МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: **Розробка ігрового додатку на мові Python**

Виконав студент групи **К-21і**
спеціальності 122 Комп'ютерні науки
Притикін Артем Сергійович

Керівник **ст. викладач**
Штефан Наталія Зінов'ївна

Консультант **док. техн. наук,**
професор
Казакова Надія Феліксівна

Рецензент **к.т.н., доцент**
Сергієнко Андрій Володимирович

Одеса 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

OpenGL – Open Graphics Library

UML – Unified Modeling Language

Arcade – бібліотека для розробки ігор мовою Python

C – мова програмуваня

Java – мова програмуваня

Python – мова програмуваня

DirectX – це набір API, розроблених для вирішення завдань, пов'язаних із програмуванням

Pygame – бібліотека для розробки ігор мовою Python

Pyglet – бібліотека для розробки ігор мовою Python

PyOpenGL –міжплатформна прив’язка Python до OpenGL

Use-Case – варіант використання

ВСТУП

Розробка ігор пройшла довгий шлях від написання тисяч рядків коду до буквально розробки повної гри з нуля, фактично не знаючи нічого про мови програмування. Однак мови програмування все ще є масовою та невід'ємною частиною створення будь-якого програмного забезпечення чи гри.

У жовтні 2021 року найпопулярнішою мовою програмування було визнано Python, що стало для нього найвагомішим досягненням за останні кілька десятиліть. Він зумів обійти своїх основних конкурентів (знамениті мови C та Java) у авторитетному рейтингу компанії Tiobe. Python, як інструмент швидкої і відносно недорогий розробки, став дуже популярний серед стартапів. Тобто там, де ідею потрібно втілити максимально швидко і просто. Python – це мова загального призначення, до того ж може розширюватися за допомогою мов C і C ++.

Мова програмування Python гнучка, сучасна та популярна мова, яка використовується практично у всіх областях нового покоління. Python ідеально для аналізу даних, в розробці WEB-додатків, ігор, програм та скриптів з автоматизації та системного адміністрування, для написання чат-ботів та штучного інтелекту. Мова з величезною кількістю готових якісних модулів на всі випадки життя, що дає можливість не винаходити колесо на кожному кроці [1].

Зараз маленькі ігри, написані мовою Python часто зустрічаються у портфоліо починаючих розробників. Ця мова ідеальна для платформерів, новел та аркадних ігор.

Метою дипломного проекту є розробка ігрового додатку на мові Python, який зацікавить молодшу цільову аудиторії ігор-платформерів.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Ігри-платформери

Ігри на платформі дозволяють гравцеві керувати персонажем, який рухається іншим світом, долаючи перешкоди, заробляючи очки та перемагаючи ворогів. З початку вісімдесятих цей вид геймплея був популярний в ігрових приставках, аркадах і портативних іграх.

Платформна гра, яку зазвичай називають «платформер», – це стиль відеоігор, у якому гравець змушує персонажа рухатися середовищем за допомогою серії рухів, заснованих на дії, як-от біг, стрибки чи розгойдування на мотузках.

Платформні ігри виникли на початку 1980-х років, коли персонажі виконували дії на одному екрані та переходили на новий екран після проходження рівня. Пізніше в цьому десятилітті рух прокручування дозволяв персонажу продовжувати рухатися в середовищі, а решта обстановки з’являлася на екрані, коли вони рухалися вертикально або горизонтально.

Назва «платформна» гра відноситься до серії платформ, на яких персонаж може бігати та стрибати на них у багатьох із цих ігор. У платформній грі аватару гравця може знадобитися розгадувати головоломки, перемагати ворогів, уникати перешкод, збирати монети, досягати пункту призначення та/або виконувати цілі іншого рівня. Платформери часто вважаються під жанром екшенів, але багато його елементів можуть з’являтися в інших жанрах, як-от пригоди, стратегії та головоломки.

Є кілька характеристик, які відрізняють ігри на платформі від інших типів ігор:

- гравець стрибає і лазить між різними платформами на ігровому полі;
- платформи часто мають нерівний рельєф і нерівну висоту розміщення;
- на шляху гравця стоять перешкоди, які потрібно подолати, щоб досягти мети.

Це лише мінімальні вимоги до ігрової платформи, і можна додавати інші функції:

- кілька рівнів зростаючої складності;
- нагороди доступні протягом всієї гри;
- життя кількох гравців;
- можливість руйнувати ігрові перешкоди.

1.2 Обґрунтування вибору мови програмування

Python – це фантастична мова програмування для розробки ігор. Розробка ігор на Python виявилася ідеальним рішенням для розробників для швидкого створення прототипів і виконання відеоігор із розвитком ігрової індустрії.

Проста і зрозуміла синтаксична структура Python відрізняє його від інших мов програмування для розробки ігор. Майже кожен розробник погодиться, що код Python легше зрозуміти, ніж код Java або C. Python є популярним вибором кодування серед розробників ігор через низьку криву навчання.

Python є об’єктно-орієнтованим, містить вбудовані високорівневі структури даних і підтримує динамічний тип і динамічне зв’язування. Python дійсно дозволяє створювати ігри, хоча він не такий популярний, як C++ із DirectX і OpenGL.

Python як мова розробки ігор став популярним в індустрії розваг. Це не лише тому, що Python популярний в інших сферах технологій чи тому, що він безкоштовний і з відкритим вихідним кодом (хоча обидві ці причини є вагомими).

Python використовується в розробці ігор, оскільки це надійна та універсальна мова програмування. Він автоматизує багато типових дій,

пов’язаних зі створенням ігор, і є численні ресурси, які допоможуть вам навчитися ефективно ним користуватися [2].

Причини вибору Python для розробки ігор:

1. Простий і зрозумілий синтаксис.

Чіткий синтаксис Python є однією з головних причин його такої популярності. Як результат, код простий для читання та розуміння, що робить його чудовим вибором для створення ігор. Код не тільки легко читати, але й легко писати, заощаджуючи час і зусилля під час розробки гри.

2. Лаконічний синтаксис допомагає плавно виконувати ідеї чи логіку, налагодження стає менш складним, а гнучкість легкого додавання функцій робить його гарним вибором для розробки ігор.
3. Гнучка об'єктна орієнтація, яка працює шляхом призначення атрибутів/функцій/властивостей класу або типу об’єкта. Ці ознаки потім успадковуються іншими категоріями, похідними від них. Отже, якщо програміст бажає розробити клас тваринного типу, він спочатку налаштує деякі основні функції, такі як eat(), sleep() тощо, а потім будь-які інші види, отримані від тварин, можуть мати ті самі атрибути.
4. Python досить універсальний з точки зору об’єктної орієнтації, що є однією з багатьох його сильних сторін. У результаті програмісти можуть просто створювати нові об’єкти та змінювати старі без необхідності писати багато коду.

Python є чудовим вибором для розробки ігор, оскільки розробникам часто доводиться генерувати й оновлювати об’єкти на льоту.

5. Ігрові бібліотеки та фреймворки.

Python для створення ігор підтримує 2D-візуальні елементи на додаток до 3D-графіки, що робить його однією з найпопулярніших мов для програмування ігор взагалі. Він має різноманітні бібліотеки та фреймворки, які спрощують розробку ігор. Pygame, Pyglet, Kivy, PySDL2, Pymunk,

PyOpenGL, PyODE та багато інших бібліотек широко використовуються в розробці ігор.

6. Динамічний набір.

Динамічна типізація Python є великою перевагою перед іншими мовами. У результаті розробникам не потрібно оголошувати змінні завчасно. Динамічний набір тексту допомагає створювати ігри, дозволяючи швидше створювати прототипи та тестувати. Програмістам Python ніколи не потрібно перетворювати змінні перед їх використанням, оскільки всі типи даних динамічно визначаються під час виконання.

7. Сумісність інтеграції зі штучним інтелектом (AI).

Python є ефективним інструментом розробки ШІ. Багато бібліотек, таких як TensorFlow, Keras, Theano та інші, спеціально створені для розробників, які хочуть включити можливості AI у свої ігри. Крім того, Python можна використовувати на різних платформах. У результаті гру можна створити на одній платформі та легко розгорнути на іншій. Python також є безкоштовним і відкритим кодом, що ще більше заощаджує кошти.

8. Надійна продуктивність.

Імпорт модулів, який дозволяє розробникам отримувати дані з інших джерел і повторно використовувати їх у своїх проектах, є загальною та впливовою функцією Python. Функція імпорту модуля також економить місце, оскільки один проект не повинен містити всі дані, необхідні для роботи; лише необхідні модулі імпортуються з інших джерел і поєднуються з оригінальними файлами.

