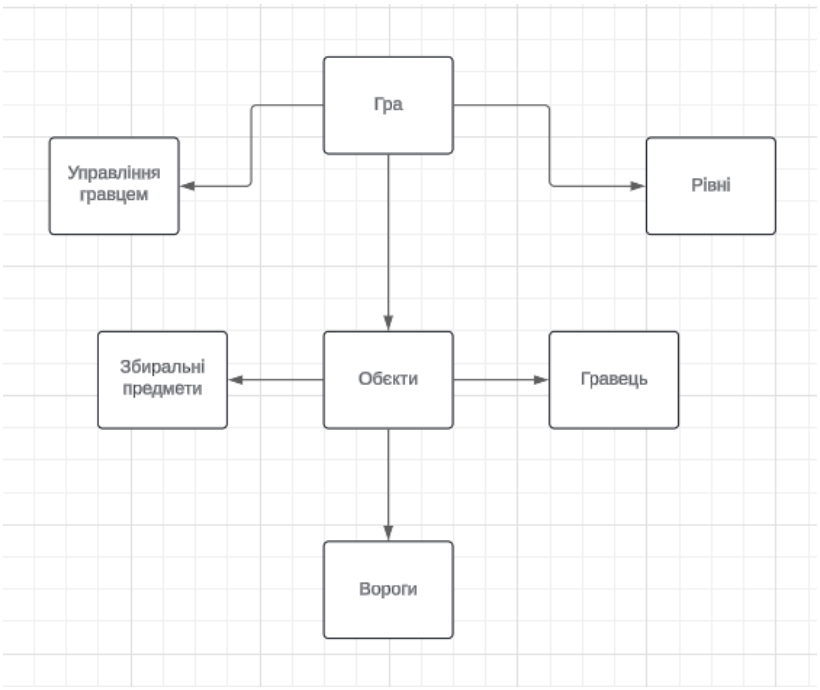

Рисунок 2.1.2 – Основні деталі ігрового процесу

Розберемо процес роботи основних функцій 2D плаформера. Цей процес можна відобразити у вигляді діаграми, як це показано на рисунку 2.1.3.

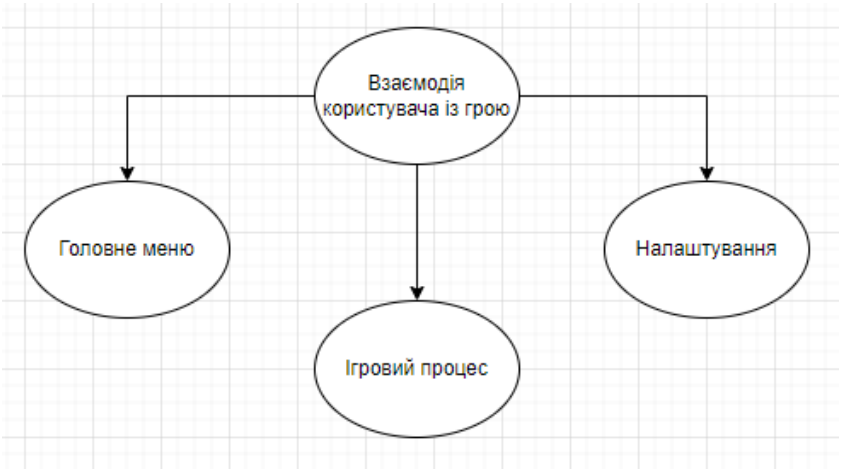

Рисунок 2.1.3 – Основні функції 2D платформера

Коли користувач запустив гру, його зустрічає головне меню. Приведемо основні функції головного меню (рисунок 2.1.4).

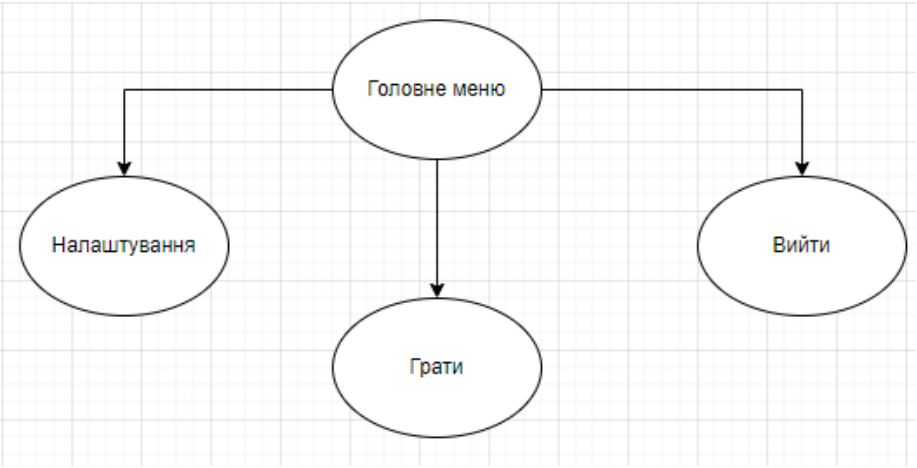

Рисунок 2.1.4 – Головне меню гри

У налаштуваннях гри можна змінювати такі параметри (рисунок 2.1.5).

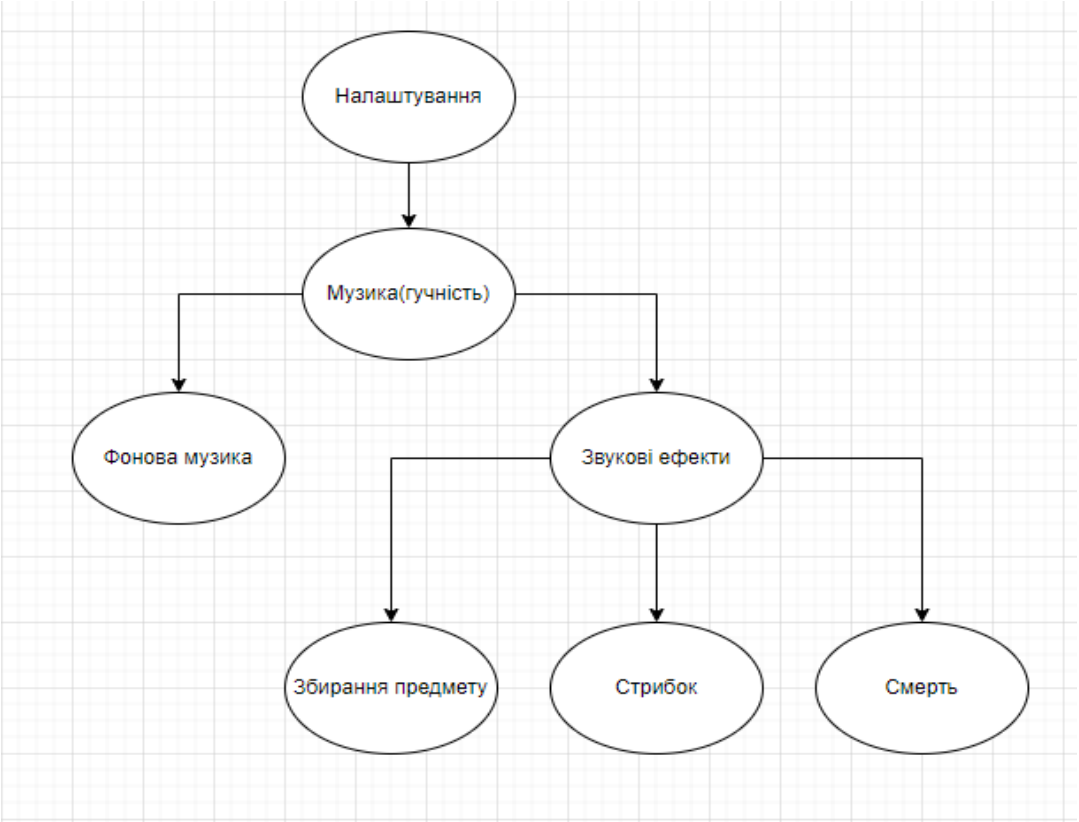

Рисунок 2.1.5 – Налаштування гри

«Грати» – розуміє під собою наступні аспекти (рисунок 2.1.6).

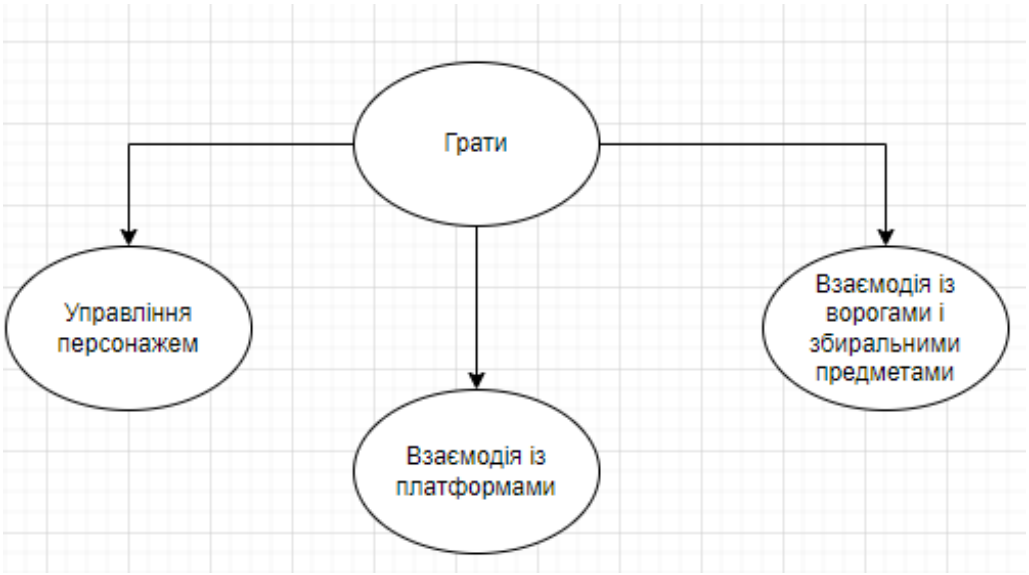

Рисунок 2.1.6 – Основні аспекти «Грати»

2.2 Проєктування структури гри

До основних функцій гри, які мають бути забезпечені, відносяться:

- Реалізація базової механіки платформера: рух гравця, стрибки, взаємодія з об'єктами.
- Створення системи рівнів з ворогами і перешкодами.
- Візуалізація ігрового процесу з використанням графіки, анімацій і ефектів.

При проєктуванні враховується зміст гри, що є однією з важливим компонентом. Тому гравцю потрібна така інформація:

- Інформація про гру: правила, цілі, особливості гри.
- Інформація про рівні: опис рівнів, цілі на кожному рівні, складність.

Гра повинна містити рівні різної складності, які гравець має проходити. Кожен рівень має мати свою структуру, складність, розташування ворогів і збиральних предметів. Також має бути відповідно підібраний дизайн і музичний супровід гри.

Діаграма компонентів гри описує основні модулі і підсистеми, з яких вона складається (рисунок 2.2.1).

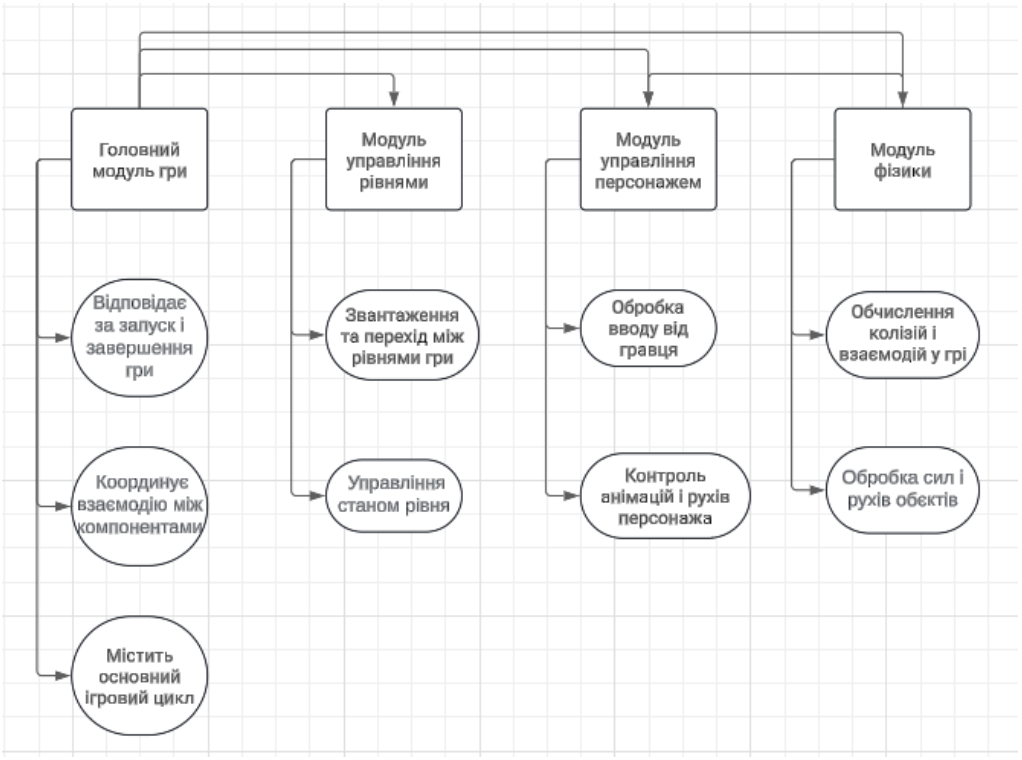

Рисунок 2.2.1 – Компоненти гри

Головний модуль гри

- Відповідає за запуск і завершення гри.
- Координує взаємодію між компонентами.
- Містить основний ігровий цикл.

Модуль управління рівнями

- Завантаження та перехід між рівнями гри.
- Управління станом рівня.
- Модуль управління персонажем

Обробка вводу від гравця.

- Контроль анімації і рухів персонажа.

Модуль фізики

- Обчислення колізій і взаємодій у грі.
- Обробка сил і рухів об'єктів.

Рівні, з яких складається сам ігровий процес, також мають бути добре спроєктовані і мають містити наступні компоненти (рисунок 2.2.2).

Рисунок 2.2.2 – Компоненти рівнів

Дизайн і анімація гри також є важливим складником для 2D платформера і для загального користувацького досвіду (рисунок 2.2.3).

