ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ НА БАЗІ ІГРОВИХ МЕХАНІК ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ШВИДКОДІЇ РОЗРАХУНКІВ»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Нікіта БІДНИК

(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. ІСД-41
Нікіта БІДНИК
Ім'я, ПРІЗВИЩЕ
Керівник: Іван ШАХМАТОВ
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем
Ступінь вищої освіти бакалавр
Спеціальність Інформаційні системи та технології
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС
_____ Каміла СТОРЧАК
«____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Біднику Нікіті Сергійовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка мобільного застосунку на базі ігрових механік для підвищення ефективності швидкодії розрахунків

керівник кваліфікаційної роботи Іван ШАХМАТОВ, викл.каф
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024р. № 36

2.Строк подання кваліфікаційної роботи «31» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи:
- Науково-технічна література з теми бакалаврської роботи.
- Методи розробки мобільних додатків.
- Основні принципи «Game Development»

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Game Development. Визначення та методи. Мови програмування.
2. Інструменти та прийоми розробки мобільних додатків на Python
3. Розробка мобільного застосунку на базі ігрових механік для підвищення ефективності швидкодії розрахунків за допомогою Python.

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «27» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	27.02 - 10.03.2024	
2	Обгрунтування актуальності роботи	11.03 – 16.03.2024	
3	Аналіз основних прийомів геймдеву	17.03 - 01.04.2024	
4	Інструменти та прийоми розробки мобільних додатків	02.04 – 18.04.2024	
5	Розробка мобільного застосунку на базі ігрових механік для підвищення ефективності швидкодії розрахунків	19.04 – 18.05.2024	
6	Оформлення роботи: вступ, висновки, реферат	19.05 - 22.05.2024	
7	Розробка демонстраційних матеріалів	23.05 - 24.05.2024	

Здобувач вищої освіти _____ Нікіта БІДНИК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Іван ШАХМАТОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 45 стор., 33 рис., 20 джерел.

Мета роботи – Порівняти та провести аналіз на придатність мови Python, її бібліотек, методик використання, ігрових двигунів, створення графіки, із більш відомими для використання у сфері Game Development мовами програмування такими як C++, C# та Java та зробити висновок про рівень придатності цієї мови в розробці ігор.

Об'єкт дослідження – Розробка мобільного застосунку на базі ігрових механік із використанням мови програмування Python.

Предмет дослідження – елементи геймдеву у застосунку та інструменти їх розробки.

Короткий зміст роботи: В цій кваліфікаційній роботі описано процес створення мобільного застосунку для покращення виконання простих математичних розрахунків в умі, з використанням ігрових технік та механік. Python показує складнощі та переваги цієї мови при розробці даного додатку в порівнянні з більш актуальнішими мовами програмування в цьому напрямку.

PYTHON, GAME DEVELOPMENT, РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ, ІГРОВИХ МЕХАНІК, МАТЕМАТИЧНИХ РОЗРАХУНКІВ

ABSTRACT

The text part of the qualification work for obtaining an educational degree bachelor's : 45 pages, 33 figures, 20 sources.

The purpose of the work is to compare and analyze the suitability of the Python language, its libraries, methods of use, game engines, graphics creation, with more well-known programming languages for use in the field of Game Development, such as C++, C# and Java, and to draw a conclusion about the level suitability of this language in game development.

The object of research is the development of a mobile application based on game mechanics using the Python programming language.

The subject of the research is game development elements in the application and tools for their development.

Summary of work: This qualifying work describes the process of creating a mobile application to improve the performance of simple mathematical calculations in the mind, using game techniques and mechanics. Python shows the difficulties and advantages of this language in the development of this application in comparison with more relevant programming languages in this direction.

PYTHON, GAME DEVELOPMENT, MOBILE APPLICATION DEVELOPMENT, GAME MECHANICS, MATHEMATICAL CALCULATIONS

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня бакалавра

Направляється здобувач Бідник Н.С до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 126, Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)
на тему: «Розробка мобільного застосунку на базі ігрових механік для підвищення ефективності швидкодії розрахунків».

Кваліфікаційна робота і рецензія додаються.

Директор ННІ _____ _____
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач_____
_____
_____
_____
_____

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача_____ на оцінку «_____» та присвоїти йому(їй) кваліфікацію _____.

Керівник кваліфікаційної роботи _____ _____
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

«____» _____ 20___ р

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач Бідник Н.С. допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою __ _____ _____ _____
(назва) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну роботу
на здобуття освітнього ступеня бакалавра

здобувача вищої освіти **Бідник Нікіта Сергійович**
на тему «**Розробка мобільного застосунку на базі ігрових механік для підвищення ефективності швидкодії розрахунків**»

Актуальність.

Розробка ігор є однією з найцікавіших та найрозповсюдженіших областей у сфері ІТ. Направленням що досліджується в цій кваліфікаційній роботі, було обрано розробку ігор та додатків на базі ігрових механік з використанням однієї із найпопулярніших мов програмування, у сьогодення – Python. Було вирішено порівняти та провести аналіз на придатність мови Python, її бібліотек, методик використання, ігрових двигунів, із більш відомими для використання у сфері Game Development мовами програмування такими як C++, C# та Java та зробити висновок про рівень придатності цієї мови в розробці ігор.

Позитивні сторони.

1. Робота демонструє ретельне дослідження можливостей і обмежень мови програмування Python у контексті розробки ігор. Детальний аналіз різних бібліотек та ігрових двигунів, зокрема Kivy, свідчить про систематичний підхід до оцінки технології, що є важливим для обґрунтованих висновків.
2. Надано порівняння Python з іншими мовами програмування для розробки ігор, такими як C++ (у зв'язці з Unreal Engine) та C# (з Unity). Це дозволяє чітко зрозуміти сильні та слабкі сторони Python у цій сфері.
3. Робота включає практичний приклад створення мобільного застосунку з використанням Python та Kivy. Це демонструє не тільки теоретичні знання, але й практичні навички, що значно підвищує цінність дослідження.

Недоліки.

1. Хоча робота вказує на деякі проблеми з продуктивністю та складнощами під час компіляції у формат .apk за допомогою buildozer, детальний аналіз цих технічних проблем не є достатньо розкритим.
2. Робота концентрується на порівнянні Python з C++ та C#, але мало уваги приділено іншим популярним мовам та інструментам для розробки ігор, таким як JavaScript (з використанням HTML5) або Lua (з використанням ігрового двигуна Love2D).

Висновок: *кваліфікаційна робота на здобуття ступеня бакалавра заслуговує оцінку "відмінно", а здобувач Бідник Нікіта Сергійович - заслуговує присвоєння кваліфікації: «бакалавр з інформаційних систем і технологій»*

Рецензент: _____ _____
науковий ступінь, вчене звання *підпис* *Ім'я, ПРІЗВИЩЕ*

ЗМІСТ

ВСТУП

Актуальність теми. Розробка ігор є однією з найцікавіших та найрозповсюдженіших областей у сфері IT. Ігри дарують гравцям можливість поринути у світ фантазій та спробувати себе у ролі найрізноманітніших персонажів. Однак ігри можуть слугувати не тільки для розваг, а і для навчання та розвитку мислення, наприклад граючи в гру з елементами вирішення математичних задач в умі. Направленням що досліджується в цій кваліфікаційній роботі, було обрано розробку ігор та додатків на базі ігрових механік з використанням однієї із найпопулярніших мов програмування, у сьогодення – Python. При вивчені та більш детальному дослідженні було виявлено, що нею користуються розробники мобільних інді-ігор та ігрових додатків в цілому. З цих причин було вирішено порівняти та провести аналіз на придатність мови Python, її бібліотек, методик використання, ігрових двигунів, створення графіки, із більш відомими для використання у сфері Game Development мовами програмування такими як C++, C# та Java та зробити висновок про рівень придатності цієї мови в розробці ігор.

Мета. Розробити мобільний застосунок на базі ігрових механік із використанням мови програмування Python.

Завдання дослідження. Виявити недоліки та переваги і зробити висновок чи є мова Python адаптованою під розробку мобільних застосунків з використанням ігрових механік.

Об'єкт дослідження. Процес розробки мобільного застосунку для покращення виконання математичних розрахунків.

Предмет дослідження: Ігрові механіки для покращення швидкодії розрахунків.

Методика дослідження. Порівняння різної документації та інформації стосовно схожих ігрових розробок, але на інших мовах програмування таких як C++, C#, Java ,з джерел таких як YouTube, Google, із власним розробленим застосунком на мові Python.

Наукова новизна: Розроблено додаток, який забезпечить практику вирішення простих арифметичних операцій в ігровій формі.

Апробація:

1. IV Всеукраїнській науково-практичний конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 року, ДУІКТ – «Інноваційні методики розробки мобільних додатків з ігровими механіками на Python для підвищення ефективності навчання», «Дослідження розробки додатків на базі ігрових механік з використанням мови програмування Python».
2. V Міжнародна науково-технічна конференція «сучасний стан та перспективи розвитку iot», 18 квітня 2024 року, ДУІКТ - «Інноваційні методики розробки мобільних додатків з ігровими механіками на Python для підвищення ефективності навчання», «Дослідження розробки додатків на базі ігрових механік з використанням мови програмування Python».

1 АНАЛІЗ МОВ ПРОГРАМУВАННЯ В GAME DEVELOPMENT

1.1 Поняття GAME DEVELOPMENT

Gamedev(геймдев) є скороченою назвою - "Game Development" - розробка ігор, що являє собою галузь програмування для створення комп'ютерних ігор, яка об'єднує технології, дизайн та креативність. Процес геймдеву включає розробку, графіки, звуку, анімації, фізики, ландшафту та багатьох інших елементів, необхідних для створення захопливого ігрового проекту.

Розробник ігор - це фахівець, який створює ігровий контент і програмне забезпечення для комп'ютерних ігор. Він працює в команді, яка включає гейм-дизайнерів, художників, музикантів та інших професіоналів, щоб втілити їхні ідеї в ігровий продукт. Особливістю індустрії gamedev є її динамічність, швидкий темп розвитку та постійне прагнення до інновацій.[8]

Індустрія геймдеву пропонує широкий спектр професій і спеціалізацій, серед них:

- Геймдев програміст: відповідає за розробку ігрових систем, геймплею, штучного інтелекту та інших технічних аспектів гри. Він використовує різні мови програмування для створення ігрових рушіїв та інструментів розробки.
- Дизайнер ігор: займається розробкою самих різних частин проекту, від створення персонажів та ландшафтів, до продумування сценаріїв гри, ігрового сюжету, квестів.
- Аніматори та художники: вони відповідають за створення графіки гри, вид персонажів, предметів, оточення та багато інших елементів гри. Вони використовують спеціалізовані програми та інструменти для створення візуальних компонентів ігрового світу.
- Звукорежисер: створює аудіоефекти, музику та звукові доріжки для гри. Він працює в тандемі з розробниками та дизайнерами, щоб забезпечити відповідну звукову атмосферу. [3]

1.2 Мова програмування C++

Мова **C++** була розроблена Б’єрном Страуструпом у 1980 році в компанії Bell Labs.

