Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему : «Розробка крос-платформеного додатку для підтримки прийняття рішень при управлінні особистими фінансами».

Виконав: студент групи ІПЗ20-1
Спеціальність 121 «Інженерія програмного забезпечення»
Солодовніков Олександр Юрійович
(прізвище та ініціали)
Керівник к.т.н., доцент Ульяновська Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент Університет митної справи та фінансів
(місце роботи)
в.о. завідувача кафедри кібербезпеки та інформаційних технологій
(посада)
к.т.н., доцент кафедри кібербезпеки та інформаційних технологій Прокопович-Ткаченко Д.І.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Солодовніков О. Ю. Розробка крос-платформеного додатку для підтримки прийняття рішень при управлінні особистими фінансами.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 121 «Інженерія програмного забезпечення» – Університет митної справи та фінансів, Дніпро, 2024.

Дана кваліфікаційна робота присвячена розробці крос-платформного додатку для підтримки прийняття рішень при управлінні особистими фінансами. Управління особистими фінансами є важливою проблемою в сучасних умовах економічної нестабільності, що викликає потребу у використанні інноваційних підходів та інструментів для ефективного контролю фінансових потоків.

Розробка додатку проводилась з використанням сучасних технологій програмування, таких як C# та Unity. Створений додаток забезпечує автоматизацію управління фінансами, надаючи користувачам можливість аналізувати витрати, планувати бюджет, а також отримувати персоналізовані рекомендації для ухвалення фінансових рішень. В додатку реалізовані методи підтримки прийняття рішень з використанням матриці Ейзенхауера та технологій експертних систем.

Використання технологій Unity та C# робить розроблене програмне забезпечення крос-платформним, що надає можливість використання як на desktop-платформі так і на мобільних пристроях.

Розроблене програмне забезпечення дозволяє користувачам зменшити фінансові ризики, поліпшити контроль за витратами, а також підвищувати загальну фінансову грамотність шляхом надання рекомендацій.

Ключові слова: управління фінансами, крос-платформний додаток, C#, Unity, матриця Ейзенхауера, експертні системи.

ABSTRACT

Solodovnikov O. Yu. Development of a Cross-Platform Application for Decision Support in Personal Finance Management.

Bachelor's thesis for obtaining a degree in Software Engineering, specialty 121. – University of Customs and Finance, Dnipro, 2024.

This bachelor's qualification work is dedicated to the development of a cross-platform application for decision support in personal finance management. Managing personal finances is a crucial issue in today's economically unstable conditions, necessitating the use of innovative approaches and tools for effective financial flow control.

The application development was carried out using modern programming technologies such as C# and Unity. The created application provides automation of financial management, allowing users to analyze expenses, plan budgets, and receive personalized recommendations for making financial decisions. The application implements decision-making support methods using the Eisenhower matrix and expert system technologies.

The use of Unity and C# technologies makes the developed software cross-platform, which provides the possibility of use both on the desktop platform and on mobile devices.

The developed software allows users to reduce financial risks, improve spending control, and increase overall financial literacy by providing recommendations.

Keywords: financial management, cross-platform application, C#, Unity, Eisenhower matrix, expert systems.

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі управління особистими фінансами набуває особливого значення через низку об'єктивних чинників, що підвищують необхідність розробки інноваційних підходів і інструментів для ефективного контролю фінансових потоків. Сучасна економічна ситуація характеризується значною нестабільністю, зокрема, інфляційними процесами, коливаннями валютних курсів та фінансовими кризами. В умовах економічної невизначеності населення стикається з необхідністю більш ретельно планувати свої фінансові операції та забезпечувати фінансову безпеку. Відсутність ефективних інструментів для управління особистими фінансами може призвести до фінансових втрат та зниження життєвого рівня громадян.

Швидкий розвиток цифрових технологій відкриває нові можливості для створення мобільних додатків, які дозволяють користувачам зручно та ефективно керувати своїми фінансами в режимі реального часу. Сучасні крос-платформні додатки забезпечують доступ до фінансової інформації з будь-якого пристрою, що значно підвищує їхню привабливість та функціональність. У зв'язку з цим, зростаюча популярність мобільних пристроїв серед населення створює потребу у розробці мобільних додатків, які забезпечують високу зручність використання та швидкий доступ до фінансових даних, що сприяє ефективному управлінню особистими фінансами.

Крім того, автоматизація фінансових процесів стає все більш важливою, оскільки багато людей стикаються з труднощами у веденні детального обліку доходів і витрат через відсутність часу або відповідних знань. Автоматизація цих процесів за допомогою спеціалізованих додатків дозволяє зменшити людський фактор, підвищити точність обліку та знизити ризики помилок, що робить управління фінансами більш ефективним та надійним.

Мобільні додатки можуть виконувати освітню функцію, надаючи користувачам інформацію про основи фінансового планування, інвестування та управління грошима. Це сприяє підвищенню фінансової обізнаності та допомагає

користувачам ухвалювати більш обґрунтовані фінансові рішення, що, в свою чергу, покращує їх фінансовий стан. Сучасні технології дозволяють створювати додатки, які враховують індивідуальні потреби та особливості користувачів, надаючи персоналізовані рекомендації та налаштування. Це робить управління фінансами більш зручним і ефективним, допомагаючи користувачам досягати своїх фінансових цілей.

Важливу роль при управлінні особистими фінансами має аспект аналізу та контролю витрат. При цьому позитивним буде наявність можливість надання рекомендацій щодо витрат.

У цьому аспекті доцільним є застосування технологій аналізу даних з використанням методів підтримки прийняття рішень, зокрема матриці Ейзенхаура, та технологій експертних систем. Матриця Ейзенхауера дозволяє швидко та ефективно розібратися зі списком справ, розподіливши ці справи за категоріями. У результаті людина ясно бачить усі найважливіші справи, а також ті справи, які взагалі не варті його уваги.

Таким чином тема кваліфікаційної роботи «Розробка крос-платформеного додатку для підтримки прийняття рішень при управлінні особистими фінансами» є надзвичайно актуальною задачею, яка відповідає сучасним викликам та потребам суспільства.

Метою роботи є автоматизація управління та контролю за особистими фінансами шляхом розробки крос-платформного застосунку.

Завданнями кваліфікаційної роботи є:

1) дослідити основні методи розробки крос-платформних додатків;

2) проаналізувати основні принципи роботи експертних систем;

3) дослідити методи виявлення закономірностей у наявних даних;

4) проаналізувати вимоги до програмного застосунку для управління фінансами;

5) спроектувати архітектуру програмного застосунку;

6) розробити програмний додаток;

7) провести тестування та налагодження програмного застосунку для забезпечення його стабільної роботи;

8) провести оцінку ефективності та зручності використання додатку через проведення користувацького тестування.

Методи та технології дослідження – методи аналізу, синтезу, узагальнення, методи аналізу даних, методи теорії підтримки прийняття рішень, технології розробки крос-платформених додатків.

Об'єктом дослідження – розробка програмного забезпечення для підтримки прийняття рішень для управління особистими фінансами.

Предметом дослідження є розробка крос-платформних додатків.

Структура роботи - робота складається з вступу, трьох розділів, списку використаних джерел з 15 найменувань, 47 рисунків

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА МЕТОДИ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

1.1. Аналіз існуючих програмних застосунків для управління та контролю за фінансами

Сучасні програмні застосунки для управління та контролю власних фінансів здебільшого орієнтовані на крос-платформність. Найпопулярнішими серед них є ті, що забезпечують своїм користувачам можливість працювати на різних пристроях.

Наприклад, застосунок Personal Capital (рис. 1.1) підтримує функціональність на телефонах, планшетах та персональних комп'ютерах [1].

Рисунок 1.1 – Приклад застосунку для управління та контролю за фінансами Personal Capital

Personal Capital – це застосунок для управління інвестиціями та фінансами, який дозволяє відстежувати інвестиційний портфель, планувати заощадження, контролювати витрати та доходи, а також генерувати звіти.

Іншим схожим програмним застосунком є Quicken (рис. 1.2), який є одним із найстаріших додатків для фінансового управління. Quicken дозволяє

відстежувати витрати, планувати бюджет, керувати інвестиціями, створювати звіти та багато іншого [2].

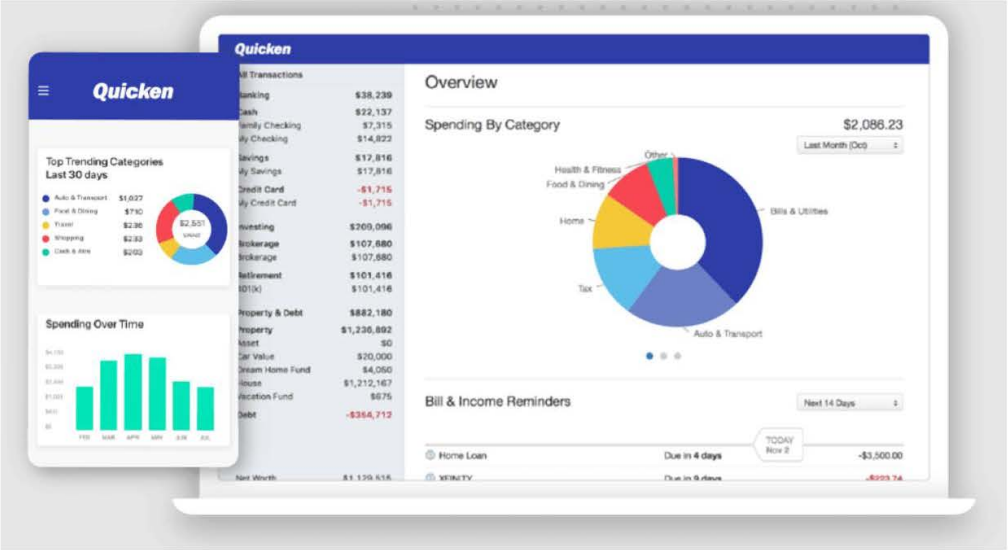

Рисунок 1.2 – Приклад застосунку для фінансового управління Quicken

PocketGuard (рис. 1.3) допомагає відстежувати витрати та доходи, створювати бюджети та контролювати свій фінансовий стан. Цей застосунок автоматично синхронізується з банківськими рахунками та надає інформацію в режимі реального часу про доступні кошти та поточні витрати. [3].

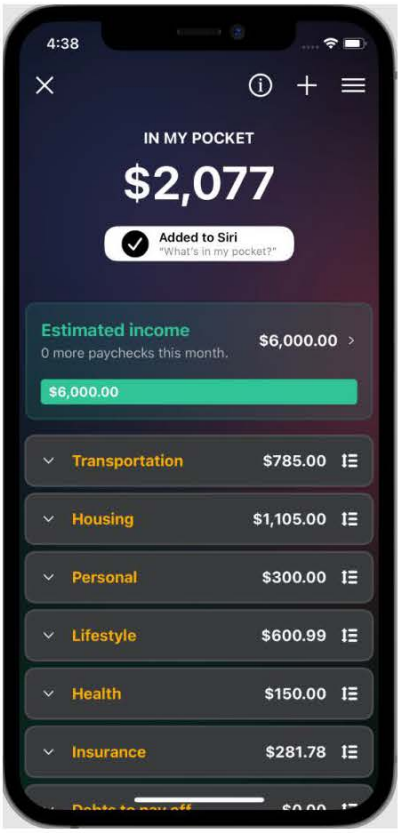

Рисунок 1.3 – Приклад мобільного застосунку для відслідковування витрат та доходів PocketGuard

1.2 Аналіз технологій та засобів розроблення крос-платформних застосунків

Для створення крос-платформних мобільних застосунків можна використовувати різноманітні технології та інструменти, які забезпечують ефективну роботу одного коду на кількох операційних системах, зокрема на Android та iOS.

Одним з варіантів є використання мови програмування Kotlin, яка є однією з офіційних мов для розробки Android-додатків і водночас підтримується для крос-платформної розробки через інструменти, як-от JetBrains Compose Multiplatform. Kotlin поєднує в собі переваги сучасної мови з потужними функціональними можливостями.

Для крос-платформної розробки також широко використовуються фреймворки, такі як React Native, Xamarin та Flutter. Ці фреймворки дозволяють розробникам писати код один раз, а потім запускати його на різних платформах,

що значно скорочує час розробки та полегшує синхронізацію функціональності між різними системами.