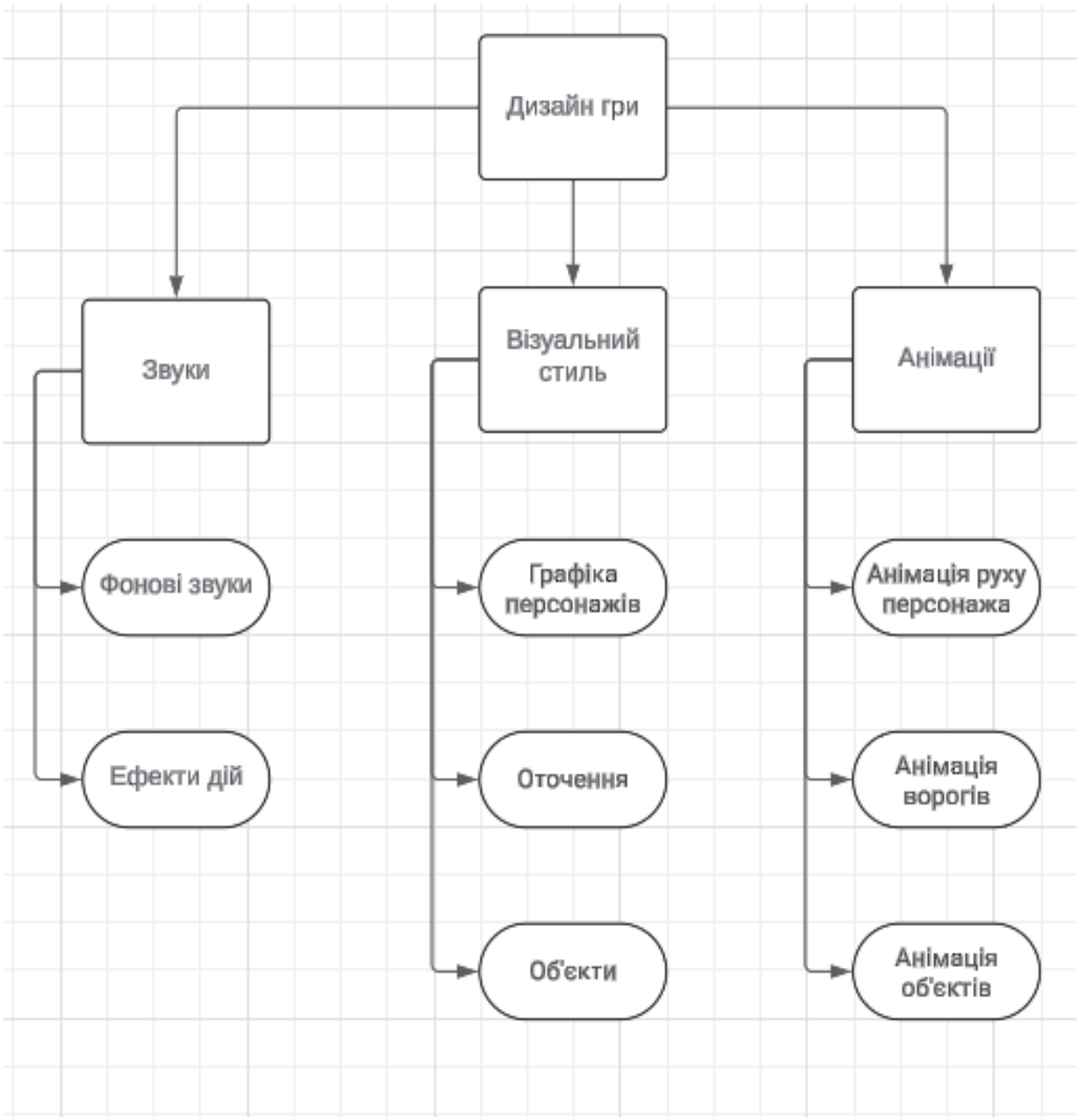

Рисунок 2.2.3 – Діаграма дизайну гри

2.3 Програмна архітектура гри

Після того, як спроєктовано загальну структуру гри, потрібно продумати, як саме це буде виглядати на програмному рівні. В цьому випадку це повинен бути один головний клас (AdventureGame) і допоміжні класи, які реалізують окремий функціонал гри – рух хмар і обєктів, розсташування об’єктів на карті і т.д. Також деякі класи будуть наслідуватися один від одного (рисунок 2.3.1).

На основі наступних сутностей проектуються відповідні класи:

- Гра – клас AdventureGame
- Розташування об'єктів – клас Tilemap
- Генерація хмар – клас Clouds
- Рухомі об'єкти – клас PhysicsEntity
- Збиральні предмети – клас Gem
- Вороги – клас Enemy
- Гравець – клас Player
- Кнопки – клас Button
- Анімація частинок – клас Particle

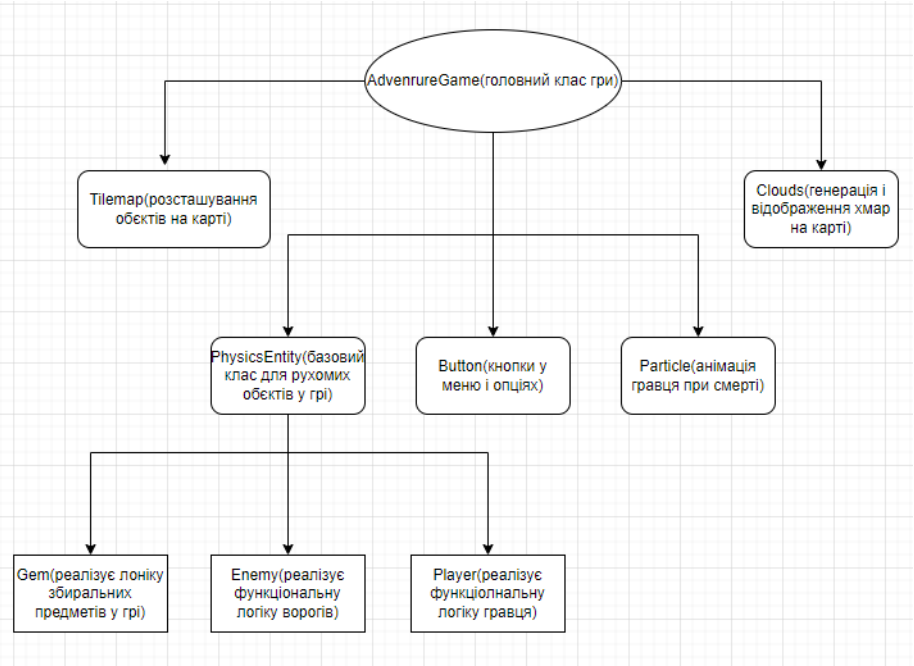

Рисунок 2.3 – Архітектура класів у грі

Основний клас:

- AdventureGame - головний клас гри, який управляє всім процесом гри.

Допоміжні класи:

- Tilemap - клас, що відповідає за розташування об'єктів на карті.
- Clouds - клас, що відповідає за генерацію і відображення хмар на карті.
- PhysicsEntity - базовий клас для рухомих об'єктів у грі, з якого успадковуються інші класи:
    - Gem - реалізує логіку збиральних предметів у грі.

    - Enemy - реалізує функціональну логіку ворогів.
    - Player - реалізує функціональну логіку гравця.
- Button - клас, що відповідає за кнопки у меню і опціях гри.
- Particle - клас, що відповідає за анімацію гравця при смерті.

2.4 Вибір і створення стилю візуалізації, графіки і анімацій

Стиль візуалізації дуже впливає на загальне враження від гри. Наприклад, використання піксельної графіки викликає асоціації з класичними іграми. Такий підхід надає грі особливого стилю і дозволяє гравцю більше зосередитися на геймплеї, ніж на деталізованій графіці.

Палітра кольорів має бути обмеженою, але яскравою. Це допомагає створити виразний і відомий вигляд гри. Осінні кольори, як оранжевий, жовтий, коричневий, дають атмосферності і тепла.

Анімації повинні бути плавними і відповідати загальному стилю гри. Важливо, щоб рухи персонажів, стрибки, взаємодія з об'єктами були добре анімовані, щоб забезпечити гравцю приємну гру.

Для створення стилю і підбору кольорів 2D платформера планується використовувати pixlr.com (рисунок 2.3.1).

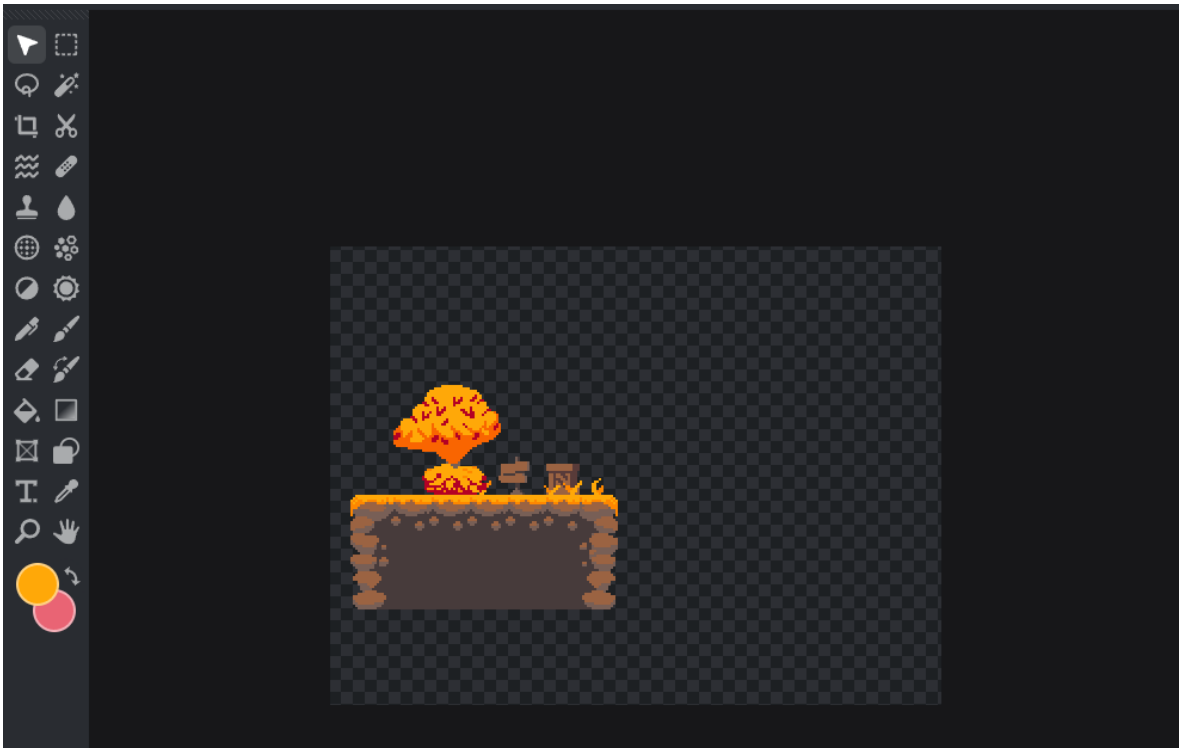

Рисунок 2.3.1 – Інтерфейс pixlr.com

За допомогою цього інструменту можна виконувати різноманітні дії із зображеннями, але основні функції, які будуть використовуватись під час створення графіки, – заливка, реверсія кольорів, пензлик, тонування. Приклад використання однієї із функцій наведено нижче.

На рисунку 2.3.2 показано використання інструменту Тонування. Інструменту також можна задавати колір, товщину пензлика у пікселях, м’якість і крок. [2]

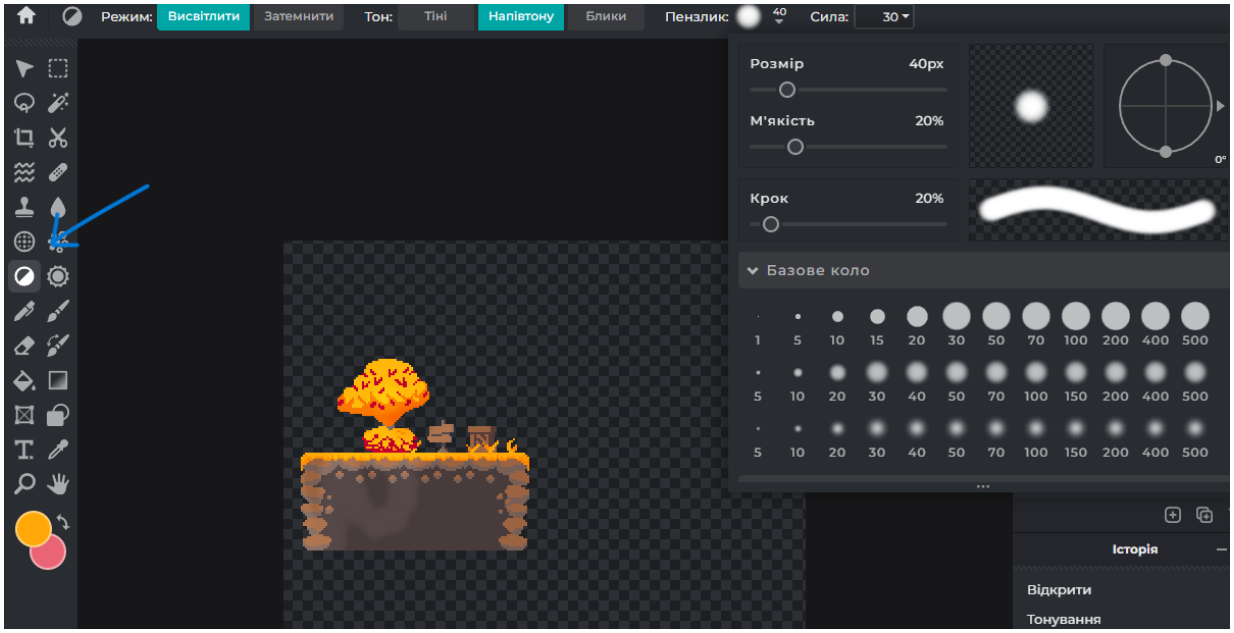

Рисунок 2.3.2 – Інструмент Тонування (pixlr.com)

Для створення анімації і графічних об’єктів (sprites) гри планується використовувати онлайн інструмент piskelapp.com (рисунок 2.3.3).

Основні функції piskelapp.com:

1. Редактор піксельної графіки.
2. Інструменти малювання (пензлі, заливки, гумки).
3. Створення анімацій з кадрами та шарами.
4. Підтримка шарів.
5. Імпорт та експорт у форматах PNG та GIF.