Рис. 1.1 – Логотип C++[12]

Сфера використання C++ дуже широка: це може бути прикладне програмне забезпечення, операційні системи, системи реального часу (наприклад, медичні прилади), драйвери, сервери, комп'ютерні ігри і т.д.[1]

Складнощі у вивчені цієї мови новачками полягає в тому, що в ній не передбачено засобів що керують введенням і виведенням інформації, оскільки ці засоби є різними в кожній моделі комп’ютера, а C++ не є орієнтованою на конкретну модель.

У геймплей-девелопменті C++ вже дуже багато років. Багато популярних ігор, такі як, GTA, The Witcher, StarCraft, TES, написані на цій мові. [6]

Зазвичай гра не розробляється з нуля, оскільки існує базове програмні рішення для цього, такі як ігрові двигуни. Вони включають в себе бібліотеки для розробки ігрової логіки, рендерингу, мережевої взаємодії та інших функцій. Працювати з ігровим двигуном зручно не лише програмістам, а й аніматором, які можуть легко додавати свої анімації, а художники – свої ігрові моделі.

Ігрові двигуни можуть бути розроблені компаніями для власного використання або продаватися за ліцензіями. Хоча ігрові двигуни загалом схожі один на одного, вони часто оптимізовані для конкретних цілей і завдань, для яких призначені. [5]

Рис. 1.2 – Логотип Unreal Engine[20]

Декілька причин чому студії обирають саме C++ при розробці:

- Вендори платформ (Sony, Microsof, Nintendo) пропонують API на C/C++, обсяг кодової бази їх ОС і SDK набагато більше, ніж ігрових двигунів, використовувати щось альтернативне просто не вийде, витрати на переробку поховають навіть нинку з її безрозмірними бюджетами.
- Портування ігор між платформами можливе лише мовами C/C++, причину я написав вище, жодної іншої спільної мови між платформами немає.
- Компілятори для плюсів оптимізувалися десятиліттями, щоб отримати відповідну продуктивність іншою мовою, йому теж доведеться пройти цей шлях, нехай не десятиліття, бо база вже є, але точні роки. Писати швидкий платформний код відносно високорівневий, на чомусь відмінному від C/C++ зараз просто не вийде.
- Legacy - успадкований код, від нього нікуди не подітися, його треба супроводжувати та лагодити, фіксувати баги. А ще треба розуміти, що цей код робить і іноді простіше написати деяку частину з нуля, ніж доопрацьовувати те, що є, але не завжди на це є люди і час.

1.3 Мова програмування Java

Мова програмування Java була створена в середині 1990-х років компанією Sun Microsystems. Історія її виникнення пов'язана з проектом, який спочатку мав назву "Green Project". Цей проект був започаткований в 1991 році групою інженерів на чолі з Джеймсом Гослінгом, Патріком Нотоном та Майком Шериданом. [1]

Основною метою Green Project було розробити мову програмування, яка б могла бути використана в різних побутових пристроях, таких як телевізійні приставки, пральні машини та інші. Це потребувало створення мови, що була б незалежною від платформи і могла б працювати на різних пристроях. Спочатку мова мала назву Oak, але згодом її перейменували на Java через юридичні проблеми з торговою маркою Oak.

Java була офіційно випущена у 1995 році як частина платформи Java 1.0. Основні принципи, на яких базувалась мова, включали простоту, об'єктно-орієнтованість, незалежність від платформи, надійність, безпеку, архітектурну нейтральність, портативність, високу продуктивність, багатозадачність та динамічність. Ці принципи дозволили Java стати популярною як для серверного, так і для клієнтського програмування. Це також призвело до її широкого використання у розробці ігор на платформі Android.

Рис. 1.3 – Логотип Java[19]

Java широко використовується для розробки ігор на платформі Android. Ця мова програмування надає безліч інструментів і бібліотек для створення мобільних ігор. Java також може використовуватися для розроблення ігор на ПК та інших платформах, але її продуктивність може бути дещо нижчою порівняно з C++.

Мова Java в деякому роді є родичем C#. Ці мови розвивались, впливаючи одна на одну, обидві підтримують збірку сміття та є об'єктно-орієнтованими. Однак, Java з самого початку була задумана незалежною від платформ,тобто здатна

працювати однаково на різних пристроях. Серед успішних ігор, написаних за допомогою Java, варто згадати Minecraft і RuneScape.[2]

Проекти що були розроблені на мові Java, компілюються в байт код що дозволяє їх виконувати на любій операційній системі в незалежності від того де проект був створений. Така особливість мови програмування використовується і при побудові комп'ютерних додатків, а також ігор. Проект що написаний на цій мові або одній з її бібліотек, може запускатися на різних приладах з різними операційними системами, головне щоб на них була встановлена JVM – віртуальна машина Java.

JavaFX — це набір інструментів для створення графічного інтерфейсу користувача (GUI) на мові Java. GUI представляє користувачу різні елементи: кнопки, меню, піктограми, списки, у вигляді графіки та зображень на дисплеї.[4]

JavaFX добре орієнтований на розробку ігор та настільних додатків і фактично замінює собою Swing, пропонуючи нові можливості для створення графічних інтерфейсів.

JavaFX також підтримує інтеграцію 3D-графіки, аудіо, відео та мережевих додатків і все в одному наборі інструментів. Він простий та прктичний у використанні, з гарною оптимізацією та підтримує велику кількість операційних систем, включаючи Windows, UNIX та MacOS.

Рис. 1.4 – Логотип JavaFX[18]

1.4 Мова програмування C#

Мова програмування C# була створена компанією Microsoft наприкінці 1990-х - початку 2000-х років. Розробка мови C# була частиною ширшого проекту .NET, який мав на меті створення нової платформи для розробки програмного забезпечення. [1]

Рис. 1.5 – Логотип C#[13]

Процес створення C# розпочався у 1999 році, коли Андерс Гейлсберг, відомий за створення мови Turbo Pascal та участю в розробці Delphi, приєднався до Microsoft. Гейлсберг очолив команду, яка працювала над новою мовою, що спочатку мала кодову назву "Cool" (C-like Object Oriented Language). Основною метою було створення сучасної об'єктно-орієнтованої мови програмування, яка б могла конкурувати з Java та C++.

C# увібрала в себе найкращі риси існуючих мов програмування, таких як C++, Java та Delphi. Вона підтримувала об'єктно-орієнтоване програмування, мала простий та чіткий синтаксис, що робило її легкою для вивчення та використання. Однією з важливих особливостей C# стала інтеграція з платформою .NET, що дозволяла програмам на C# працювати на різних операційних системах через Common Language Runtime (CLR).[11]

Перша версія C# була випущена у 2002 році як частина .NET Framework 1.0. З того часу мова постійно еволюціонувала, отримуючи нові функції та вдосконалення з кожною наступною версією. Наприклад, у версіях C# 2.0 та C# 3.0 були додані такі можливості, як узагальнення (generics), анонімні методи, лямбда-

вирази та LINQ (Language Integrated Query). Це дозволяє розробникам використовувати C# у широкому спектрі програмних проектів.

Unity — це потужний кросплатформенний рушій для розробки ігор та інтерактивного контенту, який вперше був випущений в 2005 році компанією Unity Technologies. Основна мета Unity полягає в тому, щоб надати розробникам інструменти для створення високоякісних 2D та 3D-ігор, а також віртуальної (VR) та доповненої реальності (AR).

Однією з ключових особливостей Unity є її кросплатформенність, що дозволяє розробникам створювати додатки для різних платформ, включаючи Windows, macOS, iOS, Android, Linux, а також для ігрових консолей, таких як PlayStation, Xbox і Nintendo Switch. Це забезпечує значну економію часу та ресурсів, оскільки код можна писати один раз і запускати на багатьох платформах.

Unity використовує мову програмування C#, що дозволяє створювати складні ігрові механіки та логіку. Середовище розробки Unity включає в себе редактор сцен (Scene Editor), який дозволяє легко розташовувати та налаштовувати об'єкти в грі, а також інструменти для роботи з анімацією, фізикою, звуком та інтерфейсом користувача.[10]

Однією з сильних сторін Unity є її багатий набір бібліотек і готових до використання компонентів, які значно полегшують процес розробки. Крім того, існує Unity Asset Store — онлайн-магазин, де розробники можуть купувати та продавати різні ресурси, такі як моделі, текстури, скрипти та звукові ефекти, що ще більше прискорює розробку. [2]

Unity також підтримує різні сучасні технології, включаючи підтримку VR та AR, що робить його популярним вибором для створення інтерактивних додатків у цих галузях. Unity використовують не лише для розробки ігор, але й для створення архітектурних візуалізацій, симуляторів, тренувальних програм та інших інтерактивних проектів.

З моменту свого запуску Unity постійно розвивається та оновлюється, пропонуючи розробникам все нові й нові можливості. Спільнота розробників Unity є однією з найбільших у світі, що забезпечує велику кількість навчальних

матеріалів, форумів та інших ресурсів для підтримки новачків та досвідчених користувачів.

Рис. 1.6 – Логотип Unity[16]

1.5 Мова програмування Python

Мова програмування Python була створена наприкінці 1980-х – на початку 1990-х років нідерландським програмістом Гвідо ван Россумом. Основна мета, яку ставив перед собою розробник, полягала у створенні мови, що поєднувала б високу продуктивність з простотою використання та читабельністю коду. Python був задуманий як наступник мови ABC, з якою ван Россум працював у Центрі Вісконта математичних і інформатичних наук (CWI) у Нідерландах. [1]

Рис. 1.7 – Логотип Python[15]

Перша версія Python з'явилася в грудні 1989 року, коли Гвідо ван Россум почав працювати над проектом у своїй вільний час. Офіційний реліз Python 0.9.0 відбувся в лютому 1991 року. Ця версія вже включала такі основні функції, як модулі, винятки, динамічні типи даних та інтерфейс з системними викликами. Однією з головних особливостей Python стало використання відступів для

позначення блоків коду, що робило його структурування природним та інтуїтивно зрозумілим.[9]

Python був названий на честь британського комедійного серіалу "Monty Python's Flying Circus", що підкреслює філософію розробки мови — вона мала бути приємною та веселою для використання. Від початку свого існування Python орієнтувався на простоту та мінімалізм, що відрізняло його від інших мов програмування того часу.

Python продовжував розвиватися завдяки активній підтримці спільноти розробників. У 2000 році була випущена версія Python 2.0, яка принесла багато нововведень, включаючи підтримку циклів for-each, повноцінну колекцію системних викликів та вдосконалену роботу з Unicode. Важливим етапом у розвитку мови стало створення Python Software Foundation (PSF) у 2001 році, яка взяла на себе відповідальність за управління та просування мови.