- Xamarin використовує мову C# і .NET фреймворк, дозволяючи розробникам використовувати єдину кодову базу для створення додатків для Android, iOS та Windows. Xamarin надає повний доступ до нативних API платформ, що забезпечує високу продуктивність та гнучкість у розробці.
- Flutter, розроблений Google, використовує мову Dart і надає високу продуктивність та гнучкість для створення графічно привабливих, багатофункціональних додатків за допомогою власного двигуна відмальовування.
- React Native дозволяє розробникам використовувати JavaScript для створення нативних додатків для Android та iOS, що робить його ідеальним вибором для тих, хто вже знайомий з веб-розробкою.

Сучасні ігрові рушії давно вийшли за межі ігрової індустрії. Використання їх також можливе в залежності від виду задач. Однис з таких рушіїв є Unity, що являється сучасним і потужним ігровим рушієм. Його перевага в тому, що він надає інструменти для розробки інтерактивних та графічно багатих крос-платформних додатків. Unity підтримує розробку для Android, iOS, Windows, macOS та інших платформ, що робить його універсальним вибором для створення ігор та інтерактивних застосунків.

Крім того, Android SDK та різноманітні сервіси Google, як-от Firebase, Google Maps API та Google Pay, забезпечують потужний набір інструментів для розширення функціональності мобільних додатків.

Матеріальний дизайн від Google продовжує бути актуальним у створенні консистентного і сучасного користувацького досвіду на всіх платформах, підтримуючи візуальну однорідність між різними пристроями та платформами.

Вибір певної технології для крос-платформної розробки залежить від потреб проєкту, вимог до продуктивності та особистих переваг розробника. Головною перевагою такого підходу є можливість створення якісних і сучасних застосунків, які можуть працювати на різних платформах, з мінімальними зусиллями та витратами часу.

1.3. Висновки до першого розділу. Постановка задач дослідження.

Аналіз існуючих програмних рішень для управління особистими фінансами показав, що на ринку існує велика кількість додатків, які забезпечують користувачам зручні інструменти для контролю за своїми фінансами. Основними характеристиками таких додатків є крос-платформність, можливість інтеграції з різними пристроями та доступ до фінансової інформації в режимі реального часу. Програмні продукти, такі як Personal Capital, Quicken та PocketGuard надають користувачам широкий спектр функцій для відстеження доходів та витрат, планування бюджету та управління інвестиціями.

Сучасні технології для розробки крос-платформних застосунків, такі як React Native, Xamarin, Flutter та Unity, дозволяють створювати ефективні та функціональні додатки, які можуть працювати на різних операційних системах. Це значно скорочує час та витрати на розробку, а також забезпечує консистентність та зручність у використанні.

Таким чином тема кваліфікаційної роботи «Розробка крос-платформеного додатку для підтримки прийняття рішень при управлінні особистими фінансами»

Метою роботи є автоматизація управління та контролю за особистими фінансами шляхом розробки крос-платформного застосунку.

Завданнями кваліфікаційної роботи є:

1) дослідити основні методи розробки крос-платформних додатків;

2) проаналізувати вимоги до програмного застосунку для управління фінансами;

3) спроектувати архітектуру програмного застосунку;

4) розробити програмний додаток;

5) провести тестування та налагодження програмного застосунку для забезпечення його стабільної роботи;

6) провести оцінку ефективності та зручності використання додатку через проведення користувацького тестування.

РОЗДІЛ 2. АНАЛІЗ ПІДХОДІВ ТА МЕТОДІВ РОЗРОБКИ БАЗИ ЗНАНЬ

2.1. Загальна характеристика експертних систем

У середині 70-х років XX століття з'явилися перші експертні системи як галузь прикладного штучного інтелекту. Вони були розроблені для вирішення проблем у таких сферах, як медицина та бухгалтерський облік. У 1974 році була створена експертна система MYCIN, яка допомагала діагностувати інфекційні захворювання крові та рекомендувати антибіотики для лікування пацієнтів. У 1979 році з'явилася експертна система DENDRAL, призначена для хімічних досліджень та ідентифікації молекул.

У 1980-х роках експертні системи стали застосовуватися в інших галузях, зокрема у виробництві, фінансах та праві. У 1985 році була створена експертна система XCON, яка використовувалася в автомобільній промисловості для автоматизації виробничих процесів та підвищення ефективності.

У 1990-х роках, завдяки появі комп'ютерів з великою обчислювальною потужністю та зростанню обсягу доступної інформації, експертні системи отримали широке застосування в різних сферах, включаючи інтернет-технології та медіа-індустрію.

Основною метою створення експертних систем було заміщення експерта у певній галузі комп'ютером. Знання експертів збиралися, структурувалися та передавалися комп'ютеру, який зберігав їх та робив логічні висновки, консультував користувачів у відповідних питаннях.

Експертна система – це комп'ютерна програма, здатна частково замінити спеціаліста-експерта у вирішенні проблемної ситуації. Вона базується на знаннях, отриманих від людей-експертів та/або з електронних джерел інформації.

Основна мета експертних систем - надати користувачам інструмент для прийняття рішень на основі зібраних даних та знань у визначеній галузі. Експертні системи можуть бути застосовані у різних сферах, таких як медицина, фінанси,

техніка, наука тощо. Вони виконують консультаційні функції, здійснюють аналіз, надають рекомендації та навіть можуть ставити діагнози.

Зазвичай експертні системи складаються з таких компонентів:

1) База знань – основний компонент експертної системи, в якому зберігається інформація, необхідна для її роботи. Вона містить набір правил, фактів, процедур, принципів, евристик та інших елементів, які описують експертну інформацію у певній галузі. Головною метою бази знань є забезпечення експертної системи всією необхідною інформацією для вирішення конкретних завдань або проблем у визначеній галузі. Для створення бази знань використовуються різні методи: аналіз документації та експертних звітів, інтерв'ю з експертами, опитування користувачів, аналіз існуючих баз знань, застосування машинного навчання та обробки природних мов.

2) Машина виведення (розв'язувач) – це програмний модуль, що використовує інформацію з бази знань для прийняття рішень та висновків на основі наявних знань. Машина виведення складається з двох компонентів: компонента висновку та інтерпретатора правил.

Розв'язувач працює за таким алгоритмом:

1) Отримання вхідних даних від користувача.

2) Перетворення вхідних даних у необхідний формат, зрозумілий експертній системі.

3) Пошук відповідної інформації в базі знань.

4) Використання знайденої інформації для прийняття рішень та висновків.

5) Надання результатів користувачеві у вигляді порад, рекомендацій чи висновків.

Машина виведення є потужним інструментом для прийняття рішень, оскільки здатна аналізувати велику кількість інформації та знань, які людині важко обробити самостійно. Вона може бути запрограмована для вирішення різноманітних завдань, таких як діагностика медичних захворювань, визначення оптимальних стратегій у іграх, аналіз фінансових даних та багато інших. Для

цього їй необхідна відповідна база знань та правила, які дозволяють їй аналізувати та обробляти вхідні дані.

Однією з переваг машини виведення є здатність працювати з інформацією в режимі реального часу та робити висновки на основі найновіших даних. Крім того, всі висновки, зроблені машиною виведення, є об'єктивними, оскільки вони базуються на чітких правилах та знаннях, без особистих думок та інтуїції.

Однак, машина виведення має свої обмеження. Якщо в базі знань недостатньо інформації або вхідні дані не відповідають жодному правилу, машина виведення може надати хибні або некоректні висновки. Крім того, якщо база знань не оновлюється регулярно та не підтримується, машина виведення може застаріти та не виконувати свої функції належним чином.

Механізм пояснення – це компонент експертної системи, який дозволяє системі пояснювати свої рішення користувачам. Він є важливим, оскільки допомагає зрозуміти "хід думок" експертної системи та її висновки. Механізм пояснення відображає правила, використані для прийняття конкретного рішення, і інколи показує важливість цих правил у даному контексті. Наприклад, експертна система для діагностики медичних захворювань може пояснити, які симптоми та показники були враховані для встановлення діагнозу, і які з них були найважливішими.

Механізм пояснення корисний не лише для розуміння причин прийняття рішення, але й для виявлення даних, які слід змінити для отримання іншого результату. Однією з переваг цього механізму є збільшення довіри користувачів до системи, оскільки вони можуть бачити процес ухвалення рішень. Пояснення також важливі для експертів, які звертаються до системи з складними питаннями або проблемами. Побачивши, які правила та знання були використані, експерти зможуть краще зрозуміти проблему та підвищити свої знання.

Однак механізм пояснення має свої обмеження. Якщо правила є складними або незрозумілими для користувачів, пояснення можуть стати неефективними. Крім того, якщо в базі знань недостатньо правил або вони неточні, механізм пояснення може надати недостовірну інформацію.

Інтерфейс користувача – це компонент, за допомогою якого користувач взаємодіє з експертною системою. Інтерфейси можуть бути графічними або текстовими, але обов'язково мають бути зрозумілими та зручними у використанні. Для введення та виведення даних можуть використовуватися текстові поля, кнопки, меню, чекбокси, радіокнопки, випадаючі списки, графічні елементи тощо. Для кращого розуміння висновків експертної системи можна використовувати графіки, діаграми, схеми, таблиці та інші візуальні елементи.

Особливим типом інтерфейсу користувача є діалогова система, яка дозволяє користувачам взаємодіяти з експертною системою за допомогою мовлення. Такі системи включають голосове управління та розпізнавання мовлення, що робить їх зручними для ситуацій, коли потрібно швидко та легко отримувати доступ до інформації.

При розробці інтерфейсу користувача для експертних систем особливу увагу слід приділяти його зручності та зрозумілості для користувачів. Інтерфейс повинен бути легким у використанні та надавати необхідну інформацію у зручному та доступному вигляді [11].

Модуль навчання – це компонент, який присутній у деяких експертних системах і дозволяє їм вдосконалювати свою базу знань на основі нової інформації або набутого досвіду.

Система моніторингу та контролю – це компонент, відповідальний за перевірку ефективності та надійності рішень, що приймаються експертною системою. Якщо система виявляє низьку ефективність або ненадійність результатів, вона може автоматично коригувати свої алгоритми та базу знань для підвищення продуктивності.

Ці компоненти разом забезпечують ефективне використання знань та експертної інформації для прийняття рішень у певній галузі.

Використання експертних систем може значно скоротити час і витрати, зменшити витрати на збір та аналіз інформації, підвищити точність та ефективність процесів прийняття рішень, а також покращити якість продукції та послуг.

Для створення експертних систем залучають програмістів, які розробляють необхідні компоненти, та експертів у відповідній галузі, які наповнюють базу знань. Знання експертів структуруються та вносяться до бази знань.

База знань складається з правил для аналізу інформації, отриманої від користувача щодо конкретної проблеми. Експертна система аналізує ситуацію та, залежно від своєї спеціалізації, надає рекомендації щодо вирішення проблеми.

Етапи розробки експертних систем:

1) Ідентифікація проблеми – на цьому етапі визначаються ключові завдання, що потребують рішення, формулюються цілі проекту, вибираються експерти та типи користувачів.

2) Витягання знань – проводиться аналіз області проблеми, ідентифікуються основні поняття та їхні взаємозв'язки, визначаються методи рішення задач.

3) Структуризація знань – вибираються інформаційні системи та методи подання знань, формалізуються ключові поняття, моделюється робота системи та оцінюється її відповідність поставленим цілям.

4) Формалізація знань – на цьому ключовому і трудомісткому етапі здійснюється наповнення бази знань, яка включає отримання знань від експерта, їх організацію для ефективної роботи системи, та представлення знань у форматі, доступному для експертної системи.

5) Реалізація – створення одного або декількох прототипів експертної системи, які вирішують поставлені задачі.

6) Тестування – перевірка вибраних методів представлення знань і оцінка ефективності системи в цілому.

Характеристики експертних систем:

1) Імітація експертного вирішення проблем – експертні системи повторюють механізм вирішення проблеми, як це зробив би людина-експерт, передаючи свої міркування та потенційні рішення системі.

2) Судження та висновки – система базується на знаннях з бази даних та робить висновки, використовуючи доступну інформацію.

3) Практичне спрямування – експертні системи зосереджені на реальних предметах і ситуаціях, використовуючи знання та досвід, які мають практичне значення в їх області.

4) Швидкість і надійність – на відміну від звичайних програм штучного інтелекту, експертні системи працюють швидше і забезпечують надійність рішень, знаходячи рішення, яке не гірше, ніж би прийняв експерт.