Також у цьому інструменті можна задавати розмір холста для створення спрайтів (рухомих об’єктів гри), товщину пензлика та інші налаштування інструментів. [8]

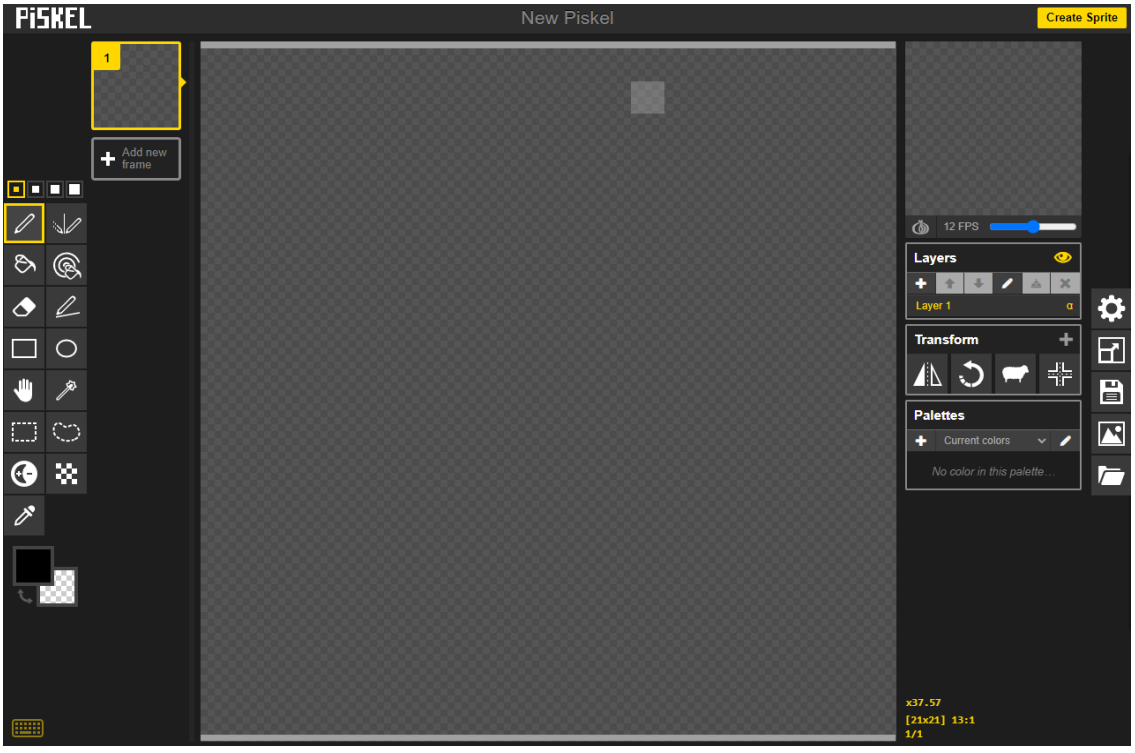

Рисунок 2.3.3 – Інтерфейс piskel.com

Приклад створення графічного об’єкта для гри наведено на рисунку 2.3.4.

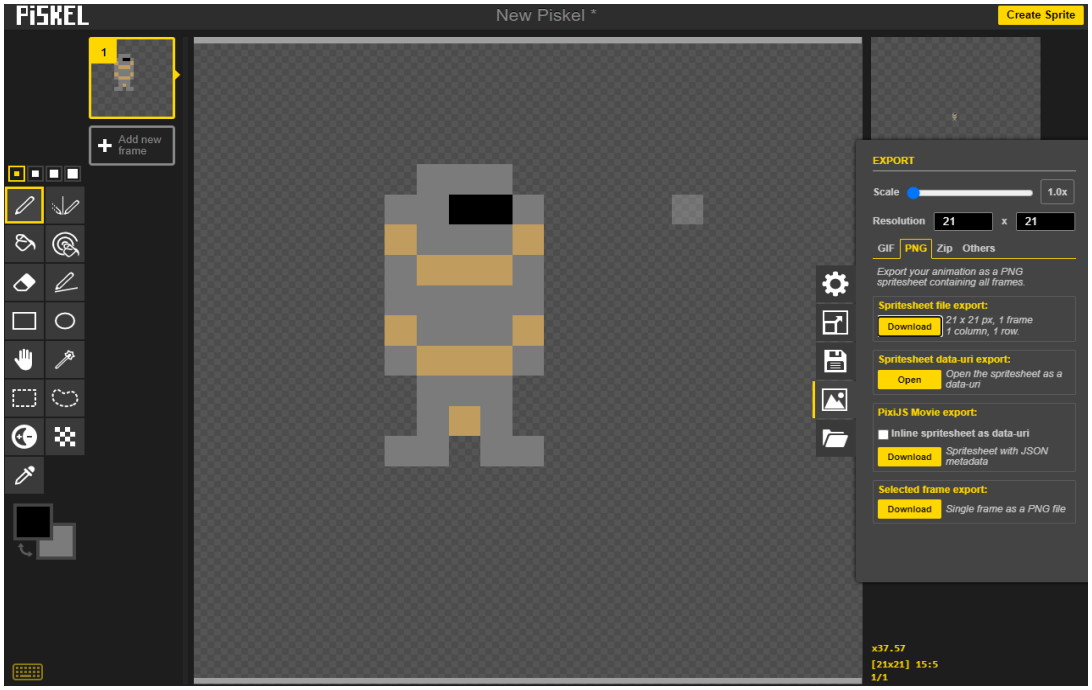

Рисунок 2.3.4 – Створення графічного об’єкта (piskel.com)

Для створення рівнів і мап для гри буде використовуватися програма Tiled.

Tiled – це безкоштовний редактор карт для створення 2D ігор. Він використовується для створення рівнів, мап та інших елементів (tiles) ігрового середовища. [7]

Основні функції Tiled:

1. Редагування тайлсетів: Створення та редагування тайлсетів, які є наборами тайлів (маленьких графічних елементів), що використовуються для створення карт.
2. Підтримка шарів: Використання шарів для створення складних мап із різними рівнями деталей.
3. Об'єкти та властивості: Додавання об'єктів з різними властивостями на карту для взаємодії в грі.
4. Експорт у різні формати: Експорт карт у формати, що підтримуються багатьма ігровими рушіями (JSON, XML, Lua). [3]

Інтерфейс програми Tiled наведено на рисунку 2.3.5.

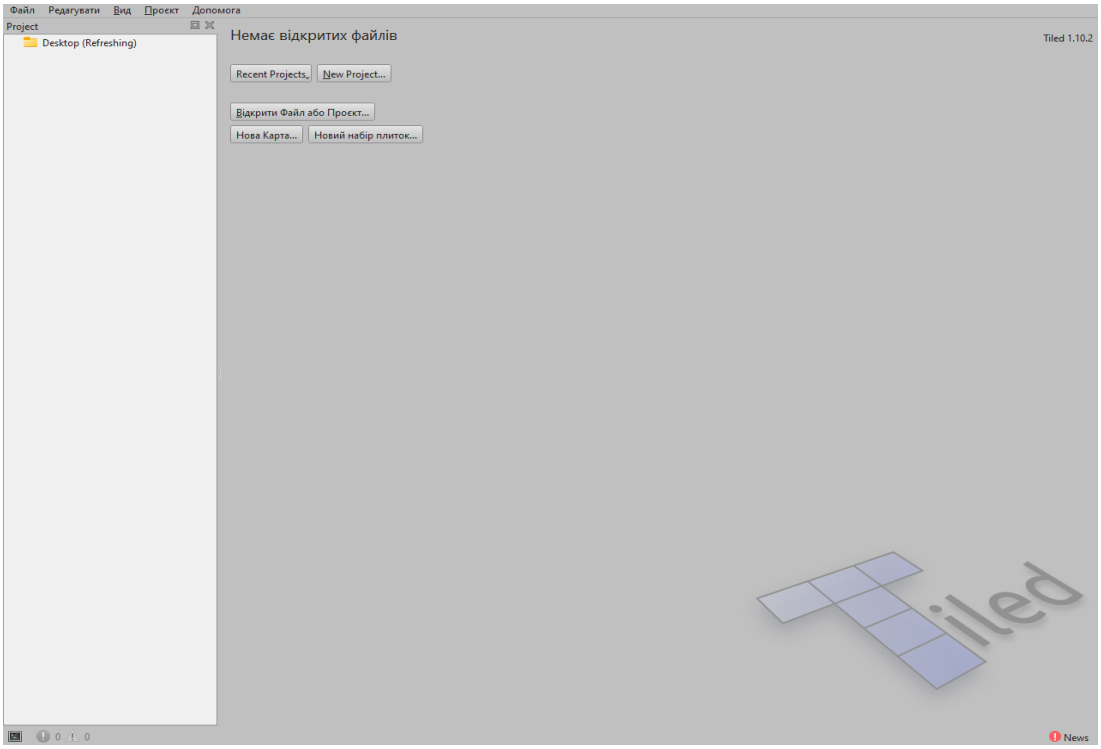

Рисунок 2.3.5 – Програма Tiled

Приклад використання програми Tiled показано на рисунку 2.3.6.

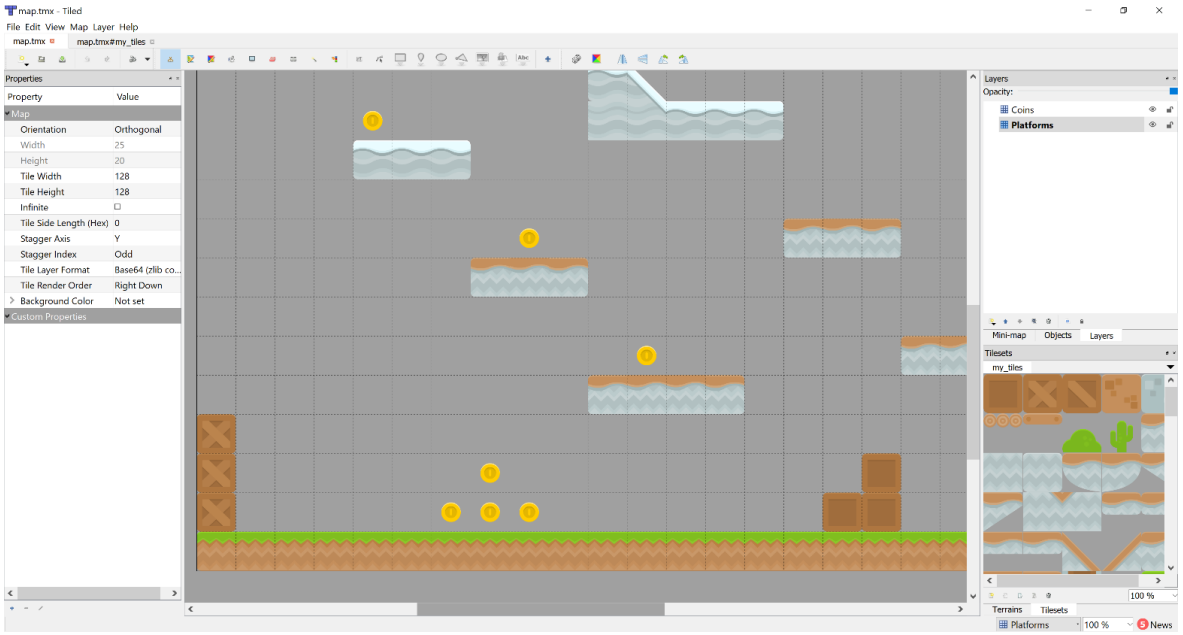

Рисунок 2.3.6 – Приклад створення рівня для гри в Tiled

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ГРИ

3.1 Вибір інструментарію розробки гри

Python – це потужна мова програмування високого рівня з простим та елегантним синтаксисом, широко використовувана для вирішення спектру завдань: від простих скриптів до складних додатків. Інтерпретована природа мови дозволяє програмістам швидко розробляти, тестувати і відлагоджувати програми без необхідності компіляції. Python підтримує модульність та повторне використання коду, що сприяє швидкому розвитку проєктів, а також – багато парадигм програмування, включаючи процедурну, об'єктно-орієнтовану, функціональну та аспектно-орієнтовану.

Однією з головних переваг Python є її велика стандартна бібліотека, яка містить тисячі корисних модулів і функцій для різних завдань, таких як робота з файлами, мережами, базами даних, обробка даних, робота з регулярними виразами та багато іншого, що дозволяє програмістам швидко розробляти рішення без написання коду з нуля. [4]

Python є платформо незалежною мовою, а це означає, що програми можна запускати на будь-якій операційній системі, на якій встановлено відповідний інтерпретатор. Це робить мову дуже гнучкою для розробки програм, оскільки можна написати код один раз, і запускати його на різних платформах без необхідності внесення змін.

Також мова програмування є відомою заявдяки своїй активній спільноті користувачів і розробників, що робить її більш привабливою для використання, оскільки постійно з'являються нові бібліотеки, фреймворки та інструменти.

Своїм походженням Python завдячує голландському програмісту Гвідо Ван Россуму, який створив концепцію цієї мови програмування наприкінці 80-х років і випустив першу версію Python 0.9.0 у лютому 1991 року в Амстердамі, а вже в січні 1994 року вийшла версія Python 1.0. Спочатку Python була задумана як нащадок мови ABC для навчання програмуванню. Це досить важливий факт в історії Python, оскільки саме він поставив на перше місце лаконічність і простоту синтаксису для

майбутньої мови програмування, якими не могла похизуватися переважна більшість мов програмування того часу. [5]

PyGame – це набір модулів для мови Python, призначений для розробки ігор та мультимедійних додатків, заснований у 2000-му році на основі бібліотеки SDL (Simple DirectMedia Layer), що забезпечує доступ до аудіо, графіки та вводу користувача. PyGame дозволяє легко створювати ігри на основі полотна та навіть із 3D-графікою.

Основні можливості PyGame включають роботу зі зображеннями у вікнах, звукові ефекти, обробку подій клавіатури та миші, а також – роботу в мережі. Бібліотека має простий та інтуїтивно зрозумілий інтерфейс, що дозволяє швидко працювати над створенням нового контенту.

PyGame надає можливості для створення анімації, обробки зіткнень та фізики об'єктів в іграх. Бібліотека підтримує роботу з різними типами об'єктів, такими як спрайти, текст та фігури, що дозволяє розробникам створювати різноманітні та цікаві графічні ефекти. Крім того, PyGame має вбудовану підтримку роботи зі шрифтами, що дозволяє створювати інтерактивні ігрові інтерфейси та повідомлення для користувача.

Щодо мережі, то PyGame має відповідні можливості, що дозволяють створювати мережеві ігри та додатки. Це відкриває шляхи для створення багатокористувацьких ігор та спільної гри разом із кількома користувачами через Інтернет.