В 2008 році вийшла Python 3.0, яка не була зворотно сумісною з попередніми версіями. Цей випуск мав на меті виправити фундаментальні недоліки мови та зробити її ще більш зручною для користувачів. В Python 3 були значно покращені можливості роботи з текстовими даними та введені інші вдосконалення, які робили код більш зрозумілим та читабельним.

З часом Python став однією з найбільш популярних мов програмування у світі. Його використовують для розробки веб-додатків, наукових досліджень, аналізу даних, штучного інтелекту та машинного навчання. Популярність Python пояснюється його простотою, великою стандартною бібліотекою, активною спільнотою та широкими можливостями для інтеграції з іншими мовами та платформами.

Pygame – це «ігрова бібліотека», набір інструментів, які допомагають програмістам створювати ігри[7]. До них відносяться:

- Графіка та анімація
- Аудіо та звук
- Управління(клавіатура, миша, геймпад тощо)

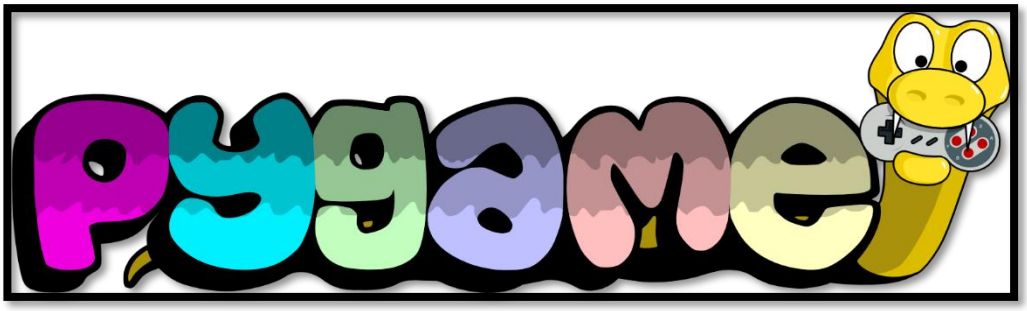

Рис. 1.8 – Логотип Pygame[17]

Pygame був створений Пітом Шінером в 2000-ому році як заміна застарілого pySDL. Ця бібліотека пропонує об’єктно-орієнтований інтерфейс, що робить її більш зручною для використання в Python порівняно з pySDL. Pygame підтримує Linux (входячи до складу багатьох популярних дистрибутивів), Windows, а також сумісний і з іншими платформами. Крім того, бібліотека отримала підтримку Android.

По своїй природі, Pygame є набором інструментів для створення ігрових об'єктів, обробки введенними користувачем даних та виведення графіки і аудіо. Завдяки великій кількості компонентів та деяким автономним концепціям, Pygame дозволяє реалізовувати проекти на більшості системах.

2 ОПИС МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Загальна ідея застосунку та його функціонал

Метою застосунку є розвиток когнітивних здібностей, зокрема навичок ментальної арифметики, що є важливим аспектом сучасної освіти та особистісного розвитку. В умовах швидкоплинного інформаційного світу здатність швидко і точно виконувати арифметичні обчислення без використання допоміжних засобів стає цінною навичкою. Створення інтерактивного застосунку для тренування цих навичок зацікавить та полегшить вивчення математичних операцій та буде набагато цікавішим. Такий застосунок, розроблений на основі фреймворку Kivy, може стати потужним інструментом для учнів, студентів та всіх бажаючих покращити свої математичні здібності.

Рис.2.1 – Логотип Kivy[14]

Застосунок має на меті створення інтуїтивно зрозумілого та інтерактивного середовища для тренування математичних навичок. Основна ідея полягає в тому,щоб шляхом постійних тренувань користувачі могли підвищити швидкість та точність виконання арифметичних операцій в умі. Застосунок пропонує різні рівні складності, які дозволяють користувачам поступово збільшувати навантаження і, таким чином, підвищувати свою майстерність.

Початкова схема застосунку мала наступний вигляд:

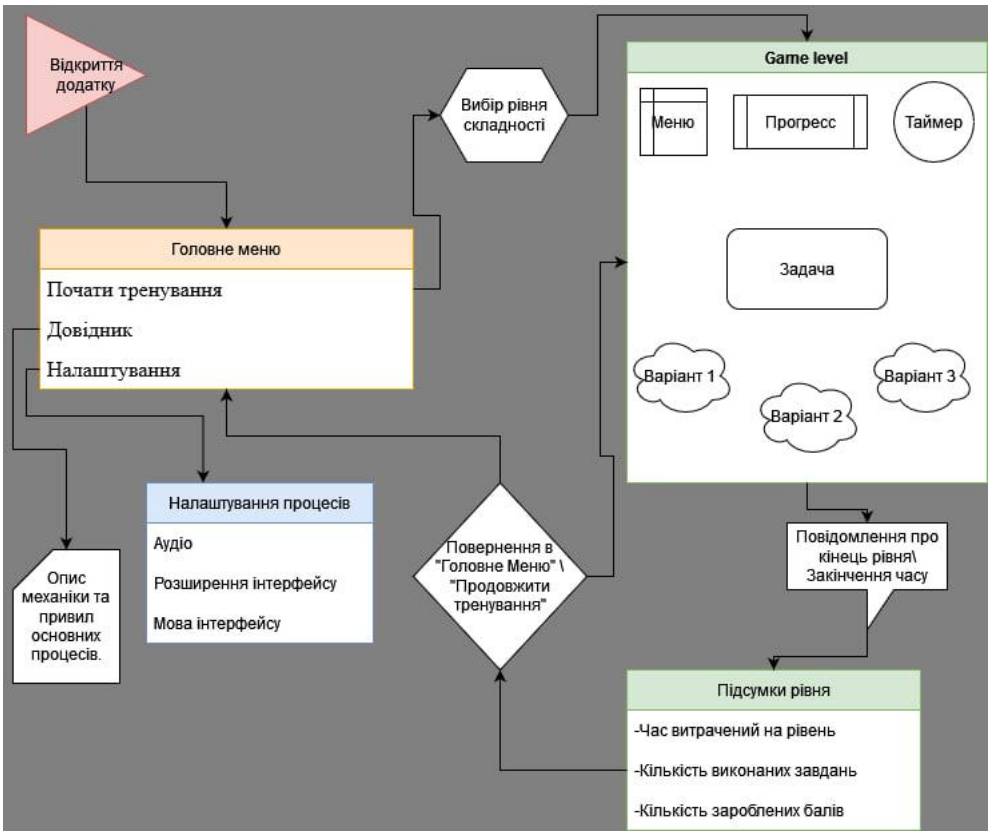

Рис.2.2 – Початкова схема застосунку

Під час розробки ця схема була спрощена для зручності користувачів.

Вона включала в себе наступні елементи додатку:

- Головне меню – яке мало наступні параметри: «Почати тренування», «Довідник», «Налаштування»;
- Вибір рівня складності;
- Ігровий рівень – «Game level»;
- Повідомлення про кінець рівня або закінчення часу;
- Підсумки рівня, що містить: «Час витрачений на рівень», «Кількість виконаних завдань», «Кількість зароблених балів»;
- Налаштування процесів, що містило: «Аудіо», «Розширення інтерфейсу», «Мова інтерфейсу».

Під час розробки схема була набагато спрощена, та представлена у наступному вигляді:

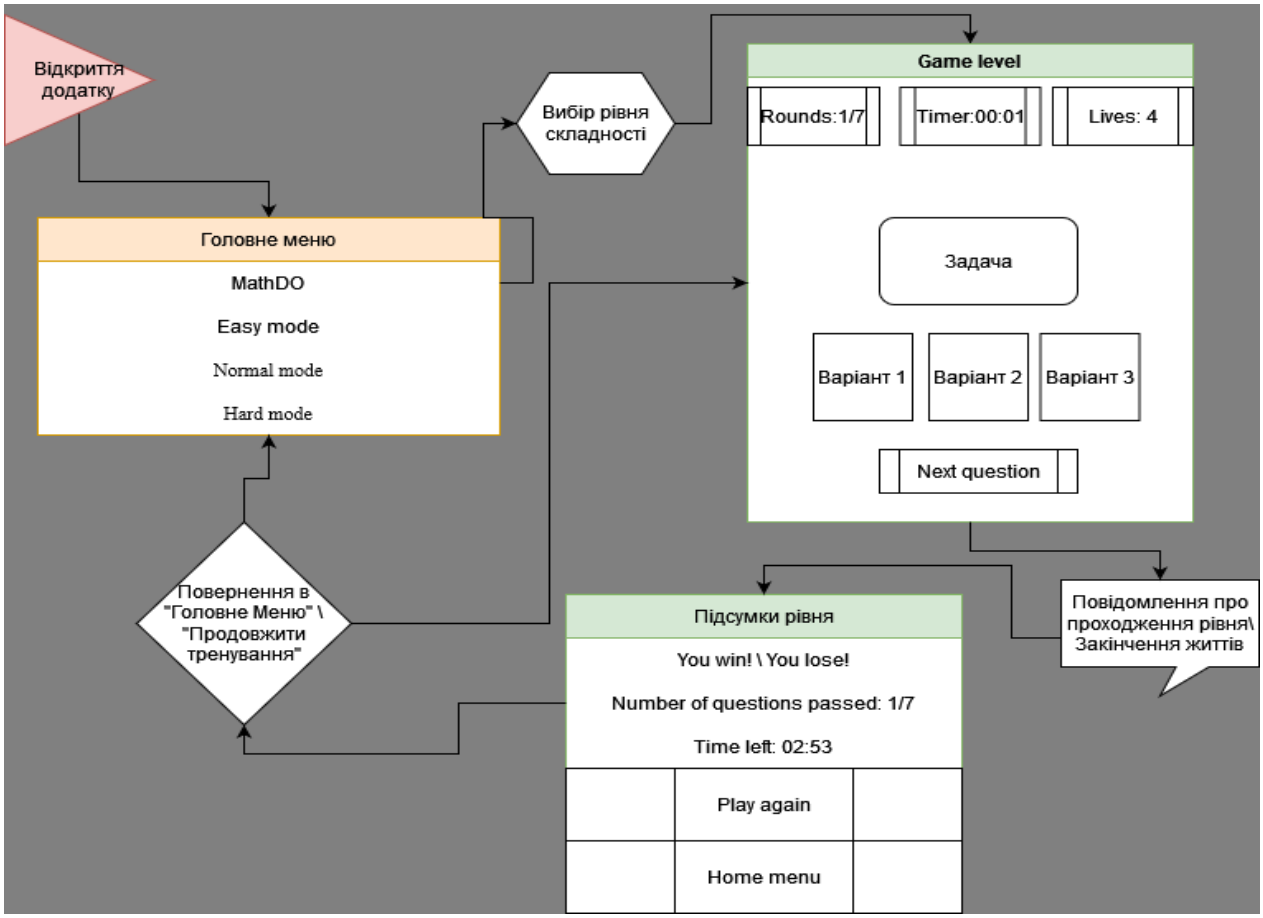

Рис.2.3 – Фінальна схема застосунку

Ця спрощена схема включає в себе наступні елементи:

- Головне меню яке мало наступні параметри: «Почати тренування», «Довідник», «Налаштування», було перероблено та замість цих елементів було одразу додано рівні складності, а саме: 1)«Easy mode», «Normal mode», «Hard mode» - при виборі рівня, змінюється кількість раундів, та кількість життів які зменшуються при виборі не вірних відповідей. 2)Також була додана назва застосунку у верхній частині головного меню із назвою «MathDo»;
- Ігровий рівень – «Game level», який в початковому вигляді мав би містити кнопку «Меню», для повернення в головне меню була прибрана, а замість неї було додано елементи: 1)«Rounds» - який показує кількість пройдених

раундів та кількість що залишилася; 2)«Lives» - що показує кількість життівщо залишилася, після закінчення яких закінчується гра;

- Підсумки рівня були перероблені таким чином: 1)По центру показується текст «You win!» або «You lose!», в залежності від того як закінчилося проходження рівня; 2)Було прибрано систему накопичення балів, у даній версії застосунку, але планується повернення у майбутньому;3)Додано два лічильники що вказують кількість пройдених рівнів «Number of questions passed:» та кількість витраченого часу на проходження гри: «Times left:»; 4)Додано дві кнопки «Play again», що починає нову гру із тим самим рівнем складності та «Home menu», що повертає у головне меню та дає можливість знову вибрати рівень складності.