5) Здатність пояснювати рішення – експертна система повинна мати можливість пояснити свої висновки, показати хід рішення та обґрунтувати його правдивість. Це дозволяє користувачам зрозуміти процес ухвалення рішення та підвищує довіру до системи.

Етапи розробки експертних систем та їх характеристики показують, як ці системи можуть ефективно імітувати людську експертизу в різних галузях, забезпечуючи швидкі, надійні та обґрунтовані рішення. Використання таких систем може значно підвищити ефективність і точність процесів прийняття рішень, а також знизити витрати на збір і аналіз інформації, що робить їх цінним інструментом у сучасному світі.

2.2 Переваги та недоліки експертних систем

Переваги експертних систем:

- Доступність

Використання експертної системи можливе на будь-якому комп'ютері, який відповідає встановленим вимогам, що дозволяє широкому колу користувачів скористатися експертними знаннями та порадами.

- Зниження витрат

Вартість надання послуг експертних знань на одного користувача значно зменшується, що робить їх економічно вигідними.

- Стабільність знань

Знання, збережені в базах знань експертних систем, не втрачаються з часом і можуть зберігатися необмежено довго, на відміну від людських експертів, які можуть звільнитися, вийти на пенсію або померти.

- Агрегація знань

Експертна система може об'єднувати знання багатьох експертів, що значно підвищує її рівень компетенції порівняно з одним людським експертом.

- Швидкість реакції

Експертна система завжди готова до консультацій, що особливо важливо в екстрених ситуаціях, де потрібна оперативна реакція.

- Об'єктивність

Експертні системи надають відповіді, незалежні від емоцій, що є критичним у стресових або екстрених ситуаціях, де людський фактор може негативно вплинути на результат.

- Інтелектуальний доступ до даних

Експертні системи можуть використовуватися для інтелектуального доступу до баз даних, наприклад, для аналізу прихованих закономірностей у даних.

Попри численні переваги, експертні системи також мають певні недоліки, що можуть обмежити їх ефективність:

- Обмежена точність

Експертні системи можуть бути неточними, оскільки їхня точність залежить від якості та кількості даних у базі знань. Недостатня або неточна інформація може призвести до хибних висновків.

- Обмежена гнучкість

Внесення змін до бази знань може бути трудомістким і вимагати спеціальної кваліфікації, що ускладнює підтримку та оновлення системи.

- Нездатність до самонавчання

Багато експертних систем не можуть навчатися на основі нової інформації, що призводить до застарілих знань і зниження їх актуальності.

- Складність використання

Складність експертних систем може робити їх доступними тільки для фахівців у відповідній галузі, ускладнюючи використання для звичайних користувачів.

- Вразливість до помилок

Без регулярного оновлення та підтримки даних система може стати вразливою до помилок, що призведе до неправильних або неточних висновків.

- Обмежена сфера застосування

Експертні системи зазвичай розробляються для конкретної галузі знань і вирішують специфічні проблеми, тому їх важко використовувати в інших сферах.

- Високі витрати

Розробка та підтримка високоякісних експертних систем можуть бути витратними, що робить їх створення складним завданням для малих компаній та організацій з обмеженим бюджетом.

Підсумовуючи, можна зазначити, що експертні системи мають значні переваги, але також і певні недоліки, які необхідно враховувати перед їх використанням та розробкою. У кожному конкретному випадку слід ретельно зважувати всі переваги та недоліки для прийняття обґрунтованого рішення.

2.3. Аналіз технології DataMining

Оскільки для заповнення бази знань експертної системи необхідно аналізувати, класифікувати та структурувати великий обсяг інформації, знання методів і алгоритмів технології Data Mining може значно допомогти у цьому процесі.

Data Mining – це технологія, що використовується для аналізу великих наборів даних. Вона застосовує специфічні методи для виявлення цінної інформації та раніше невідомих закономірностей. Основні типи закономірностей, які можна виявити за допомогою Data Mining, включають:

- Асоціація

Виявлення залежностей і закономірностей між взаємопов'язаними подіями. Правило асоціації виглядає так: якщо відбувається подія X, то слідує подія Y. Цей метод вперше був використаний у сфері торгівлі для визначення товарів, які часто купуються разом. В медицині його застосовують для виявлення симптомів, які найчастіше пов'язані з певними захворюваннями.

- Послідовність

Виявлення закономірностей у часі, коли відбуваються події. Правило послідовності формулюється так: якщо відбувається подія X, то через час t відбудеться подія Y. Цей метод часто використовується в торгівлі та для аналізу навігації користувачів по сайту.

- Класифікація

Віднесення об'єктів або подій до певних груп на основі визначених ознак. Цей метод широко використовується у науці для класифікації живих організмів, у психології для класифікації типів темпераменту та у медицині для класифікації хвороб.

- Кластеризація

Схожа з класифікацією, але ознаки для групування не визначені заздалегідь. Методи Data Mining самі визначають споріднені дані та знаходять спільні ознаки. Кластеризація використовується в аналізі текстів, сегментації зображень, розпізнаванні об'єктів, а також у маркетингу для визначення типів споживачів.

- Прогнозування

Опис майбутніх подій на основі аналізу попередніх. Використовується для прогнозування демографічних змін, метеорологічних умов та в багатьох інших сферах.

До методів і алгоритмів Data Mining належать:

- Штучні нейронні мережі
- Дерева рішень
- Символьні правила
- Методи найближчого сусіда і k-найближчого сусіда
- Метод опорних векторів

- Байєсовські мережі
- Лінійна регресія
- Кореляційно-регресійний аналіз
- Ієрархічні методи кластерного аналізу
- Неієрархічні методи кластерного аналізу
- Алгоритми k-середніх і k-медіан
- Методи пошуку асоціативних правил
- Метод обмеженого перебору
- Еволюційне програмування
- Генетичні алгоритми
- Різноманітні методи візуалізації даних

Отже, більшість методів технології Data Mining – це відомі математичні методи та алгоритми, багато з яких належать до теорії штучного інтелекту. Data Mining об'єднує методи багатьох напрямів і розвивається разом з розвитком інших систем.

На рис. 2.1 показано науки, системи та компоненти, знання яких включає в себе DataMining.

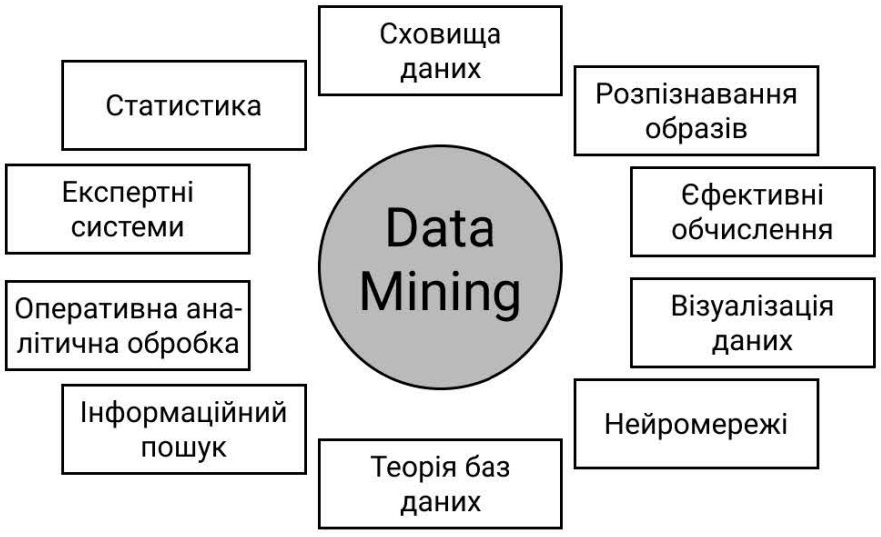

Рисунок 2.1 - Компоненти DataMining

Знання, які добуваються за допомогою методів DataMining прийнято представляти у вигляди закономірностей (патернів) (рис. 2.2).

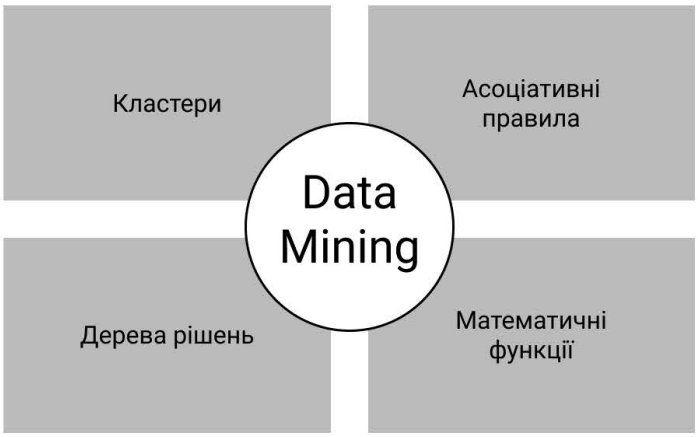

Рисунок 2.2 - Способи представлення знань DataMining

2.4 Дерева рішень як спосіб представлення закономірностей DataMining

Метод дерева рішень використовується для класифікації, опису даних, регресії та прогнозування. Цей метод дозволяє класифікувати великий обсяг даних за допомогою візуального представлення ієрархічної структури. Основою методу дерева рішень є підпорядкованість, розгортаємість і ранжування цілей.

Дерево рішень складається з таких компонентів: кореневий вузол, вузол рішень і лист (див. рис. 2.3). Усі ці компоненти з'єднуються між собою ребрами, які зазвичай є відповідями на запитання (так або ні). Графічно дерево рішень малюється догори ногами.

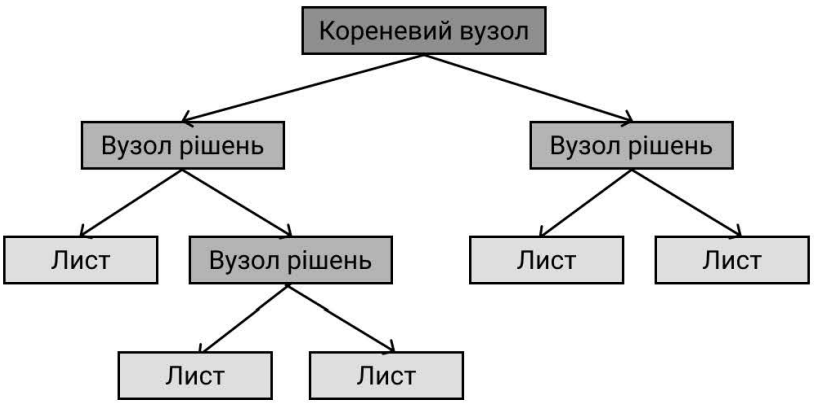

Рисунок 2.3 – Дерево рішень і його компоненти

На вершині дерева рішень знаходиться кореневий вузол, який є початком дерева. Зазвичай він містить умову або запитання, що називається "вирішальне правило". Від кореневого вузла, якщо це бінарне дерево, йдуть два переходи: відповідь "так" веде до лівої частини дерева, а "ні" - до правої.

Якщо після переходу слідує наступне запитання або умова, це внутрішній вузол або вузол рішення. Після наступного розгалуження може бути ще один вузол рішення або лист.

Лист – це кінцевий вузол дерева, також відомий як термінальний вузол. Він містить висновок або рішення, залежно від мети створення дерева рішень.

Окрема частина дерева рішень називається гілкою або піддеревом.

При використанні цього методу завжди потрібно пам'ятати, що велике дерево не завжди є ефективним. Такі дерева називають "гіллястими" або "кущистими", вони складаються з невиправдано великої кількості вузлів і гілок, що призводить до розбиття вихідної множини на велику кількість підмножин, які складаються з дуже малої кількості об'єктів. У результаті такого "переповнення" здатність дерева до узагальнення зменшується, і побудовані моделі не можуть давати правильні відповіді.

У методах Data Mining дерева рішень можуть бути наступних видів:

- Класифікаційне дерево – використовується, коли необхідний результат - це клас, до якого належить об'єкт.

- Регресійне дерево – використовується, коли необхідний результат - це числове значення (наприклад, ціна товару або час, необхідний для виконання завдання).

Як і будь-який метод чи алгоритм, дерево рішень має свої переваги та недоліки.

Переваги методу:

1) Легкість освоєння та простота у використанні.

2) Відсутність потреби у попередній обробці даних.

3) Інтуїтивно зрозумілий процес переходу по вузлах та отримання результату.