Python добре відомий своєю надійністю. У результаті ігровий додаток зможе обробляти велику кількість дій без уповільнення. Це зручно для створення ігор, заснованих на дії, оскільки дозволяє обробляти інформацію одночасно без зависань або затримок. Хорошим прикладом є популярна гра «Nintendo Mario Kart 8 Deluxe». У всьому світі було продано понад 45 мільйонів копій гри, демонструючи потужність мови програмування [3].

1.3 Фреймворки для розробки ігор на Python

Хоча Python робить навчання коду доступнішим для кожного, вибір для написання відеоігор може бути обмеженим, особливо якщо необхідно писати аркадні ігри з чудовою графікою та привабливими звуковими ефектами. Протягом багатьох років розробники ігор Python обмежувалися фреймворком pygame. Тепер є інший вибір.

Ігрові рущії для Python найчастіше мають форму бібліотек Python, які можна встановити різними способами. Більшість доступні на PyPI і можуть бути встановлені за допомогою «pip». Однак деякі з них доступні лише на GitHub, GitLab або в інших місцях обміну кодом, і для них можуть знадобитися інші кроки встановлення.

Pyglet – це потужна, але проста у використанні бібліотека Python для розробки ігор та інших візуально багатих програм у Windows, Mac OS X і Linux. Він підтримує вікна, обробку подій інтерфейсу користувача, джойстики, графіку OpenGL, завантаження зображень і відео, а також відтворення звуків і музики.

Рисунок 1 – Логотип Pyglet

Pyglet надається за ліцензією BSD з відкритим вихідним кодом, що дозволяє використовувати його як для комерційних, так і для інших проектів з відкритим кодом з дуже невеликими обмеженнями. Особливості:

- Pyglet не вимагає жодних зовнішніх джерел чи налаштувань;
- він містить вбудований посібник для зображень та аудіо;

– він обробляє гнучкі локальні вікна;

– Pyglet був написаний на природній мові Python.

Pygame – це безкоштовна платформа Python для розробки мультимедійних програм, наприклад відеоігор, із відкритим кодом. Він заснований на відомій бібліотеці SDL. У цьому модулі використовуються C, Python, Native і OpenGL.

Рисунок 2 – Логотип Pygame

Pygame дозволяє користувачам використовувати програмування на Python для створення повнофункціональних ігор і мультимедійних пакетів. Він неймовірно портативний і сумісний майже з усіма платформами та операційними системами.

Особливості:

– процесори легкодоступні;

– Pygame найкраще використовує код C і Assembles для центральних функцій;

– він простий у використанні та портативний;

– потрібно лише мінімум рядків коду.

PyOpenGL – це популярна міжплатформна прив’язка Python до OpenGL і пов’язаних API. Зв’язування PyOpenGL створено за допомогою стандартної бібліотеки ctypes.

Для роботи з «непитоновскими» бібліотеками (наприклад, OpenGL) необхідні модулі, що забезпечують можливість виклику функцій бібліотеки безпосередньо з програми на мові Python. Бібліотека PyOpenGL – модуль, який

дозволяє в програмах на мові Python легко працювати з функціями OpenGL, GLU і GLUT, а також поруч із розширенням OpenGL [4].

PyOpenGL працює з різноманітними зовнішніми бібліотеками графічного інтерфейсу Python, включаючи Pygame, PyQt, Raw XLib та багато інших.

Arcade – це сучасна структура Python для створення ігор із привабливою графікою та звуком. Об’єктно-орієнтована за дизайном, Arcade надає авторам ігор сучасний набір інструментів для створення чудових ігор на Python.

Розроблений професором Полом Крейвеном із коледжу Сімпсона в Айові, США, Arcade побудовано на основі віконної та мультимедійної бібліотеки pyglet.

Рисунок 3 – Логотип Arcade

Він надає набір покращень, модернізацій які вигідно відрізняються як від Pygame, так і від Pygame Zero:

- підтримує сучасну графіку OpenGL;
- підтримує підказки типу Python 3;
- має підтримку кадрових анімованих спрайтів;
- містить узгоджені назви команд, функцій і параметрів;
- заохочує відокремлення логіки гри від коду відображення;
- вимагає менше шаблонного коду;

- забезпечує добре підтримувану та актуальну документацію, включаючи кілька навчальних посібників і повні приклади ігор на Python;
- має вбудовані фізичні механізми для ігор зверху вниз і платформних ігор.

Усе в Arcade відбувається у вікні, яке створюється з розміром, визначеним користувачем. Система координат передбачає, що початок координат (0, 0) розташований у нижньому лівому куті екрана, а координати у збільшуються в міру просування вгору. Це відрізняється від багатьох інших ігрових движків, які розміщують (0, 0) у верхньому лівому куті та збільшують у-координати, рухаючись вниз.

За своєю суттю Arcade є об'єктно-орієнтованою бібліотекою. Хоча програму Arcade можна писати процедурно, її справжня сила виявляється, коли ви створюєте повністю об'єктно-орієнтований код.

Arcade, як і Pygame Zero, надає вбудований ігровий цикл і чітко визначену модель подій, тому ви отримуєте дуже чистий і читабельний код гри. Також, як і Pygame Zero, Arcade надає потужний клас спрайтів, який допомагає рендерингу, позиціонуванню та виявленню зіткнень. Крім того, спрайти Arcade можна анімувати за допомогою кількох зображень [5].

Ren'Py – це сучасний нащадок чисто текстової пригоди, а саме візуальна новела, яка підкреслює сюжетний аспект гри, обмежуючи взаємодію гравців, додаючи візуальні елементи та звук для покращення враження. Візуальні романи – це графічні романи ігрового світу, сучасні, інноваційні та надзвичайно привабливі для створення та споживання.

Ren'Py – це інструмент, заснований на Pygame і розроблений спеціально для створення візуальних романів. Ren'Py отримав свою назву від японського слова, що означає романтичне кохання, і надає інструменти та основу для створення переконливих візуальних романів.

Ren’Py не є суто бібліотекою Python, яку можна встановити та використовувати. Ігри Ren’Py створюються за допомогою Ren’Py Launcher, який постачається з повним пакетом SDK Ren’Py. Ця програма запуску також містить редактор ігор, хоча ви можете редагувати свою гру в обраному редакторі. Re’Py також має власну мову сценаріїв для створення ігор. Однак Ren’Py базується на Pygame, і його можна розширити за допомогою Python, що гарантує його появу тут.

1.3 Огляд аналогів

Для того, щоб поставити задачі та вимоги до проекту розробки, слід оглянути існуючі аналоги. Це допоможе зорієнтуватися щодо функціональних можливостей гравця та художнього оформлення ігор-платформерів.

Перший аналогом було розглянуто гру «Super Meat Boy» (рис. 4).

Рисунок 4 – Скріншот сцени гри «Super Meat Boy»

Платформери використовують багато цікавих трюків, щоб зберегти досвід свіжим. Вони часто приносять веселі історії, дурних персонажів або

сильніших ворогів. Але сенс платформера полягає в платформі, і Super Meat Boy максимально використовує це буквальне визначення.

Гравці будуть долати перешкоди від безпечного місця до безпечного. Вікна в цих безпечних місцях стають меншими, коли розгортається кожен неймовірний розділ. Згодом лише найзапекліші шанувальники ігрових платформ із елітною координацією рук і очей зможуть оголосити про перемогу.

«The Sims 4» – одна з найкращих ігор, розроблених на Python (рис. 5). Цей симулятор життя представляв повноцінний досвід віртуального життя, який дозволяв гравцям створювати власного персонажа або «симулятора» у своєму власному середовищі.

Рисунок 5 – Скрін епізоду гри «The Sims 4»

Це класична гра-симулятор, розроблена компанією Maxis і видана Electronic Arts. Довговічність цієї франшизи є чимось на що дивитися,

оскільки вона змінювалася з роками. У 2022 році він змінився на безкоштовну модель. Через вісім років після випуску він все ще розвивається.

Є пакети розширення – це найдорожчий рівень завантажуваного вмісту, і, як не дивно, вони містять найбільшу кількість функцій і елементів. Для переважної більшості пакетів розширення ви також отримуєте новий повністю деталізований світ, який доповнює тему.

Наступна гра «Shovel Knight: Treasure Trove» – це повне видання «Shovel Knight», класична пригодницька сага з чудовим геймплеєм, персонажами, що запам'ятовуються, і 8-бітною ретро-естетикою (рис. 6).