Функціонал застосунку:

1. Генерація математичних завдань:

- Застосунок автоматично генерує випадкові математичні завдання, використовуючи операції додавання, віднімання, множення, ділення та піднесення в степінь.
- Генерація завдань відбувається з урахуванням рівня складності, який може бути налаштований користувачем.
- Залежно від рівня складності, використовуються різні діапазони чисел та комбінації операцій.

2. Інтерфейс користувача:

- Головне меню: Включає кнопки для вибору рівня складності. Користувач може легко переходити між різними режимами гри та налаштовувати параметри відповідно до своїх вподобань.
- Екран завдань: Відображає поточне математичне завдання та варіанти відповідей. Інтерфейс забезпечує швидкий доступ до інформації та можливість миттєвого вибору відповіді, а також побачити кількість пройдених рівнів, таймер який показує витрачений час та кількість життів що залишились.

- Кнопки відповідей: Три кнопки з можливими відповідями, з яких користувач обирає правильну. Відповіді перемішуються, щоб уникнути звикання до їх розташування.

3. Оцінка відповідей:

Після вибору відповіді користувачем, застосунок миттєво перевіряє її на правильність і відображає відповідне повідомлення «Your answer is correct! Good job!» або «Incorrect answer! Try better next time!».

Застосунок надає короткий відпочинок перед генерацією наступного завдання, що дозволяє користувачеві підготуватися до наступного питання. Перехід до наступного завдання відбувається за допомогою кнопки «Next question», яка з'являється лише при виборі правильної відповіді.

4. Рівні складності:

- Користувач може вибирати різні рівні складності «Easy mode», «Normal mode», «Hard mode», які впливають на кількість раундів які необхідно пройти щоб закінчити рівень та кількість життів які даються на цю гру, чим простіша складність тим менша кількість раундів та більша кількість життів. Це дозволяє користувачам поступово збільшувати навантаження і покращувати свої навички.

5. Відстеження прогресу:

- Застосунок зберігає статистику пройдених рівнів, а також час, витрачений на кожну гру, які виводяться у вікні результатів в разі проходження всіх рівнів, або при витраченні всіх життів.

6. Інтеграція з іншими платформами:

- Застосунок може бути розроблений для різних платформ, включаючи Android та десктопні операційні системи.
- Використання Kivy дозволяє легко портувати застосунок на різні платформи, що забезпечує широку аудиторію користувачів.

Технічні аспекти розробки:

1. Використання Kivy:

- Kivy є фреймворком з відкритим вихідним кодом, який дозволив створити багатоплатформний застосунок з інтерактивним інтерфейсом.
- Завдяки можливостям Kivy, було створено інтуїтивно зрозумілий та привабливий інтерфейс, який добре працює як на мобільних пристроях, так і на настільному комп'ютері.

2. Структура застосунку:

- Застосунок побудований на основі BoxLayout, що забезпечує гнучке розміщення елементів інтерфейсу.
- Використання різних віджетів Kivy, таких як Label, Button та Clock, дозволило створити динамічний та інтерактивний інтерфейс.
- Можливість налаштування вигляду та поведінки кожного елемента забезпечує високу гнучкість у розробці.

Застосунок для покращення здібностей розрахунків в умі, розроблений на основі фреймворку Kivy, є потужним інструментом для тренування математичних навичок. Його основні функції, включаючи генерацію математичних завдань, оцінку відповідей, відстеження прогресу та налаштування рівня складності, створюють інтерактивне та мотивуюче середовище для користувачів. Такий застосунок може бути корисним як для освітніх цілей, так і для індивідуального розвитку когнітивних здібностей, сприяючи підвищенню швидкості та точності виконання арифметичних операцій в умі.

2.2 Області використання даного застосунку в повсякденному житті

Розвиток ментальної арифметики є важливим компонентом сучасної освіти та особистісного розвитку. Розроблений застосунок, спрямований на покращення навичок ментальних обчислень який може знайти застосування у різних аспектах повсякденного життя. Завдяки інтерактивному інтерфейсу, гнучкості налаштувань та зручності використання, такий застосунок може бути корисним для широкої аудиторії, від школярів до дорослих професіоналів.

Освітня сфера:

1. Шкільна освіта:

- Застосунок може бути інтегрований у навчальний процес в школах для покращення математичних навичок учнів.
- Використання інтерактивних завдань сприяє кращому засвоєнню матеріалу та підвищує зацікавленість учнів у вивченні математики.
- Вчителі можуть використовувати застосунок як додатковий інструмент для закріплення пройденого матеріалу.

1. Позашкільна освіта:

- Застосунок може використовуватися в різних позашкільних математичних гуртках та секціях. Це допомагає дітям покращити свої математичні навички поза рамками шкільної програми, сприяючи їх всебічному розвитку.

2. Домашнє навчання:

- Батьки можуть використовувати застосунок для додаткових занять з дітьми вдома. Це створює можливість для батьків активно залучатися до навчального процесу та допомагати своїм дітям у розвитку важливих навичок.

Професійна сфера:

1. Фінансові спеціалісти:

- Застосунок може бути корисним для фінансових аналітиків, бухгалтерів та інших професіоналів, які регулярно працюють з числами. Регулярні тренування з ментальної арифметики допоможуть їм швидше та точніше виконувати розрахунки, що сприятиме підвищенню ефективності їхньої роботи.

2. Інженери та технічні спеціалісти:

- В інженерних професіях часто виникає потреба в швидких розрахунках, особливо в польових умовах. Застосунок може використовуватися для підтримання навичок ментальних обчислень, що дозволить інженерам більш ефективно виконувати свої завдання.

Повсякденне життя:

1. Повсякденні покупки:

- Застосунок може допомогти людям швидко виконувати розрахунки під час покупок, наприклад, підраховувати загальну вартість товарів у кошику або визначати суму здачі. Це корисно для тих, хто хоче контролювати свої витрати та уникати помилок при оплаті.

2. Особисті фінанси:

- Управління особистими фінансами включає багато розрахунків, таких як підрахунок бюджету, розподіл витрат та інвестиційні розрахунки. Регулярні тренування з використанням застосунку допоможуть швидше і точніше виконувати ці завдання, що сприятиме кращому управлінню фінансами.

3. Розвиток когнітивних здібностей:

- Ментальні тренування з арифметики сприяють загальному розвитку когнітивних функцій, включаючи пам'ять, увагу та логічне мислення. Це може бути корисним для людей будь-якого віку, допомагаючи підтримувати мозок у тонусі та покращувати загальні розумові здібності.

Гейміфікація та розваги:

1. Ігрові застосунки та платформи:

- Застосунок може бути інтегрований у різні освітні ігрові платформи, де користувачі можуть змагатися один з одним у швидкості та точності розрахунків. Така гейміфікація сприяє підвищенню мотивації та зацікавленості користувачів у тренуваннях.

2. Сімейні ігри:

- Застосунок може стати частиною сімейних ігор та розваг, де члени сім'ї можуть змагатися між собою у виконанні арифметичних завдань. Це не тільки розвиває навички, але й зміцнює родинні зв'язки через спільні заняття.

Застосунок має широкий спектр застосувань у повсякденному житті. Він може бути корисним як в освітній сфері, так і в професійній діяльності, а також у повсякденних ситуаціях. Розроблений для розвитку математичних навичок, застосунок сприяє підвищенню когнітивних здібностей, що робить його цінним інструментом для користувачів різного віку та професій.

2.3 Інструменти та методики для розробки мобільного додатку

Для розробки додатку, використовувалися різноманітні інструменти та методики, які забезпечують ефективність та зручність процесу створення, тестування та розгортання додатку. Нижче описані ключові інструменти та методики, використані в розробці цього додатку.

Інструменти для розробки:

1. Kivy**:**

- Фреймворк Kivy був основним інструментом для розробки користувацького інтерфейсу та логіки додатку. Kivy є бібліотекою з відкритим вихідним кодом для створення багатоплатформових додатків, яка підтримує розробку для Windows, macOS, Linux, Android та iOS.
- Переваги Kivy включають його гнучкість, можливість створення інтерактивних інтерфейсів та підтримку мультитачу, що робить його ідеальним для розробки освітніх ігор.

2. Python:

- Мова програмування Python використовувалася для написання основної логіки додатку, відомий своєю простотою та читабельністю, що дозволило швидко розробляти та тестувати нові функції.
- Бібліотеки та модулі Python, такі як Kivy, допомогли значно скоротити час розробки та спростити процес інтеграції додаткових функцій.

3. Buildozer:

- Buildozer застосовувався для компіляції Python-коду у формат APK для Android. Buildozer автоматизує процес створення, налаштування та компіляції додатку, забезпечуючи легкість та швидкість розгортання.
- Конфігураційний файл buildozer.spec дозволяє налаштовувати різні параметри проекту, такі як залежності, дозволи та налаштування графіки, що робить процес збірки більш керованим.

Методики розробки:

1. Гнучка розробка (Agile Development)

- Методологія Agile була обрана для управління процесом розробки. Agile дозволяє розробникам швидко адаптуватися до змін вимог, проводити часті релізи та отримувати зворотний зв'язок від користувачів.
- Ітераційний підхід забезпечує можливість постійного вдосконалення додатку, впровадження нових функцій та виправлення помилок на основі відгуків користувачів.

2. Тестування:

- Автоматизоване та ручне тестування використовувалося для забезпечення якості додатку. Автоматизовані тести дозволяють швидко перевіряти працездатність ключових функцій після внесення змін у код.

3. Розробка інтерфейсу користувача (UI/UX Design):

- Дизайн інтерфейсу був спрямований на створення інтуїтивно зрозумілого та зручного інтерфейсу для користувачів. Враховувалися принципи доступності та юзабіліті, щоб додаток був зрозумілим для користувачів різного віку.
- Інтерактивні елементи та анімації використовувалися для підвищення залученості користувачів та надання їм позитивного досвіду взаємодії з додатком.

Використання інструментів і методик у розробці додатку:

1. Kivy та Python були використані для створення основної логіки гри та розробки користувацького інтерфейсу. Зокрема, Kivy дозволяє створювати адаптивні інтерфейси, що відображаються коректно на різних розмірах екранів.
2. Buildozer забезпечував автоматизовану компіляцію та підготовку додатку для розгортання на платформі Android, спрощуючи процес створення APK-файлів.
3. Методологія Agile дозволила ефективно управляти проектом, швидко реагувати на зміну вимог та забезпечувати регулярні оновлення додатку.
4. Тестування забезпечило високу якість продукту, мінімізуючи кількість помилок та проблем перед випуском.

5. UI/UX дизайн гарантував, що додаток буде зручним та привабливим для користувачів, сприяючи їхньому залученню та задоволенню від використання.

Висновок:

Розробка застосунку включала використання сучасних інструментів та методик, таких як Kivy, Python, Buildozer та Agile. Цей комплексний підхід дозволив створити ефективний, зручний та функціональний продукт, який може бути корисним у різних сферах життя. Використання перевірених інструментів та методик забезпечило високу якість та стабільність додатку, сприяючи його успішному впровадженню та використанню.

2.4 Платформи мобільних додатків для застосунку

Зважаючи на характеристики мого застосунку та використання фреймворку Kivy, було розглянуто детальніше кожну з платформ мобільних додатків:

1. Android: Android - це найбільш популярна платформа для мобільних додатків у світі. За допомогою Kivy можливо розробити додаток, який легко розгортати на пристроях Android. Крім того, Kivy надає зручний інтерфейс для взаємодії з різними функціями пристрою, такими як камера, геолокація, датчики тощо.
2. iOS: iOS також підтримується Kivy, але розгортання додатків на цій платформі може бути складнішим через вимоги Apple. Щоб розгорнути додаток на iOS, доведеться виконати деякі додаткові кроки, такі як налаштування сертифікатів і профілів розробника, а також виконання вимог App Store.
3. Windows Mobile: Хоча платформа Windows Mobile втратила свою популярність, Kivy також підтримує розробку додатків для цієї платформи. Це може бути корисно, якщо потрібно забезпечити сумісність додатку з різними операційними системами.
4. Cross-Platform Solutions: Крім традиційних платформ, також можливо використовувати різні фреймворки та інструменти для розробки мобільних

додатків для кількох платформ одночасно. Наприклад, Flutter, React Native або Xamarin дозволяють створювати додатки, які працюють як на Android, так і на iOS, з використанням спільного коду.

Для розробки застосунку було обрано платформу Android через ряд переваг:

1. Широка аудиторія користувачів: Android має найбільшу кількість користувачів серед усіх мобільних платформ, що робить його привабливим вибором для розповсюдження мого застосунку серед широкого кола користувачів.
2. Розмаїття пристроїв і варіантів: Android працює на різних пристроях, від доступних смартфонів до планшетів та інших пристроїв, що дозволяє максимально охопити цільову аудиторію.
3. Відкрита платформа для розробників: Android є відкритою платформою, що надає розробникам більше свободи у створенні та розгортанні додатків. Це означає більше можливостей для інновацій та експериментів.
4. Багатий функціонал і можливості: Android надає доступ до різних функцій та можливостей пристрою, таких як камера, геолокація, датчики, NFC тощо. Це дозволяє розробникам створювати додатки з розширеним функціоналом та інтерактивними можливостями.
5. Магазин додатків Google Play: Розгортання додатку в Google Play дозволяє забезпечити його доступність для мільйонів користувачів Android по всьому світу. Google Play також надає зручні інструменти для розгортання, оновлення та монетизації додатків.

Обираючи платформу Android для мого застосунку, я можу використовувати ці переваги для забезпечення максимального охоплення аудиторії та успішного розгортання мого продукту на ринку мобільних додатків.

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1 Процес розробки застосунку

Середовищем розробки мобільного застосунку було обрано PyCharm, в якому розроблявся наведений код далі.

```
import random
import kivy
from kivy.app import App
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.boxlayout import BoxLayout
from kivy.clock import Clock
from datetime import datetime
from random import randint, choice

kivy.require('2.0.0')
```

Рис.3.0 – Імпорт біліотек

В даному модулі приведено список імпортованих бібліотек необхідних для розробки, а саме:

- Модуль random, який надає функції для генерації випадкових чисел.
- Модуль kivy, головний модуль фреймворку Kivy, який використовується для розробки багатодотикових застосунків.
- Клас App з модуля kivy.app, який є базовим класом для створення застосунків Kivy.
- Клас Label з модуля kivy.uix.label, який використовується для відображення тексту на екрані.
- Клас Button з модуля kivy.uix.button, який використовується для створення кнопок.
- Клас boxlayout з модуля kivy.uix.boxlayout, який використовується для організації дочірніх елементів в рядок або стовпець.
- Клас Clock з модуля kivy.clock, який використовується для створення таймерів та розпорядження часом.
- Клас datetime з модуля datetime, який надає функціональність для роботи з часом та датами.

- Функції randint і choice з модуля random для генерації випадкових чисел і вибору випадкового елемента зі списку відповідно.
- Вказано версію фреймворку Kivy, яку використовує програма, в даному випадку - версія 2.0.0.

Імпорт бібліотек є однією з перших кроків при написанні коду та може змінюватись в ході розробці.

```
class MathGame(App):
    def build(self):
        self.operators = ['+', '-', '*', '/', '^']
        self.main_layout = BoxLayout(orientation='vertical')
        # Показать главное меню при первом старте приложения
        self.show_main_menu()

        return self.main_layout
```

Рис.3.1 – Метод build

В даному блоці коду визначаємо клас MathGame, який є підкласом класу App з фреймворку Kivy. Цей клас буде відповідати за функціональність нашого застосунку. Визначаємо метод build, який є обов'язковим для підкласу App і повертаємо головний інтерфейс застосунку. Ініціалізуємо змінну operators, яка містить список математичних операторів. BoxLayout є одним з основних макетів (layouts) у бібліотеці Kivy, цей макет дозволяє розміщувати віджети у вертикальному або горизонтальному напрямку, в залежності від вказаної орієнтації. Створюємо екземпляр BoxLayout з орієнтацією vertical і присвоює його змінній main_layout.

Цей контейнер буде головним контейнером для розміщення інших елементів інтерфейсу. Викликаємо метод show_main_menu(), який відображає головне меню на головному вмісті main_layout. Повертаємо головний вміст застосунку, який є екземпляром BoxLayout, що містить головне меню.

```
def show_main_menu(self, instance=None):
    self.main_layout.clear_widgets()

    title_label = Label(text="MathDo", font_size='70sp')
    title_label.font_name = 'Danfo-Regular-VariableFont_ELSH.ttf'
    title_label.color = '#fff6db'
    self.main_layout.add_widget(title_label)
```

```
    easy_button = Button(text="Easy mode", font_size='40sp',
on_press=self.start_easy_mode)
    easy_button.size_hint_y = 0.5
    easy_button.size_hint_x = 0.5
    easy_button.pos_hint = {'center_x': 0.5, 'center_y': 0.5}  #
Розміщуємо кнопку по центру відносно батьківського контейнера
    easy_button.background_color = (0,0,0,0)
    easy_button.color = '#59f23a'
    easy_button.font_name = 'PoetsenOne-Regular.ttf'
    self.main_layout.add_widget(easy_button)

    normal_button = Button(text="Normal mode", font_size='40sp',
on_press=self.start_normal_mode)
    normal_button.size_hint_y = 0.5
    normal_button.size_hint_x = 0.5
    normal_button.pos_hint = {'center_x': 0.5, 'center_y': 0.5}  #
Розміщуємо кнопку по центру відносно батьківського контейнера
    normal_button.background_color = (0, 0, 0, 0)
    normal_button.color = '#f7bd40'
    normal_button.font_name = 'PoetsenOne-Regular.ttf'
    self.main_layout.add_widget(normal_button)

    hard_button = Button(text="Hard mode", font_size='40sp',
on_press=self.start_hard_mode)
    hard_button.size_hint_y = 0.5
    hard_button.size_hint_x = 0.5
    hard_button.pos_hint = {'center_x': 0.5,'center_y': 0.5}  #
Розміщуємо кнопку по центру відносно батьківського контейнера
    hard_button.background_color = (0, 0, 0, 0)
    hard_button.color = '#f00c36'
    hard_button.font_name = 'PoetsenOne-Regular.ttf'
    self.main_layout.add_widget(hard_button)
```

Рис.3.2 – Метод show_main_menu

Далі йде опис метода show_main_menu(), який згадувався раніше. Очищуємо вміст головного контейнера main_layout, видаляючи всі дочірні віджети. Створюємо кнопку easy_button з текстом "Easy mode", розміром шрифту 40sp. Встановлюємо розміри кнопки, розташування та оформлення. При натисканні на кнопку буде викликана функція start_easy_mode. Додаємо цю кнопку до головного контейнера main_layout. Створюємо кнопку normal_button з текстом "Normal mode", розміром шрифту 40sp. Встановлюємо розміри кнопки, розташування та оформлення. При натисканні на кнопку буде викликана функція start_normal_mode. Додаємо цю кнопку до головного контейнера main_layout. Створюємо кнопку hard_button з текстом "Hard mode", розміром шрифту 40sp. Встановлюємо розміри

кнопки, розташування та оформлення. При натисканні на кнопку буде викликана функція start_hard_mode. Додаємо також цю кнопку до головного контейнера main_layout.

Цей блок відповідає за елементи головного меню та їх вид, а також за виклик цього меню в цілому.

```
def start_easy_mode(self, instance):
    self.roundF = 7
    self.lives = 4
    self.difficulty = 'easy'
    self.start_game()

def start_normal_mode(self, instance):
    self.roundF = 15
    self.lives = 3
    self.difficulty = 'normal'
    self.start_game()

def start_hard_mode(self, instance):
    self.roundF = 24
    self.lives = 2
    self.difficulty = 'hard'
    self.start_game()
```

Рис.3.3 – Метод start_easy_mode

Далі йде блок з встановленням складності гри, оскільки ці методи подібні між собою розглянемо на прикладі одного з них:

1. def start_easy_mode(self, instance): Ця строка визначає метод start_easy_mode, який приймає параметр instance.
2. self.roundF = 7: Встановлює змінну roundF на значення 7. Ця змінна визначає кількість раундів для режиму "легко".
3. self.lives = 4: Встановлює змінну lives на значення 4. Ця змінна представляє кількість життів у грі.
4. self.difficulty = 'easy': Встановлює змінну difficulty на значення 'easy'. Ця змінна вказує на обраний рівень складності.
5. self.start_game(): Викликає метод start_game(), який розпочинає гру з обраними параметрами для режиму "легко".

Такі самі дії виконуються і для методів start_normal_mode та start_hard_mode, але з відповідними змінами у значеннях змінних roundF, lives і difficulty, відповідно до обраного рівня складності.

```
def start_game(self):
    self.layout = BoxLayout(orientation='vertical')

    # Верхня частина інтерфейсу з відображенням поточного рівня та
таймера
    self.top_layout = BoxLayout(orientation='horizontal',
size_hint_y=None, height='50dp')
    self.level_label = Label(text=f"Rounds:1/{self.roundF}",
font_size='50sp')
    self.level_label.font_name = 'Honk-Regular-
VariableFont_MORF,SHLN.ttf'
    self.top_layout.add_widget(self.level_label)

    self.time_label = Label(text="Timer:00:00", font_size='57sp',
halign='right')
    self.time_label.font_name = 'Jacquard12-Regular.ttf'
    self.top_layout.add_widget(self.time_label)

    self.lives_label = Label(text=f"Lives: {self.lives}",
font_size='50sp')
    self.lives_label.font_name = 'Jaini-Regular.ttf'
    self.top_layout.add_widget(self.lives_label)

    self.layout.add_widget(self.top_layout)

    # Частина інтерфейсу з питанням та кнопками
    self.question_label = Label(text='', font_size='45sp')
    self.question_label.font_name = 'PoetsenOne-Regular.ttf'
    self.layout.add_widget(self.question_label)

    self.buttons_layout = BoxLayout(orientation='horizontal')
    self.buttons_layout.size_hint_y = 0.7
    self.buttons_layout.size_hint_x = 0.7
    self.buttons_layout.pos_hint = {'center_x': 0.5, 'center_y':
0.5}
    self.buttons = []
    for _ in range(3):
        button = Button(font_size='40sp',
on_press=self.check_answer)
        button.color = '#3de048'
        button.font_name = 'PoetsenOne-Regular.ttf'
        button.background_color= '#c369cf'
        self.buttons.append(button)
        self.buttons_layout.add_widget(button)
    self.layout.add_widget(self.buttons_layout)
```

```
    # Нижня частина інтерфейсу з кнопкою "Next question"
    self.next_button = Button(text='Next question',
font_size='50sp', on_press=self.next_question)
    self.next_button.size_hint_y = 0.5
    self.next_button.size_hint_x = 0.5
    self.next_button.pos_hint = {'center_x': 0.5, 'center_y': 0.5}
    self.next_button.height = '50dp'
    self.next_button.opacity = 0
    self.next_button.disabled = True
    self.next_button.background_color = (0, 0, 0, 0)
    self.next_button.color = 'orange'
    self.next_button.font_name = 'Honk-Regular-
VariableFont_MORF,SHLN.ttf'
    self.layout.add_widget(self.next_button)

    # Змінні для відстеження поточного рівня
    self.roundT = 1  # Початкова кількість пройдених рівнів

    self.main_layout.clear_widgets()
    self.main_layout.add_widget(self.layout)

    # Генерація першого питання
    self.generate_question()

    # Запускаємо таймер
    self.start_timer()
```

Рис.3.4 – Метод start_game

Далі йде метод start_game() що відповідає за початок гри і ось пояснення його коду:

1. self.layout = BoxLayout(orientation='vertical'): Створюється головний контейнер (BoxLayout) вертикальної орієнтації, який буде містити інтерфейс гри.
2. self.top_layout = BoxLayout(orientation='horizontal', size_hint_y=None, height='50dp'): Створюється контейнер для верхньої частини інтерфейсу з горизонтальною орієнтацією. Висота контейнера встановлюється на 50 пікселів.
3. self.level_label = Label(text=f"Rounds:1/{self.roundF}", font_size='50sp'): Створюється елемент мітки, який показує поточний раунд та загальну кількість раундів. Розмір тексту встановлюється на 50 сп.

4. self.time_label = Label(text="Timer:00:00", font_size='57sp', halign='right'): Створюється елемент мітки для відображення таймера. Розмір тексту встановлюється на 57 сп.
5. self.lives_label = Label(text=f"Lives: {self.lives}", font_size='50sp'): Створюється елемент мітки, який показує кількість життів гравця. Розмір тексту встановлюється на 50 сп.
6. self.layout.add_widget(self.top_layout): Додає контейнер верхньої частини інтерфейсу до головного контейнера.
7. self.question_label = Label(text='', font_size='45sp'): Створюється елемент мітки для відображення питання. Початковий текст порожній. Розмір тексту встановлюється на 45 сп.
8. self.buttons_layout = BoxLayout(orientation='horizontal'): Створюється контейнер для кнопок з горизонтальною орієнтацією.
9. self.buttons = []: Створюється пустий список для зберігання об'єктів кнопок.
10. for _ in range(3): Цикл для створення трьох кнопок.
11. button = Button(font_size='40sp', on_press=self.check_answer): Створюється кнопка з розміром тексту 40 сп та додається обробник події натискання, який вказує на метод check_answer.
12. self.layout.add_widget(self.next_button): Додає кнопку "Next question" до головного контейнера.
13. self.roundT = 1: Ініціалізує змінну roundT, яка відслідковує кількість пройдених раундів.
14. self.main_layout.clear_widgets(): Очищає головний контейнер від усіх попередніх віджетів.
15. self.main_layout.add_widget(self.layout): Додає головний контейнер до головного віджету, щоб відобразити інтерфейс гри.
16. self.generate_question(): Викликає метод generate_question(), щоб згенерувати перше питання.
17. self.start_timer(): Викликає метод start_timer(), щоб запустити таймер гри.

Цей блок відповідає за початок меню гри та його елементів, та візуалізацію подій таких як раунди, життя та час.

```
def start_timer(self):
    self.start_time = datetime.now()  # Запоминаем время начала игры
    Clock.schedule_interval(self.update_timer, 1)  # Обновляем
таймер каждую секунду

def update_timer(self, dt):
    current_time = datetime.now()
    elapsed_time = current_time - self.start_time
    minutes = int(elapsed_time.total_seconds() // 60)
    seconds = int(elapsed_time.total_seconds() % 60)
    self.time_label.text = f"Timer:{minutes:02d}:{seconds:02d}"  #
Форматируем время в виде "MM:SS"
```

Рис.3.5 – Метод start_timer та update_timer

Далі йдуть функції для обробки таймера:

1. self.start_time = datetime.now(): Запам'ятовує час початку гри, використовуючи поточний час і дату.
2. Clock є частиною бібліотеки Kivy, яка використовується для керування та планування подій у часі. Clock.schedule_interval(self.update_timer, 1):Розкладає виклик функції update_timer кожну секунду (1 секунда).
3. current_time = datetime.now():Отримує поточний час.
4. elapsed_time = current_time - self.start_time: Обчислює час, що пройшов з моменту початку гри.
5. minutes = int(elapsed_time.total_seconds() // 60): Визначає кількість хвилин, які пройшли з початку гри, шляхом цілочисельного ділення загальної кількості секунд на 60.
6. seconds = int(elapsed_time.total_seconds() % 60): Визначає кількість секунд, які пройшли з початку гри, використовуючи залишок від ділення загальної кількості секунд на 60.
7. self.time_label.text = f"Timer:{minutes:02d}:{seconds:02d}": Оновлює текст мітки таймера з форматом "MM:SS", де minutes і seconds форматуються так, щоб завжди складали дві цифри, доповнюючи нулями зліва за потреби.

Цей модуль відповідає за функціональну частину таймера та корректність його роботи.

```
def generate_question(self):
    if self.roundT <= self.roundF and self.lives > 0:  # Проверяем,
не завершилась ли игра
        num1 = randint(1, 100)
        num2 = randint(1, 12)  # avoiding division by zero
        operator = choice(self.operators)

        correct_answer = None  # Инициализируем переменную перед
началом условий проверки оператора '^'

        if operator == '+':
            correct_answer = num1 + num2
        elif operator == '-':
            correct_answer = num1 - num2
        elif operator == '*':
            correct_answer = num1 * num2
        elif operator == '/':
            while num1 % num2 != 0:  # Повторяем генерацию, пока
результат деления не будет целым
                num1 = randint(1, 100)
                num2 = randint(1, 12)  # avoiding division by zero
                if num1 % num2 == 0:  # Check if division results in
an integer
                    correct_answer = num1 // num2
                    break
        elif operator == '^':
            num1 = randint(1, 10)  # keeping base small for
simplicity
            num2 = randint(1, 3)  # exponent should be <= 3
            correct_answer = num1 ** num2

        if correct_answer is not None:  # Проверяем, было ли
установлено значение для correct_answer
            self.correct_answer = correct_answer
            self.question_label.text = f'{num1} {operator} {num2} =
?'

            answers = [correct_answer]
            for _ in range(2):
                wrong_answer = correct_answer + randint(-10, 10)
                while wrong_answer in answers:  # Avoid duplicates
                    wrong_answer = correct_answer + randint(-10, 10)
                answers.append(wrong_answer)

            random.shuffle(answers)  # Перемешать список с ответами

            for i, button in enumerate(self.buttons):
```

```
                button.text = str(answers[i])

            self.next_button.opacity = 0
            self.next_button.disabled = True
    else:
        self.end_game()
```

Рис.3.6 – Метод generate_question

Одна з головних функцій generate_question, яка відповідає за генерацію питань у грі:

1. if self.roundT <= self.roundF and self.lives > 0: Перевіряє, чи не закінчилася гра (тобто, чи кількість раундів не перевищує максимальну і кількість життів більше за 0).
2. num1 = randint(1, 100): Генерує випадкове число num1 від 1 до 100.
3. num2 = randint(1, 12): Генерує випадкове число num2 від 1 до 12, уникаючи ділення на нуль.
4. operator = choice(self.operators): Вибирає випадковий оператор зі списку доступних операторів.
5. correct_answer = None: Ініціалізує змінну correct_answer перед початком умов перевірки оператора "^".
6. if operator == '+': Якщо оператор "+" - обчислює правильну відповідь як суму чисел num1 і num2.
7. Аналогічні перевірки для інших операторів (-, *, /), де правильна відповідь обчислюється відповідно до вибраного оператора.
8. elif operator == '^': Якщо оператор "^", генерує числа num1 і num2, а потім обчислює num1 у степені num2.
9. if correct_answer is not None: Перевіряє, чи було встановлено значення для correct_answer.
10. self.correct_answer = correct_answer: Зберігає правильну відповідь.
11. self.question_label.text = f'{num1} {operator} {num2} = ?': Встановлює текст питання на мітці, щоб показати вираз, який треба вирішити.
12. Генерується список варіантів відповідей, який включає правильну відповідь та дві неправильні. Ці неправильні відповіді випадковим чином

підбираються близькими до правильної відповіді числами, з уникненням дублювання.