4) Можливість роботи з категоріальними та інтервальними змінними.

5) Надійність і точність.

6) Не потребує спеціального програмного забезпечення для роботи з великими наборами даних.

Недоліки методу:

1) Складність побудови правильного дерева рішень, яке не буде перенасиченим великою кількістю розгалужень.

2) Висока ймовірність помилок при побудові дерева рішень, особливо при визначенні наступного вузла і принципів розбиття [11].

2.5. Матриця Ейзенхауера

Важливим аспектом при прийнятті рішень є впорядкування необхідних рішень за важливостями. Для виконання цього завдання в теорії прийняття рішень використовується серед іншого матриця Ейзенхауера.

Як правило, люди складають настільки довгі списки всі справи з них апріорі неможливо переробити. В результаті в них накопичуються недороблені та незавершені справи. А це небезпечно, бо це не лише гальмує рух до мети, а й взагалі може зашкодити подальшому розвитку життя. Адже велика ймовірність розпорошення сил та енергії на те, що людині не потрібно.

Матриця Ейзенхауера дозволяє швидко та ефективно розібратися зі списком справ, розподіливши ці справи за категоріями. У результаті людина ясно бачить усі найважливіші справи, а також ті справи, які взагалі не варті його уваги.

Для цього необхідно скласти комплексний список всіх задач, включаючи терміни їхнього виконання. До кожної із них поставити питання:

1. Чи узгоджується ця задача з життєвими цілями та пріоритетами?
2. Чи обов'язково виконувати її саме зараз?
3. Чи можна відкласти її або делегувати комусь іншому?

Матриця Ейзенхауера розподіляє задачі на важливі і термінові (які необхідно виконати у першу чергу), не термінові важливі справи (у другу чергу), термінові не важливі (у третю чергу) і не термінові і не важливі (в останню чергу) (рис. 2.4). Завдання з першої категорії це речі, які необхідно зробити. Це робота, яка має значний вплив на довгострокове планування і має бути вирішена терміново.

До другої категорії відносяться завдання які важливі в перспективі, але не є терміновими. Тому набагато легше спланувати, коли їх виконувати.

Термінові та неважливі справи - це справи, які заважають ефективно працювати, оскільки потребують термінової уваги. Потрібно бути особливо уважним до таких справ. Часто їх плутають із завданнями першої категорії. Але не все термінове важливе. Основний критерій для розрізнення термінового та важливого – наближає цю справу вас до мети чи ні. До третього квадранта входять завдання, ніяк не пов'язані з вашою метою.

Неважливі та нетермінові справи - сюди потрапляє усе те, що ми робимо майже щодня, але це не пов'язано з нашою роботою: перегляд телевізора, порожні телефонні розмови, комп'ютерні ігри, відвідування форумів, соціальних мереж тощо. буд. Це зазвичай приємні справи, але не обов'язкові. Ці справи Ейзенхауер назвав справжніми "пожирателями часу", що знижують продуктивність дня.

Тому завжди треба чітко формувати свої цілі, пам'ятати про них та бути сфокусованим на них. Найпростіше - завжди мати їх опис перед очима.

Таким чином матриця Ейзенхауера дозволяє швидко та ефективно розібратися зі списком справ, розподіливши ці справи за категоріями. У результаті людина ясно бачить усі найважливіші справи, а також ті справи, які взагалі не варті його уваги. Багато рутинних справ з цього квадранта можна і потрібно делегувати.

Рисунок 2.4 – Матриця Ейзенхауера

2.6. Висновки до другого розділу

Підсумовуючи все вище сказане, можна сказати, що експертна система представляє собою метод вирішення проблем чи питань в певній галузі, за допомогою накопичення експертних знань, які збираються у експертів даної галузі та структуруються.

Експертні системи, засновані на використанні накопичених знань та алгоритмів штучного інтелекту, мають великий потенціал у різних галузях, таких як медицина, фінанси, виробництво та багато інших. Вони можуть значно покращити процес прийняття рішень, забезпечуючи об'єктивність і зниження ризику помилок.

Для виявлення нових закономірностей у вже наявній інформації, наданій експертами, корисно застосовувати технологію Data Mining, яка включає численні методи, спрямовані на вирішення цієї задачі. Одним зі способів представлення таких закономірностей є дерева рішень, які дозволяють наочно та зрозуміло

структурувати інформацію. При створенні таких дерев важливо уникати надмірного ускладнення та не додавати зайвих вузлів, оскільки це може призвести до неправильних результатів.

Для планування на короткий термін ефективною є матриця Ейзенхауера, яка буде використана у даній кваліфікаційній роботі. Це ефективний метод опрацювання списків справ, які людина планує зробити за визначений термін. Як правило, люди складають настільки довгі списки всі справи з них апріорі неможливо переробити.

В результаті в них накопичуються недороблені та незавершені справи. А це небезпечно, бо це не лише гальмує рух до мети, а й взагалі може зашкодити подальшому розвитку життя. Адже велика ймовірність розпорошення сил та енергії на те, що людині не потрібно.

Програмна реалізація матриці Ейзенхауера буде наведена в пункті 3.6 кваліфікаційної роботи.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПІДТРИМКИ ПРИЙНЯТТЯ РШЕНЬ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ.

3.1. Структура проекту та інструменти розробки

Для ефективної розробки програмного забезпечення потрібно визначити структуру системи, базу знань та відповідно правила по яким працює програмний застосунок (рис. 3.1).

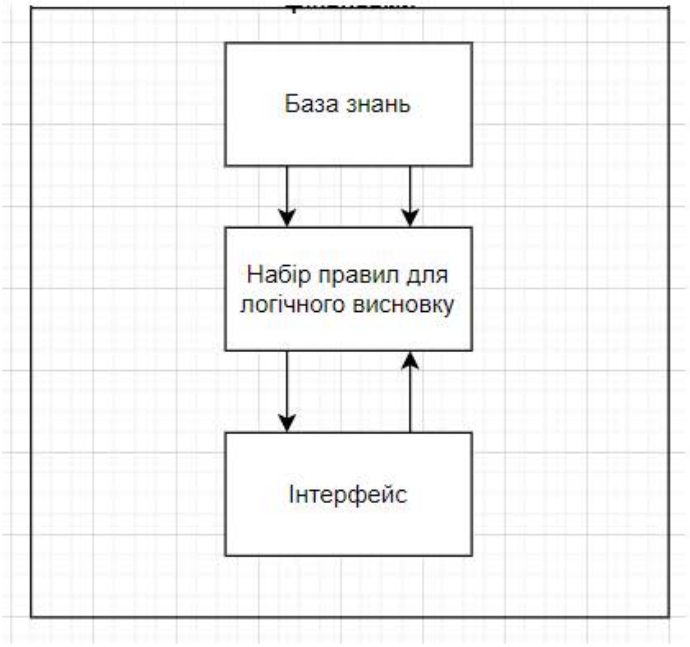

Рисунок 3.1 – Структура програмного застосунку

Для розробки програмного забезпечення у відповідності до мети кваліфікаційної роботи візьмемо за основу принципи побудови експертних систем як вид інтелектуальних систем підтримки прийняття рішень. Основні підходи та принципи експертних систем були розглянуті в другому розділі. Нам потрібно розробити 3 елементи системи:

- База знань
- Набір правил для логічного висновку
- Інтерфейс, що слугує для виведення та введення даних [10]

Для розробки даного додатку використовується мова програмування C#. C# є сучасною, об'єктно-орієнтованою мовою програмування, яка була розроблена корпорацією Microsoft як частина платформи .NET. Вперше представлена у 2000 році, C# здобула широку популярність завдяки своїй гнучкості, здатності до масштабування та потужним засобам для розробки додатків різного типу, від настільних програм до веб-додатків та мобільних застосунків.

Основні особливості мови C# включають:

1. Об'єктно-орієнтований підхід: C# підтримує всі основні принципи об'єктно-орієнтованого програмування, такі як інкапсуляція, наслідування та поліморфізм. Це дозволяє створювати масштабовані та легко підтримувані системи.

2. Безпека типів: Строга типізація в C# забезпечує високий рівень безпеки під час виконання програм, запобігаючи типовим помилкам, як-от невірне перетворення типів або неправильне використання об'єктів.

3. Інтеграція з .NET Framework: C# була спеціально створена для роботи з .NET Framework, що надає величезний набір готових до використання класів і бібліотек, що спрощують розробку програмного забезпечення [10].

Для створення додатку з візуально приємним інтерфейсом використовується Unity. Unity є потужним кросплатформеним рушієм, який часто асоціюється з розробкою ігор, але також ефективно використовується для створення різноманітних візуально привабливих інтерактивних застосунків, включаючи ті, що вимагають складної візуалізації даних та управління користувацьким інтерфейсом. Його гнучкість роблять Unity ідеальним інструментом для створення інтерфейсу який може бути привабливим та зручним для користувача [8].

Основні переваги:

- Широкі можливості візуалізації: Unity забезпечує потужні засоби для 3D та 2D візуалізації, що дозволяють створювати детальні та анімовані графічні елементи, які можуть підвищити візуальну привабливість додатку.

- Кросплатформеність: Unity підтримує експорт проектів на багато платформ, включаючи Windows, macOS, iOS, Android, та веб-браузери. Це робить його відмінним рішенням для розробки додатків, які можуть бути доступні на різних пристроях.
- Багатий набір інструментів для розробки UI: Unity надає розширені можливості для створення інтерфейсів користувача через компонент Unity UI, який дозволяє легко інтегрувати текст, кнопки, зображення та інші інтерактивні елементи.
- Підтримка C#: Unity використовує C# для скриптів, що дозволяє розробникам використовувати потужні та гнучкі можливості цієї мови програмування включаючи обробку подій користувача, управління даними та інше [9].

3.2. Розробка програмного застосунку

Розробка додатку складається з декількох частин а саме: розробка алгоритмів аналізу, розробка критеріїв аналізу фінансових завдань, які перед собою ставить користувач та розробка способу виведення результату у вигляді порядку виконання фінансових завдань та розподілення коштів на всі дні виконання фінансових завдань (рис. 3.2).

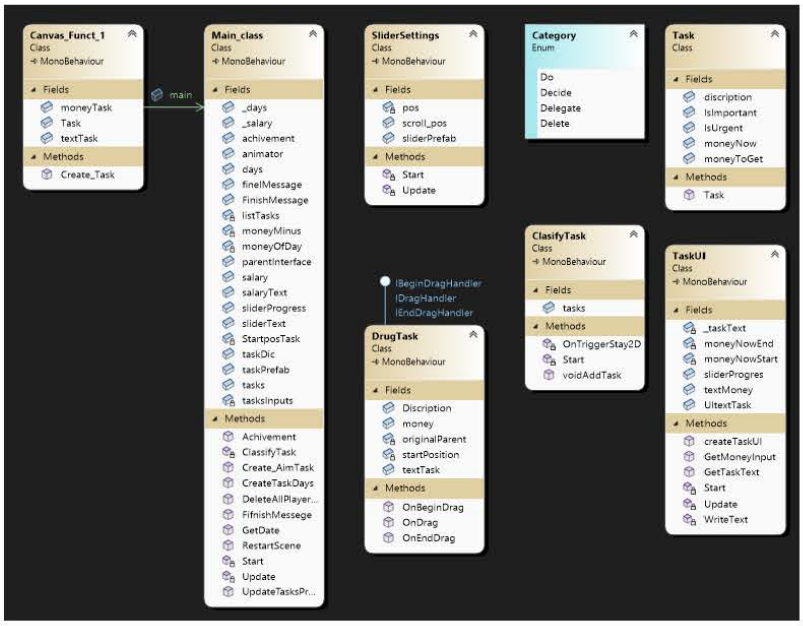

Рисунок 3.2 – Діаграма класів проекту

Для роботи програми, потрібно розглянути основні класи, що забезпечують роботу програми.