Рисунок 6 – Скрін епізоду гри «Shovel Knight: Treasure Trove»

Гравець має змогу бігати, стрибати і боротися в ролі Лопатного лицаря, повелителя лопатного клинка, якому необхідно знайти його втрачену кохану. Перемога горезвісних лицарів з «Ордену Без Пощади» і виклик його ватажку, Чарівниці – все це головні задачі гри.

«Shovel Knight: Treasure Trove» включає чотири додаткові ігри! Грайте за Чумного лицаря, Примарного лицаря та Короля-лицаря у їхніх власних

пригодах або боріться з друзями локально у поєдинках із 4 гравців у режимі Showdown. Разом вони являють собою велику та масштабну сагу.

Ця гра є одним з найкращіх прикладів «графічний контент – ігровий геймплей», цікава сюжетна лінія, продумані характери героїв, генерація рівнів.

Наступна гра «Braid» – це пазл-платформер, намальований у живописному стилі, де гравець маніпулює плином часу дивними та незвичайними засобами (рис. 7). З будинку в місті герой вирішує в низку світів і вирішує головоломки, щоб врятувати викрадену принцесу. У кожному світі гравець має різну силу впливати на те, як поводиться час, і саме незвичайність часу створює загадки.

Рисунок 7 – Скрін епізоду гри «Braid»

Поведінка в часі включає: здатність перемотувати назад, об’єкти, стійкі до перемотування, час, прив’язаний до простору, паралельні реальності, затягування часу та, можливо, багато іншого. «Braid» цінує ваш час і увагу; в цій грі немає наповнювача. Кожна головоломка показує щось нове та цікаве про ігровий світ [6].

«Braid» – це двовимірна гра на платформі, у якій ви ніколи не зможете померти та ніколи не програєте. Незважаючи на це, «Braid» є складним, але завдання полягає у розв’язуванні головоломок, а не в тому, щоб гравець повторював складні стрибки.

Подорож серією світів, пошук частини головоломки та її вирішення, маніпулюючи часом: перемотуючи назад, створюючи паралельні всесвіти, створюючи кишені розширеного часу. Геймплей виглядає свіжим і новим; головоломки покликані надихнути на нові способи мислення [7].

1.4 Постановка задачі

За результатами першого розділу дипломної роботи прийнято рішення поставити наступні задачі до об’єкту розробки:

1. Розробити 2Д-гру у стілі платформер.
2. В ролі мови програмування обрати Python та бібліотеку Arcade, яка ідеально підходить до таких проектів.
3. Розробити концепт-документ, який буде описувати ідеї та механіки гемплею.
4. Провести тестування та надати опис подальшого розвитку проекту.

Аналітичний огляд аналогів допоможе обрати графічний контент (ассети та спрайти) для моделювання ігрового світу.

2 ПРОЕКТНА ЧАСТИНА

Перш ніж вкладати час, гроші та інші ресурси, дуже важливо чітко визначити своє бачення проекту. До єтапу розробки, слід постійно ставити собі певний набір запитань, щоб оцінити план і напрямок дій щодо розробки гри. Попередня оцінка майбутніх починань розробника служить тестом на виживання проекту [8].

На першому етапі описують концепт-документ, який включає основну ідею, опис цільової аудиторії, які особливості будуть у даного проекту та інше.

2.1 Концепт-документ

1. Вступ

Головний герой – прибулець Кенні, який блукає просторами всесвіту у пошуках пригод. Одного разу він потрапляє на невідому планету. Кенні вирішує вийти на поверхню щоб зібрати більше інформації для свого довідника локацій. Його чекають декілька рівнів, кожен з яких буде представлений своїми кліматичними умовами, а також невідомими істотами, та нові випробування власних сил. По дорозі прибулець може збирати монети, які використовуються для відновлення рівня життєвих ресурсів. Кенні не має зброї, тому він повинен уникати будь-яких небезпечних перешкод на шляху.

2. Жанр та аудиторія

Гра відноситься до жанру “platformer” з 2D графікою зі скролінгом ігрового шляху та звуковим супроводом. Гра орієнтована на широку аудиторію, не містить вікового обмеження, мінімальний вік гравця – 6 років. Додаткову привабливість гра має для власників не найсучаснішої конфігурації PC і людей, які полюбляють мультиплікаційну графіку.

Гра не використовує торгові марки або другу власність, що підлягає ліцензуванню. Всі сети графічних спрайтів та звукові ефекти завантажено на умовах “use free”.

3. Ключові особливості гри.

Користувач керує головним героєм Кенні через клавіатуру. Для проходження рівнів знадобиться уважність та швидкість реагування на перешкоди.

Невисокі вимоги до ПК при чудовій яскравій графіці. Прибулець знаходиться на незнайомій планеті, тому вороги можуть бути де завгодно і будь-що. Оскільки планета чужа, сила тяжіння може бути іншою, що може вплинути на здатність Кенні стрибати та рухатися. Гра підтримує підхід “Violent-free”, тобто “вбивство” ворогів виконується в кращих традиціях дитячих ігор – заморожування об’єкту чи зникнення.

4. Опис гри

Основне завдання гравця – проходження ігрового сюжету кожного рівня. Кенні повинен слідкувати за станом своїх життєвих сил на невідомій планеті, для цього йому потрібно збирати золоті монети. Для переходу на наступний левел йому необхідно знайти золотий ключ, який відмикає двері.

Побудова дизайну рівнів заснована на виконанні міні-місій і постійно ставить завдання ухвалення рішень про спосіб проходження тієї чи іншої ділянки. Гравець отримує нові навички щодо прибування у невідомій місцевості, а також вивчає мешканців кожного левела, їх поведінку та можливі наслідки перетину з ними.

Перешкодами на шляху гравця є природні умови (лід чи лава), та мешканці рівнів (мухи, равлики, крижані кульки та інші). Крім цього, для прибульця Кенні умови гравітації також невідомі, тому він вчиться переміщуватись та стрибати.

Сюжет гри розрахований на 3-5 хвилин проходження одного рівня.

2.2 Ігрова механіка

Ігрова механіка – набір правил та способів, що реалізує певним чином деяку частину інтерактивної взаємодії гравця та гри. Вся безліч ігрових механік гри формують конкретну реалізацію її ігрового процесу. Геймплей описує гру взагалі, а ігрові механіки дозволяють безпосередньо реалізувати те, як гра функціонує.

Ігрова механіка – розділяє такі поняття, як правила гри (як моделюється ігровий світ) і дії, доступні гравцеві (способи взаємодії гравця з цим світом). Тобто, це безліч дій і способів, доступних гравцю [8].

Маючи на увазі приблизний дизайн можна визначити, як гравець буде контролювати ігровий процес. Для переміщення прибульця по ігровому полю потрібен спосіб керування кількома різними рухами: «Left» і «Right» для переміщення по платформі, «Up» та «Down», щоб піднятися по сходах між платформами або задати направлення руху при стрибку.

Перш ніж почати писати будь-який код, завжди добре підготувати дизайн:

1. Проект являє собою гру уникнення ворога з горизонтальним прокручуванням.
2. Гравець починає гру з лівого боку екрана.
3. Вороги розміщені на платформі та у повітрі.
4. Вороги рухаються ліворуч по прямій лінії, поки не зникнуть з екрана.
5. Гравець може рухатися вліво, вправо, вгору або вниз, щоб уникнути ворогів.
6. Гравець не може відійти від екрана.
7. Гра закінчується, коли гравця вдаряє ворог або користувач закриває вікно.

Наступний крок після опису ідеї гри та механізмів керування слід попрацювати над графікою та звуковим супроводом. Зображення, спрайти, звуки та навіть текст, які використовуються для відображення партитури, спільно відомі як ресурси. Вони визначають гру в очах потенційних гравців.

Але створення їх може бути складним завданням, що займає стільки ж часу, якщо не більше, ніж написання коду гри.

Зазвичай, при розробці ігрового додатку крім розробника-программіста є артдизайнер. Тому було прийнято рішення звернутися до ассетів спрайтів, які є на багатоьох інтернет-ресурсах у вільному для завантаження доступі.

Так, спрайти головного герою – прибульця Кенні було взято з одного ассету, а матеріали для моделювання ігрових сцен ще з декількох ассетів [9]. Все це допоможе сформувати графічний контент, на основі якого будуть реалізовані механіки гри мовою Python.

2.3 Моделювання варіантів використання

Будь-яка гра – це не просто гарна графіка і завдання, яке потрібно виконати, тут є повна сюжетна лінія, і різні рішення і дії призведуть до різних результатів. Діаграма варіантів використання – це спосіб узагальнити деталі системи та користувачів у цій системі. Зазвичай це відображається як графічне зображення взаємодії між різними елементами в системі.