13. random.shuffle(answers): Перемішує список варіантів відповідей, щоб розмістити їх у випадковому порядку.
14. Для кожного кнопки встановлюється текст варіанта відповіді, який відображатиметься на кнопках.
15. self.next_button.opacity = 0: Задає прозорість кнопці "Next question".
16. self.next_button.disabled = True: Вимикає кнопку "Next question", щоб вона не була доступною для натискання.
17. else: Викликає self.end_game(), якщо умова if не виконується, тобто гра закінчена.

Ця функція є одніє з головних так, як відповідає за генерацію питань та перевірку на правильність відповіді гравця.

```
def end_game(self):
    # При завершении игры меняем интерфейс на экран с результатами
    elapsed_time = datetime.now() - self.start_time
    minutes = int(elapsed_time.total_seconds() // 60)
    seconds = int(elapsed_time.total_seconds() % 60)
    time_text = f"Time left: {minutes:02d}:{seconds:02d}"

    self.layout.clear_widgets()
    if self.lives > 0:
        win_text1 = f"You win!"
        win_text2 = f"\nNumber of questions passed: {self.roundT -
1} / {self.roundF} \n{time_text}"
    else:
        win_text1 = f"You lose!"
        win_text2 = f"\nNumber of questions passed: {self.roundT -
1} / {self.roundF} \n{time_text}"

    rate_label = Label(text=win_text1,font_size='80sp',
halign='center')
    rate_label.font_name = 'Jacquard12-Regular.ttf'
    rate_label.size_hint_y = 0.7
    rate_label.size_hint_x = 0.7
    rate_label.pos_hint = {'center_x': 0.5, 'center_y': 0.5}
    rate_label.color = 'red'

    combined_label = Label(text=win_text2, font_size='50sp',
halign='center')
    combined_label.font_name = 'Jacquard12-Regular.ttf'
```

```
    combined_label.size_hint_y = 0.7
    combined_label.size_hint_x = 0.7
    combined_label.pos_hint = {'center_x': 0.5, 'center_y': 0.5}
    combined_label.color = 'yellow'
    self.layout.add_widget(rate_label)
    self.layout.add_widget(combined_label)

    play_again_button = Button(text='Play again', font_size='60sp',
on_press=self.reset_game)
    play_again_button.font_name = 'Honk-Regular-
VariableFont_MORF,SHLN.ttf'
    play_again_button.size_hint_y = 0.7
    play_again_button.size_hint_x = 0.7
    play_again_button.pos_hint = {'center_x': 0.5,'center_y': 0.5}
# Размещаем кнопку по центру родительского контейнера
    play_again_button.color = 'orange'
    play_again_button.background_color = (0,0,0,0)

    self.layout.add_widget(play_again_button)

    home_menu_button = Button(text='Home menu', font_size='60sp',
on_press=self.show_main_menu)
    home_menu_button.font_name = 'Honk-Regular-
VariableFont_MORF,SHLN.ttf'
    home_menu_button.size_hint_y = 0.7
    home_menu_button.size_hint_x = 0.7
    home_menu_button.pos_hint = {'center_x': 0.5,'center_y': 0.5}  #
Размещаем кнопку по центру родительского контейнера
    home_menu_button.color = 'orange'
    home_menu_button.background_color = (0, 0, 0, 0)
    self.layout.add_widget(home_menu_button)
```

Рис.3.7 – Метод end_game

Розглянемо функцію end_game, яка викликається при завершенні гри і змінює інтерфейс на екран з результатами гри. По більшій частині в цьому блоці описані встановлення розмірів та положення для кнопок та тексту, що виконуються за допомогою size_hint_y, size_hint_x, pos_hint, rate_label.color тому їх опис ми опустимо, а основні функції цього метода наступні:

1. Обчислює час, який пройшов з початку гри.
2. Перевіряє, чи є ще життя у гравця. Якщо так, відображається повідомлення "You win!". Якщо ні, відображається повідомлення "You lose!".
3. Встановлює текст результату гри та часу для двох міток з використанням обчислених значень.
4. Створює дві мітки з результатами гри та додає їх до макету.

5. Створює дві кнопки - "Play again" і "Home menu" - для відтворення гри знову або повернення до головного меню.
6. Додає ці кнопки до макету.

Ця функція відповідає за закінчення гри та виведення результатів гри в залежності від пройдення усіх рівнів або поразки.

```
def reset_game(self, instance):
    if self.difficulty == 'easy':
        self.roundF = 7
        self.lives = 4
    elif self.difficulty == 'normal':
        self.roundF = 15
        self.lives = 3
    elif self.difficulty == 'hard':
        self.roundF = 24
        self.lives = 2

    # Reset game variables
    self.roundT = 1
    self.layout.clear_widgets()

    # Rebuild the interface
    self.build_interface()
    self.generate_question()
    self.start_timer()
```

Рис.3.8 – Метод reset_game

Функція reset_game відновлює гру до початкового стану при натисканні на кнопку "Play again":

1. Перевіряє рівень складності гри (easy, normal, або hard) і встановлює відповідні значення для кількості раундів і кількості життів.
2. Скидає змінні гри, такі як кількість пройдених раундів і відображення на макеті.
3. Очищує макет, щоб видалити попередні результати гри.
4. Перебудовує інтерфейс гри за допомогою функції build_interface.
5. Генерує перше питання.
6. Запускає таймер для нової гри.

Функція reset_game відповідає за повторний початок гри при цьому зберігши рівень складності що був обраний на початку.

```
def build_interface(self):
    # Верхняя часть интерфейса с отображением текущего уровня и
таймера
    self.top_layout = BoxLayout(orientation='horizontal',
size_hint_y=None, height='50dp')
    self.level_label = Label(text=f"Rounds:1/{self.roundF}",
font_size='50sp')
    self.level_label.font_name = 'Honk-Regular-
VariableFont_MORF,SHLN.ttf'
    self.top_layout.add_widget(self.level_label)

    self.time_label = Label(text="Timer:00:00", font_size='57sp',
halign='right')
    self.time_label.font_name = 'Jacquard12-Regular.ttf'
    self.top_layout.add_widget(self.time_label)

    self.lives_label = Label(text=f"Lives: {self.lives}",
font_size='50sp')
    self.lives_label.font_name = 'Jaini-Regular.ttf'
    self.top_layout.add_widget(self.lives_label)

    self.layout.add_widget(self.top_layout)

    # Часть интерфейса с вопросом и кнопками
    self.question_label = Label(text='', font_size='60sp')
    self.question_label.font_name = 'PoetsenOne-Regular.ttf'
    self.layout.add_widget(self.question_label)

    self.buttons_layout = BoxLayout(orientation='horizontal')
    self.buttons_layout.size_hint_y = 0.7
    self.buttons_layout.size_hint_x = 0.7
    self.buttons_layout.pos_hint = {'center_x': 0.5, 'center_y':
0.5}

    self.buttons = []
    for _ in range(3):
        button = Button(font_size='40sp',
on_press=self.check_answer)
        button.color = '#3de048'
        button.font_name = 'PoetsenOne-Regular.ttf'
        button.background_color = '#c369cf'
        self.buttons.append(button)
        self.buttons_layout.add_widget(button)
    self.layout.add_widget(self.buttons_layout)

    # Нижняя часть интерфейса с кнопкой "Next question"
    self.next_button = Button(text='Next question',
font_size='50sp', on_press=self.next_question)
    self.next_button.size_hint_y = 0.5
    self.next_button.size_hint_x = 0.5
    self.next_button.pos_hint = {'center_x': 0.5, 'center_y': 0.5}
```

```
    self.next_button.height = '50dp'
    self.next_button.opacity = 0
    self.next_button.disabled = True
    self.next_button.background_color = (0, 0, 0, 0)
    self.next_button.color = 'orange'
    self.next_button.font_name = 'Honk-Regular-
VariableFont_MORF,SHLN.ttf'
    self.layout.add_widget(self.next_button)
```

Рис.3.9 – Метод build_interface

Цей метод build_interface створює інтерфейс гри, який містить:

1. Верхню частину інтерфейсу: ця частина містить відображення поточного раунду, таймера та кількості життів гравця.

- top_layout: Вертикальний контейнер, що містить рівень, таймер та кількість життів.
- level_label: Мітка, що відображає поточний рівень гри.
- time_label: Мітка, що відображає таймер.
- lives_label: Мітка, що відображає кількість життів гравця.

2. Частину з питанням та кнопками: ця частина містить питання та кнопки з варіантами відповідей.

- question_label: Мітка, що відображає питання.
- buttons_layout: Вертикальний контейнер для розташування кнопок з варіантами відповідей.
- buttons: Список кнопок з варіантами відповідей.

3. Нижню частину інтерфейсу з кнопкою "Next question": ця частина містить кнопку "Next question" для переходу до наступного питання.

- next_button: Кнопка "Next question", яка викликає метод next_question при натисканні.

Цей метод відповідає за будову інтерфейсу гри, а саме кнопок та текстів раундів часу, життів.

```
def check_answer(self, instance):
    if instance.text == str(self.correct_answer):
        self.question_label.text = "Your answer is correct! Good
job!"
        self.question_label.font_size = 60
        self.next_button.opacity = 1
        self.next_button.disabled = False
```

```
    else:
        self.lives -= 1
        self.lives_label.text = f"Lives: {self.lives}"
        self.question_label.text = "Incorrect answer! Try better
next time!"
        self.question_label.font_size = 60

        if self.lives == 0:
            self.end_game()
```

Рис.3.10 – Метод check_answer

У цьому методі check_answer перевіряється відповідь гравця на питання:

1. Перевіряється, чи відповідь гравця співпадає з правильною відповіддю correct_answer. Якщо так, відображається повідомлення про правильну відповідь, і кнопка "Next question" стає видимою і активною.
2. Якщо відповідь гравця не є правильною:

- Зменшується кількість життів гравця lives на одиницю.
- Відображається повідомлення про неправильну відповідь.
- Якщо кількість життів гравця досягла нуля, викликається метод end_game(), щоб завершити гру.

Метод відповідає за перевірку правильності відповіді та віднімання життів при не вірному виборі.

```
def next_question(self, instance):
    self.roundT += 1  # Увеличиваем количество пройденных уровней
    self.level_label.text = f"Rounds:{self.roundT}/{self.roundF}"  #
Обновляем отображение текущего уровня
    self.generate_question()
```

Рис.3.11 – Метод next_question

У методі next_question відбувається підготовка до наступного питання гри:

1. Збільшується лічильник пройдених раундів roundT на одиницю.
2. Оновлюється текст мітки level_label, що відображає поточний рівень гри, вказуючи на кількість пройдених раундів і загальну кількість раундів.
3. Викликається метод generate_question(), який генерує нове питання для гравця.

Цей метод відповідає за перехід до іншого питання у разі правильної відповіді.

```
if __name__ == '__main__':
    MathGame().run()
```

Рис.3.12 – Ініціалізація гри

Ця конструкція умови перевіряє, чи файл був запущений безпосередньо як виконуваний, а не імпортований як модуль у інший файл. Якщо це так, то створюється і запускається екземпляр класу MathGame, який є підкласом App з бібліотеки Kivy. Виклик методу .run() запускає головний цикл програми, який обробляє події, відображає вікна та виконує різні функції гри до тих пір, поки програма не буде закрита.