Головний клас, що працює та організовує з роботу з усіма об'єктами Main_class. Даний клас приймає дані практично з усіх об'єктів ведення та аналізує їх (рис. 3.3).

```
*/
public GameObject FinishMessage;
public Text finelMessage;
//
public Slider sliderProgress;
public Text sliderText;
//AboutTask
public GameObject taskPrefab;
public GameObject parentInterface;// зробити батьківським
private List<TaskUI> listTasks;
private List<GameObject> tasksInputs;// Список для зручного видалення об'єкта
private Vector2 StartposTask = new Vector2(-100, 200);
//
public InputField _salary;
public InputField _days;
public Text salaryText;
//
public int salary;
public int days;
private float moneyOfDay; // можлива сума на день
public List<Task> tasks;
public Dictionary<int, List<Task>> taskDic;
//
private int moneyMinus = 0;//суму чисел які користувач ввів з усіх полів
//Achivement
public Text achivement;
public Animator animator;
Unity Message | 0 references
```

Рисунок 3.3 – Елементи інтерфейсу, які контролює Main_class

Main_class має декілька важливих класів, що слугують основою для роботи даного додатку.

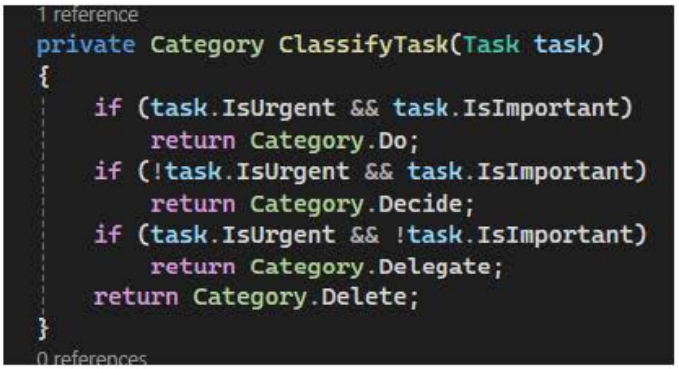

Рисунок 3.4 – Функція ClassifyTask

Функція ClassifyTask (рис. 3.4) класифікує об'єкти типу Task (завдання) відповідно до їх важливості та терміновості. Це дозволяє організувати задачі за категоріями на основі матриці Ейзенхауера. Категорія визначається за 2 змінними task.IsUrgent та task.IsImportant, тоді Task надається одна з цих категорій: Category.Do, Category.Decide, Category.Delegate, Category.Delete.

- Якщо задача одночасно є терміновою (IsUrgent) та важливою (IsImportant), то вона класифікується як Category.Do.
- Якщо задача не є терміновою, але важлива, то вона класифікується як Category.Decide, тобто потрібно запланувати час для виконання задачі у майбутньому.
- Якщо задача термінова, але не важлива, то її категорія – Category.Delegate.
- Якщо задача не є ні терміновою, ні важливою, то вона класифікується як Category.Delete. Такі задачі можна відкинути або відкласти на невизначений термін.

```
0 references
public void GetDate()
{
    Debug.Log("///GetData");
    salary = int.Parse(_salary.text);
    days = int.Parse(_days.text);
    moneyOfDay = salary / days;
    PlayerPrefs.SetInt("SizeDays", days);
    salaryText.text = "У вас всього " + salary + " " + "Ваша денна сума
}
0 references
```

Рисунок 3.5 – Функція GetDate

Функція GetDate використовується для зчитування з елементів інтерфейсу таких як InputField _salary, InputField _days, та оновлення тексту в елементі salaryText.

Після виконання даної, функції можуть отримувати дані про кількість днів на виконання фінансових операцій та фінансові надходження які отримав користувач.

```
1 reference
public void CreateTaskDays()
{
    tasks = ClasifyTask.tasks;
    List<Task> tasksDay = new List<Task>();
    for(int i = 0; i < days; i++)
    {
        taskDic[i] = new List<Task>();
        foreach(Category cat in Enum.GetValues(typeof(Category)))
        {
            tasksDay = tasks.Where(t=>ClassifyTask(t)==cat && t.moneyNow < t.moneyToGet).
                OrderByDescending(t=>t.IsImportant).
                ThenByDescending(t=>t.IsUrgent).
                ThenBy(t=> t.moneyNow).
                Take(1).
                ToList();
            taskDic[i].AddRange(tasksDay);

        }
    }
}
1 reference
```

Рисунок 3.6 – Функція CreateTaskDays

Функція CreateTaskDays (рис. 3.6) організує задачі за днями на основі їх категорії та певних критеріїв вибору. Ця функція призначена для визначення найбільш відповідних задач для кожного дня, що ґрунтуються на їх важливості, терміновості, та фінансовому пріоритеті.

Спочатку відбувається ініціалізація списку tasks, де зберігаються всі задачі, отримані з класу ClasifyTask. Далі створюється новий список для зберігання під

час роботи циклу: List<Task> tasksDay = new List<Task>(). Цикл проходить через кожен день (кількість днів - days), для якого потрібно визначити задачі. Для кожного дня створюється новий список, який буде зберігати відібрані задачі для конкретного дня: taskDic[i] = new List<Task>(). TasksDay = tasks.Where(t=>ClassifyTask(t)==cat && t.moneyNow < t.moneyToGet) - здійснюється відбір задач за критеріями: задача повинна відповідати поточній категорії (cat), сума грошей на даний момент (moneyNow) має бути меншою за суму, яку потрібно отримати (moneyToGet).

Задачі сортуються за важливістю (IsImportant), терміновістю (IsUrgent), та сумою грошей на даний момент (moneyNow), з вибором однієї задачі (Take(1)).

```
1 reference
public void Create_AimTask()
{
    int NumberIndex = 0;//буде індексом
    int NumberDay = 1;//Буде номером дня
                      //Коли не перший раз
    Debug.Log(NumberIndex);
    if (PlayerPrefs.HasKey("NumberDay") && PlayerPrefs.HasKey("NumberIndex"))
    {
        NumberDay = PlayerPrefs.GetInt("NumberDay");
        NumberIndex = PlayerPrefs.GetInt("NumberIndex");
    }
    else
    {

        sliderText.text = NumberDay.ToString();
        PlayerPrefs.SetInt("NumberDay", NumberDay);
        PlayerPrefs.SetInt("NumberIndex", NumberIndex);
        sliderProgress.maxValue = days;
        sliderProgress.value = NumberDay;
    }
    Vector2 pos = StartposTask;
    if(taskDic[NumberIndex].Count>0)
    {
        foreach (Task task in taskDic[NumberIndex])
        {
            GameObject newTaskUI = Instantiate(taskPrefab, pos, Quaternion.identity);
            newTaskUI.transform.SetParent(parentInterface.transform, false);
            TaskUI taskUI = newTaskUI.GetComponent<TaskUI>();
            taskUI.createTaskUI(task.moneyToGet, task.moneyNow, task.discription);
            listTasks.Add(taskUI);
            tasksInputs.Add(newTaskUI);// Додаємо до списку інпути
            pos.y = pos.y - 130;
        }
    }
    else
    {
        FifnishMessege(false, false);
    }

}
```

Рисунок 3.7 – Функція Create_AimTask

Функція Create_AimTask (рис. 3.7) призначена для ініціалізації та відображення завдань користувача у графічному інтерфейсі, з використанням заданих даних про дні та індекси завдань. Вона виконує ряд операцій з управлінням станом завдань і зберігає необхідні параметри між сесіями за допомогою PlayerPrefs.

Змінні NumberIndex, NumberDay зберігають індекс поточного номеру списку завдань та номер дня, коли повинні відображатися дані завдання.

```
foreach (Task task in taskDic[NumberIndex])
{
    GameObject newTaskUI = Instantiate(taskPrefab, pos, Quaternion.identity);
    newTaskUI.transform.SetParent(parentInterface.transform, false);
    TaskUI taskUI = newTaskUI.GetComponent<TaskUI>();
    taskUI.createTaskUI(task.moneyToGet, task.moneyNow, task.discription);
    listTasks.Add(taskUI);
    tasksInputs.Add(newTaskUI);// Додаємо до списку інпути
    pos.y = pos.y - 130;
}
```

Рисунок 3.8 – Цикл створення завдань в графічному інтерфейсу

За допомогою циклу (рис. 3.8), розглядаються всі завдання, та створюється відповідний елемент інтерфейсу кожному завданню. В іншому випадку виконується функція FifnishMessege.

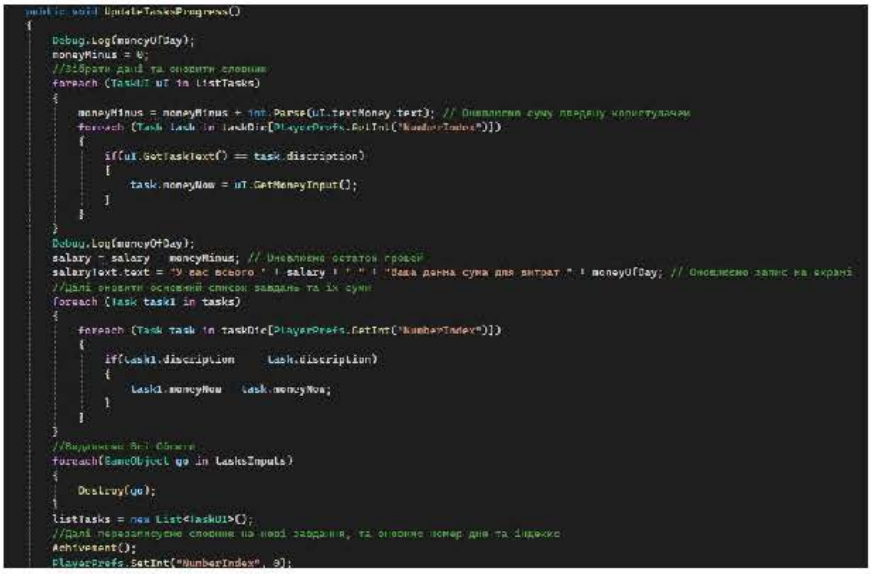

Рисунок 3.9 – Функція UpdateTasksProgress

UpdateTasksProgress (рис. 3.9) використовується для оновлення всіх масивів даних пов’язаних з виконанням завдань та фінансів користувача. Функція складається з декількох важливих частин.

```
Debug.Log(moneyOfDay);
moneyMinus = 0;
//Зібрати дані та оновити словник
foreach (TaskUI uI in listTasks)
{
    moneyMinus = moneyMinus + int.Parse(uI.textMoney.text); // Оновлюємо суму введену користувачем
    foreach (Task task in taskDic[PlayerPrefs.GetInt("NumberIndex")])
    {
        if(uI.GetTaskText() == task.discription)
        {
            task.moneyNow = uI.GetMoneyInput();
        }
    }
}
```

Рисунок 3.10 – Перша частина функції

Перша частина функції (рис. 3.10), це цикл foreach (TaskUI uI in listTasks), даний цикл використовується для отримання даних які ввів користувач та оновлення прогресу виконання фінансових завдань.

```
]
Debug.Log(moneyOfDay);
salary = salary - moneyMinus; // Оновлюємо остаток грошей
salaryText.text = "У вас всього " + salary + " " + "Ваша денна сума для витрат " + moneyOfDay; // Оновлюємо запис на екрані
//Далі оновити основний список завдань та їх суми
foreach (Task task1 in tasks)
{
    foreach (Task task in taskDic[PlayerPrefs.GetInt("NumberIndex")])
    {
        if(task1.discription == task.discription)
        {
            task1.moneyNow = task.moneyNow;
        }
    }
]
//Видаляємо Всі Обекти
foreach(GameObject go in tasksInputs)
{
    Destroy(go);
}
listTasks = new List<TaskUI>();
//Далі перезаписуємо словник на нові завдання, та оновимо номер дня та індекс
Achivement();
PlayerPrefs.SetInt("NumberIndex", 0);
days = days - 1;
```

Рисунок 3.11 – Видалення всіх завдання

Наступним кроком є видалення всіх завдань з інтерфейсу (рис. 3.11), також за допомогою циклів foreach. Далі останнім кроком є створення нових завдань з врахуванням оновлених даних прогресу виконання завдань (рис. 3.12).

```
days = days - 1;
if (salary > 0)
{
    if (days > 0)
    {
        CreateTaskDays();//Перезаписуємо словник із завданнями
                         //Переходимо на наступний день
        int newNumberDay = PlayerPrefs.GetInt("NumberDay") + 1;
        PlayerPrefs.SetInt("NumberDay", newNumberDay);
        sliderText.text = newNumberDay.ToString();
        sliderProgress.value = PlayerPrefs.GetInt("NumberDay");
        //Створюємо нові завдання на вікні
        Create_AimTask();
    }
    else
    {
        //Якщо не залишилося
        FifnishMessege(true, false);
    }
}
else
{
    FifnishMessege(true,true);
}
```