Ще однією важливою особливістю бібліотеки PyGame є можливість експортувати ігри на різні платформи, включаючи Windows, macOS, Linux та Android. Це дозволяє розробникам створювати ігри для широкої аудиторії та публікувати їх у відповідних магазинах додатків. [6]

3.2 Реалізація функціоналу гри

Ресурси створюються поетапно. Першим кроком є імпорт додаткових бібліотек (рисунок 3.2.1).

```
import sys
import pygame
import random
import math
import os

from support import load_image, load_images, Animation, Text, scale_images
from entities import PhysicsEntity, Player, Enemy, Gem
from tilemap import Tilemap
from clouds import Clouds
from particle import Particle
from button import Button
```

Рисунок 3.2.1 – Імпорт бібліотек

Для ініціалізації PyGame, створення вікна гри, завантаження ресурсів (зображення, анімації, звуки), встановлення початкових значень для ігрових змінних та завантаження першого рівня гри створено метод __init__ у класі AdventureGame (рисунок 3.2.2).

```
class AdventureGame:
    def __init__(self) -> None:
        pygame.init()
        WIDTH, HEIGHT = 1024, 768
        pygame.display.set_caption('Осінні Пригоди Кота')
        self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
        self.display = pygame.Surface((WIDTH / 2, HEIGHT / 2), pygame.SRCALPHA)
        self.display_2 = pygame.Surface((WIDTH / 2, HEIGHT / 2))
        self.clock = pygame.time.Clock()
        self.movement = [False, False]
        player_sprite = load_image('entities/player.png')
        self.assets = {
```

```
        self.vol_music = 0.05
        self.vol_ambience = 0.6
        self.vol_explosion = 0.8
        self.vol_jump = 0.7
        self.vol_gem = 0.9
        self.sfx['ambience'].set_volume(self.vol_ambience)
        self.sfx['explosion'].set_volume(self.vol_explosion)
        self.sfx['jump'].set_volume(self.vol_jump)
        self.sfx['gem'].set_volume(self.vol_gem)
        pygame.mixer.music.load('assets/music.mp3')
        pygame.mixer.music.set_volume(self.vol_music)
        pygame.mixer.music.play(-1)
        self.clouds = Clouds(self.assets['clouds'], count=16)
        self.player = Player(self, (50, 50), (11, 16))
        self.tilemap = Tilemap(self, tile_size=16)
        self.level = 0
        self.load_level(self.level)
        self.screenshake = 0
        self.last_state = 'menu'
```

Рисунок 3.2.2 – Ініціалізація гри

Для завантаження шрифту з файлу для використання в текстових елементах створено метод get_font у класі AdventureGame (рисунок 3.2.3).

```
def get_font(self, size):
    return pygame.font.Font('assets/font.ttf', size)
```

Рисунок 3.2.3 – Завантаження шрифту

Для завантаження карти рівня з файлу та розміщення ігрових елементів на карті створено метод load_level у класі AdventureGame (рисунок 3.2.4).

```
def load_level(self, lvl_id):
    self.tilemap.load('assets/maps/' + str(lvl_id) + '.json')

    self.enemies = []
    self.gems = []
    for spawner in self.tilemap.extract([('spawners', 0), ('spawners', 1), ('spawners', 2)]):
        if spawner['variant'] == 0:
            self.player.rect_pos.x = spawner['pos'][0]
            self.player.rect_pos.y = spawner['pos'][1]
            self.player.air_time = 0
        elif spawner['variant'] == 2:
            self.gems.append(Gem(self, spawner['pos'], (16, 16)))
        else:
            self.enemies.append(Enemy(self, spawner['pos'], (11, 16)))

    self.dead = 0
    self.particles = []
    self.collided = 0
    self.transition = -30
    self.level_ended = False
    self.wait_time = 0
    self.scroll = [0, 0]
    self.score = 0
    self.max_score = len(self.gems)
```

Рисунок 3.2.4 – Завантаження рівня

Для реалізації основного ігрового циклу, відображення ігрових елементів, обробки події, оновлення ігрового стану та переходу між рівнями створено метод run у класі AdventureGame (рисунок 3.2.5).

```
def run(self):
    self.sfx['ambience'].play(-1)
    while True:
        self.display.fill((0, 0, 0, 0))
        self.display_2.fill((59, 107, 156))
        self.scroll[0] += (self.player.rect_pos.centerx - self.display.get_width() / 2 - self.scroll[0]) / 30
        self.scroll[1] += (self.player.rect_pos.centery - self.display.get_height() / 2 - self.scroll[1]) / 90
        render_scroll = (int(self.scroll[0]), int(self.scroll[1]))
        Text(self.display, f'Рахунок:{self.score}/{self.max_score}', self.get_font(8), (10, 10)).render()
        if int(self.scroll[1]) >= 18:···
        if int(self.scroll[0]) <= -100:···
        speed = 0.3
        for x in range(-10, 200):
            self.display_2.blit(self.assets['back_trees'],···
            self.display_2.blit(self.assets['back_trees'],···
        self.clouds.update()
        self.clouds.render(self.display_2, offset=render_scroll)
        self.screenshake = max(0, self.screenshake - 1)
        if self.dead:···
        self.tilemap.render(self.display, offset=render_scroll)
        if (self.score == self.max_score) and self.player.rect_pos.colliderect(self.tilemap.end_tile()):···
        if self.level_ended:···
        if self.wait_time > 40:···
        if self.transition < 0:···
```

Рисунок 3.2.5 – Запуск гри

Метод menu у класі AdventureGame відображає головне меню гри, де гравець може вибрати необхідні опції, та обробляє вибір гравця (рисунок 3.2.6).

```
def menu(self):
    self.sfx['ambience'].stop()
    self.load_level(self.level)
    self.transition = 0
    switch = 1
    if self.level == 0:···
    else:···
    button_width, button_height = 100, 40
    screen_center_x = self.display.get_width() / 2
    play_button = Button(···
    option_button = Button(self.display, self.get_font(11), 'Опції', button_width, button_height,···
    quit_button = Button(self.display, self.get_font(11), 'Вийти', button_width, button_height,···
    while True:
        self.display.fill((0, 0, 0, 0))
        self.display_2.fill((59, 107, 156))
        self.display_2.blit(self.assets['menu_trees'], (0, 0))
        self.display_2.blit(self.assets['menu_water'], (0, 0))
        self.display.blit(self.assets['menu_grass'], (screen_center_x / 2, 0))
        if not switch:···
        if self.transition < 0:···
        Text(self.display, 'Осінні Пригоди Котеняти', self.get_font(15), (screen_center_x-160, 50)).render()
        play_button.render()
        if play_button.done == True:···
        option_button.render()
        if option_button.done == True:···
        quit_button.render()
        if quit_button.done == True:···
        for event in pygame.event.get():···
        display_mask = pygame.mask.from_surface(self.display)
        display_sillhouette = display_mask.to_surface(setcolor=(0, 0, 0, 180), unsetcolor=(0, 0, 0, 0))
        for offset in [(-2, 0), (2, 0), (0, -2), (0, 2)]:···
        if self.transition:···
        self.display_2.blit(self.display, (0, 0))
        self.screen.blit(pygame.transform.scale(self.display_2, self.screen.get_size()), (0, 0))
        pygame.display.update()
        self.clock.tick(60)
```

Рисунок 3.2.6 – Створення меню гри

Метод pause_menu у класі AdventureGame реалізовує меню паузи в грі. Він зупиняє всі рухи в грі, зупиняє звуковий ефект атмосфери, відображає на екрані кнопки для вибору дії (продовження гри, перехід до меню опцій або виходу в головне меню гри) і чекає на вибір гравця. Код реалізує також анімацію переходу між екранами паузи та гри (рисунок 3.2.7).

```
def pause_menu(self):
    self.movement = [False, False]
    self.sfx['ambience'].stop()
    self.transition = 0
    switch = 1
    play_button = Button(self.display, self.get_font(11), 'Назад', 100, 40, (205, 150), 6)
    option_button = Button(self.display, self.get_font(11), 'Опції', 100, 40, (205, 225), 6)
    quit_button = Button(self.display, self.get_font(11), 'Меню', 100, 40, (205, 300), 6)
    while True:
        self.display.fill((0, 0, 0, 0))
        self.display_2.fill((59, 107, 156))
        if not switch: ···
        if self.transition < 0: ···
        Text(self.display, 'Пауза', self.get_font(22), (192, 50)).render()
        play_button.render()
        if play_button.done == True: ···
        option_button.render()
        if option_button.done == True: ···
        quit_button.render()
        if quit_button.done == True: ···
        for event in pygame.event.get(): ···
        display_mask = pygame.mask.from_surface(self.display)
        display_sillhouette = display_mask.to_surface(setcolor=(0, 0, 0, 180), unsetcolor=(0, 0, 0, 0))
        for offset in [(-2, 0), (2, 0), (0, -2), (0, 2)]:
            self.display_2.blit(display_sillhouette, offset)
        if self.transition: ···
        self.display_2.blit(self.display, (0, 0))
        self.screen.blit(pygame.transform.scale(self.display_2, self.screen.get_size()), (0, 0))
        pygame.display.update()
        self.clock.tick(60)
```

Рисунок 3.2.7 – Пауза в грі

Метод option_menu у класі AdventureGame відображає меню опцій в грі. Гравець може вибрати опції звуку (змінити тип звуку і налаштувати гучність) та повернутися до головного меню гри або меню паузи. Метод також відображає анімацію переходу між екранами (рисунок 3.2.8).

```
def option_menu(self):
    self.movement = [False, False]
    self.sfx['ambience'].stop()
    self.transition = 0
    switch = 1
    back_button = Button(self.display, self.get_font(11), 'Назад', 100, 40, (205, 300), 6)
    sound_button_l = Button(self.display, self.get_font(16), '<', 30, 25, (200, 125), 6)
    sound_button_r = Button(self.display, self.get_font(16), '>', 30, 25, (415, 125), 6)
    volume_buttom_l = Button(self.display, self.get_font(16), '<', 30, 25, (200, 215), 6)
    volume_buttom_r = Button(self.display, self.get_font(16), '>', 30, 25, (415, 215), 6)
    sounds = list(self.sfx.copy())
    sounds.append('music')
    sound = 1
    sound_translations = {'jump': 'Стрибок', 'explosion': 'Зіткнення', 'ambience': 'Фон', 'gem': 'Рибка', 'music': 'Музика'}
    while True:
        self.display.fill((0, 0, 0, 0))
        self.display_2.fill((59, 107, 156))
        if not switch: ···
        if self.transition < 0: ···
        Text(self.display, 'Опції', self.get_font(22), (180, 50)).render()
        Text(self.display, 'Звук:', self.get_font(16), (50, 125)).render()
        Text(self.display, 'Гучність:', self.get_font(16), (50, 215)).render()
        sound_name = sounds[sound]
        translated_sound_name = sound_translations.get(sound_name, sound_name)
        Text(self.display, translated_sound_name, self.get_font(16), (250, 125)).render()
        sound_button_l.render()
```

Рисунок 3.2.8 – Опції в грі

Клас Button визначає кнопку для гри на PyGame. Під час ініціалізації приймається об'єкт гри, шрифт для тексту кнопки, сам текст кнопки, розміри (ширина і висота), позиція кнопки на екрані та висота "підняття" кнопки під час

натискання. Метод render відображає кнопку на екрані, перевіряє, чи кнопка була натиснута, і встановлює відповідний стан, а також – перевіряє, чи мишка знаходиться над кнопкою, і змінює колір кнопки відповідно. Метод check_click перевіряє, чи відбувається натискання кнопки мишею (рисунок 3.2.9).

```
class Button:
    def __init__(self, game, font, text, width, height, pos, elevation) -> None: ···

    def render(self):
        self.top_rect.y = self.org_y_pos - self.dynamic_elevation
        self.text_rect.center = self.top_rect.center
        self.bottom_rect.midtop = self.top_rect.midtop
        self.bottom_rect.height = self.top_rect.height + self.dynamic_elevation
        pygame.draw.rect(self.game, self.bottom_color, self.bottom_rect, border_radius=7)
        pygame.draw.rect(self.game, self.border_color, self.top_rect, self.border_thickness, border_radius=7)
        pygame.draw.rect(self.game, self.top_color, self.top_rect.inflate(-self.border_thickness, -self.border_thickness), border_radius=7)
        self.game.blit(self.text_surf, self.text_rect)
        self.check_click()
        return self.done

    def check_click(self):
        mpos = pygame.mouse.get_pos()
        mpos = (mpos[0] / SCALE, mpos[1] / SCALE)
        if self.top_rect.collidepoint(mpos):
            self.top_color = '#c8e0e0'
            if pygame.mouse.get_pressed()[0]:
                self.dynamic_elevation = 0
                self.pressed = True
            else:
                self.dynamic_elevation = self.elevation
                if self.pressed:
                    self.done = True
                    self.pressed = False
        else:
            self.dynamic_elevation = self.elevation
            self.top_color = '#FFFFFF'
```

Рисунок 3.2.9 – Клас Button

Клас Cloud відповідає за одну хмаринку. Він має атрибути позиції, зображення, швидкості руху та глибини. Метод update оновлює позицію хмаринки на екрані, відповідно до його швидкості. Метод render відображає хмаринку на поверхні з урахуванням зсуву та глибини хмаринки.