Таким чином ми маємо готовий застосунок з використанням мови програмування Python та використавши в основному бібліотеку Kivy та деякі її можливості в розробці ігор та приємного графічного інтерфейсу.

3.2 Недоліки які виникли під час розробки застосунку

Під час розробки застосунку на мові програмування мною були виявлені наступні недоліки:

1. Продуктивність:

- Python є інтерпретованою мовою, що може призвести до нижчої продуктивності порівняно з компільованими мовами, такими як C++ чи Java. Це особливо важливо для ігор, де висока продуктивність є критичною для плавного ігрового процесу.

2. Обмеженість графічних можливостей:

- Kivy, хоч і потужна бібліотека для створення графічних інтерфейсів на Python, не є настільки розвиненою та оптимізованою для створення складних ігор, як інші ігрові рушії (Unity, Unreal Engine).

3. Бібліотеки та інструменти:

- Обмежена кількість доступних бібліотек та інструментів для розробки ігор на Python порівняно з іншими мовами програмування, що можуть ускладнити реалізацію певних функцій.

Чому Python не підходить для повноцінної розробки ігор:

1. Продуктивність:

- Як вже зазначено, Python є інтерпретованою мовою, що призводить до меншої швидкості виконання коду. Ігри часто потребують реального часу обробки для графіки, фізики та інших обчислювальних завдань, де Python може не впоратись з необхідною швидкістю.

2. Відсутність розвинених інструментів:

- Хоча є декілька бібліотек для розробки ігор (наприклад, Pygame, Kivy), вони не можуть конкурувати з розвинутими рушіями, такими як Unity чи Unreal Engine, які пропонують широкий набір інструментів, підтримку 3D графіки та інтеграцію з різними платформами.

3. Обмежена підтримка багатопотоковості:

- Python має GIL (Global Interpreter Lock), який може бути обмежуючим фактором при спробі реалізувати багатопотокову обробку, що важливо для складних ігор.

Складнощі при компіляції у формат .apk через buildozer:

1. Налаштування середовища:

- Налаштування buildozer та його залежностей може бути складним і часозатратним процесом. Неправильні версії інструментів, відсутні бібліотеки або конфлікти версій можуть викликати помилки.

2. Відсутність документації та підтримки:

- Хоча buildozer має базову документацію, вона може бути недостатньо деталізованою для вирішення специфічних проблем. Це може призвести до труднощів у пошуку рішення при виникненні помилок під час компіляції.

3. Залежності:

- Buildozer залежить від багатьох зовнішніх інструментів та бібліотек (таких як SDK Android, NDK, Python-for-android). Неправильне або неповне налаштування цих компонентів може призвести до помилок компіляції.

4. Проблеми з кросплатформенністю:

- Розробка і компіляція для різних платформ (особливо для мобільних пристроїв) може бути складною через різницю в архітектурах та вимогах до виконуваних файлів.

5. Відлагодження:

- Виявлення та виправлення помилок у процесі компіляції та при запуску готового .apk файлу може бути складним через відсутність детальних повідомлень про помилки та обмежену можливість відлагодження.

Ці недоліки і складнощі можуть зробити розробку повноцінних ігор на Python досить викликовим завданням, особливо для великих та продуктивно-інтенсивних проектів.

ВИСНОВКИ

Під час виконання роботи я дослідив напрямок розробок ігор та додатків на базі ігрових механік з використанням мови програмування Python.

Було порівняно та проведено аналіз на придатність цієї мови, її бібліотек, методик використання, ігрових двигунів, створення графіки, із більш відомими для використання у сфері Game Development мовами програмування такими як C++, C# та Java та було виявлено наступні недоліки:

- Низька продуктивність. Python є інтерпретованою мовою, що значно впливає на швидкість виконання коду. У розробці ігор, де важлива висока продуктивність та швидка обробка графіки і фізики, це стає суттєвим недоліком. Мови, такі як C++ та C#, надають кращу продуктивність завдяки компіляції коду перед виконанням.
- Обмеженість бібліотек. Хоча для Python існують бібліотеки для розробки ігор, такі як Pygame та Kivy, вони не можуть конкурувати з більш потужними ігровими двигунами, такими як Unity (C#) або Unreal Engine (C++). Ці двигуни надають розширені можливості для створення складної графіки та фізики, чого часто не вистачає бібліотекам Python.
- Відсутність підтримки 3D-графіки. Розробка 3D-ігор на Python є складним завданням через обмежену підтримку 3D-графіки. Інші мови програмування мають краще розвинуті інструменти та середовища для роботи з 3D-графікою.
- Складність компіляції в .apk. Процес компіляції Python-застосунків у формат .apk для Android є складним та нестабільним. Використання таких інструментів, як Buildozer, часто супроводжується численними проблемами, що ускладнює розробку та розгортання ігор.
- Недостатня документація та підтримка. Багато популярних ігрових двигунів та інструментів мають обширну документацію і активні спільноти, що допомагають розробникам. У випадку з Python, документація та підтримка для ігрових бібліотек часто є обмеженими, що ускладнює навчання та використання.

Мною було розроблено мобільний застосунок на базі ігрових механік для підвищення ефективності швидкодії розрахунків з використанням мови Python, її біліотек та проведено детальний опис на етапі розробки та демонстрацію кінцевого вигляду програми.

Хоча Python є чудовою мовою для швидкого прототипування та створення простих ігор, його обмеження у продуктивності, підтримці графіки та складності компіляції роблять його менш придатним для повноцінної розробки ігор порівняно з іншими мовами програмування, такими як C++, C# та Java. Для створення комплексних ігор з високими вимогами до графіки та продуктивності, доцільніше використовувати мови та інструменти, спеціально розроблені для цієї мети, а саме:

C++ та Unreal Engine забезпечують розробників потужними інструментами та ресурсами для створення високоякісних AAA-ігор. Висока продуктивність, гнучкість, потужні графічні можливості та розвинена екосистема роблять цей вибір оптимальним для розробників, що прагнуть створювати складні та реалістичні ігрові проекти. Завдяки цьому, C++ та Unreal Engine стали стандартом у індустрії AAA-ігор, забезпечуючи гравців незабутніми враженнями та вражаючою графікою.

C# та Unity є потужними інструментами для розробки високоякісних A+ (тобто високоякісних ігор, але не на рівні AAA) ігор. Простота використання C#, його інтеграція з Unity, а також широкі можливості та кросплатформеність Unity роблять їх ідеальним вибором для розробників. Завдяки цим інструментам, розробники можуть створювати складні та захоплюючі ігри з високою продуктивністю та чудовою графікою, що забезпечує їм успіх на ринку ігор.

Апробація:

1. V Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку Iot» ,18 квітня 2024 року, ДУІКТ – «Інноваційні методики розробки мобільних додатків з ігровими механіками на python для підвищення ефективності навчання», «Дослідження розробки додатків на базі ігрових механік з використанням мови програмування python»

2. IV Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 року, ДУІКТ -«Інноваційні методики розробки мобільних додатків з ігровими механіками на python для підвищення ефективності навчання», «Дослідження розробки додатків на базі ігрових механік з використанням мови програмування python»

ПЕРЕЛІК ПОСИЛАНЬ

1. Історичний огляд розвитку мов програмування. *Промінчики віртуальності*. URL: https://promichikivirt.blogspot.com/2017/12/blog-post_7.html (дата звернення: 01.05.2024).
2. Мови програмування, на яких написані популярні комп'ютерні ігри. *Комп'ютерна Академія ШАГ | Кам'янське*. URL: https://kam.itstep.org/blog_3/programming-languages-in-which-popular-computer-games-are-written (дата звернення: 19.05.2024).
3. C++Developer: понятие, особенности профессии, важность в GameDev • VOKI Games. *Voki Games*. URL: https://vokigames.com/ru/c-developer-ponyatie-osobennosti-professii-vazhnost-v-gamedev/ (дата звернення: 02.05.2024).
4. Введення в Java FX. *JavaRush*. URL: https://javarush.com/ua/groups/posts/uk.2560.vvedennja-v-java-fx (date of access: 08.05.2024).
5. Kushnirenko S. Каков C++ в gamedev'е?. *Хабр*. URL: https://habr.com/ru/articles/795869/ (дата звернення: 02.05.2024).
6. OhNxuB. Кто такой C++ разработчик и как с помощью этого языка создаются GTA, The Witcher, StarCraft, TES. *dev.ua*. URL: https://dev.ua/ru/news/s-1667241787 (дата звернення: 02.05.2024).
7. Учасники проектів Вікімедіа. Pygame – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Pygame (дата звернення: 19.05.2024).
8. Професія аніматора в ігровій індустрії: хто створює анімації в іграх. *Українська Ідентичність*. URL: https://viter.zapisi.cx.ua/ukraincyam/profesiya-animatora-v-igroviy-industrii-khto-stvoryuie-animacii-v-igrakh.html (дата звернення: 19.05.2024).
9. Що таке Python? - aCode. *aCode*. URL: https://acode.com.ua/intro-python/ (дата звернення: 19.05.2024).
10. URL: https://kk.nau.edu.ua/article/4182 (дата звернення: 19.05.2024).

11. URL:https://library.econom.zp.ua:85/xmlui/bitstream/handle/123456789/27/2022_Semenyuk_IPZ_118k9.pdf?sequence=1&isAllowed=y (дата звернення: 19.05.2024).
12. Зображення C++. URL: https://dzen.ru/a/ZK_Phtp0pmJSB8cb (дата звернення: 14.05.2024).
13. Зображення C#. URL: https://rat.in.ua/shop/c/ (дата звернення: 01.05.2024).
14. Зображення Kivy. URL: https://www.publish0x.com/python-kivy-basics (дата звернення: 02.05.2024).
15. Зображення Python. URL: https://dzen.ru/a/X_WIGbsU1U_7jxvg (дата звернення: 06.05.2024).
16. Перевод визуальных новелл на Unity. *Anivisual.net – визуальные новеллы, игры и их переводы – Визуальные новеллы на русском языке*. URL: https://anivisual.net/blog/2023-01-23-844 (дата звернення: 08.05.2024).
17. Contributors to Python вики. Pygame. *Python вики*. URL: https://python.fandom.com/ru/wiki/Pygame (date of access: 08.05.2024).
18. JavaFX Build software better, together. *GitHub*. URL: https://github.com/topics/obfuscation?o=asc&s=stars (date of access: 01.05.2024).
19. Java TimeZone TimeZone CodeDSlab | DSlab. *DSlab*. URL: https://dslab.us/tag/java-timezone-timezone-code/ (date of access: 01.05.2024).
20. Unreal Engine – Siluette. *Siluette – Онлайн видео курсы на любые темы*. URL: https://siluette.com.ua/ru/product/unreal-engine/ (дата звернення: 15.05.2024).