Діаграми варіантів використання вказуватимуть події в системі та те, як ці події протікають, однак діаграма варіантів використання не описує, як ці події реалізуються. Варіант використання – це методологія, яка використовується в системному аналізі для визначення, уточнення та організації системних вимог.

У цьому контексті термін «система» відноситься до чогось, що розробляється або використовується, наприклад, веб-сайт продажу продуктів і послуг, що надаються поштою. Діаграми варіантів використання використовуються в UML (Unified Modeling Language), стандартній нотації для моделювання об’єктів і систем реального світу. Наявність діаграми варіантів використання має низку переваг перед аналогічними діаграмами, такими як блок-схеми [10].

Діаграма Use-Case UML представлено на рисунку 8:

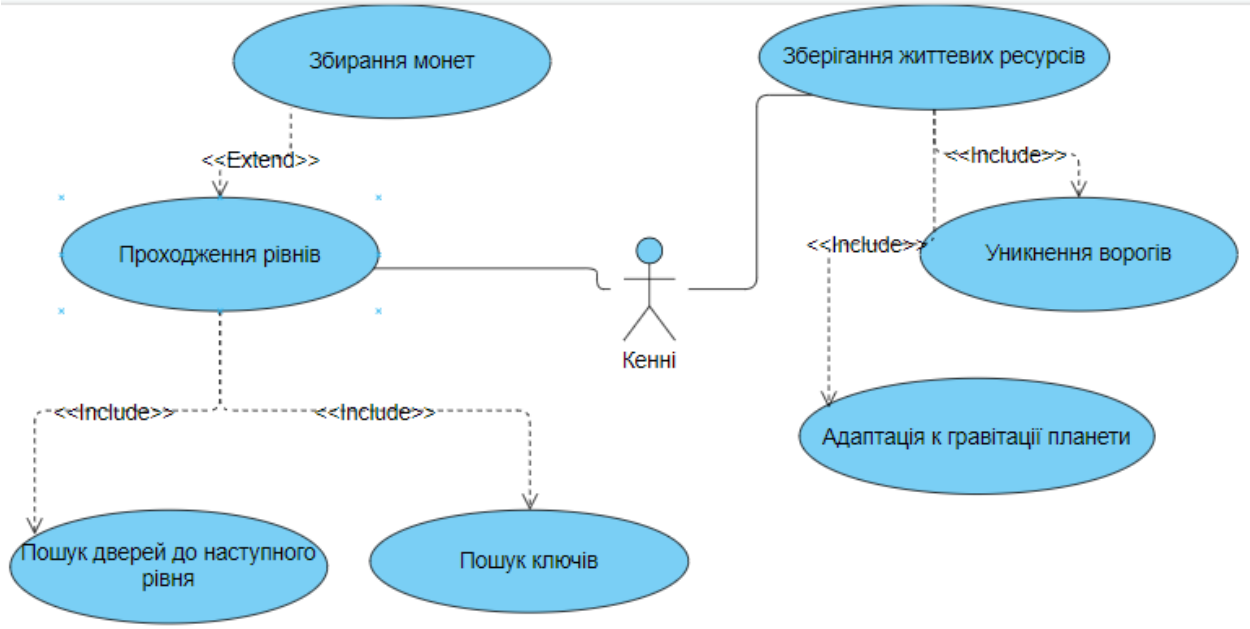


Рисунок 8 – Діаграма Use-Case

3 ОПИС РЕАЛІЗАЦІЇ РОБОТИ

3.1 Структура гри

Аркадна бібліотека – це сучасна структура Python для створення ігор із привабливою графікою та звуком. Об’єктно-орієнтований і створений для Python 3.6 і вище, arcade надає програмісту сучасний набір інструментів для створення чудових ігор на Python.

Проект з розробки комп’ютерної гри складаються з графічних і звукових ресурсів, а також коду. Зберігання ігрових ресурсів і коду належним чином організовано дозволяє швидко вносити цілеспрямовані зміни в дизайн або поведінку гри, мінімізуючи вплив на інші аспекти гри. У проекті використовується така структура:

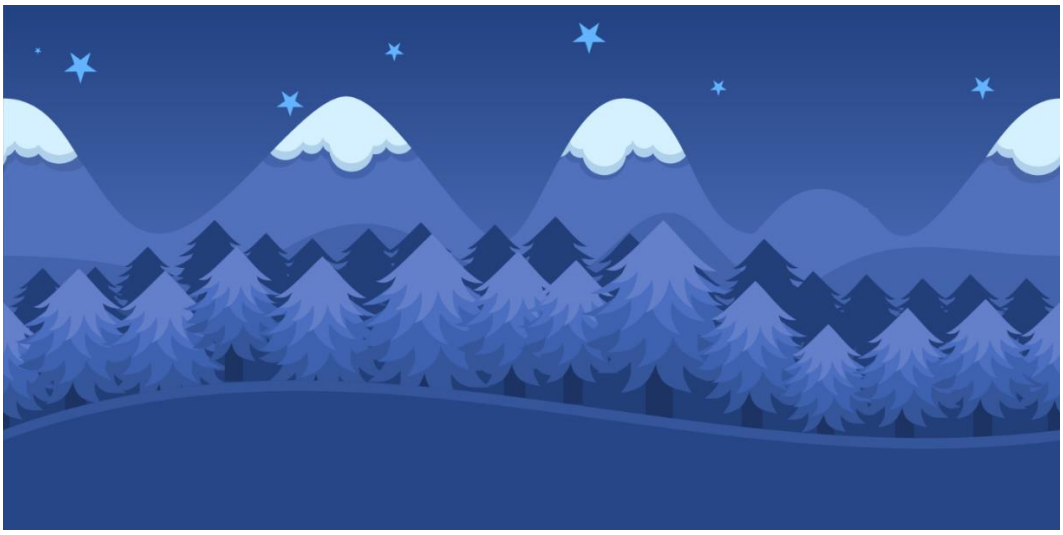
```
arcade_platformer/
|
├── arcade_platformer/
|
├── assets/
    |
    ├── images/
    |    |
    |    ├── enemies/
    |    |
    |    ├── ground/
    |    |
    |    ├── HUD/
    |    |
    |    ├── items/
    |    |
    |    ├── player/
    |    |
    |    └── tiles/
    |
    └── sounds/
```

«Arcade_platformer» містить увесь код Python для гри, а «assets» складається з усіх ігрових зображень, шрифтів, звуків і карт плиток. Для завантаження даних у «assets» достатньо просто перетягнути потрібні папки з локального диску до середи розробки (у роботі використовується VSCode).

Відразу з’являється перелік ресурсів та можна перевірити чи правильну директорію було завантажено ці дані.

3.2 Опис реалізації інтерфейсу

В проекті реалізовано два рівні, тобто дві різні локації: темна сторона планети (рис. 1а) – де завжди холод та ніч, та сонячна сторона (рис. 8б) – тут завжди світло та спекотно). Для кожної з них використовувався свій бекгроунд.



а)

б)

Рисунок 8 – Background двух рівнів

Для локацій характерна своя природа та мешканці. Для моделювання кожного рівня використовувались декілька сетів страйтів. По-перше, це спрайти для відображення головного героя – прибульця Кенні (рис. 9). В проекті реалізовано анімацію, яка відображає рухи об’єктів. Так, повний колаж спрайтів для Кенні має наступний вигляд:

Рисунок 9 – Спрайти героя для відображення руху

На стартовому левелі Кенні опиняється на тіньовій стороні планети, де панує холод та нічна темрява. Мешканці не люблять чужинців, тож кожен з них несе свій рівень загрози для Кенні.

До «enemies» (вороги) відносяться наступні істоти (рис. 10):

- летучі миші (рис. 10а) – при перетині з ними у індикаторі життя Кенні віднімається «пів серця»;
- хижа рибка (рис. 10б) – якщо герой попадає на неї, в індикаторі життя зникає одне серце;

1 а)

1 б)

Рисунок 10 – Спрайти мешканців першого левела

- крижана капля (рис. 11а) – ця істота заморожує Кенні на 10 секунд;
- спінер (рис. 11б) – завдяки гострим шипам відбирає «одне життя» Кенні.

а)

б)

Рисунок 11 – Спрайти мешканців першого левела

Для контролю життєвих сил Кенні гравець використовує індикатори серця (рис. 12):

Рисунок 12 – Спрайти індикатору життя

Для поповнення життєвих сил прибульцю необхідно збирати джерела енергії – золоті монети (рис. 13а), десять монет дарують «пів серця» життя. Крижаний кристал (рис. 13б) дістати не просто, але він може поповнити сили на ціле серце. За умовами левелів, кристал тільки один раз з’являється у ігровому просторі.