Рисунок 3.12 – створення нових завдань

Функція Achivement, використовується для надання користувачу певних досягнень. Під час роботи програми, функції виводить повідомлення, якщо користувач досягнув певного результату у виконанні фінансових завдань (рис. 3.13).

```
1 reference
public void Achivement()
{
    animator.SetBool("IsSet", true);
    Debug.Log("/////");
    float fullSizeListTask = tasks.Count();
    float sizeCompleteTask = 0;
    foreach (Task task in tasks)
    {
        if(task.monayNow>= task.moneyToGet)
        {
            sizeCompleteTask++;
        }
    }
    Debug.Log("SizeTask "+fullSizeListTask);
    Debug.Log("Complete " + sizeCompleteTask);
    Debug.Log((sizeCompleteTask / fullSizeListTask) * 100+);
    float percent = (sizeCompleteTask / fullSizeListTask) * 100f;
    Debug.Log("achivo " + percent);
    if (percent >= 28 && percent <= 30 && !PlayerPrefs.HasKey("Ach1"))
    {

        achivement.text = "Продовжуйти в тому ж дусі, ви пройшли третину шляху 25%";
        animator.SetTrigger("Ach");
        PlayerPrefs.SetString("Ach1", "1");


    }
    if(percent >= 50 && percent <=60 && !PlayerPrefs.HasKey("Ach2"))
    {
        Debug.Log("Check");
        achivement.text = "Ви вже пройшли половину шляху 50%";
        animator.SetTrigger("Ach");
        PlayerPrefs.SetString("Ach2", "2");

    }
    if(percent >=70 && percent <=80 && !PlayerPrefs.HasKey("Ach3"))
    {
        achivement.text = "Ви майже дійшли до кінця";
        animator.SetTrigger("Ach");
        PlayerPrefs.SetString("Ach3", "3");
```

Рисунок 3.13 – Функція Achivement

Під час роботи програми функції перевіряє за допомогою виразу !PlayerPrefs.HasKey("Ach1"), чи користувач отримував дане досягнення, якщо так то відповідне досягнення вже не виводиться.

```
public void FifnishMessege( bool isDayOrTask, bool moneyFalse)
{
    //якщо гроші закінчилися
    if(!moneyFalse)
    {
        //true якщо закінчилися дні
        if (isDayOrTask)
        {
            FinishMessage.SetActive(true);
            finelMessage.text = "Дні виконання завдань закінчилися, почніть заново";
        }
        else
        {
            //Якщо закінчилися завдання
            finelMessage.text = "Вітаємо ви виконали всі свої плани";
            FinishMessage.SetActive(true);
        }
        Debug.Log("Дні закінчилися");
    }
    else
    {
        FinishMessage.SetActive(true);
        finelMessage.text = "Гроші закінчилися, почніть заново";
    }

}
```

Рисунок 3.14 – Функція FifnishMessege

Остання важлива функція FifnishMessege (рис. 3.14). Використовується для виведення повідомлення про завершення роботи додатку. Повідомлення виводить текст в якому зазначається про виконання всіх завдань або закінчення днів виконання чи грошей користувача.

Наступний клас ClasifyTask. Даний клас забезпечує зберігання новостворених завдань користувача та додавання їх у список tasks. Також даний клас визначає структуру завдання Task та його параметрів (рис. 3.15).

```
public class Task
{
    public string discription;
    public float moneyToGet;
    public float moneyNow;
    public bool IsUrgent;
    public bool IsImportant;
    4 references
    public Task(string disc, float money, bool isugent, bool isImp ) {
        discription = disc;
        moneyToGet = money;
        IsUrgent = isugent;
        IsImportant = isImp;
    }
}
Unity Script (4 asset references) | 1 reference
```

Рисунок 3.15 – Структура Task

Головною функцією є OnTriggerStay2D, тобто дана функція використовує компонент BoxCollider2D для її коректної роботи.

Рисунок 3.16 – Функція OnTriggerStay2D

Функція OnTriggerStay2D (рис. 3.16) виконується кожен кадр під час роботи програми та записує завдання відповідно в якому колайдері ці завдання знаходяться (_1c, _2c, _3c, _4c), далі просто визначає параметри task.IsUrgent, task.IsImportant за якими визначається категорія завдання.

Рисунок 3.17 – Функція voidAddTask

VoidAddTask (рис. 3.17), використовується для спрощеного додавання завдань в список.

Наступний клас DrugTask (рис. 3.18) використовується для роботи елементу інтерфейсу, щоб користувач міг «перетягувати» створене завдання до потрібної категорії.

```
0 references
public void OnBeginDrag(PointerEventData eventData)
{
    startPosition = transform.position;  // Зберігаємо почa
    originalParent = transform.parent;  // Зберігаємо почат
    transform.SetParent(transform.parent); // Переміщуємо с
    GetComponent<CanvasGroup>().blocksRaycasts = false; //
}

0 references
public void OnDrag(PointerEventData eventData)
{
    transform.position = Input.mousePosition; // Оновлюємо
}

0 references
public void OnEndDrag(PointerEventData eventData)
{
    transform.SetParent(originalParent); // Відновлюємо пер
    GetComponent<CanvasGroup>().blocksRaycasts = true; // Д
}
}
```

Рисунок 3.18 –Клас DrugTask

Для переміщення об’єкту слугують 3 функції: OnBeginDrag, OnDrag, OnEndDrag. Ці три функції керують процесом перетягування об'єкта у графічному інтерфейсі користувача в Unity. OnBeginDrag ініціює процес перетягування, зберігаючи початкову позицію та батьківський об'єкт для. Також ця функція встановлює поточний об'єкт на передній план і тимчасово вимикає блокування подій raycast, що дозволяє вільно переміщувати його над іншими елементами інтерфейсу. OnDrag відповідає за оновлення позиції об'єкта, рухаючи його за курсором миші в момент перетягування. OnEndDrag закінчує процес перетягування, повертаючи об'єкт до його оригінального батьківського контейнера та знову активує блокування raycast, що дозволяє об'єкту знову реагувати на події в інтерфейсі.

Останній клас, що відповідає за створення фінансового завдання для користувача є TaskUI (рис. 3.19).

```
public class TaskUI : MonoBehaviour
{
    // Start is called before the first frame update
    public InputField textMoney;
    public Slider sliderProgres;
    public Text UItextTask;
    private float moneyNowStart;// Коли на початку до зберігання змін
    private float moneyNowEnd; // після натискання на кнопку
    private string _taskText;
    Unity Message | 0 references
    void Start()
    {

    }
    1 reference
    public void createTaskUI(double maxValue, double moneyNow ,string taskText)
    {
        sliderProgres.maxValue = float.Parse(maxValue.ToString());
        UItextTask.text = taskText;
        sliderProgres.value = float.Parse(moneyNow.ToString());
        moneyNowStart = float.Parse(moneyNow.ToString());
        _taskText = taskText;
        WriteText();
    }
    1 reference
    public float GetMoneyInput()
    {
        return moneyNowEnd;
    }
    1 reference
    public string GetTaskText()
    {
        return _taskText;
    }
    // Update is called once per frame
```

Рисунок 3.19 – Клас TaskUI

Функція createTaskUI (рис. 3.19) виконує створення фінансового завдання для користувача. Під час створення у вікні програми з'являється 2 елемента інтерфейсу, 1 це поле в якому зазначено назву завдання та зображено прогрес її виконання, інший елемент це поле для введення суми оплати, яку користувач вводить, щоб її виконати повністю або частково. Таким чином при створенні завдання, в кожного з них буде по 2 елемента інтерфейсу.

```
Unity Message | 0 references
void Update()
{
    try
    {
        if(textMoney.text != "")
        {
            sliderProgres.value = moneyNowEnd = float.Parse(textMoney.text) + moneyNowStart;
        }
    }
    catch (Exception e)
    {
        Debug.LogException(e);
    }

}
1 reference
void WriteText()
{
    UItextTask.text = UItextTask.text + " " + moneyNowStart + "/" + sliderProgres.maxValue;
}
```

Рисунок 3.20 – Функція Update

Функція Update (рис. 3.20) відповідає за оновлення прогресу виконання завдання, як тільки користувач введе нову суму коштів.

3.3. Розробка інтерфейсу

Для ефективної роботи експертної системи потрібно створити інтерфейс, який матиме перш за все елементи вводу та виведення інформації на екран користувача.

Unity надає численні елементи інтерфейсу які можуть легко налаштовуватися під різні потреби користувача (рис. 3.21).

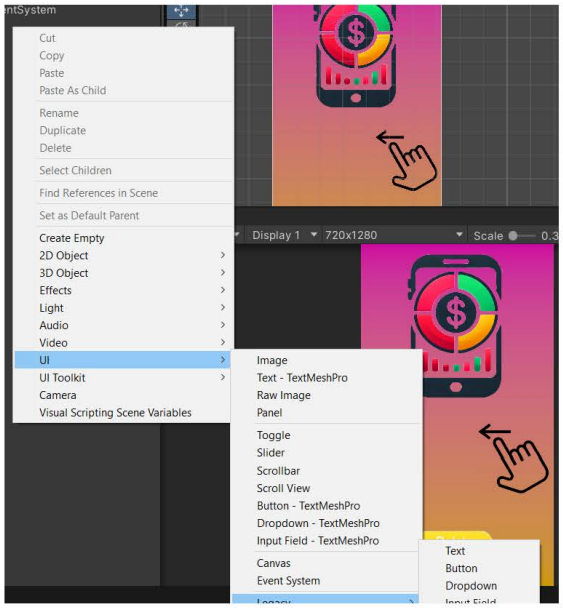

Рисунок 3.21 – Всі елементи інтерфейсу в Unity

Перш ніж створити користувацький інтерфейсом, спочатку потрібно створити базовий елемент Canvas. Даний елемент інтерфейсу має важливі параметри від регулювання розмірів елементів інтерфейсу до встановлення адаптивності під різні типи пристроїв як ПК так і смартфони. Для регуляції цих параметрів використовують компонент Canvas Scaler (рис. 3.22).

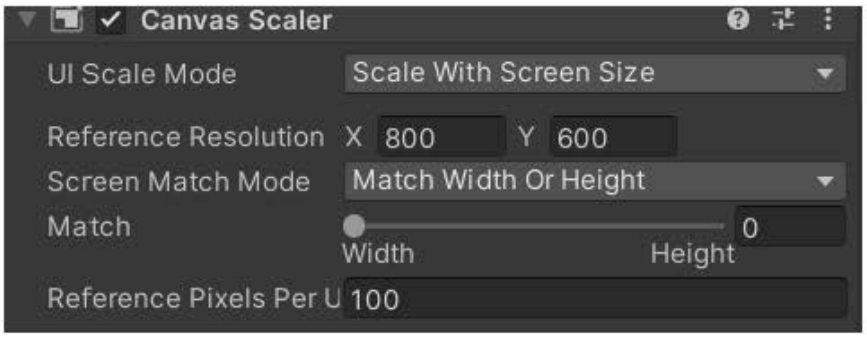

Рисунок 3.22 – Компонент Canvas Scaler

В даному компоненті важливим параметром є UI Scale Mode, що вказує яким чином буде підлаштовуватися інтерфейс. Scale With Screen Size визначає що всі елементи будуть підлаштовуватися під розмір екрану (рис. 3.22).

Зовнішній вигляд інтерфейсу можна перевірити на сцені при виборі розмірів екрану (рис. 3.23).

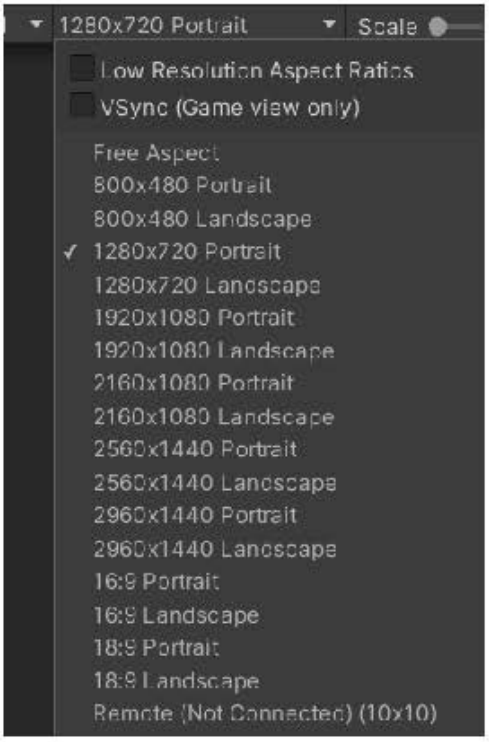

Рисунок 3.23 – Різні шаблони розмірів екрану

Наступним кроком є створення користувацького меню, що дозволяв би переміщатися між різними вікнами інтерфейсу. В даному проекті використовується елемент ScrollView.