Клас Clouds управляє групою хмар. Під час ініціалізації він створює об'єктів класу Cloud з випадковими параметрами і додає їх до списку. Метод update оновлює всі хмари в групі, а метод render відображає їх на екрані. Хмари сортуються за глибиною, щоб правильно рендерити їх у відношенні один до одного (рисунок 3.2.10).

```python
class Cloud:
    def __init__(self, pos, img, speed, depth) -> None:
        self.pos = list(pos)
        self.img = img
        self.speed = speed
        self.depth = depth
    def update(self):
        self.pos[0] += self.speed
    def render(self, surf, offset=(0, 0)):
        render_pos = (self.pos[0] - offset[0] * self.depth, self.pos[1] - offset[1] * self.depth)
        surf.blit(self.img, (render_pos[0] % (surf.get_width() + self.img.get_width()) - self.img.get_width(), ···


class Clouds:
    def __init__(self, cloud_images, count=16) -> None:
        self.clouds = []
        for i in range(count):
            self.clouds.append(Cloud((random.random() * 99999, random.random() * 99999), random.choice(cloud_images), ···
        self.clouds.sort(key=lambda x: x.depth)
    def update(self):
        for cloud in self.clouds:
            cloud.update()
    def render(self, surf, offset=(0, 0)):
        for cloud in self.clouds:
            cloud.render(surf, offset=offset)
```

Рисунок 3.2.10 – Клас Cloud/s

Клас PhysicsEntity визначає фізичний об'єкт у грі, який може рухатися, взаємодіяти з іншими об'єктами та відображатися на екрані (рисунок 3.2.11).

```python
class PhysicsEntity:
    def __init__(self, game, entity_type, pos, size) -> None: ···

    def set_action(self, action):
        if action != self.action:
            self.action = action
            self.animation = self.game.assets[self.type + '/' + self.action].copy_anim()

    def update(self, tilemap, movement=(0, 0)):
        self.collisions = {'up': False, 'down': False, 'right': False, 'left': False}
        frame_movement = (movement[0] + self.velocity[0], movement[1] + self.velocity[1])
        self.rect_pos[0] += frame_movement[0]
        for rect in tilemap.physics_rects_around(self.rect_pos): ···
        self.rect_pos[1] += frame_movement[1]
        for rect in tilemap.physics_rects_around(self.rect_pos): ···
        if movement[0] > 0:
            self.flip = True
        if movement[0] < 0:
            self.flip = False
        self.velocity[1] = min(5, self.velocity[1] + 0.1)
        if self.collisions['down'] or self.collisions['up']:
            self.velocity[1] = 0
        self.animation.update()

    def render(self, surf, offset=(0, 0)):
        surf.blit(pygame.transform.flip(self.animation.img(), self.flip, False), (
        self.rect_pos[0] - offset[0] + self.anim_offset[0], self.rect_pos[1] - offset[1] + self.anim_offset[1]))
```

Рисунок 3.2.11 – Клас PhysicsEntity

Клас Gem унаслідує функціональність від класу PhysicsEntity та додає до цього базового функціоналу можливість взаємодії з гравцем та звуковим ефектом (рисунок 3.2.12).

```python
class Gem(PhysicsEntity):
    def __init__(self, game, pos, size) -> None:
        super().__init__(game, 'gem', pos, size)
        self.anim_offset = (0, 0)

    def update(self, tilemap, movement=(0, 0)):
        super().update(tilemap, movement)
        self.set_action('idle')
        if self.rect_pos.colliderect(self.game.player.rect_pos):
            self.game.sfx['gem'].play()
            return True

    def render(self, surf, offset=(0, 0)):
        super().render(surf, offset=offset)
```

Рисунок 3.2.12 – Клас Gem

Клас Enemy представляє ворожий об'єкт у грі, який може рухатися та взаємодіяти з оточенням (рисунок 3.2.13).

```python
class Enemy(PhysicsEntity):
    def __init__(self, game, pos, size) -> None:
        super().__init__(game, 'enemy', pos, size)
        self.walking = 0
        self.anim_offset = (-4, -4)

    def update(self, tilemap, movement=(0, 0)):
        if self.walking:
            if tilemap.solid_check((self.rect_pos.centerx + (-7 if not self.flip else 7), self.rect_pos[1] + 23)):
                if (self.collisions['right'] or self.collisions['left']):
                    self.flip = not self.flip
                else:
                    movement = (movement[0] - 0.35 if not self.flip else 0.35, movement[1])
            else:
                self.flip = not self.flip
            self.walking = max(0, self.walking - 1)
        elif random.random() < 0.01:
            self.walking = random.randint(30, 120)
        super().update(tilemap, movement=movement)
        if movement[0] != 0:
            self.set_action('run')
        else:
            self.set_action('idle')

    def render(self, surf, offset=(0, 0)):
        super().render(surf, offset=offset)
```

Рисунок 3.2.13 – Клас Enemy

Клас Player визначає об'єкт гравця у грі (рисунок 3.2.14).

```python
class Player(PhysicsEntity):
    def __init__(self, game, pos, size) -> None:
        super().__init__(game, 'player', pos, size)
        self.air_time = 0
        self.jumps = 1

    def update(self, tilemap, movement=(0, 0)):
        super().update(tilemap, movement=movement)
        self.air_time += 1
        if self.air_time > 200:
            self.game.dead += 1
        if self.collisions['down']:
            self.air_time = 0
            self.jumps = 1
        if self.air_time > 4:
            self.set_action('jump')
        elif movement[0] != 0:
            self.set_action('run')
        else:
            self.set_action('idle')

    def render(self, surf, offset=(0, 0)):
        super().render(surf, offset=offset)

    def jump(self):
        if self.jumps:
            self.velocity[1] = -3
            self.jumps -= 1
            self.air_time = 5
            return True
```

Рисунок 3.2.14 – Клас Player

Клас Particle представляє частинки в грі, які можуть використовуватися для візуальних ефектів (рисунок 3.2.15).

```python
class Particle:
    def __init__(self, game, p_type, pos, velocity=[0, 0], frame=0) -> None:
        self.game = game
        self.type = p_type
        self.pos = list(pos)
        self.velocity = list(velocity)
        self.animation = self.game.assets['particle/' + p_type].copy_anim()
        self.animation.frame = frame

    def update(self):
        kill = False
        if self.animation.done:
            kill = True
        self.pos[0] += self.velocity[0]
        self.pos[1] += self.velocity[1]
        self.animation.update()
        return kill

    def render(self, surf, offset=(0, 0)):
        img = self.animation.img()
        surf.blit(img,
                  (self.pos[0] - offset[0] - img.get_width() // 2, self.pos[1] - offset[1] - img.get_height() // 2))
```

Рисунок 3.2.15 – Клас Particle

Завантаження та обробка зображень, створення анімаційних об'єктів та відображення тексту на екрані гри виконується за допомогою рядків коду, що зображений на рисунку 3.2.16.

```python
def load_image(path):
    img = pygame.image.load(BASE_PATH + path).convert()
    img.set_colorkey((0, 0, 0))
    return img

def load_images(path):
    images = []
    for img_name in sorted(os.listdir(BASE_PATH + path)):···
    return images


class Animation:
    def __init__(self, images, img_dur=5, loop=True) -> None:···

    def copy_anim(self):
        return Animation(self.images, self.img_duration, self.loop)

    def update(self):
        if self.loop:···
        else:···

    def img(self):
        return self.images[int(self.frame / self.img_duration)]


class Text:
    def __init__(self, display, text, font, pos) -> None:···

    def render(self):
        text = self.font.render(self.text, False, (255, 255, 255))
        self.game.blit(text, self.pos)
```

Рисунок 3.2.16 – Зображення, анімація та текст

Клас Tile представляє платформи в грі. Клас CustomEncoder відповідає за кодування об'єктів класу Tile у формат JSON (рисунок 3.2.17).

```python
class Tile:
    def __init__(self, type: str, variant: int, pos) -> None:···

    def __getitem__(self, key):
        return self.tile_dict[key]

    def set_variant(self, new_variant):
        self.variant = new_variant
        self.tile_dict['variant'] = new_variant

    def __json__(self):
        return self.tile_dict


class CustomEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, Tile):
            return obj.__json__()
        return json.JSONEncoder.default(self, obj)
```

Рисунок 3.2.17 – Класи Tile та CustomEncoder

Клас Tilemap керує тайловою карткою: витягає тайли за ID, зберігає та завантажує карту, перевіряє наявність твердих об'єктів, рендерить карту, застосовує автотайлинг (рисунок 3.2.18).

```python
class Tilemap:
    def __init__(self, game, tile_size=16) -> None:...

    def extract(self, id_pairs, keep=False):
        matches = []
        for tile in self.offgrid_tiles.copy():...
        for loc in self.tilemap:...
        return matches

    def tiles_around(self, pos):
        tiles = []
        tile_loc = (int(pos[0] // self.tile_size), int(pos[1] // self.tile_size))
        for offset in CLOSEST_TILES_OFFSETS:...
        return tiles

    def save(self, path):
        f = open(path, 'w')
        json.dump({'tilemap': self.tilemap, 'tile_size': self.tile_size, 'offgrid': self.offgrid_tiles}, f,...
        f.close

    def load(self, path):
        f = open(path, 'r')
        map_data = json.load(f)
        f.close()
        self.tilemap = map_data['tilemap']
        self.tile_size = map_data['tile_size']
        self.offgrid_tiles = map_data['offgrid']
```

```python
    def physics_rects_around(self, pos):
        rects = []
        for tile in self.tiles_around(pos):...
        return rects

    def end_tile(self):
        for tile in self.offgrid_tiles:...
        for loc in self.tilemap:...
        return pygame.Rect(0, 0, 2, 2)

    def autotile(self):
        for loc in self.tilemap:
            tile = self.tilemap[loc]
            neighbors = set()
            for shift in [(1, 0), (-1, 0), (0, -1), (0, 1)]:...
            neighbors = tuple(sorted(neighbors))
            if (tile['type'] in PHYSICS_TILES) and (neighbors in AUTOTILE_MAP):...
```

Рисунок 3.2.18 – Клас Tilemap

3.3 Тестування програмного забезпечення

Після розробки програмного продукту важливим аспектом є його тестування. Кроки тестування створеного 2D платформера описано нижче.

Першочергово було протестовано кнопки на головному меню гри (рисунок 3.3.1).

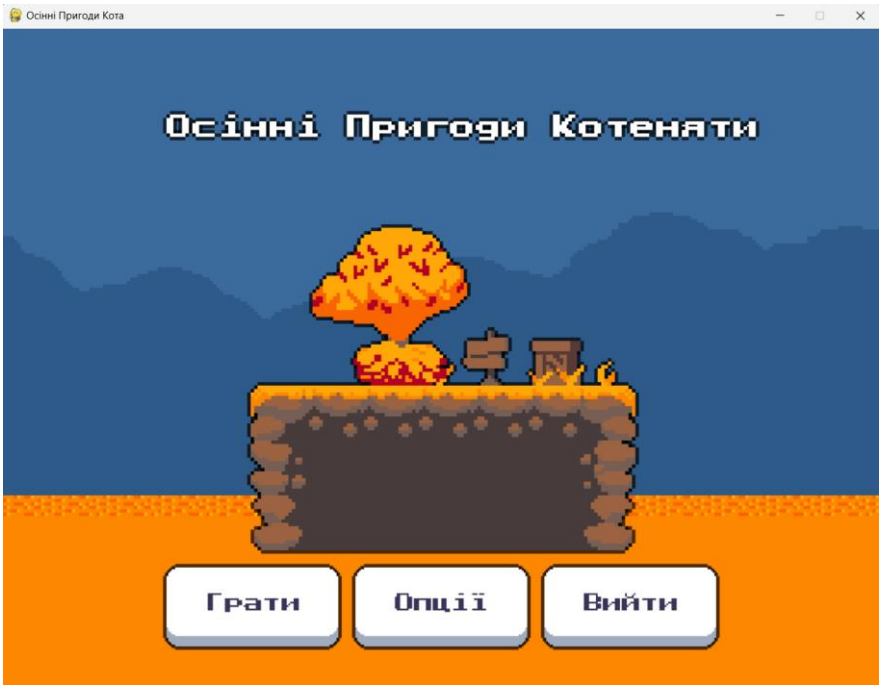

Рисунок 3.3.1 – Кнопки головного меню гри

Коли натиснуто кнопку «Грати», запускається перший рівень гри, де користувач має можливість керувати геймплеєм (рисунок 3.3.2).

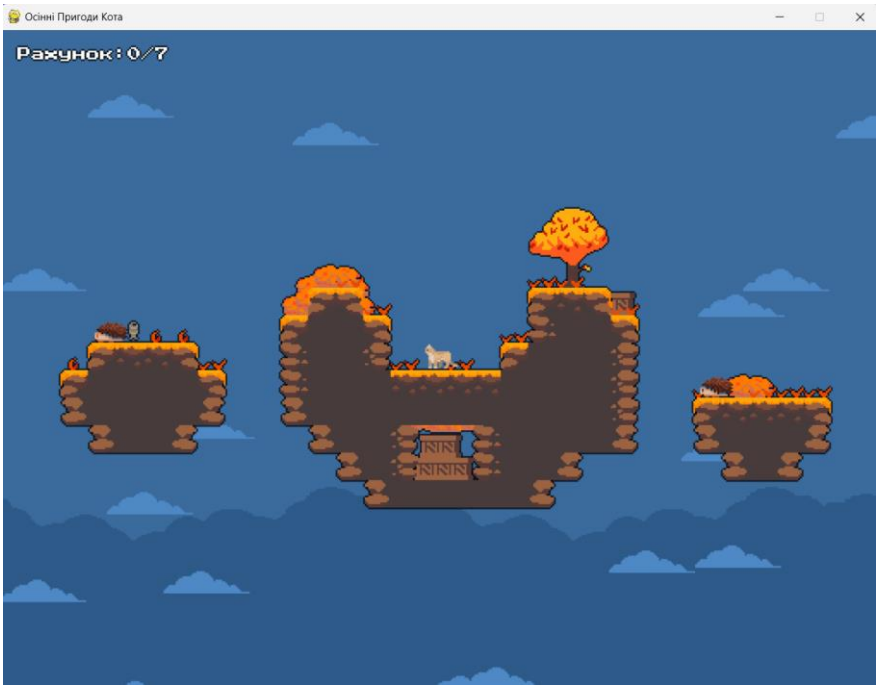

Рисунок 3.3.2 – Результат натискання кнопки «Грати»

Коли натиснуто кнопку «Опції», відкривається нове вікно, на якому можна змінити звук та його гучність (рисунок 3.3.3).

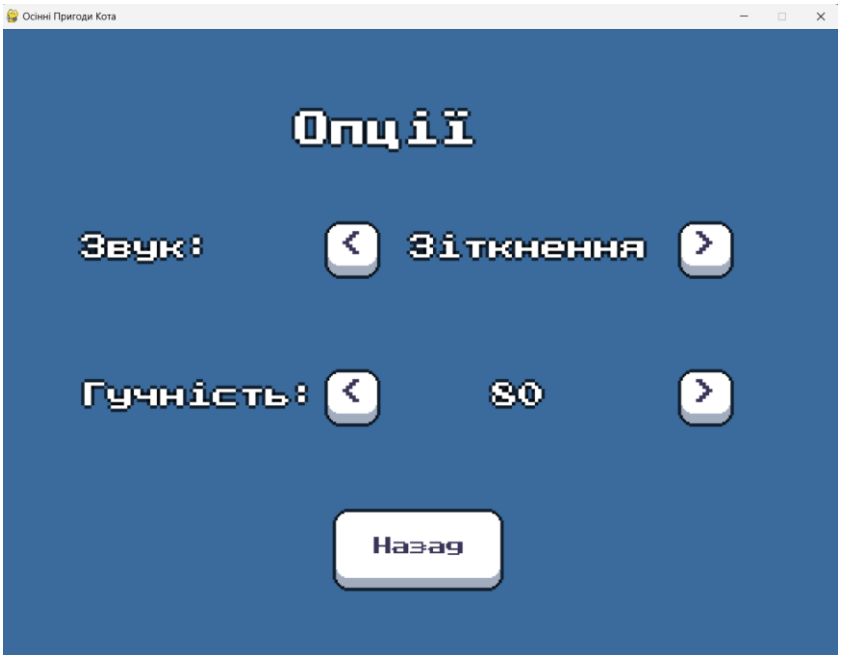

Рисунок 3.3.3 – Результат натискання кнопки «Опції»

Коли натиснуто кнопку «Вийти», гра успішно завершується.