1 а) 1 б)

Рисунок 13 – Спрайти джерел енергії

Ключі (рис. 14): синій дає силу заморозити летучіх мишей на 10 секунд, золотий ключ відкриває двері до наступного рівня.

Рисунок 14 – Спрайти ключів

Далі розглянемо елементи ігрового простору. Це крижані частинки скель та твердих поверхонь для першого рівня (рис. 15):

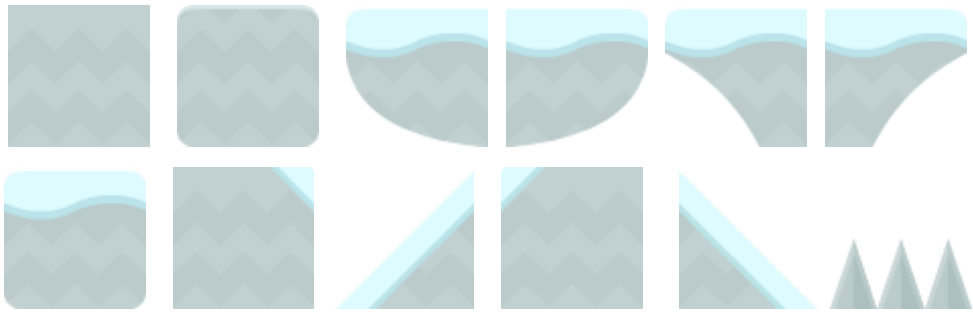
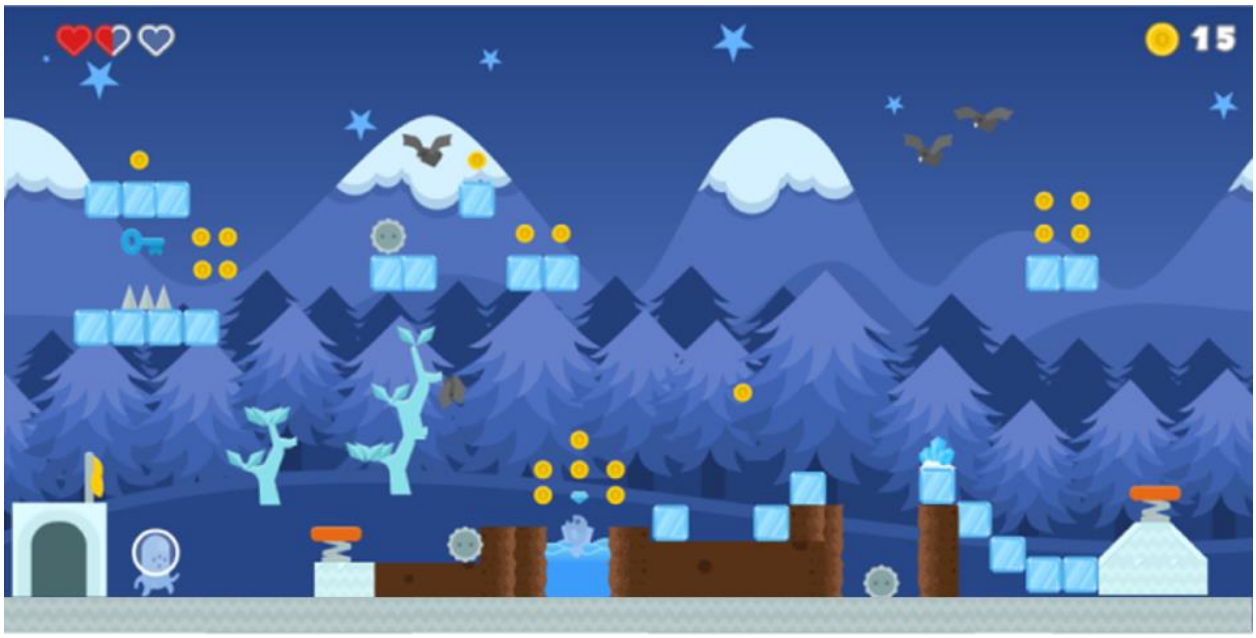


Рисунок 15 – Спрайти будувальних елементів

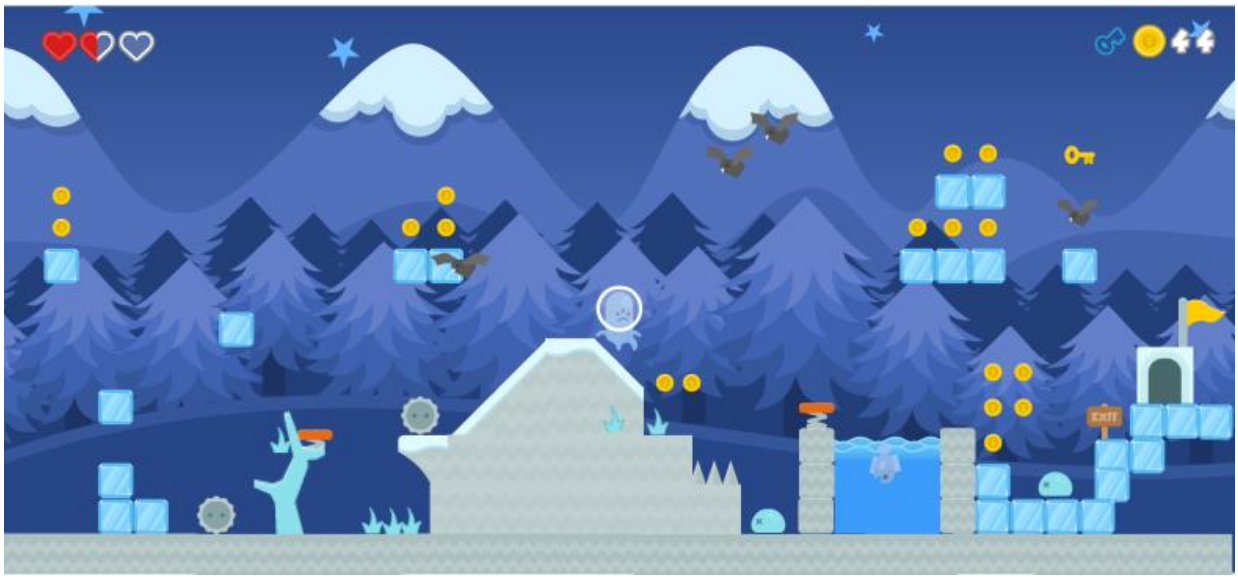
Відповідно до історії гри, прибулець прилаштовується до умов гарвітації на незнайомій планеті. Висота стрибків в нбого буде обмежена, тож для досягання висот він може використовуват елемент-пружину (рис. 16):

Рисунок 16 – Спрайти для анімації роботи пружини

Приклад ігрового простору стартового рівня представлено на рисунку 17.



1 а)



1 б)

Рисунок 17 – Скриншоти ігрового простору першого рівня

На наступному рівні Кенні зустрічає нових мешканців, замість летучих мишей тут літають мухи (рис. 18а), а крижаних краплинок змінили равлики (рис. 18б).

Реакції та наслідки з такими істотами залишаються як і у першому рівні.

1 а)

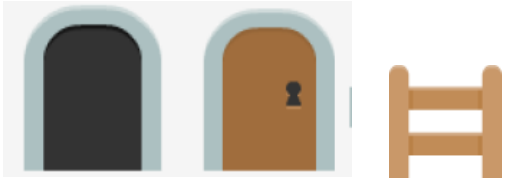
1 б)

Рисунок 18 – Спрайти для анімації руху тварин другого левела

На «сонячному» рівні забрати один індикатор життя може пекуча лава (рис. 19). Для анимації руху лави «вгору-вниз» використовуються наступні спрайти (коли лава підіймається – буде зле обличчя, коли вниз – сумне що не вдалося схопити Кенні):

Рисунок 19 – Сет спрайтів для лави

На двох левелах є двері (рис. 20а): відкриті на початку (з них з’являється прибулець), та зачинені у кінці рівня. Більша частина локацій другого рівня розміщенно у повітрі, тож прибулець буде використовувати дробинки (рис. 20б) та пружину щоб дістатися цілі.



1 а) 1 б)

Рисунок 20 – Спрайти дверей та дробини

Елементи твердих поверхонь для «сонячного» рівня представлено на рисунку 21. З них будуть складатися повітряні платформи. Приклад сцени другого рівня представлено на рисунку 22.

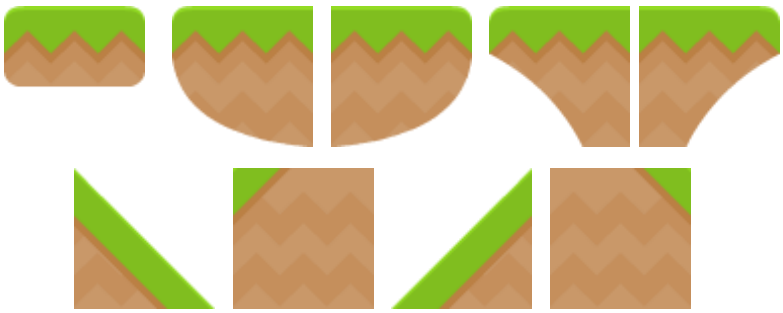


Рисунок 21 – Спрайти твердих поверхонь

Рисунок 22 – Скріншот сцени другого рівня гри

3.3 Реалізація дінаміки ігрового процесу

3.3.1 Основні налаштування гри

Першим кроком створення гри є підключення бібліотеки «Arcade», яка дозволить побудувати модель ігрового простору. Далі нам знадобиться

обов’язково рендом. Через константи задаємо висоту та ширину ігорового простору (вікна):

```
import arcade
import pathlib
from random import randint

WIDTH = 1200
HEIGHT = 800
SCREEN_TITLE = "MyDipl project game"
```

Керувати всіма спрайтами може бути складно. Відстеження їх усіх – робота для списку спрайтів, вони забезпечують три важливі поведінки:

- оновлення всіх спрайтів в списку за допомогою одного виклику SpriteList.update();
- намалювання всіх спрайтів зі списку за допомогою одного виклику SpriteList.draw().
- перевірка, чи один спрайт зіткнувся з будь-яким спрайтом у списку.

Слід прописати звідки будуть підтягуватись спрайти, тобто шлях до наших ассетів у проекті (ASSETS_PATH зберігає шлях до папки ресурсів, використовуючи шлях до поточного файлу як основу):

```
ASSETS_PATH = pathlib.Path(__file__).resolve().parent.parent
```

Для запуску всієї гри буде використано клас Platformer. Методи цього класу викликаються для оновлення стану гри, обробки введених користувачем даних і малювання елементів на екрані.

```
class Platformer(arcade.Window):
def __init__(self) -> None:
        super().__init__(WIDTH, HEIGHT, SCREEN_TITLE)
```

Нам знадобляться декілька списків, що будуть містити різні набори спрайтів: монетки, фон гри, істоти, кількість отриманих бонусів тощо.

```
        self.coins = None
        self.background = None
        self.goals = None
        self.enemies = None
        self.player = None
```

Остання строка оголошує об’єкт програвача, який буде правильно визначено у .setup(). Гра має кілька рівнів та гравець має кілька життів. Замість того, щоб перезапускати всю гру, викликаючи .__init__(), будемо викликати .setup(), щоб повторно ініціалізувати гру до відомої початкової точки або встановити новий рівень. Визначемо спрайт гравця:

```
# Налаштування плеєру для звукого супроводу
    self.player = arcade.Sprite("images/kenny.png", SCALING)
    self.player.center_y = self.height/2

    self.player.left = 10
    self.all_sprites.append(self.player)
```

По-перше, необхідно створити новий об’єкт arcade.Sprite, вказуючи зображення для відображення та коефіцієнт масштабування. Доцільно організувати зображення в одній підпапці, особливо у великих проектах.

Наступний крок: встановлюємо позицію у спрайту на половину висоти вікна та позицію x спрайту, розміщуючи лівий край на кілька пікселів від лівого краю вікна. Далі слід використовувати .append() для додавання спрайту до списку .all_sprites, який ви використовуватимете для малювання.

3.3.2 Позиціонування та переміщення спрайтів

Тут сі спрайти в аркаді мають певний розмір і положення у вікні, а саме, розмір, визначений Sprite.width і Sprite.height, визначається графікою, яка використовується під час створення спрайту.

Позиція початково встановлена таким чином, щоб центр спрайту, визначений Sprite.center_x і Sprite.center_y, становив (0,0) у вікні. Коли відомі

координати .center_x і .center_y, arcade може використовувати розмір для обчислення країв Sprite.left, Sprite.right, Sprite.top і Sprite.bottom.

Це також працює в зворотному напрямку. Якщо встановите задане значення Sprite.left, Arcade також перерахує решту атрибутів позиції. Такий підхід зручно використовувати щоб знайти спрайт або перемістити його у вікні. Це надзвичайно корисна та потужна характеристика аркадних спрайтів, вона допомогає зменшити код проекту.

Функція планування arcade.schedule() необхідна буде для управління періодом з’явлення ворогів (їх спрайтів) у ігровому просторі. Тут потрібно два аргументи:

- ім'я функції для виклику;
- інтервал часу очікування між кожним викликом, у секундах.

За планом, ворожі спрайти повинні з’являлися протягом усієї гри. У такому разі слід налаштовати заплановану функцію для створення нових ворогів. Цей код переходить у .setup(). Ось як цей код виглядає:

```
#Породжує нового ворога кожні 0,25 секунди
arcade.schedule(self.add_enemy, 0.25)
```

У грі вороги обох левелів мають три ключові властивості:

- вони з’являються у випадкових місцях у правій частині вікна;
- вони рухаються вліво по прямій;
- вони зникають, коли йдуть з екрана.

```
def create_enemy(self, delta_time: float):
    #Додає нового ворога на екран

    #Спочатку створюємо нового спрайта ворога
    enemy = arcade.Sprite("images/bat.png", SCALING)

    #Встановлення його положення на випадкову висоту та поза
екраном праворуч
    enemy.left = random.randint(self.width, self.width + 80)
    enemy.top = random.randint(10, self.height - 10)
```

Метод .add_enemy() приймає єдиний параметр, delta_time, який показує, скільки часу минуло з моменту останнього виклику. Тут йде аналогічна послідовність як у випадку зі спрайтом гравця: спочатку необхідно створити нову аркаду. Спрайт із зображенням і коефіцієнтом масштабування. Встановлюється позиція за допомогою .left і .top у випадкову позицію десь праворуч від екрана.

Рухомі спрайти – це вороги на ігровому просторі. Щоб перемістити спрайт, слід змінити його положення під час фази оновлення ігрового циклу. Хоча можна зробити це самостійно, Arcade має деякі вбудовані функції.

Кожен arcade.Sprite має не тільки набір атрибутів позиції, але також має набір атрибутів руху. Кожного разу, коли спрайт оновлюється, аркада використовуватиме атрибути руху для оновлення позиції, надаючи відносного руху спрайту.

Атрибут Sprite.velocity – це кортеж, що містить зміни в положеннях x і y. Є можливість отримати прямий доступ до Sprite.change_x і Sprite.change_y. Кожного разу, коли спрайт оновлюється, його позиція змінюється на основі .velocity. Для цього необхідно в .add_enemy() встановити швидкість:

```
enemy.velocity = (random.randint(-20, -5), 0)

# Додати до enemies_list
self.enemies_list.append(enemy)
self.all_sprites.append(enemy)
```

Оскільки вороги завжди рухаються ліворуч, коли вони виходять з екрана, вони не повертаються. У такому разі слід слідкувати за переповненням списків спрайтів ворогів, а саме, видаляти з нього об’єкти, коли вороги достигають лівої межі ігрового простору.

Можна визначити, що ворог знаходиться поза екраном, якщо enemy.right менше нуля, тобто лівого краю вікна. У такому випадку буде викликатися enemy.remove_from_sprite_lists(), щоб видалити його з усіх списків, до яких він належить, і звільнити цей об’єкт із пам’яті:

```
if enemy.right < 0:
    enemy.remove_from_sprite_lists()

for enemy in enemies_list:
    enemy.update()
```

Arcade – це об'єктно-орієнтована бібліотека, тобто, можна створювати власні класи на основі аркадних класів і перевизначати методи, які необхідно змінити. У цьому випадку було створено новий клас на основі arcade.Sprite і лише замінюєте .update():

```
class FlyingSprite(arcade.Sprite):
    def update(self):
        """Оновлення положення спрайту коли він переміститься
           за межі екрана вліво, його видаляємо"""

        #Переміщення спрайту
        super().update()

        # Видалення спрайту
        if self.right < 0:
            self.remove_from_sprite_lists()
```

FlyingSprite – це всі об’єкти ворогів, які будуть літати у грі. Потім змінюємо .update(), спочатку викликаючи super().update() для належної обробки руху. Настцупний крок, виконання перевірки поза екраном.

```
def add_enemy(self, delta_time: float):
    # Створюємо нового спрайта ворога
enemy = FlyingSprite("images/missile.png", SCALING)
```

Також потрібен фізичний двигун та список для запам’ятовування кількості отриманих бонусів, бо ця величина буде в подальшому давати прибульцю поповнення життевих сил. За замовчуванням гра буде починатися з першого левела (всього поки що 2 левела у грі).

```
        self.physics_engine = None
        self.score = 0
        self.level = 1
```

Останньою базовою вимогою до гри буде використання звукових ефектів під час ключових дій під час проходження гри: це звук збирання золотих монеток, стрибок прибульця та фінальний трек під час завершення левела. Тобто будемо використовувати визначену раніше константу ASSETS_PATH для пошуку та завантаження звукових файлів.

```
self.coin_sound = arcade.load_sound(
    str(ASSETS_PATH / "sounds" / "coin.wav")
)
self.jump_sound = arcade.load_sound(
    str(ASSETS_PATH / "sounds" / "jump.wav")
)
self.victory_sound = arcade.load_sound(
    str(ASSETS_PATH / "sounds" / "victory.wav")
)
```

Уся дія практично в кожній грі відбувається в центральному ігровому циклі. Він починається після того, як гру налаштовано та ініціалізовано, і завершується, коли гра робиться. Кілька речей відбувається послідовно всередині цього циклу. Як мінімум ігровий цикл виконує такі чотири дії:

1) програма визначає, чи закінчилася гра. якщо так, то цикл закінчується;
2) введені користувачем дані обробляються;
3) стани ігрових об’єктів оновлюються на основі таких факторів, як введення користувача або час;
4) гра відображає візуальні ефекти та відтворює звукові ефекти на основі нового стану.

Все в Arcade відбувається у вікні, яке створюється за допомогою open_window(). Arcade використовує декартову систему координат: вікно знаходиться в квадранті I, з початковою точкою (0, 0), розташованою в нижньому лівому куті екрана. Координата x збільшується, коли об’єкт рухається праворуч, а координата y збільшується, коли він рухаєтеся вгору.

Під час вбудованого ігрового циклу Arcade викликає набір методів Window для реалізації всіх функцій. Усі назви цих методів починаються з «on_

і» їх можна розглядати як обробники завдань або подій. Коли циклу аркадної гри потрібно оновити стан усіх ігрових об’єктів Python, він буде викликати .on_update(). Коли йому потрібно перевірити рух миші, він викликатиме .on_mouse_motion().

3.3.3 Реалізація керування кнопками

Наступний крок – це реалізація динаміки руху Кенні, бо герой гри повинен пересуватись між об’єктами ігрового простору. У більшості платформерів користувач переміщує гравця за допомогою джойстика або клавіатури. Вони можуть змусити гравця стрибнути або звести його з платформи.

Коли гравець опиниться в повітрі, йому не потрібно нічого робити, щоб змусити його впасти на нижчу платформу. Контроль того, куди гравець може ходити та як він падає після стрибка чи сходження з платформи, обробляється фізичним механізмом.

У грі фізичний механізм забезпечує наближення фізичних сил, які діють на гравців та інші об’єкти гри. Ці сили можуть впливати на рух ігрових об’єктів, включаючи стрибки, лазіння, падіння та блокування руху.

У Python arcade включено три фізичні механізми:

1) arcade.PhysicsEngineSimple – це простий механізм, який обробляє рух і взаємодію спрайтів для одного гравця та списку спрайтів стін. Це корисно для ігор зверху вниз, де гравітація не є фактором.
2) arcade.PhysicsEnginePlatformer – це більш складний движок, створений для використання в платформних іграх. Окрім основного руху, він забезпечує силу тяжіння, яка тягне об’єкти до нижньої частини екрана. Це також надає гравцеві можливість стрибати та лазити по драбинах.
3) arcade.PymunkPhysicsEngine створено на основі Pymunk, двовимірної бібліотеки фізики, яка використовує бібліотеку Chipmunk. Pymunk

робить надзвичайно реалістичні фізичні обчислення доступними для аркадних програм.

Для даного проекту буде використано саме arcade.PhysicsEnginePlatformer. Клас «Arcade.Window» має дві функції для обробки введення з клавіатуриа саме: виклик у коді .on_key_press() щоразу, коли натискається клавіша, і .on_key_release() щоразу, коли клавішу відпускають. Модуль Arcade.key містить усі константи клавіатури, які користувач може використовувати.

В проекті використовано чотири клавіші зі стрілками вгору, вниз, ліворуч і праворуч. У методі .on_key_release() йде перевірка лише ключів, які впливатимуть на рух спрайту гравця.

```
def on_key_press(self, symbol, modifiers):
    """ Обробляти введення користувача з клавіатури
         symbol {int} - яка клавіша була натиснута
         modifiers {int} - які модифікатори були натиснуті"""

    if symbol == arcade.key.Q:
        # Quit immediately
        arcade.close_window()

    if symbol == arcade.key.P:
        self.paused = not self.paused

    if symbol == arcade.key.UP:
        self.player.change_y = 5

    if symbol == arcade.key.DOWN:
        self.player.change_y = -5

    if symbol == arcade.key.LEFT:
        self.player.change_x = -5

    if symbol == arcade.key.RIGHT:
        self.player.change_x = 5

def on_key_release(self, symbol: int, modifiers: int):
    """ Скасувати вектори руху, коли клавіші руху відпущені
        Аргументи:
         symbol {int} - Яка клавіша була натиснута
         modifiers {int} - Які модифікатори були натиснуті"""

    if (
```

```
        or symbol == arcade.key.UP
        or symbol == arcade.key.DOWN
    ):
        self.player.change_y = 0

    if (
        or symbol == arcade.key.LEFT
        or symbol == arcade.key.RIGHT
    ):
        self.player.change_x = 0
```

Оновлення ігрових об’єктів в ігровому циклі допопмогає рухатися спрайтам в ігровому просторі. Arcade контролює ігровий цикл Python, він також контролює, коли потрібні оновлення, викликаючи .on_update(). Перевизначення цього методу допомогає забезпечити правильну поведінку гри, включаючи рух у грі та іншу поведінку.

```
def on_update(self, delta_time: float):
        #Якщо гру призупинено, нічого не оновлювати
    if self.paused:
        return

    #Повністю оновити
    self.all_sprites.update()

    #Контроль переміщення спрайду в межах екрану
    if self.player.top > self.height:
        self.player.top = self.height
    if self.player.right > self.width:
        self.player.right = self.width
    if self.player.bottom < 0:
        self.player.bottom = 0
    if self.player.left < 0:
        self.player.left = 0
```

Тут кожен спрайт є членом списку self.all_sprites. Виклик self.all_sprites.update() призводить до виклику .update() для кожного спрайту в списку. Кожен спрайт у списку має .velocity (складається з атрибутів .change_x і .change_y) і оброблятиме власний рух під час виклику .update().

3.3.4 Перевірка зіткнень з перешкодами

Малювання ігрових об’єктів відбувається в методі .on_draw().

```
def on_draw(self):
     arcade.start_render()
     self.all_sprites.draw()
```

Усі малюнки починаються з виклику arcade.start_render(). Подібно до оновлення, віддмальовуємо всі спрайти одночасно шляхом виклику self.all_sprites.draw().

Коли гравець стикається з перешкодою або користувач закриває вікно, гра закінчується. Зазвичай, ігри пов’язані зі зіткненнями тієї чи іншої форми. Виявлення зіткнень вимагає від програміста визначити, чи два ігрові об’єкти частково займають той самий простір на екрані.

Для цієї мети використовується функція виявлення зіткнень, щоб контролювати життєві сили, обмежувати рух гравця стінами та підлогою та створювати перешкоди, яких слід уникати. Залежно від задіяних ігрових об’єктів і бажаної поведінки логіка виявлення зіткнень може вимагати потенційно складної математики.

Є три різних метода Sprite для швидкого виявлення зіткнень:

1. Sprite.collides_with_point((x,y)) повертає True, якщо дана точка (x,y) знаходиться в межах поточного спрайту, і False в іншому випадку.
2. Sprite.collides_with_sprite(Sprite) повертає True, якщо даний спрайт перекривається з поточним спрайтом, і False в іншому випадку.
3. Sprite.collides_with_list(SpriteList) повертає список, що містить усі спрайти зі списку SpriteList, які збігаються з поточним спрайтом. Якщо спрайтів, що перекриваються, немає, список буде порожнім, тобто матиме нульову довжину.

Оскільки нас цікавить, чи зіткнувся спрайт для одиночної гри з будь-яким із ворожих спрайтів, використовуємо self.player.collides_with_list

(self.enemies_list) і перевіряємио, чи список, який він повертає, містить спрайти. Якщо так, то гру завершено з урахуванням наявності життєвих ресурсів прибульця.

```
def on_update(self, delta_time: float):

    #Якщо призупинено, нічого не оновлювати
    if self.paused:
        return

    #Перевірка на зітхнення
    if self.player.collides_with_list(self.enemies_list):
        if player_life == 0:
        arcade.close_window()
           else: -= player_life

    self.all_sprites.update()
```

Частота кадрів в грі є частотою оновлення графіки на екрані. Вища частота кадрів зазвичай забезпечує більш плавний процес гри, а нижча частота кадрів дає більше часу для виконання складних обчислень. Частотою кадрів аркадної гри Python керує ігровий цикл у arcade.run(). Він викликає .on_update() і .on_draw() приблизно 60 разів на секунду. Тому гра має частоту кадрів 60 кадрів в секунду або 60 FPS.

Цей покажчик може змінюватися вгору або вниз залежно від багатьох факторів, наприклад, навантаження на комп'ютер або довший, ніж зазвичай, час оновлення (відстань=час*швидкість), тобто так можна переміщувати свої спрайти з постійною швидкістю незалежно від частоти кадрів.

Метод .on_update() приймає параметр delta_time (це кількість часу в секундах, що минув з моменту останнього виклику .on_update()). Для гри, яка працює зі швидкістю 60 FPS, delta_time становитиме 1/60 секунди або приблизно 0,0167 секунди.

Таким чином, для оновлення руху спрайтів слід оновити їх положення вручну: іде заміна положення кожного спрайту вручну, множачи .change_x і .change_y на delta_time. Це гарантує, що спрайт рухається на постійну відстань

щосекунди, а не на постійну відстань кожного кадру, що може згладити процес гри.

```
    def on_update(self, delta_time: float):
        """ Оновлення позиції та статуси всіх ігрових об'єктів
         Якщо призупинено, нічого не робити  """

        # Якщо призупинено, нічого не оновлювати
        if self.paused:
            return

        # Перевірка за зітхнення з ворогом
        if
len(self.player.collides_with_list(self.enemies_list)) > 0:
            arcade.play_sound(self.collision_sound)
            arcade.close_window()

        #Оновлення
        for sprite in self.all_sprites:
            sprite.center_x = int(
                sprite.center_x + sprite.change_x * delta_time
            )
            sprite.center_y = int(
                sprite.center_y + sprite.change_y * delta_time
            )
```

3.4 Подальший розвиток ігрового додатку

Гра повинна бути привабливою для гравця. Під час розробки гри треба пам’ятати не тільки про виконання завершеного проекту, але й про якість розваги, яка б привернула аудиторію. Гра повинна бути розроблена так, щоб вона була складною, але по силам обраній цільовий аудіторії.

Для графічного контенту має бути обраний стиль, який сподобається цільовій аудиторії, а персонажі мають бути привабливими. Гра повинна бути плавною та інтуїтивно зрозумілою та залучати гравця у світ, створений грою. Для того щоб оминути даний ризик потрібно притримуватися головного правила «необхідність тестування» на всіх етапах розробки проекту. Це допоможе створити успішний проект.

Під час розробки існує ризик щодо не вірної розстановки пріоритетів для завдань. Наприклад, під час прототипування можна знехтувати якістю художнього оформлення. Вже після розробки усіх деталей можна зайнятися підвищенням якості артів. Також треба вміти вчасно зупинятися щодо додавання нових властивостей й вмінь у грі, треба правильно розуміти свої сили. Ще одним кроком у подальшому розвитку проекту стане створення авторських ассетів спрайтів ігрових елементів та музичного супроводу гри. Унікальні елементи допоможуть відокремити проект серед аналогів.

Наступний крок: впровадження нових властивостей в грі. Під час розробки можна зіткнутися з тим, що в грі можуть бути впровадженні рішення чи властивості, які не були розроблені до цього. З одного боку це дуже великий плюс, який свідчить про оригінальність гри, з іншого боку виникає і певний ризик. Він полягає у тому, що гравець може не відразу зрозуміти які дії від нього вимагає гра, чи як у ігровому світі керувати героєм.

Щоб полегшити процес «адаптації» до гри треба під час розробки надати прототип на тестування задля написання пошарового туторіалу щодо процесу керування в грі, та підказки для властивостей гри, з якими виникли труднощі.

Генерація рівнів за складності гри буде основною задачею для подальшого розвитку. Для реалізації цього під час розробки гри для полегшення навантаження на розробника та задля прискорення процесу розробки гри найчастіше набирається команда людей, яка потім буде розділена відповідно за своєю роботою (аналіз, художня частина та розробка).

Головний ризик під час цього процесу полягає у невідповідальності навичок кандидатів та рівнем роботи, яку вони мають виконувати. Наприклад через низьку кваліфікацію розробника можна «забути» про складну архітектуру проекту чи використання певних технологій, через низьку кваліфікацію художника про добре промальовану концепцію. Цю проблему можна вирішити за допомогою запровадження інтерв’ю, де буде перевірятися кваліфікація кандидата.

Як подальший розвиток гри слід в першу чергу розробити нові левели з унікальними мешканцями. Крім того, додати функцію «паузи» та зберігання стану гри. Як і всі ігрові додатки, проект буде протестовано як функціональні помилки, так і за візуальними ефектами.

ВИСНОВКИ

Аркадні ігри залишаються популярними серед користувачів за рахунок нескладності алгоритмів та графіки. Для початківця у game developer такий проект є гарним рішенням для вивчення основ цього напрямку та поповнення свого портфоліо.

На етапі аналітичного огляду були розглянуті аналоги та інструменти їх реалізації. Хоча Python робить навчання коду доступнішим для кожного, вибір для написання відеоігор може бути обмеженим, особливо якщо стоїть мета писати аркадні ігри з чудовою графікою та привабливими звуковими ефектами. Протягом багатьох років розробники ігор Python обмежувалися фреймворком pygame. Тепер є інший вибір. Для диплому було обрано засоби розробки та поставлені задачі до проекту (використовувалась мова програмування Python з бібліотекою Arcade).

На етапі проектування розроблено концепт-документ, який описує мету, ідеї та механіки об’єкту розробки. Крім цього, побудовано діаграму Use-Case для відображення основних задач і можливостей взаємодії героя з ігровим світом.

За результатами у роботі згенеровано два рівня зі своїми умовами існування головного героя. На кожен рівень гравцю дається 5 хвилин на проходження всіх перешкод. Для підвищення зацікавленості цільової аудіторії на основні механіки додано звукові ефекти.

Крім цього, розглянуті питання щодо ризиків під час розробки гри і можливості його подальшого розвитку до комерційного продукту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ:

1. Top Eight Best Games Made Using Python Language. URL: https://www.pythonblogs.com/top-best-games-made-using-python-language/ (дата звернення 21.02.2023)

2. Is Python good for developing games? Why or why not? URL: https://www.tutorialspoint.com/is-python-good-for-developing-games-why-or-why-not#:~:text=Python%20is%20utilized%20in%20game,how%20to%20use%20it%20efficiently (дата звернення 21.02.2023)

3. Python for game development: Why is it a good choice? URL: https://enlear.academy/python-for-game-development-why-is-it-a-good-choice-a4845832cfc2 (дата звернення 21.02.2023)

4. PyOpenGL для начинающих и немного новогоднего настроения. URL: https://habr.com/ru/post/246625/ (дата звернення 21.02.2023)

5. Top Python Game Engines. URL: https://realpython.com/top-python-game-engines/ (дата звернення 21.02.2023)

6. https://www.metacritic.com/game/xbox-360/braid

7. Most Famous Games Built With Pygame. URL: https://brightchamps.com/blog/most-famous-games-built-with-pygame/ (дата звернення 21.02.2023)

8. Top Eight Best Games Made Using Python Language. URL: https://www.pythonblogs.com/top-best-games-made-using-python-language/ (дата звернення 21.02.2023)

9. 7 Best Games That Use Python. URL: https://gamerant.com/best-games-that-use-python/#battlefield-2 (дата звернення 21.02.2023)

10. Use case diagram (UML use case diagram). URL: https://www.techtarget.com/whatis/definition/use-case-diagram (дата звернення 15.04.2023)

11. Ассети спрайтів. URL: https://kenney.nl/assets (дата звернення 15.04.2023)

12. Ресурс аудіо-файлів. URL: https://freesound.org (дата звернення 15.04.2023)