Елемент інтерфейсу ScrollView надає можливість плавно переходити між різними вікнами інтерфейсу програми як на персональному комп'ютері так і на екрані власного смартфону.

Далі інтерфейс розподіляються на 4 вікна інтерфейсу (рис. 3.24):

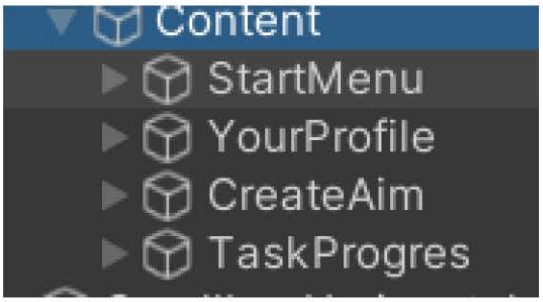

Рисунок 3.24 – Всі вікна програми

StartMenu – це стартове меню яке бачить користувач вперше, під час завантаження програми. Має кнопку для видалення всіх збережених даних та логотип програми (рис. 3.25).

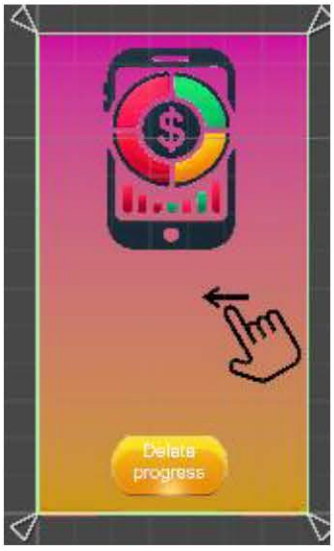

Рисунок 3.25 – Вікно StartMenu

YourProfile – дане вікно інтерфейсу використовується для введення даних про фінансові надходження користувача та кількість днів під час яких користувач планує виконати власні фінансові завдання (рис. 3.26).

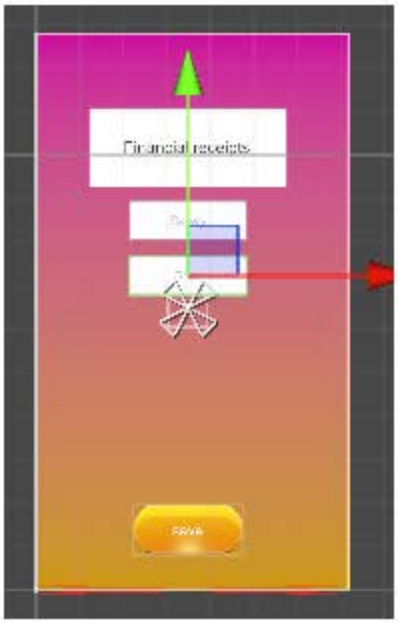

Рисунок 3.26 – Вікно YourProfile

Наступне вікно CreateAim – використовується для створення фінансових завдань користувачем, та визначення категорії, при переміщенні завдання в певну область (рис. 3.27).

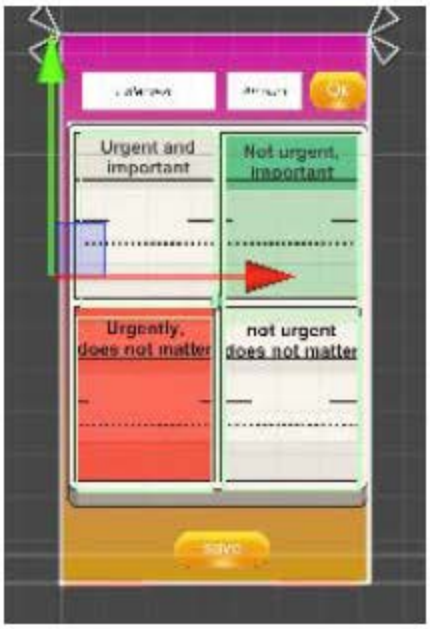

Рисунок 3.27 – Вікно CreateAim

Останнє вікно TaskProgres, в якому з'являються вже розподілені за категоріями завдання, які користувач має виконати (рис. 3.28):

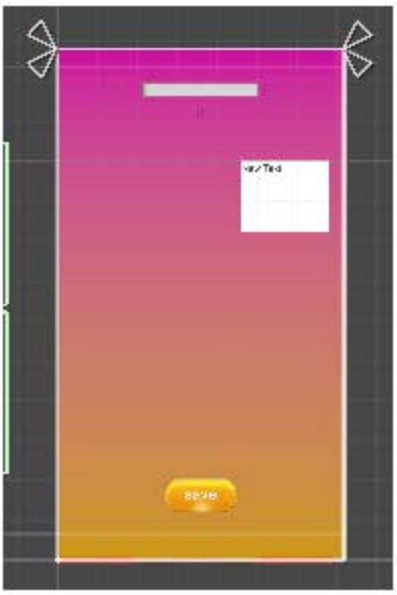

Рисунок 3.28 – Вікно TaskProgres

Unity надає можливість створити робочу збірку проекту, що надає можливість запускати програму на будь-якому пристрої. Для цього потрібно відкрити вікно BuildSettings та обрати потрібну платформу для створення збірки проекту (рис. 3.29).

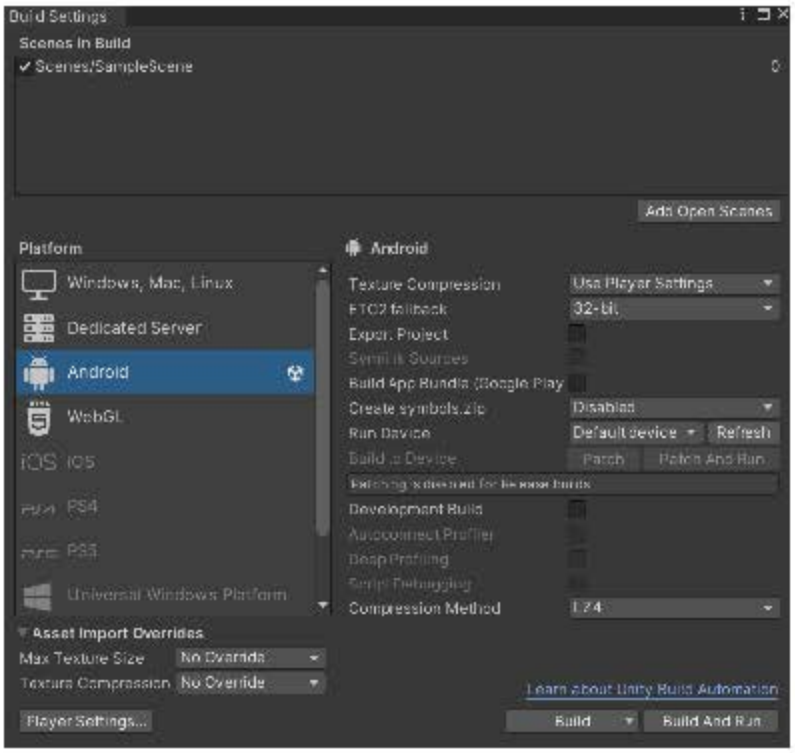

Рисунок 3.29 – Вікно BuildSettings

3.4. Тестування проекту

Наступним кроком є тестування програми, перевірка всіх її функції та коректної роботи інтерфейсу. Спочатку потрібно завантажити програму та запустити її. Після цього користувач повинен спостерігати наступне вікно (рис. 3.30):

Рисунок 3.30 – Стартове вікно програми

У вікні інтерфейсу спостерігається анімоване зображення, що показує користувачу в якому напрямку рухати палець по сенсору, що перегорнути меню на інше вікно. Після того як користувач перегорнув меню, він потрапляє в вікно де йому потрібно вказати кількість днів для планування виконання завдань та фінансові надходження на цей період, за допомогою яких він буде оплачувати ті цілі які він зазначить в подальшому. Тож введемо 30 днів та суму 10000 гривень та натискаємо зберегти (рис. 3.31).

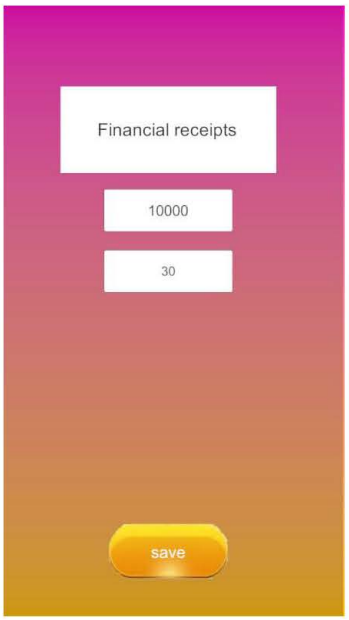

Рисунок 3.31 – Введення днів та фінансових надходжень

Далі користувач повинен знову перейти на наступне вікно, для створення вже самих цілей на вказаний період днів (рис. 3.32).

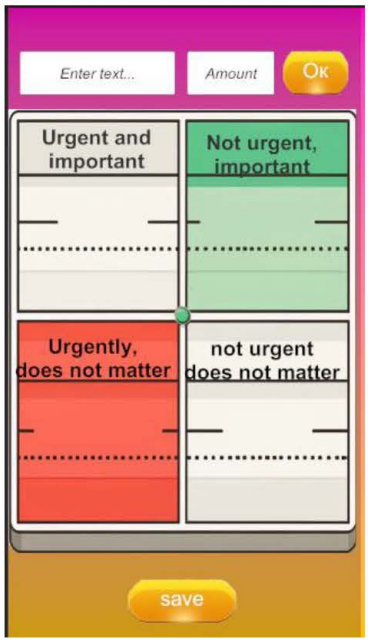

Рисунок 3.32 – Вікно створення цілей

Варто зазначити, що користувач спостерігає собою програмну реалізацію матриці Ейзенхауера. Для початку користувач має створити якесь завдання, для цього він має ввести опис завдання та суму для його виконання. Для прикладу

створимо завдання «Комунальні послуги» та введемо суму приблизно в 5000 гривень та натиснемо «Ок» (рис. 3.33).

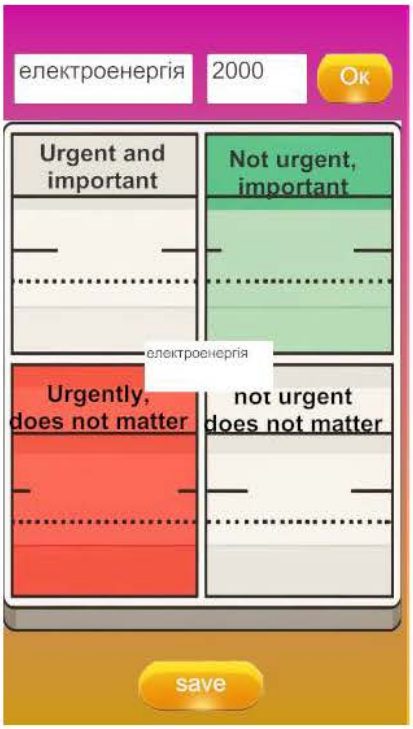

Рисунок 3.33 – Створення нової цілі

Далі користувач може перемістити його у відповідну категорію по важливості та по строку виконання (рис. 3.34).

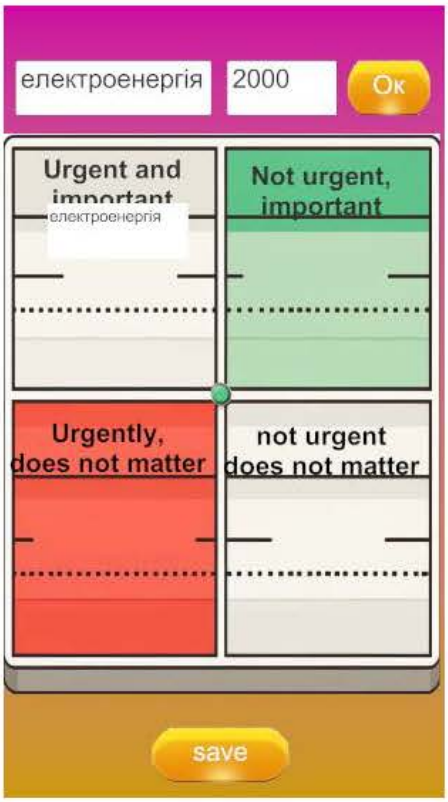

Рисунок 3.34 – Створення цілі, та визначення категорії цілі

Створимо декілька цілей та розподілимо їх по всіх категоріях (рис. 3.35).