Під час основного рівня користувач має можливість керувати персонажем за допомогою кнопок на клавіатурі, зокрема, здійснювати стрибки (рисунок 3.3.4).

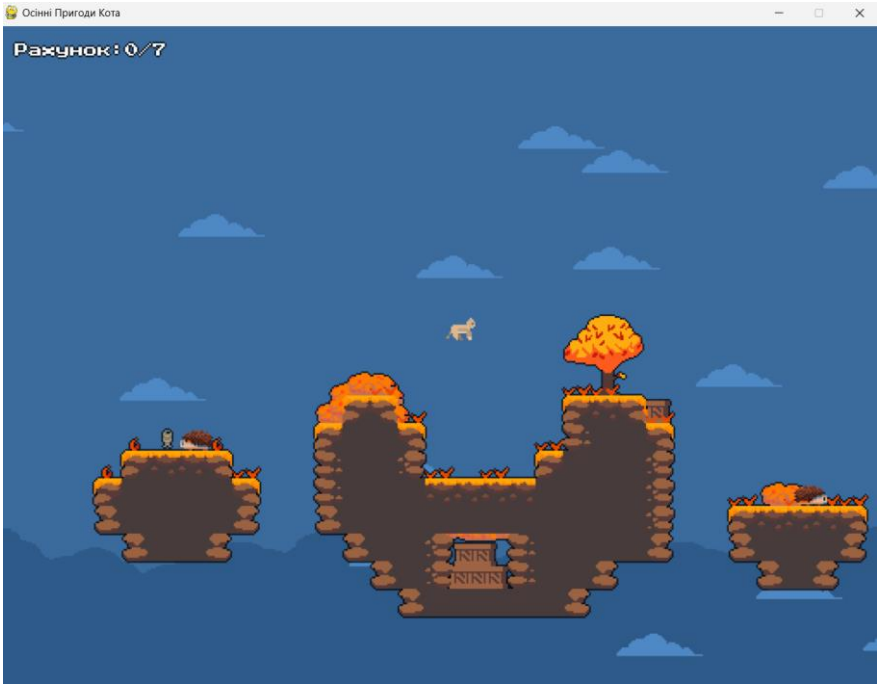

Рисунок 3.3.4 – Процес гри

Також під час процесу гри після натискання кнопки Esc на клавіатурі гра зупиняється на паузу, відкривається відповідне вікно з надписом і функціонуючими кнопками «Назад», «Опції» та «Меню» (рисунок 3.3.5).

Рисунок 3.3.5 – Пауза гри

На карті існують ворожі персонажі (їжаки), після дотику з якими (рисунок 3.3.6 (а)) гравець програє та починає раунд заново (рисунок 3.3.6 (б)).

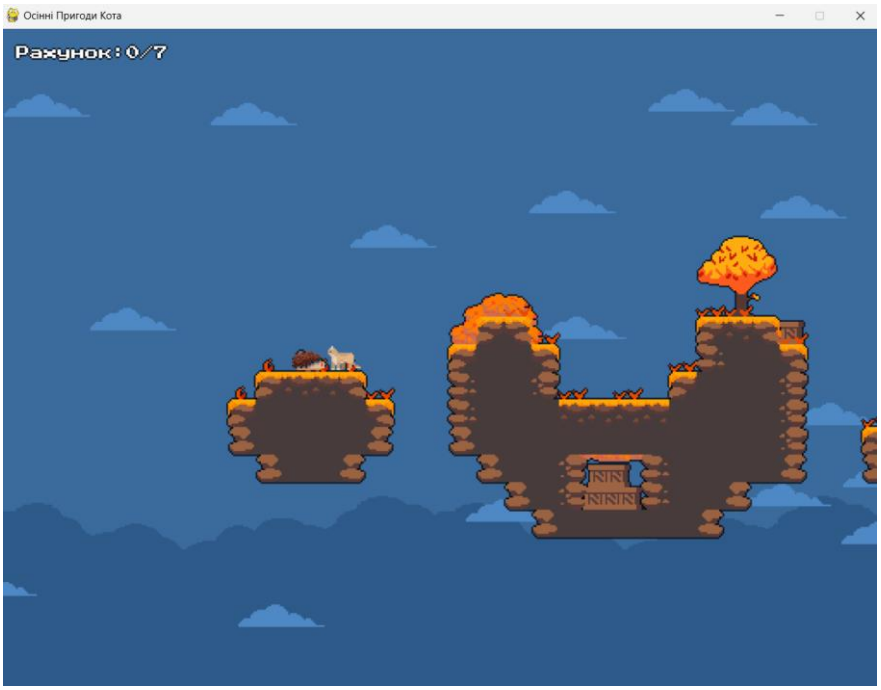

Рисунок 3.3.6 (а) – Дотик до ворожого об’єкта

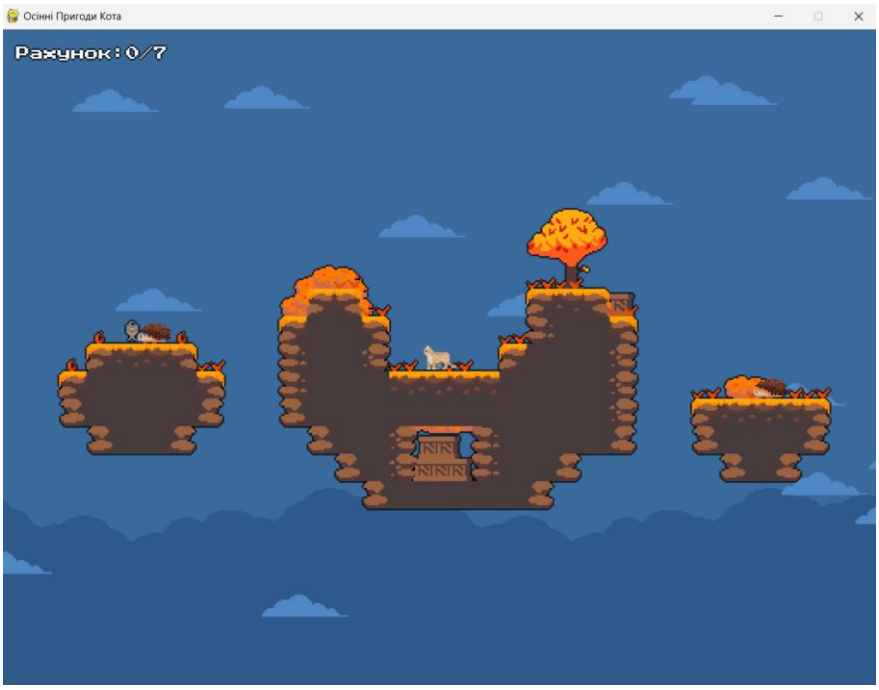

Рисунок 3.3.6 (б) – Результат дотику до ворожого об’єкта

Окрім ворогів, у грі існує їжа для головного персонажа – риба. Коли кіт з’їдає рибу, ігновий рахунок збільшується на одну одиницю (рисунок 3.3.7).

Рисунок 3.3.7 – Збільшення ігрового рахунку

Після повного проходження двох рівнів гри на екрані з’являється повідомленння про завершення ігрового процесу (рисунок 3.3.8).

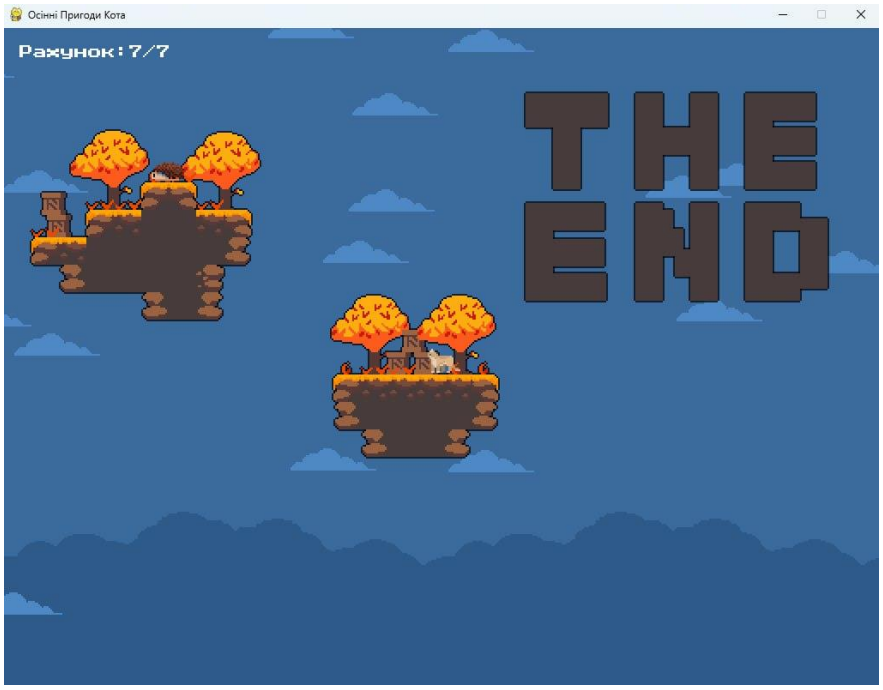

Рисунок 3.3.8 – Завершення гри

Отже, завдяки тестуванню основних розроблених функцій критичних помилок виявлено не було. Успішно здійснено запуск гри, а код виконано без помилок.

4.БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Вимоги до організації робочого місця оператора ПК

Психологія безпеки праці є важливим розділом загальної психології, що вивчає психологічні аспекти, пов'язані з забезпеченням безпечних умов праці. Вона охоплює аналіз психологічних факторів, які можуть впливати на поведінку працівників, їх реакції на стресові ситуації, взаємодію в колективі та дотримання правил безпеки. Ось кілька ключових аспектів цієї теми:

Психологічні фактори ризику

Вплив стресу і втоми:

Стрес та втома можуть знижувати концентрацію уваги, що збільшує ймовірність помилок та нещасних випадків на робочому місці. Наприклад, працівники, які знаходяться під постійним стресом, можуть бути менш уважними до деталей та інструкцій з безпеки. Психофізіологічний стан працівника:

Фізичний і психічний стан працівника, включаючи рівень втоми, стресу та емоційного вигоряння, може впливати на його здатність виконувати роботу безпечно. Наприклад, втомлені працівники можуть бути схильнішими до неуважності та необачних дій. Грибан В.Г., Негодченко О.В. Охорона праці. – К.: Центр учбової літератури, 2009. 209 с.

1. Мотивація та ставлення до безпеки

Мотивація до безпечної поведінки:

Мотивація працівників дотримуватися правил безпеки може бути підвищена через належну організацію праці, створення стимулів та позитивного клімату на робочому місці. Наприклад, премії за дотримання правил безпеки можуть стимулювати працівників бути більш уважними.

Ставлення до ризику:

Різні працівники мають різний рівень толерантності до ризику, що впливає на їх поведінку. Деякі можуть свідомо ігнорувати правила безпеки, недооцінюючи небезпеку.

2. Вплив колективу та організаційної культури

Взаємодія в колективі та підтримка серед колег можуть сприяти більшій відповідальності за дотримання правил безпеки. Колектив, де прийнято нагадувати один одному про небезпеку та підтримувати заходи безпеки, зазвичай має менше нещасних випадків.

Організаційна культура:

Організаційна культура, що підкреслює важливість безпеки, створює середовище, де працівники відчувають підтримку і мають високу мотивацію дотримуватися правил. Наприклад, регулярні тренінги з безпеки та відкритий діалог між працівниками та керівництвом можуть покращити ситуацію з безпекою на підприємстві. Яремко З.М. Безпека життєдіяльності. Навчальний посібник. – Львів: Видавничий центр ЛНУ ім. І. Франка, 2005. 301 с.

3. Навчання та тренінги

Психологічна підготовка:

Проведення тренінгів та навчання з питань безпеки може підвищити обізнаність працівників та їх готовність діяти у надзвичайних ситуаціях. Працівники, які добре знають правила безпеки, більш схильні дотримуватися їх у повсякденній роботі.

Роль лідерства:

Керівники та лідери груп можуть відігравати важливу роль у формуванні ставлення до безпеки серед працівників. Вони можуть слугувати прикладом для наслідування та впливати на поведінку підлеглих.

Психологія безпеки праці в загальній проблемі психології охоплює багато аспектів, які важливі для створення безпечних умов праці. Це включає управління стресом, мотивацію, вплив колективу та організаційної культури, а також роль навчання та лідерства. Важливо враховувати ці фактори для забезпечення безпеки та здоров'я працівників на робочому місці.

3.2 Методи оцінки соціальної та соціально-економічної ефективності заходів щодо покращення умов та охорони праці.

Ефективність заходів щодо поліпшення умов і охорони праці оцінюється, в першу чергу, за показниками соціальної ефективності, які передбачають створення умов праці, що відповідають санітарним нормам і вимогам правил безпеки. Покращення умов і охорони праці призводить до зменшення кількості виробничих травм, загальної і професійної захворюваності; до скорочення чисельності працівників, що працюють в умовах, які не відповідають санітарно-гігієнічним нормам; зменшення кількості випадків виходу на пенсію за інвалідністю внаслідок травматизму чи професійної захворюваності; скорочення плинності кадрів через незадовільні умови праці тощо.

Соціально-економічна ефективність розраховується з метою:

– економічного обґрунтування планових заходів, необхідних для вибору оптимальних варіантів технологічних, ергономічних та організаційних рішень;

– визначення фактичної ефективності заходів щодо поліпшення умов і охорони праці;

– оцінки результатів управління виробництвом на різних рівнях;

– розрахунку необхідних витрат для приведення умов праці на робочих місцях у відповідність до нормативних вимог;

– визначення раціональних розмірів матеріального стимулювання працівників підприємства, науково-дослідних, конструкторських і проектних організацій за розробку і запровадження працеохоронних заходів.

Оцінка соціально-економічної ефективності працеохоронних заходів здійснюється на підприємствах усіх форм власності, у тому числі на робочому місці, дільниці, в цеху. Вона може визначатися також по галузі та в державі в цілому.

Показники соціальної і соціально-економічної ефективності розраховуються як відношення величин соціальних або соціально-економічних результатів до витрат, необхідних для їх здійснення.

Такі показники характеризують кількість умовних одиниць сукупного об'єму соціального чи соціально-економічного результату в розрахунку на одиницю витрат.

Показники соціальної і соціально-економічної ефективності використовуються для визначення фактичного рівня питомих витрат, необхідних для зменшення кількості працюючих у незадовільних умовах, зниження рівня травматизму, захворюваності, плинності кадрів на різних підприємствах та в економіці в цілому.

Оцінювати економічні аспекти охорони праці слід за допомогою методів оцінки соціальної й економічної ефективності заходів по створенню умов праці, що відповідають чинним нормативним актам з охорони праці.

Для оцінки соціальної ефективності заходів з удосконалення умов та охорони праці використовуються такі показники:

1. Скорочення кількості робочих місць (А/С), що не відповідають вимогам нормативних актів щодо безпеки праці, розраховується за формулою:

$$\Delta K = \frac{K_1 - K_2}{K_з} \cdot 100\%,$$

де Ки К2 – кількість робочих місць, що не відповідають санітарним нормам до і після проведення заходів; К3 – загальна кількість робочих місць.

2. Скорочення чисельності працівників (АЧ), які працюють в умовах, що не відповідають санітарним нормам, визначається за формулою:

$$\Delta Ч = \frac{Ч_1 - Ч_2}{Ч_з} \cdot 100\%,$$

де Чи Ч2 – чисельність працівників, які працюють в умовах, що не відповідають санітарним нормам до і після здійснення заходу, осіб;

Ч3 – річна середньо-облікова чисельність працівників, осіб.

3. Збільшення кількості машин, механізмів (ДЛ/) та виробничих приміщень (Д£), приведених до вимог норм охорони праці, визначається за формулами:

$$\Delta M = \frac{M_1 - M_2}{M_з} \cdot 100\%,$$

$$\Delta Б = \frac{Б_1 - Б_2}{Б_з} \cdot 100\%,$$

де М, М2 – кількість машин, механізмів, що не відповідають нормативним вимогам до і після проведення заходу, шт.;

М3 – загальна кількість машин і механізмів, шт.;

Б і, 2>2 – кількість виробничих приміщень, які не відповідають нормативним вимогам до і після здійснення заходу, шт.;

Б3 – загальна кількість виробничих приміщень, шт.

4. Зменшення коефіцієнта частоти травматизму (АКЧ) визначається за формулою:

$$\Delta K_ч = \frac{N_1 - N_2}{Ч_з} \cdot 1000,$$

де N1, N2 – кількість випадків травматизму відповідно до і після проведення заходу;

Ч3 – річна середньооблікова чисельність працівників, осіб.

5. Зменшення коефіцієнта тяжкості травматизму (АКТ):

$$\Delta K_Т = \frac{Д_1}{N_1} - \frac{Д_2}{N_2},$$

де Д, Ді – кількість днів непрацездатності через травматизм відповідно до і після проведення заходу.

6. Зменшення коефіцієнта частоти професійних захворювань через несприятливі умови праці:

$$\Delta K_з = \frac{З_1 - З_2}{Ч_з} \cdot 100,$$

де Зі, 32 – кількість випадків професійних захворювань відповідно до і після проведення заходу.

7. Зменшення коефіцієнта тяжкості захворювання:

$$\Delta K_{зт} = \frac{Д_{з1}}{N_{з1}} - \frac{Д_{з2}}{N_{з2}},$$

де Д3, Д32 – кількість днів тимчасової непрацездатності через хвороби відповідно до і після проведення заходу;

N зі" Мз2 – кількість випадків захворювання відповідно до і після проведення заходу.

8. Зменшення кількості випадків виходу на пенсію за інвалідністю внаслідок травматизму чи професійного захворювання:

$$\Delta Ч_i = \frac{Ч_{i1} - Ч_{i2}}{Ч_з} \cdot 100,$$

де Чц, 4(2 – чисельність працівників, що стали інвалідами відповідно до і після проведення заходу, осіб.

9. Скорочення плинності кадрів через несприятливі умови праці:

$$\Delta Ч_П = \frac{Ч_{П1} - Ч_{П2}}{Ч_з} \cdot 100,$$

де Чп, Чпг – кількість працівників, що звільнилися за власним бажанням через несприятливі умови праці відповідно до і після проведення заходу, осіб. Гандзюк М.П., Желібо Є.П., Халімовський М.О. Основи охорони праці: Підручник. 5-е вид. / За ред. М.П. Гандзюка. – К.: Каравела, 2011. – 384 с.

Оцінка соціально-економічної ефективності працеохоронних заходів проводиться на всіх рівнях підприємства, а також по галузі та в державі в цілому. Показники ефективності розраховуються як відношення соціальних або соціально-економічних результатів до витрат, необхідних для їх здійснення. Ці показники використовуються для визначення рівня витрат, необхідних для зменшення кількості працюючих у незадовільних умовах, зниження рівня травматизму, захворюваності та плинності кадрів.

Для оцінки соціальної ефективності використовуються конкретні формули, що дозволяють оцінити скорочення кількості небезпечних робочих місць, чисельності працівників у несприятливих умовах, зменшення коефіцієнтів частоти та тяжкості травматизму, професійних захворювань, а також зниження плинності кадрів.

Таким чином, комплексний підхід до оцінки ефективності заходів з охорони праці дозволяє підприємствам приймати обґрунтовані рішення для покращення умов праці та підвищення безпеки на робочих місцях, що сприяє підвищенню продуктивності та зменшенню витрат, пов’язаних з виробничими травмами та захворюваннями.

ВИСНОВКИ

Результатом дипломного проєкту є 2D платформер, який розроблений за допомогою бібліотеки PyGame, що забезпечує цікавий геймплей та креативний рівень дизайну. Загальною метою 2D платформера є створення ігрового досвіду для гравців, де вони зможуть насолоджуватися дизайном та ігровим процесом загалом.

Під час розробки диплому були розглянуті основні принципи створення ігрових додатків та проаналізовано прогрес технологій розробки ігор. Спроєктровано 2D платформер із використанням бібліотеки PyGame. Для успішної розробки програмного продукту створено головне меню та налаштування, реалізовано основну логіку гри, враховано аспекти графіки, анімації, звукового супроводу та обробки подій. Проведено аналіз мов програмування та інших інструментів, які використовуються для розробки ігор.

Розроблена гра містить різноманітні функції та елементи, що забезпечують захоплюючий ігровий процес. У грі є опції для зміни музики і гучності, що дозволяє гравцям налаштовувати аудіосупровід. Сюжет гри розгортається навколо кота, яка стрибає по платформах-островах та збирає рибу. Гравець має уникати ворожих об'єктів, оскільки під час дотику до них – програє. Цей ігровий механізм додає елемент виклику та стратегії, що робить гру більш інтерактивною.

Проведений тест гри забезпечує необхідну стабільну функціональність і відповідає очікуванням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Платформер. URL: https://uk.wikipedia.org/wiki/Платформер (дата звернення 07.05.2024).
2. Pixlr. URL: https://en.wikipedia.org/wiki/Pixlr (дата звернення 08.05.2024).
3. Tiled. URL: https://github.com/mapeditor/tiled/wiki (дата звернення 08.05.2024).
4. Мова програмування Python. URL: https://edu.cbsystematics.com/ua/blog/pythonstart-blog (дата звернення 14.05.2024).
5. Що таке мова Python та які сфери її застосування. URL: https://university.sigma.software/what-is-python/ (дата звернення 14.05.2024).
6. PyGame. URL: https://uk.wikipedia.org/wiki/Pygame (дата звернення 14.05.2024).
7. PyGame Tiles. URL: https://pygame.readthedocs.io/en/latest/tiles/tiles.html (дата звернення 18.05.2024).
8. PyGame Sprite. URL: https://www.pygame.org/docs/ref/sprite.html (дата звернення 20.05.2024).
9. Якими є нові вимоги до роботи за комп'ютером? URL: https://news.dtkt.ua/labor/labor-relations/51073-iakimi-je-novi-vimogi-do-roboti-za-kompiuterom-nagaduvannia-vid-derzpraci (дата звернення 21.05.2024).
10. Безпека в розробці ігор. URL: https://dzone.com/articles/security-in-game-development (дата звернення 21.05.2024).
11. Game Development using Python. https://styluspub.presswarehouse.com/browse/book/9781683926276/Game-Development-Using-Python
12. Грибан В.Г., Негодченко О.В. Охорона праці. – К.: Центр учбової літератури, 2009. 209 с.
13. Яремко З.М. Безпека життєдіяльності. Навчальний посібник. – Львів: Видавничий центр ЛНУ ім. І. Франка, 2005. 301 с.
14. Гандзюк М.П., Желібо Є.П., Халімовський М.О. Основи охорони праці: Підручник. 5-е вид. / За ред. М.П. Гандзюка. – К.: Каравела, 2011. – 384 с.
15. Мудрик І. Орієнтований на дані дизайн. Аналіз та переваги 181-182 с. (ТНТУ)

ДОДАТКИ

Додаток А

Програмний код проєкту

Лістинг А1 – Код файлу main.py

```
class AdventureGame:
    def load_level(self, lvl_id):
        self.tilemap.load('assets/maps/' + str(lvl_id) + '.json')
        self.enemies = []
        self.gems = []
        for spawner in self.tilemap.extract([('spawners', 0),
('spawners', 1), ('spawners', 2)]):
            if spawner['variant'] == 0:
                self.player.rect_pos.x = spawner['pos'][0]
                self.player.rect_pos.y = spawner['pos'][1]
                self.player.air_time = 0
            elif spawner['variant'] == 2:
                self.gems.append(Gem(self, spawner['pos'], (16, 16)))
            else:
                self.enemies.append(Enemy(self, spawner['pos'], (11,
16)))

    def run(self):
        self.sfx['ambience'].play(-1)
        while True:
            self.display.fill((0, 0, 0, 0))
            self.display_2.fill((59, 107, 156))
            self.scroll[0] += (self.player.rect_pos.centerx -
self.display.get_width() / 2 - self.scroll[0]) / 30
            self.scroll[1] += (self.player.rect_pos.centery -
self.display.get_height() / 2 - self.scroll[1]) / 90
            render_scroll = (int(self.scroll[0]), int(self.scroll[1]))
            Text(self.display,
f'Рахунок:{self.score}/{self.max_score}', self.get_font(8), (10,
10)).render()
            self.clouds.update()
            self.clouds.render(self.display_2, offset=render_scroll)
            self.screenshake = max(0, self.screenshake - 1)
            self.tilemap.render(self.display, offset=render_scroll)
            display_mask = pygame.mask.from_surface(self.display)
            display_sillhouette = display_mask.to_surface(setcolor=(0,
0, 0, 180), unsetcolor=(0, 0, 0, 0))
            for offset in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
                self.display_2.blit(display_sillhouette, offset)
            if not self.dead:
                self.player.update(self.tilemap, (self.movement[1] -
self.movement[0], 0))
                self.player.render(self.display, offset=render_scroll)
            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN and not
self.level_ended:
                    if event.key == pygame.K_LEFT:
                        self.movement[0] = True
                    if event.key == pygame.K_RIGHT:
                        self.movement[1] = True
```

Продовження лістинга А1

```
                    if event.key == pygame.K_UP:
                        if self.player.jump():
                            self.sfx['jump'].play()
                    if event.key == pygame.K_a:
                        self.movement[0] = True
                    if event.key == pygame.K_d:
                        self.movement[1] = True
                    if event.key == pygame.K_w:
                        if self.player.jump():
                            self.sfx['jump'].play()
                    if event.key == pygame.K_ESCAPE:
                        self.pause_menu()
                if event.type == pygame.KEYUP:
                    if event.key == pygame.K_LEFT:
                        self.movement[0] = False
                    if event.key == pygame.K_RIGHT:
                        self.movement[1] = False
                    if event.key == pygame.K_a:
                        self.movement[0] = False
                    if event.key == pygame.K_d:
                        self.movement[1] = False
            self.display_2.blit(self.display, (0, 0))
            screenshake_offset = (random.random() * self.screenshake /
2, random.random() * self.screenshake - self.screenshake / 2)
            self.screen.blit(pygame.transform.scale(self.display_2,
self.screen.get_size()), screenshake_offset)
            pygame.display.update()
            self.clock.tick(60)

    def menu(self):
        self.sfx['ambience'].stop()
        self.load_level(self.level)
        self.transition = 0
        switch = 1
        if self.level == 0:
            play_button_text = 'Грати'
        else:
            play_button_text = 'Продовжити'
        button_width, button_height = 100, 40
        screen_center_x = self.display.get_width() / 2
        play_button = Button(
            self.display,
            self.get_font(11),
            play_button_text,
            button_width,
            button_height,
            (screen_center_x - button_width // 2 - 110, 320),
            6
        )
        option_button = Button(self.display, self.get_font(11),
'Опції', button_width, button_height,
                               (screen_center_x - button_width // 2,
320), 6)
```

Продовження лістинга А1

```
        quit_button = Button(self.display, self.get_font(11), 'Вийти',
button_width, button_height,
                             (screen_center_x - button_width // 2 +
110, 320), 6)
        while True:
            self.display.fill((0, 0, 0, 0))
            self.display_2.fill((59, 107, 156))
            self.display_2.blit(self.assets['menu_trees'], (0, 0))
            self.display_2.blit(self.assets['menu_water'], (0, 0))
            self.display.blit(self.assets['menu_grass'],
(screen_center_x / 2, 0))
            if not switch:
                self.transition += 1
                if self.transition > 30:
                    self.transition = - 30
                    self.run()
            if self.transition < 0:
                self.transition += 1
            Text(self.display, 'Осінні Пригоди Котеняти',
self.get_font(15), (screen_center_x-160, 50)).render()
            play_button.render()
            if play_button.done == True:
                switch = 0
            option_button.render()
            if option_button.done == True:
                self.last_state = 'menu'
                self.option_menu()
                option_button.done = False
            quit_button.render()
            if quit_button.done == True:
                pygame.quit()
                sys.exit()
            display_mask = pygame.mask.from_surface(self.display)
            display_sillhouette = display_mask.to_surface(setcolor=(0,
0, 0, 180), unsetcolor=(0, 0, 0, 0))
            for offset in [(-2, 0), (2, 0), (0, -2), (0, 2)]:
                self.display_2.blit(display_sillhouette, offset)
            self.display_2.blit(self.display, (0, 0))
            self.screen.blit(pygame.transform.scale(self.display_2,
self.screen.get_size()), (0, 0))
            pygame.display.update()
            self.clock.tick(60)

    def pause_menu(self):
        self.movement = [False, False]
        self.sfx['ambience'].stop()
        self.transition = 0
        switch = 1
        play_button = Button(self.display, self.get_font(11), 'Назад',
100, 40, (205, 150), 6)
        option_button = Button(self.display, self.get_font(11),
'Опції', 100, 40, (205, 225), 6)
        quit_button = Button(self.display, self.get_font(11), 'Меню',
100, 40, (205, 300), 6)
```

Продовження лістинга А1

```
        while True:
            self.display.fill((0, 0, 0, 0))
            self.display_2.fill((59, 107, 156))
            if not switch:
                self.transition += 1
                if self.transition > 30:
                    self.transition = - 30
                    self.run()
            if self.transition < 0:
                self.transition += 1
            Text(self.display, 'Пауза', self.get_font(22), (192,
50)).render()
            play_button.render()
            if play_button.done == True:
                switch = 0
            option_button.render()
            if option_button.done == True:
                self.last_state = 'pause'
                self.option_menu()
                option_button.done = False
            quit_button.render()
            if quit_button.done == True:
                self.menu()
            display_mask = pygame.mask.from_surface(self.display)
            display_sillhouette = display_mask.to_surface(setcolor=(0,
0, 0, 180), unsetcolor=(0, 0, 0, 0))
            for offset in [(-2, 0), (2, 0), (0, -2), (0, 2)]:
                self.display_2.blit(display_sillhouette, offset)
            self.display_2.blit(self.display, (0, 0))
            self.screen.blit(pygame.transform.scale(self.display_2,
self.screen.get_size()), (0, 0))
            pygame.display.update()
            self.clock.tick(60)

    def option_menu(self):
        self.movement = [False, False]
        self.sfx['ambience'].stop()
        self.transition = 0
        switch = 1
        back_button = Button(self.display, self.get_font(11), 'Назад',
100, 40, (205, 300), 6)
        sound_button_l = Button(self.display, self.get_font(16), '<',
30, 25, (200, 125), 6)
        sound_button_r = Button(self.display, self.get_font(16), '>',
30, 25, (415, 125), 6)
        volume_buttom_l = Button(self.display, self.get_font(16), '<',
30, 25, (200, 215), 6)
        volume_buttom_r = Button(self.display, self.get_font(16), '>',
30, 25, (415, 215), 6)
        sounds = list(self.sfx.copy())
        sounds.append('music')
        sound = 1
        sound_translations = {
            'jump': 'Стрибок',
```

Продовження лістинга А1

```
            'explosion': 'Зіткнення',
            'ambience': 'Фон',
            'gem': 'Рибка',
            'music': 'Музика'
        }
        while True:
            self.display.fill((0, 0, 0, 0))
            self.display_2.fill((59, 107, 156))
            Text(self.display, 'Опції', self.get_font(22), (180,
50)).render()
            Text(self.display, 'Звук:', self.get_font(16), (50,
125)).render()
            Text(self.display, 'Гучність:', self.get_font(16), (50,
215)).render()
            sound_name = sounds[sound]
            translated_sound_name = sound_translations.get(sound_name,
sound_name)
            Text(self.display, translated_sound_name,
self.get_font(16), (250, 125)).render()
            sound_button_1.render()
            volume_buttom_l.render()
            volume_buttom_r.render()
            if sounds[sound] in self.sfx or sounds[sound] == 'music':
                current_vol = getattr(self, f'vol_{sounds[sound]}',
None)
                if current_vol is not None:
                    Text(self.display, f'{round(current_vol * 100)}',
self.get_font(16), (300, 215)).render()
            back_button.render()
            display_mask = pygame.mask.from_surface(self.display)
            display_sillhouette = display_mask.to_surface(setcolor=(0,
0, 0, 180), unsetcolor=(0, 0, 0, 0))
            for offset in [(-2, 0), (2, 0), (0, -2), (0, 2)]:
                self.display_2.blit(display_sillhouette, offset)
            self.display_2.blit(self.display, (0, 0))
            self.screen.blit(pygame.transform.scale(self.display_2,
self.screen.get_size()), (0, 0))
            pygame.display.update()
            self.clock.tick(60)

AdventureGame().menu()
```

Лістинг А1 включає методи для керування гравцем, обробку подій, оновлення дисплея та відображення меню та опцій.

Лістинг А2 – Код файлу button.py

```
class Button:
    def render(self):
        self.top_rect.y = self.org_y_pos - self.dynamic_elevation
        self.text_rect.center = self.top_rect.center
```

Продовження лістинга А2

```
        self.bottom_rect.midtop = self.top_rect.midtop
        self.bottom_rect.height = self.top_rect.height +
self.dynamic_elevation

        pygame.draw.rect(
            self.game,
            self.bottom_color,
            self.bottom_rect,
            border_radius=7
        )

        pygame.draw.rect(
            self.game,
            self.border_color,
            self.top_rect,
            self.border_thickness,
            border_radius=7
        )

        pygame.draw.rect(
            self.game,
            self.top_color,
            self.top_rect.inflate(-self.border_thickness, -
self.border_thickness),
            border_radius=7
        )

        self.game.blit(self.text_surf, self.text_rect)

        self.check_click()
        return self.done

    def check_click(self):
        mpos = pygame.mouse.get_pos()
        mpos = (mpos[0] / SCALE, mpos[1] / SCALE)
        if self.top_rect.collidepoint(mpos):
            self.top_color = '#c8e0e0'
            if pygame.mouse.get_pressed()[0]:
                self.dynamic_elevation = 0
                self.pressed = True
            else:
                self.dynamic_elevation = self.elevation
                if self.pressed:
                    self.done = True
                    self.pressed = False
        else:
            self.dynamic_elevation = self.elevation
            self.top_color = '#FFFFFF'
```

Лістинг А2 визначає методи для відображення кнопки на екрані та перевіряє натискання миші і змінює стан кнопки відповідно.

Лістинг А3 – Код файлу clouds.py

```
class Cloud:
    def update(self):
        self.pos[0] += self.speed

    def render(self, surf, offset=(0, 0)):
        render_pos = (self.pos[0] - offset[0] * self.depth,
self.pos[1] - offset[1] * self.depth)
        surf.blit(self.img, (render_pos[0] % (surf.get_width() +
self.img.get_width()) - self.img.get_width(),
                             render_pos[1] % (surf.get_height() +
self.img.get_height()) - self.img.get_height()))


class Clouds:
    def update(self):
        for cloud in self.clouds:
            cloud.update()

    def render(self, surf, offset=(0, 0)):
        for cloud in self.clouds:
            cloud.render(surf, offset=offset)
```

Лістинг А3 визначає методи для оновлення позиції хмар та відображає їх

Лістинг А4 – Код файлу entities.py

```
class PhysicsEntity:
    def set_action(self, action):
        if action != self.action:
            self.action = action
            self.animation = self.game.assets[self.type + '/' +
self.action].copy_anim()

    def update(self, tilemap, movement=(0, 0)):
        self.collisions = {'up': False, 'down': False, 'right': False,
'left': False}
        frame_movement = (movement[0] + self.velocity[0], movement[1]
+ self.velocity[1])

        self.rect_pos[0] += frame_movement[0]
        self.rect_pos[1] += frame_movement[1]

        if movement[0] > 0:
            self.flip = True
        if movement[0] < 0:
            self.flip = False

        self.velocity[1] = min(5, self.velocity[1] + 0.1)

        if self.collisions['down'] or self.collisions['up']:
            self.velocity[1] = 0

        self.animation.update()
```

Продовження лістинга А4

```
    def render(self, surf, offset=(0, 0)):
        surf.blit(pygame.transform.flip(self.animation.img(),
self.flip, False), (
        self.rect_pos[0] - offset[0] + self.anim_offset[0],
self.rect_pos[1] - offset[1] + self.anim_offset[1]))


class Gem(PhysicsEntity):
    def update(self, tilemap, movement=(0, 0)):
        super().update(tilemap, movement)

        self.set_action('idle')

        if self.rect_pos.colliderect(self.game.player.rect_pos):
            self.game.sfx['gem'].play()
            return True

    def render(self, surf, offset=(0, 0)):
        super().render(surf, offset=offset)


class Enemy(PhysicsEntity):
    def update(self, tilemap, movement=(0, 0)):
        if self.walking:
            if tilemap.solid_check((self.rect_pos.centerx + (-7 if not
self.flip else 7), self.rect_pos[1] + 23)):
                if (self.collisions['right'] or
self.collisions['left']):
                    self.flip = not self.flip
                else:
                    movement = (movement[0] - 0.35 if not self.flip
else 0.35, movement[1])
            else:
                self.flip = not self.flip
            self.walking = max(0, self.walking - 1)
        elif random.random() < 0.01:
            self.walking = random.randint(30, 120)

        super().update(tilemap, movement=movement)

        if movement[0] != 0:
            self.set_action('run')
        else:
            self.set_action('idle')

    def render(self, surf, offset=(0, 0)):
        super().render(surf, offset=offset)

class Player(PhysicsEntity):
    def update(self, tilemap, movement=(0, 0)):
        super().update(tilemap, movement=movement)

        self.air_time += 1
```

Продовження лістинга А4

```
        if self.air_time > 200:
            self.game.dead += 1

        if self.collisions['down']:
            self.air_time = 0
            self.jumps = 1

        if self.air_time > 4:
            self.set_action('jump')
        elif movement[0] != 0:
            self.set_action('run')
        else:
            self.set_action('idle')

    def render(self, surf, offset=(0, 0)):
        super().render(surf, offset=offset)

    def jump(self):
        if self.jumps:
            self.velocity[1] = -3
            self.jumps -= 1
            self.air_time = 5
            return True
```

Лістинг А4 визначає класи для фізичних об'єктів гри та керування їхнім рухом.

Лістинг А5 – Код файлу particle.py

```
class Particle:
    def update(self):
        kill = False
        if self.animation.done:
            kill = True

        self.pos[0] += self.velocity[0]
        self.pos[1] += self.velocity[1]

        self.animation.update()

        return kill

    def render(self, surf, offset=(0, 0)):
        img = self.animation.img()
        surf.blit(img,
                  (self.pos[0] - offset[0] - img.get_width() // 2,
self.pos[1] - offset[1] - img.get_height() // 2))
```

Лістинг А5 визначає клас для оновлення позиції частинки, перевірки, чи завершилася анімація для її видалення, та відображення частинки на екрані.

Лістинг А6 – Код файлу support.py

```
BASE_PATH = 'assets/images/'

def load_image(path):
    img = pygame.image.load(BASE_PATH + path).convert()
    img.set_colorkey((0, 0, 0))
    return img

def load_images(path):
    images = []
    for img_name in sorted(os.listdir(BASE_PATH + path)):
        images.append(load_image(path + '/' + img_name))
    return images

class Animation:
    def copy_anim(self):
        return Animation(self.images, self.img_duration, self.loop)

    def update(self):
        if self.loop:
            self.frame = (self.frame + 1) % (self.img_duration *
len(self.images))
        else:
            self.frame = min(self.frame + 1, self.img_duration *
len(self.images) - 1)
            if self.frame >= self.img_duration * len(self.images) - 1:
                self.done = True

    def img(self):
        return self.images[int(self.frame / self.img_duration)]

class Text:
    def render(self):
        text = self.font.render(self.text, False, (255, 255, 255))
        self.game.blit(text, self.pos)

def scale_images(images, size):
    return [pygame.transform.scale(image, size) for image in images]
```

Лістинг А6 відповідає за завантаження зображень та списків зображень із диска, керування анімаціями та відображення тексту на екрані.

Лістинг А7 – Код файлу tilemap.py

```
class Tile:
    def __init__(self, type: str, variant: int, pos) -> None:
        self.type = type
        self.variant = variant
        self.pos = tuple((int(pos[0]), int(pos[1])))
        self.tile_dict = {'type': self.type, 'variant': self.variant,
'pos': self.pos}
```

Продовження лістинга А7

```
    def __getitem__(self, key):
        return self.tile_dict[key]

    def set_variant(self, new_variant):
        self.variant = new_variant
        self.tile_dict['variant'] = new_variant

    def __json__(self):
        return self.tile_dict

class CustomEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, Tile):
            return obj.__json__()
        return json.JSONEncoder.default(self, obj)

class Tilemap:
    def extract(self, id_pairs, keep=False):
        matches = []
        for tile in self.offgrid_tiles.copy():
            if (tile['type'], tile['variant']) in id_pairs:
                matches.append(tile.copy())
                if not keep:
                    self.offgrid_tiles.remove(tile)

        for loc in self.tilemap:
            tile = self.tilemap[loc]
        return matches

    def tiles_around(self, pos):
        tiles = []
        tile_loc = (int(pos[0] // self.tile_size), int(pos[1] //
self.tile_size))
        return tiles

    def solid_check(self, pos):
        tile_loc = str(int(pos[0] // self.tile_size)) + ';' +
str(int(pos[1] // self.tile_size))
        if tile_loc in self.tilemap:
            if self.tilemap[tile_loc]['type'] in PHYSICS_TILES:
                return self.tilemap[tile_loc]

    def physics_rects_around(self, pos):
        rects = []
        for tile in self.tiles_around(pos):
            if tile['type'] in PHYSICS_TILES:
                rects.append(
                    pygame.Rect(tile['pos'][0] * self.tile_size,
tile['pos'][1] * self.tile_size, self.tile_size,
                                self.tile_size))
        return rects

    def end_tile(self):
```

```
        for tile in self.offgrid_tiles:
```

Продовження лістинга А7

```
            if tile['type'] == 'decor' and tile['variant'] == 3:
                return pygame.Rect(tile['pos'][0], tile['pos'][1],
self.tile_size, self.tile_size)
        return pygame.Rect(0, 0, 2, 2)

    def autotile(self):
        for loc in self.tilemap:
            tile = self.tilemap[loc]
            neighbors = set()
            for shift in [(1, 0), (-1, 0), (0, -1), (0, 1)]:
                check_loc = str(tile['pos'][0] + shift[0]) + ';' +
str(tile['pos'][1] + shift[1])
                if check_loc in self.tilemap:
                    if self.tilemap[check_loc]['type'] ==
tile['type']:
                        neighbors.add(shift)
            neighbors = tuple(sorted(neighbors))
            if (tile['type'] in PHYSICS_TILES) and (neighbors in
AUTOTILE_MAP):
                try:
                    tile['variant'] = AUTOTILE_MAP[neighbors]
                except:
                    pass
                try:
                    tile.set_variant(AUTOTILE_MAP[neighbors])
                except:
                    pass

    def render(self, surf, offset=(0, 0)):
        for tile in self.offgrid_tiles:
            surf.blit(self.game.assets[tile['type']][tile['variant']],
                      (tile['pos'][0] - offset[0], tile['pos'][1] -
offset[1]))

        for x in range(offset[0] // self.tile_size, (offset[0] +
surf.get_width()) // self.tile_size + 1):
            for y in range(offset[1] // self.tile_size, (offset[1] +
surf.get_height()) // self.tile_size + 1):
                loc = str(x) + ';' + str(y)
                if loc in self.tilemap:
                    tile = self.tilemap[loc]

surf.blit(self.game.assets[tile['type']][tile['variant']], (
                    tile['pos'][0] * self.tile_size - offset[0],
tile['pos'][1] * self.tile_size - offset[1]))
```

Лістинг А7 визначає класи для керування платформами в грі.

Додаток Б

Диск із матеріалами кваліфікаційної роботи