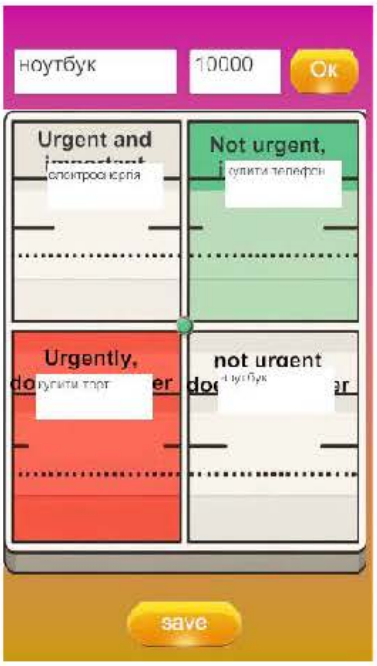

Рисунок 3.35 – Створення декількох фінансових цілей

Далі натискаємо на кнопку «Зберегти», та переходимо на наступне вікно. На екрані користувач спостерігає декілька завдань, та прогрес їх виконання тобто яка сума була внесена на кожну ціль. Також можна спостерігати у верхній частині інтерфейсу загальну суму, та рекомендовану суму витрат на день (рис. 3.36).

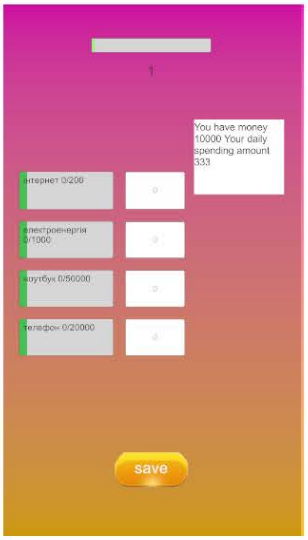

Рисунок 3.36 – Вікно з фінансовими цілями

Тепер потрібно ввести суму для кожного завдання. При введенні користувач вже спостерігає на скільки збільшиться прогрес оплати обраного фінансового завдання.

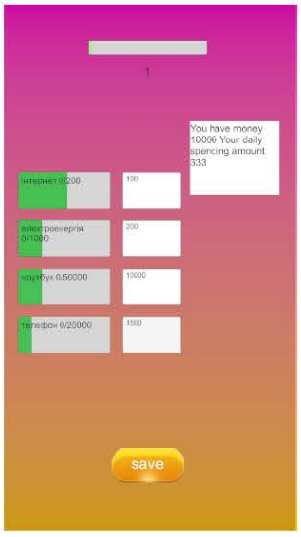

Рисунок 3.37 – Введення суми для кожного завдання

Далі натискаємо «Зберегти». Після цих дій користувач може спостерігати як зменшилась загальна сума, та оновилися дані про прогрес оплати цілі для кожного завдання та номер дня у верхній частині екрану (рис. 3.38).

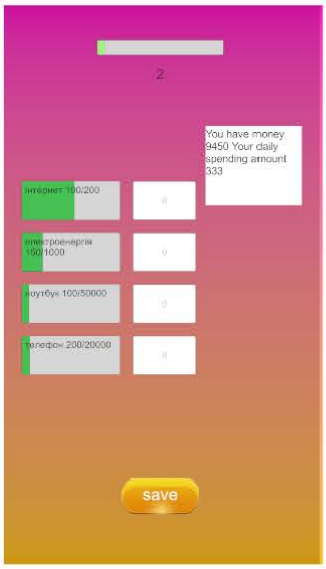

Рисунок 3.38 – Оновлення даних завдань

Спробуємо внести всю суму для одного завдання. Після кнопки «Зберегти» отримаємо виведення досягнення та спостерігаємо, що кількість завдань зменшилась (рис. 3.39).

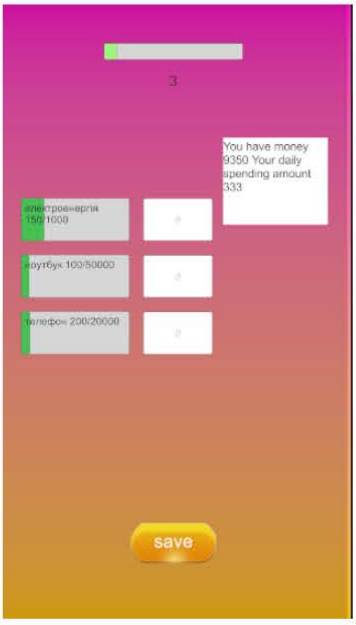

Рисунок 3.39 – Виведення досягнення

Тепер спробуємо виконати всі завдання. Після виконання всіх завдання раніше зазначеного терміну, виводиться відповідне повідомлення:

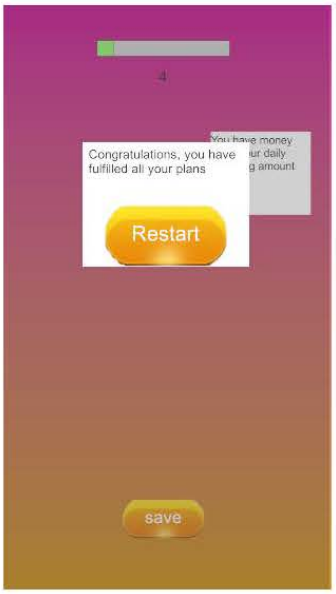

Рисунок 3.40 – Виведення кінцевого повідомлення

3.5. Висновки до третього розділу

Розробка експертної системи для керування особистими фінансами демонструє значний прогрес у використанні сучасних технологічних рішень для покращення фінансового добробуту користувачів. Розробка такої системи з використанням мови програмування C# та інтеграцією з Unity для кросплатформенної розробки дозволила створити візуально привабливий та інтуїтивно зрозумілий інтерфейс, що є ключовим аспектом для ефективного взаємодії з користувачами. Використання експертних систем у сфері особистих фінансів відкриває нові можливості для розвитку цифрових фінансових послуг і забезпечує значний внесок у фінансову незалежність та стабільність користувачів.

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці крос-платформного застосунку для управління та контролю за фінансами, що є актуальною задачею в умовах сучасної економічної нестабільності. Основна мета роботи полягала у створенні зручного та ефективного інструменту для планування і контролю особистих фінансів, який би допомагав користувачам більш ефективно розпоряджатися своїми грошовими потоками, досягати фінансової стабільності та підвищувати якість життя.

У першому розділі роботи було проведено аналіз існуючих програмних застосунків для управління фінансами. Було встановлено, що більшість з них забезпечують зручні інструменти для контролю за фінансами, але потребують удосконалення в частині інтеграції та крос-платформності. Було виявлено, що основними характеристиками успішних фінансових додатків є крос-платформність, можливість інтеграції з різними пристроями та доступ до фінансової інформації в режимі реального часу. Особлива увага була приділена технологіям та засобам розроблення крос-платформних застосунків, зокрема використанню React Native, Xamarin та Flutter, що значно скорочують час розробки і забезпечують стабільну роботу на різних операційних системах. Було також розглянуто приклади успішних фінансових додатків, таких як Personal Capital, Quicken та PocketGuard, які надають користувачам широкий спектр функцій для відстеження доходів та витрат, планування бюджету та управління інвестиціями.

Другий розділ роботи зосереджено на аналізі підходів та методів розробки бази знань для експертних систем, що є основою для прийняття рішень у розробленому застосунку. Було розглянуто основні характеристики експертних систем, їхні переваги та недоліки, а також методи Data Mining для виявлення закономірностей у фінансових даних. Експертні системи дозволяють автоматизувати процеси аналізу та прийняття рішень, що значно підвищує ефективність та точність управління фінансами. Особлива увага приділялась

використанню дерев рішень як способу представлення закономірностей Data Mining, що дозволяє наочно і зрозуміло структурувати інформацію та приймати обґрунтовані рішення.

У третьому розділі роботи було детально описано процес розробки самого застосунку. Основні етапи включали проектування архітектури системи, розробку інтерфейсу користувача та тестування застосунку. Для реалізації застосунку використовувалась мова програмування C# та платформа Unity, що забезпечило високу продуктивність та зручність використання. Unity надає потужні засоби для візуалізації та створення інтерактивних елементів інтерфейсу, що робить додаток привабливим і зручним для користувачів. Було створено інтуїтивно зрозумілий інтерфейс, який включає функції моніторингу доходів і витрат, бюджетування, системи рекомендацій та нагадувань.

Результати тестування показали, що розроблений застосунок є ефективним інструментом для управління особистими фінансами. Він дозволяє користувачам більш свідомо підходити до планування своїх фінансових операцій, що в свою чергу сприяє підвищенню їх фінансової стабільності та добробуту. Застосунок забезпечує зручний доступ до фінансової інформації в режимі реального часу, дозволяє легко планувати і контролювати витрати, а також надає персоналізовані рекомендації для досягнення фінансових цілей.

Водночас, було виявлено деякі недоліки, зокрема неінформативний рейтинг користувача та відсутність синхронізації з календарем і банківською карткою. Для їх усунення запропоновано розширити функціонал рейтингу, додаючи деталі про досягнуті цілі та суми, які було заощаджено, а також інтегрувати можливості синхронізації з календарем та банківськими рахунками, що дозволить автоматизувати введення фінансових даних і полегшить процес планування.

Загалом, розроблений застосунок демонструє великий потенціал для покращення фінансової грамотності та незалежності користувачів. Використання сучасних технологій та методів розробки забезпечує високу ефективність та надійність застосунку, що робить його важливим інструментом для сучасного суспільства. Застосунок не лише полегшує управління фінансами, але й сприяє

підвищенню фінансової обізнаності користувачів, допомагаючи їм ухвалювати більш обґрунтовані фінансові рішення та досягати фінансових цілей.

Подальший розвиток додатку може включати впровадження додаткових функцій, таких як інтеграція з іншими фінансовими сервісами, розширення можливостей персоналізації та аналітики, а також підвищення безпеки даних користувачів. Ці вдосконалення дозволять зробити застосунок ще більш корисним та зручним для широкого кола користувачів, сприяючи їх фінансовій стабільності та добробуту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Приклад застосунку для управління та контролю за фінансами Personal Capital. URL: https://www.personalcapital.com/

2. Приклад застосунку для управління та контролю за фінансами Quicken. URL: https://www.quicken.com/

3. Приклад застосунку для управління та контролю за фінансами PocketGuard. URL: https://pocketguard.com/

4. Приклад застосунку для управління та контролю за фінансами Wally. URL: https://www.wally.me/

5. Chechetova, N. F. & Chechetova-Terashvili, T. M. (2019). Financial literacy as a key to the success of personal finance management. World Science, 2(10(50), 14-20. http://doi.org/10.31435/rsglobal_ws/31102019/6725.

6. Babyre, O. V. (2017). Gamification as an instrument of persuasion in the context of environmental culture. Movni i kontseptualni kartyny svitu, (59), 21-26.

7. Safa, M. A. & Beheshti, S. (2018). Interactionist and Interventionist Group Dynamic Assessment (GDA) and EFL Learners' Listening Comprehension Development. Iranian Journal of Language Teaching Research, 6(3), 37–56. Retrieved from http://ijltr.urmia.ac.ir/article_120600_7a028bbb2ca75a31452b853d337b9fab.pdf.

8. Unity 3D [Електронний ресурс] – Режим доступу: https://docs.unity.com

9. Майк Гейг. Розробка ігор на Unity за 24 години, 2020. 464 с.

10. Алекс Окіта. Вивчаємо програмування на C# з Unity 3D, друге видання, 2019. 690 с.

11. Peter J.F. Lucas, Principles of Expert systems – 426 c

12. Firebase [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs

13. Що таке UML-діаграми? – [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html

14. UML 2.5 Diagrams Overview [Електронний ресурс]. – Режим доступу: https://www.uml-diagrams.org/uml-25-diagrams.html

15. Офіційний документація C# [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview