НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення

на тему: «Гра-симулятор «Життя програміста»

Виконав:
студент IV курсу, групи ТВ-13
Ушаков Віктор Віталійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
Старший викладач, Дацюк О. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант із задачі розробки симулятора:
Доцент, к.т.н., Залевська О. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:

_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____2025р.

ЗАВДАННЯ

на дипломну роботу студенту

Ушакова Віктора Віталійовича
(прізвище, ім'я, по батькові)

1. Тема роботи: Гра-симулятор «Життя програміста»

керівник роботи Оксана Антонівна Дацюк, старший викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02» червня 2025р. № 1875-с

2. Строк подання студентом роботи «09» червня 2025р.

3. Вихідні дані до роботи: фреймворк Flutter, мова програмування Dart, середовище розробки Visual Studio Code.

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): Розробка інтерактивного симулятора з геймплеєм написання коду, керування ресурсами та взаємодії з робочими локаціями, інтегрованого з профорієнтаційними тестами для оцінки технічних навичок користувача.

5. Перелік ілюстративного матеріалу: Архітектура ПЗ, схема взаємодії компонентів системи, алгоритм ПЗ, алгоритм ігрового циклу, інтерфейс програми.

6. Дата видачі завдання «31» жовтня 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	30.10.2024	Виконано
2	Дослідження предметної області	31.10.2024 – 01.11.2024	Виконано
3	Дослідження існуючих рішень	02.11.2024 – 01.12.2024	Виконано
4	Постановка вимог до проектування системи	02.12.2024 – 12.01.2025	Виконано
5	Розробка програмного продукту	13.01.2025 – 11.05.2025	Виконано
6	Тестування	12.05.2025 – 15.05.2025	Виконано
7	Захист програмного продукту	12.05.2025	Виконано
8	Оформлення дипломної роботи	19.05.2025 – 01.06.2025	Виконано
9	Передзахист	02.06.2025 – 06.06.2025	Виконано
10	Захист	16.06.2025 – 27.06.2025	Виконано

Студент _____ Віктор УШАКОВ
(підпис) (ім’я, прізвище)

Керівник роботи _____ Оксана ДАЦЮК
(підпис) (ім’я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 54 сторінки, 22 рисунків, 2 додатки та 9 посилань.

Метою роботи є розробка інтерактивного симулятора «Життя програміста» для професійної орієнтації абітурієнтів та майбутніх студентів у сфері інформаційних технологій з використанням сучасних технологій кросплатформної розробки.

Практичне значення одержаних результатів полягає в створенні інноваційного інструменту профорієнтації, який дозволяє користувачам у реалістичній ігровій формі ознайомитися з повсякденними реаліями роботи програміста, оцінити власну схильність до професії та прийняти більш усвідомлене рішення щодо вибору кар'єрного шляху в IT-сфері.

Ключові слова: симулятор, професійна орієнтація, програміст, Flutter, Dart, гейміфікація, кар'єрний розвиток, керування ресурсами, кросплатформна розробка, інтерактивне навчання.

ABSTRACT

Structure and scope of the thesis. The work contains 54 pages, 22 figures, 2 appendices and 9 references.

The purpose of the work is to develop an interactive simulator "Programmer's Life" for career guidance of applicants and future students in the field of information technology using modern cross-platform development technologies.

The practical value of the obtained results lies in creating an innovative career guidance tool that allows users to familiarize themselves with the daily realities of a programmer's work in a realistic game format, assess their own inclination to the profession and make a more informed decision about choosing a career path in the IT field.

Keywords: simulator, career guidance, programmer, Flutter, Dart, gamification, career development, resource management, cross-platform development, interactive learning.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API	Application Programming Interface – інтерфейс програмування застосунків
Dart	Об'єктно-орієнтована мова програмування, розроблена Google
Flutter	Фреймворк з відкритим кодом для розробки кросплатформених додатків
Git	Розподілена система контролю версій
IT	Інформаційні технології
JSON	JavaScript Object Notation – текстовий формат обміну даними
Junior	Початковий рівень розробника з мінімальним досвідом роботи
Middle	Середній рівень розробника з 2-5 роками досвіду
MVVM	Model-View-ViewModel – архітектурний шаблон розділення логіки
ПЗ	Програмне забезпечення
Provider	Бібліотека Flutter для керування станом додатку
QTE	Quick Time Event – міні-гра на швидку реакцію
SDK	Software Development Kit – набір засобів розробки
Senior	Старший рівень розробника з експертними знаннями та досвідом
SharedPreferences	Бібліотека для збереження простих даних на пристрої користувача
Віджет	Елемент керування
Коміт	Фіксація змін

ВСТУП

Сучасний ІТ-ринок характеризується високою конкуренцією та стрімким розвитком технологій, що вимагає від майбутніх фахівців свідомого вибору професії та розуміння специфіки роботи програміста. Одним із ефективних методів профорієнтації та ознайомлення з професією є гейміфікація – використання ігрових механік для моделювання реальних робочих процесів. Саме тому розробка симулятора «Життя програміста», що дозволяє в інтерактивній формі ознайомитись із реаліями роботи у сфері програмування, є актуальним завданням.

Симулятор «Життя програміста» розроблений насамперед для абітурієнтів та майбутніх студентів, щоб допомогти їм краще визначитися, чи підходить їм професія програміста. Гра моделює реалії робочого процесу (написання коду, керування енергією та стресом, взаємодія з колегами), забезпечує інтерактивний підхід до профорієнтації та сприяє саморозвитку користувачів через ігрові механіки.

Програмний продукт розроблено з використанням фреймворку Flutter, що дозволяє створювати кросплатформні додатки, які працюють як на мобільних пристроях, так і у браузерах. Це забезпечує широку доступність симулятора для різних категорій користувачів незалежно від використовуваних ними пристроїв.

1 ЗАДАЧА РОЗРОБКИ СИМУЛЯТОРА «ЖИТТЯ ПРОГРАМІСТА»

1.1 Постановка задачі

Головною метою розробки симулятора «Життя програміста» є створення інтерактивного ігрового середовища, яке дозволить абітурієнтам і майбутнім студентам у цікавій формі ознайомитися з основними аспектами роботи програміста та оцінити власні схильності до цієї професії. Симулятор має реалістично відображати робочі процеси, забезпечувати можливість керування ресурсами і пропонувати різноманітні сценарії розвитку кар'єри.

Передусім планується створення інтерактивного профорієнтаційного тесту, який допоможе користувачам оцінити свою схильність до роботи програмістом на основі особистісних характеристик, системного мислення та здатності вирішувати логічні задачі.

Важливим компонентом стане система моделювання кар'єрного росту від Junior до Senior розробника з відповідною прогресією складності завдань та рівня відповідальності. Паралельно буде реалізовано механіку керування ресурсами, що включає енергію, стрес та прогрес коду, відображаючи необхідний баланс між продуктивністю, особистим благополуччям та результатами роботи.

Для створення реалістичного досвіду передбачається розробка різноманітних локацій, таких як робоче місце, кімната відпочинку та конференц-зал, які відтворюють типове робоче середовище програміста й дозволяють користувачам взаємодіяти з віртуальними колегами. Доповнюватиме цей досвід система непередбачуваних подій та криз, що вимагатимуть від користувача прийняття рішень в умовах обмеженого часу, моделюючи реальні стресові ситуації в роботі програміста.

Технічна реалізація проекту передбачає створення інтуїтивно зрозумілого користувацького інтерфейсу, адаптованого для різних платформ, включаючи

мобільні пристрої та веб. Також буде впроваджено систему збереження прогресу та статистики проходження для аналізу результатів та надання корисних рекомендацій користувачу.

Основною цільовою аудиторією симулятора є абітурієнти та школярі старших класів, які розглядають можливість вступу на IT-спеціальності, але не мають чіткого уявлення про специфіку роботи програміста. Крім того, гра може бути корисною для першокурсників, які хочуть краще зрозуміти свої перспективи в обраній професії, та для всіх, хто цікавиться сферою програмування.

Симулятор повинен надавати користувачам можливість «приміряти» на себе роль програміста, випробувати віртуальний досвід професійної діяльності і на основі цього зробити більш усвідомлений вибір щодо свого майбутнього навчання та кар'єри.

1.2 Особливості професії програміста в контексті симулятора

При розробці симулятора було важливо відобразити ключові особливості професії програміста, які впливають на повсякденну роботу та загальний рівень задоволеності спеціалістів.

Симулятор концентрується на керуванні ресурсами, адже робота програміста вимагає постійного балансування між енергією та стресом. Тривала розумова діяльність призводить до втоми, а технічні проблеми та дедлайни підвищують рівень стресу. У симуляторі це відображено через відповідні шкали ресурсів, які динамічно змінюються залежно від дій гравця.

Важливим аспектом є моделювання кар'єрного зростання, оскільки в професії програміста чітко виділяються три основні етапи розвитку – Junior, Middle та Senior. Кожен етап характеризується різним рівнем складності задач, відповідальності та очікувань від роботодавця. Симулятор моделює поступовий перехід між цими етапами, демонструючи зростання вимог та винагород.

Галузь програмування відрізняється стрімким розвитком технологій, що вимагає від спеціалістів постійного навчання та адаптації. У грі це відображено

через необхідність розв'язувати нові типи завдань на кожному рівні складності. Також врахована взаємодія з колегами, адже програмування – не суто індивідуальна робота, а діяльність, що часто відбувається в команді. Симулятор включає елементи командної взаємодії через механіки допомоги колегам, перевірка коду та участі в нарадах, що впливають на загальну успішність проекту.

Значну частину робочого часу програміста займає вирішення різноманітних технічних проблем та усунення помилок. Гра моделює ці аспекти через випадкові події та міні-ігри на швидку реакцію, що репрезентують виправлення критичних помилок.

У симуляторі особлива увага приділяється психологічним аспектам професії, таким як керування стресом, пошук балансу між роботою та відпочинком, а також побудова ефективної комунікації в команді. Ці елементи дозволяють користувачам отримати більш повне уявлення про професію програміста не лише з технічної сторони, але й з точки зору повсякденного досвіду та емоційної складової.

1.3 Вимоги до функціональності симулятора

Для забезпечення ефективного навчального та профорієнтаційного досвіду симулятор «Життя програміста» має відповідати комплексним функціональним вимогам.

Симулятор повинен починатися з комплексного профорієнтаційного тестування для визначення схильності користувача до професії програміста. Тест має оцінювати такі параметри як стресостійкість, здатність до командної роботи, мотивація, робоча етика та здатність до навчання. Результати тесту мають зберігатися та впливати на подальший ігровий процес.

Гра повинна забезпечувати багаторівневу систему прогресії з трьома рівнями кар'єрного розвитку (Junior, Middle, Senior), кожен з яких складається з семи ігрових днів. Складність завдань та інтенсивність випадкових подій повинні зростати з кожним рівнем. Нові рівні мають розблоковуватися після успішного проходження попередніх.

Важливим елементом є система ресурсів, яка включає три основні показники – енергію, стрес та прогрес коду. Енергія зменшується при виконанні завдань, стрес збільшується через випадкові події та дедлайни, а прогрес коду відображає просування до завершення завдання. Досягнення 100% прогресу коду є основною умовою проходження ігрового дня.

Симулятор має містити різноманітні локації, мінімум три різні (робоче місце, кімната відпочинку, конференц-зал), кожна з яких пропонує унікальні механіки взаємодії та способи відновлення ресурсів чи підвищення прогресу. Під час ігрового процесу повинні виникати непередбачувані події, такі як критичні помилки в коді, термінові наради, зміни вимог, що вимагають миттєвої реакції гравця та впливають на ресурси і загальний прогрес.

Гра повинна надавати можливість використання тимчасових підсилень, таких як допомога колеги, енергетичний напій, інструмент для виправлення коду, медитація та оцінка коду, для покращення певних параметрів. Кожне підсилення може бути використане лише один раз на рівень.

Кожен ігровий день має обмеження часу, протягом якого гравець має досягти 100% прогресу рівня. Важливою складовою є інтерактивна механіка імітації написання коду через введення тексту або натискання клавіш, що впливає на прогрес та споживання ресурсів.

Симулятор повинен відстежувати та відображати детальну статистику проходження, включаючи кількість написаних рядків коду, точність введення, кількість помилок, та інші параметри успішності. Також має бути забезпечено автоматичне збереження загального прогресу користувача, включаючи розблоковані рівні, результати тестів та досягнення.

Графічний інтерфейс повинен бути інтуїтивно зрозумілим, відображати всі необхідні показники та забезпечувати легку навігацію між різними екранами та локаціями. Додаток має характеризуватися адаптивним дизайном, коректно працювати на різних пристроях і забезпечувати однаково комфортний досвід як на мобільних платформах, так і у вебверсії.

Реалізація вказаних функціональних вимог забезпечить створення повноцінного симулятора, який надає користувачам реалістичний досвід роботи програміста та допоможе їм оцінити власну схильність до цієї професії.

Висновки до розділу 1

У першому розділі було розглянуто задачу розробки симулятора «Життя програміста» як інноваційного інструменту профорієнтації для абітурієнтів та майбутніх студентів IT-спеціальностей.

Постановка задачі визначила головну мету проекту – створення інтерактивного ігрового середовища, яке дозволить користувачам у цікавій формі ознайомитися з основними аспектами роботи програміста та оцінити власні схильності до цієї професії. Було встановлено, що симулятор має забезпечувати реалістичне відображення робочих процесів, можливість керування ресурсами та різноманітні сценарії розвитку кар'єри.

Аналіз особливостей професії програміста в контексті симулятора показав необхідність моделювання ключових аспектів діяльності, включаючи керування балансом між енергією та стресом, кар'єрне зростання через етапи Junior-Middle-Senior, постійне навчання та адаптацію до нових технологій, командну взаємодію та вирішення технічних проблем. Особливу увагу було приділено психологічним аспектам професії, таким як керування стресом та пошук балансу між роботою і відпочинком.

Визначені вимоги до функціональності симулятора охоплюють комплексне профорієнтаційне тестування, багаторівневу систему прогресії з трьома етапами кар'єрного розвитку, систему керування ресурсами (енергія, стрес, прогрес коду), різноманітні локації для взаємодії, систему випадкових подій та криз, можливість використання тимчасових підсилень, обмеження часу для кожного ігрового дня та детальну статистику проходження.

2 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ ТА АНАЛОГІВ

2.1 GameDev Tycoon

GameDev Tycoon – симулятор, в якому гравець керує власною компанією з розробки відеоігор, що проілюстровано на рисунку 2.1 [1]. Гравець приймає рішення щодо створення ігор, визначає жанри, розробляє стратегію просування та управляє ресурсами компанії.

Рисунок 2.1 – Інтерфейс гри GameDev Tycoon

Сильні сторони GameDev Tycoon:

- глибока симуляція управлінських процесів, що дозволяє гравцю відчути себе керівником IT-компанії;
- детальний аналіз ринку відеоігор, що стимулює стратегічне мислення та розуміння бізнес-процесів;
- можливість розвивати компанію від невеликого стартапу до великого гравця індустрії, що демонструє еволюцію IT-бізнесу;
- реалістичне моделювання технологічного прогресу та його впливу на розробку.

Обмеження для профорієнтації програміста:

- основна увага приділяється керуванню бізнес-процесами, а не безпосередньо професійним навичкам програмування;
- відсутність елементів тестування технічних компетенцій чи керування особистими ресурсами (стрес, енергія);
- фокус на підприємницьких аспектах замість повсякденних реалій роботи програміста;
- недостатня увага до соціальних аспектів роботи в команді розробників.

2.2 Software Inc.

Software Inc. – симулятор, де гравець створює та управляє компанією з розробки програмного забезпечення [2]. Гра включає елементи керування персоналом, розробки програмних продуктів, маркетингу та фінансів, що можна побачити на рисунку 2.2.

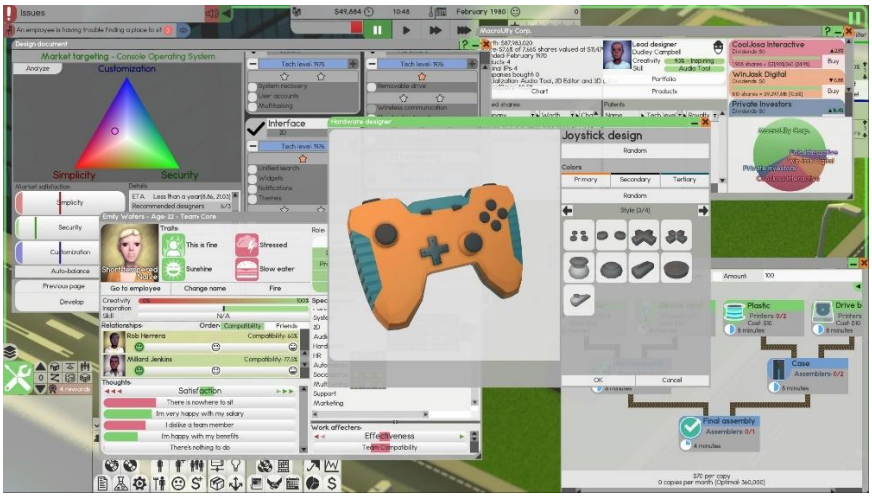


Рисунок 2.2 – Робочий процес у Software Inc.

Сильні сторони Software Inc.:

- детальне моделювання процесів розробки програмного забезпечення, включаючи планування проектів та керування співробітниками;
- наявність сценаріїв для вирішення кризових ситуацій, що імітують реальний тиск на програмістів;

- можливість налаштування робочих процесів та методологій розробки;
- реалістична симуляція життєвого циклу розробки ПЗ від концепції до релізу.

Обмеження для профорієнтації програміста:

- основний акцент робиться на керуванні компанією, а не на розвитку особистих професійних навичок;
- відсутність інтегрованих тестів для оцінки програмістських знань чи керування стресом;
- недостатнє відображення повсякденних завдань програміста (написання коду, дебаг, code review);
- фокус на стратегічному плануванні замість операційної діяльності розробника.

2.3 CodeCombat

CodeCombat є освітньою платформою, яка використовує ігрові механіки для навчання програмуванню [3]. Гравці керують персонажами в фантастичному світі, знімок екрану ігрового процесу зазначено на рисунку 2.3, написавши код для виконання різних завдань та подолання перешкод.

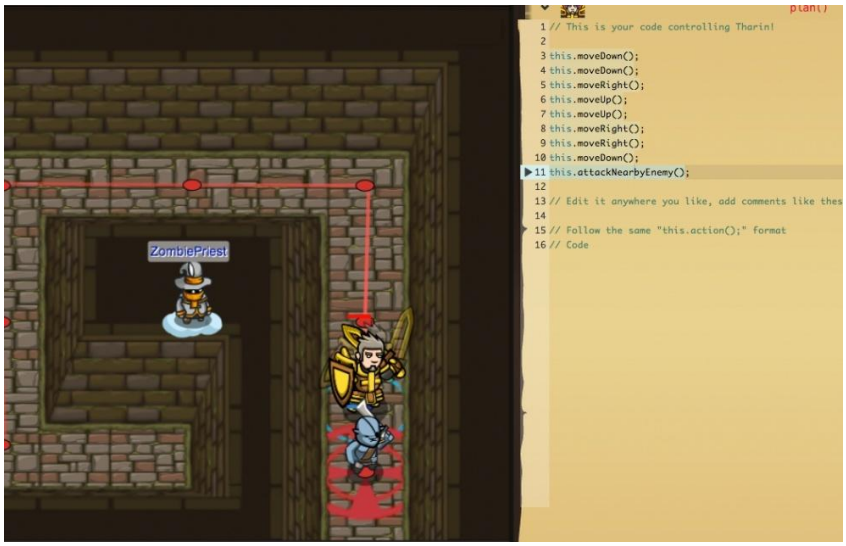


Рисунок 2.3 – Ігровий інтерфейс CodeCombat

Сильні сторони CodeCombat:

- захоплюючий ігровий процес, що робить навчання програмуванню цікавим та мотивуючим;
- підтримка множини мов програмування (Python, JavaScript, Java, C++);
- поступове ускладнення завдань від базових концепцій до складних алгоритмів;
- візуальний зворотний зв'язок – гравець бачить результат виконання свого коду в реальному часі;
- можливість вивчення як основ програмування, так і поглиблених тем (веброзробка, ігрова розробка).

Обмеження для профорієнтації програміста:

- фокус виключно на написанні коду без моделювання інших аспектів роботи програміста;
- відсутність симуляції робочого середовища та корпоративної культури;
- не розглядаються питання керування часом, стресом та energy management;
- відсутні елементи командної роботи та взаємодії з колегами;
- не відображає реальні виклики професії програміста поза технічними навичками.

2.4 CheckiO

CheckiO – ігрова платформа, інтерфейс якої зображений на рисунку 2.4, для вивчення програмування, де користувачі розв'язують задачі на Python та TypeScript у формі головоломок та викликів [4]. Платформа організована як острови з різними типами завдань.

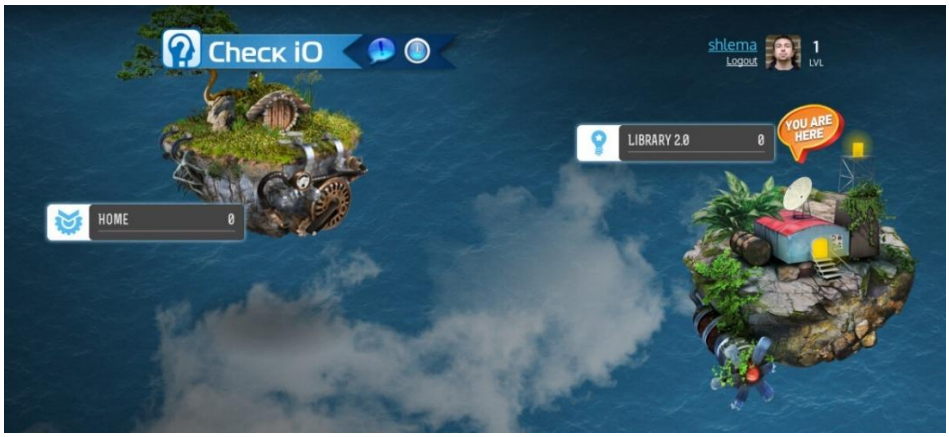


Рисунок 2.4 – Платформа CheckiO для вивчення Python

Сильні сторони CheckiO:

- структурований підхід до навчання з чітким поділом на рівні складності;
- спільнота розробників, де можна обговорювати рішення та вчитися у інших;
- можливість порівняння власних рішень з рішеннями інших учасників;
- різноманітні типи завдань від простих алгоритмів до складних математичних проблем;
- система рейтингу та досягнень, що стимулює до продовження навчання.

Обмеження для профорієнтації програміста:

- обмежена кількість мов програмування (лише Python та TypeScript);
- відсутність моделювання реальних робочих сценаріїв та проектів;
- фокус на алгоритмічних задачах без урахування інших аспектів розробки ПЗ;
- не розглядаються питання архітектури, дизайну коду та best practices;
- відсутність симуляції стресових ситуацій та керування ресурсами.

2.5 Унікальні особливості симулятора «Життя програміста»

На відміну від розглянутих аналогів, розроблений симулятор «Життя програміста» поєднує кілька унікальних особливостей, що роблять його особливо ефективним для професійної орієнтації:

Інтеграція профорієнтаційного тестування безпосередньо в ігровий процес дозволяє не тільки оцінити схильність до професії, але й врахувати результати в подальшому проходженні гри, створюючи персоналізований досвід для кожного користувача.

Реалістичне моделювання балансу між продуктивністю (прогрес коду), особистим благополуччям (енергія) та психічним здоров'ям (рівень стресу) відображає реальні виклики професії програміста, з якими стикаються фахівці у повсякденній роботі.

Чітка кар'єрна прогресія через три етапи професійного розвитку (Junior, Middle, Senior) з відповідним зростанням складності завдань та інтенсивності непередбачуваних подій дає користувачам розуміння шляху професійного розвитку.

Акцент на соціальних аспектах професії через механіки взаємодії з віртуальними колегами, участь у нарадах та code review розвиває не лише технічні, але й комунікативні навички, важливі для успішної кар'єри.

Система випадкових подій та криз, що вимагають швидкого реагування та прийняття рішень, моделює реальні стресові ситуації в роботі програміста, допомагаючи користувачам оцінити свою готовність до таких викликів.

Різноманітні локації (робоче місце, кімната відпочинку, конференц-зал) дозволяють гравцю експериментувати з різними стратегіями керування ресурсами та продуктивністю, відображаючи важливість балансу особистого життя та роботи.

Висновки до розділу 2

Проведений аналіз існуючих аналогів показав, що більшість з них зосереджується або на керуванні бізнес-процесами (GameDev Tycoon, Software

Inc.), або на розвитку конкретних технічних навичок (CodeCombat, CheckiO). Проте жоден з них не пропонує комплексного підходу до профорієнтації, який би поєднував симуляцію робочого процесу програміста, керування ресурсами, кар'єрний розвиток та соціальну взаємодію.

GameDev Tycoon та Software Inc., хоча і надають глибоке розуміння бізнес-аспектів IT-індустрії, не розкривають повсякденні реалії роботи програміста на операційному рівні. CodeCombat надає захоплюючий досвід навчання програмуванню через ігрові механіки, але обмежується виключно технічними аспектами. CheckiO пропонує структурований підхід до вирішення алгоритмічних задач, проте не моделює реальне робоче середовище.

Усі розглянуті аналоги мають спільну проблему – вони не враховують психологічні та соціальні аспекти роботи програміста, такі як керування стресом, взаємодія в команді, прийняття рішень в умовах дедлайнів та балансування між різними проектами.

Розроблений симулятор «Життя програміста» заповнює цю прогалину, пропонуючи користувачам можливість у реалістичній ігровій формі випробувати різні аспекти професії програміста та оцінити власну схильність до цієї сфери діяльності. Це особливо цінно для абітурієнтів та майбутніх студентів, які стоять перед вибором професійного шляху.

Унікальність симулятора полягає в тому, що він не обмежується лише технічними аспектами програмування або управлінськими процесами, а пропонує цілісне бачення професії, включаючи керування стресом, енергією, взаємодію з колегами та розвиток кар'єри. Це дозволяє користувачам отримати більш глибоке розуміння повсякденних реалій роботи програміста та зробити більш усвідомлений вибір щодо свого майбутнього.

Таким чином, симулятор «Життя програміста» є інноваційним і актуальним інструментом профорієнтації, що відповідає сучасним потребам освіти та ринку праці в IT-сфері, заповнюючи суттєву прогалину в існуючих рішеннях для професійної орієнтації майбутніх програмістів.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Фреймворк Flutter

Для розробки мобільного додатку «Життя Програміста» використано фреймворк Flutter – сучасну технологію з відкритим кодом від Google для створення кросплатформних додатків [5]. Flutter дозволяє розробляти програмне забезпечення для різних платформ (Android, iOS, Windows) з використанням єдиної кодової бази, що значно прискорює процес розробки та спрощує підтримку продукту.

Flutter працює на основі мови програмування Dart і використовує підхід, заснований на віджетах – будівельних блоках для створення користувацького інтерфейсу. Кожен віджет представляє собою незмінний опис частини користувацького інтерфейсу. Flutter відрізняється від інших фреймворків тим, що не використовує нативні компоненти операційної системи, а малює кожен елемент інтерфейсу самостійно, використовуючи власний графічний рушій Skia.

Основні переваги використання Flutter для реалізації проекту:

- швидкий рендеринг інтерфейсу з використанням власного графічного рушія skia [6];
- багата бібліотека віджетів, які забезпечують нативний вигляд на різних платформах;
- гаряче перезавантаження (hot reload) для миттєвого відображення змін у процесі розробки;
- висока продуктивність додатку завдяки компіляції в нативний код;
- можливість створення кастомних анімацій і складних інтерфейсів;
- активна підтримка від спільноти розробників та google.

Flutter використовує декларативний підхід до побудови інтерфейсу, що означає, що розробники описують, як інтерфейс має виглядати для кожного стану

додатку, а фреймворк самостійно оновлює відображення при зміні стану. Це спрощує розробку реактивних інтерфейсів і зменшує кількість помилок.

Архітектура Flutter включає три ключових шари:

1) фреймворк flutter (написаний на dart) – надає api високого рівня для розробників;

2) рушій flutter (написаний на c++) – відповідає за рендеринг, анімації та жести;

3) embedder – інтегрує flutter в операційну систему пристрою.

У проекті «Життя Програміста» активно використовуються багато компонентів Flutter, зокрема:

- material design віджети для стандартних елементів інтерфейсу;
- stateful віджети для компонентів з динамічним станом;
- анімації для забезпечення плавних переходів;
- система навігації для переходів між екранами;
- обробка жестів для реалізації взаємодії з користувачем.

3.2 Мова програмування Dart

Проект розроблено з використанням мови програмування Dart, яка є основною мовою для розробки Flutter-додатків [7]. Dart – сучасна об'єктно-орієнтована мова програмування з відкритим кодом, розроблена компанією Google.

Dart поєднує в собі найкращі особливості різних мов програмування. Вона має C-подібний синтаксис, що робить її знайомою для розробників, які працювали з Java, C++, JavaScript або C#. Водночас, Dart включає сучасні функції, такі як null safety, pattern matching та сильну типізацію.

Ключові особливості Dart, які були використані в проекті:

- система типів з підтримкою статичної та динамічної типізації;
- підтримка асинхронного програмування з використанням future та async/await;

- ефективна робота з колекціями даних;
- зручний синтаксис для роботи з функціями та класами;
- null safety для запобігання помилкам, пов'язаним з null;
- вбудована підтримка для серіалізації та десеріалізації json;
- генерація коду та анотації для зменшення шаблонного коду;
- каскадний оператор (..) для послідовного виклику методів на одному об'єкті;
- розширення функціональності існуючих класів за допомогою extension methods;
- мікси (mixins) для повторного використання коду без множинного наслідування.

Dart має ефективну систему керування пакетами (pub.dev), що дозволяє легко інтегрувати сторонні бібліотеки в проект. У нашому додатку було використано кілька популярних пакетів, які значно розширили функціональність базового Flutter.

Dart також має потужний компілятор, який може виконувати JIT (Just-In-Time) компіляцію під час розробки для швидкого циклу розробки і AOT (Ahead-Of-Time) компіляцію для випуску додатку з оптимальною продуктивністю.

Мова має вбудовану підтримку для ізоляцій (isolates) – спосіб реалізації багатопотоковості в Dart, який дозволяє виконувати паралельні обчислення без блокування основного потоку інтерфейсу.

3.3 Бібліотека Provider для керування станом

Для керування станом додатку та оптимізації обміну даними між компонентами використано бібліотеку Provider [8]. Ця бібліотека реалізує патерн «Provider» та спрощує керування даними і станом програми, а також забезпечує ефективне оновлення інтерфейсу при зміні даних.

Provider – це рішення для керування станом, яке базується на InheritedWidget з Flutter SDK, але надає більш зручний API. Він дозволяє передавати дані через дерево віджетів без необхідності вручну передавати їх через конструктори.

3.4 Бібліотека SharedPreferences для зберігання даних

Для зберігання прогресу гравця та налаштувань додатку використано бібліотеку SharedPreferences, яка надає доступ до постійного сховища даних на пристрої користувача [9]. SharedPreferences є легковаговим рішенням для зберігання пар ключ-значення, призначеним для збереження невеликих обсягів даних, які повинні залишатися доступними між запусками додатку. Дана бібліотека використовує нативні механізми зберігання даних платформи: на Android це SharedPreferences API, на iOS – NSUserDefaults, а для вебверсій – localStorage.

Головними перевагами використання SharedPreferences у проекті є простий API, підтримка різноманітних типів даних включно з рядками, числами, булевими значеннями та списками, можливість асинхронного доступу до даних та автоматична синхронізація з нативними сховищами цільової платформи. Завдяки впровадженню цієї бібліотеки в проекті «Життя Програміста» стало можливим ефективне збереження прогресу гравця щодо проходження рівнів та розблокованих досягнень, результатів орієнтаційного тесту, різноманітних налаштувань гри, а також статистики та метрик гравця. Особливо важливою функцією SharedPreferences стало забезпечення стійкості даних при перезапуску додатку, що критично для підтримки безперервного ігрового досвіду.

У проекті дана бібліотека реалізована через набір методів для збереження та завантаження даних, що дозволяє абстрагувати роботу зі сховищем від основної логіки додатку. Такий підхід не лише спрощує розробку, але й підвищує надійність системи збереження прогресу, що є однією з ключових вимог до будь-якого ігрового додатку. Для збереження складних об'єктів додатково використовується серіалізація в JSON-формат, що робить систему збереження даних ще більш гнучкою.

3.5 Система контролю версій Git

Для керування версіями коду використано систему контролю версій Git, яка дозволяє відстежувати зміни у вихідному коді та забезпечує ефективну співпрацю розробників. Git зберігає повну історію змін проекту, надаючи можливість повертатися до попередніх версій, порівнювати їх та ідентифікувати причини помилок, а також спрощує паралельну розробку через систему гілок.

У процесі розробки проекту «Життя Програміста» застосовувалися основні можливості Git: локальні репозиторії для зберігання історії, комміти для фіксації змін, гілки для розробки нових функцій та процедури злиття для їх об'єднання. Впровадження Git дозволило ефективно відстежувати зміни, організовувати паралельну розробку та безпечно експериментувати з новими функціями без ризику для основного коду.

Робота з Git здійснювалася як через вбудований у Visual Studio Code інтерфейс, так і через командний рядок. Процес розробки організовано за методологією Git Flow з використанням основних гілок: main для стабільної версії, develop для інтеграції змін, feature/* для розробки нових функцій, release/* для релізів та hotfix/* для швидкого виправлення критичних помилок. Такий підхід забезпечив ефективне керування змінами та підтримку стабільності коду проекту.

Висновки до розділу 3

У третьому розділі було детально розглянуто засоби розробки програмного забезпечення симулятора «Життя програміста» та обґрунтовано вибір технологічного стеку.

Вибір фреймворку Flutter як основи для розробки виявився оптимальним рішенням завдяки його кросплатформності, високій продуктивності та багатій бібліотеці віджетів. Flutter забезпечує можливість створення додатку для різних платформ з єдиної кодової бази, що значно прискорює розробку та спрощує

подальшу підтримку продукту. Власний графічний рушій Skia гарантує швидкий рендеринг інтерфейсу та можливість створення складних анімацій.

Мова програмування Dart показала свою ефективність завдяки сучасним функціям, таким як null safety, асинхронне програмування та зручна робота з колекціями. Інтеграція Dart з Flutter забезпечує оптимальну продуктивність та зручність розробки.

Бібліотека Provider для керування станом виявилася ідеальним вибором для реалізації реактивного інтерфейсу та ефективного обміну даними між компонентами. Вона спрощує керування складним станом додатку та забезпечує прозору архітектуру.

SharedPreferences як рішення для збереження даних повністю відповідає потребам проекту, забезпечуючи надійне зберігання прогресу користувача між сеансами гри на всіх підтримуваних платформах.

Використання системи контролю версій Git дозволило ефективно управляти змінами в коді та організувати структурований процес розробки за методологією Git Flow.

Обраний технологічний стек забезпечує всі необхідні можливості для реалізації поставлених вимог, гарантує високу продуктивність, масштабованість та підтримуваність розробленого програмного продукту. Комбінація сучасних технологій дозволяє створити якісний кросплатформний додаток, який відповідає сучасним стандартам розробки мобільних застосунків.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Архітектура додатку

Архітектурно симулятор «Життя Програміста» побудовано за принципом багатошарової архітектури з використанням шаблону проектування Model-View-ViewModel (MVVM), що забезпечує чіткий розподіл відповідальності між компонентами системи та полегшує підтримку й тестування.

Архітектура системи складається з трьох основних шарів:

- рівень даних (Model) – відповідає за зберігання стану гри, обробку ігрової логіки та бізнес-правил. Включає механізми збереження та завантаження даних користувача. Реалізований через GameStateProvider, який виступає єдиним джерелом правди для всього додатку;
- рівень представлення (View) – забезпечує візуалізацію даних для користувача та обробку його дій. Включає екрани та інтерактивні компоненти, що взаємодіють з користувачем;
- рівень сполучення (ViewModel) – з'єднує Model і View, оброблюючи дані та передаючи їх між компонентами системи. Реалізований через систему Provider, яка забезпечує реактивне оновлення інтерфейсу при зміні даних.

Загальна схема архітектури додатку представлена на рисунку 4.1.

Рисунок 4.1 – Архітектура програмного застосунку

Така структура дозволяє легко орієнтуватися в проекті та підтримувати розділення відповідальності між компонентами

4.2 Основні компоненти системи

Модуль керування станом є головним компонентом системи, який керує всіма даними та логікою гри. Центральним елементом є клас GameStateProvider, що використовує шаблон проектування Provider для надання доступу до даних усім компонентам системи, що зображено на рисунку 4.2.

```
class GameStateProvider with ChangeNotifier {
  // Resources
  double _energy = 100.0;
  double _stress = 0.0;
  double _codeProgress = 0.0;

  // Game state
  CareerStage _currentStage = CareerStage.Junior;
  int _currentDay = 1;
  bool _isGameActive = false;
  bool _isPaused = false;
  DateTime? _levelStartTime;
  int _remainingSeconds = 360; // 6 minutes
  Location _currentLocation = Location.workplace;
```

Рисунок 4.2 – клас GameStateProvider

Взаємодія між компонентами системи показана на рисунку 4.3, де GameStateProvider виступає центральним елементом, з яким взаємодіють усі інші компоненти системи.

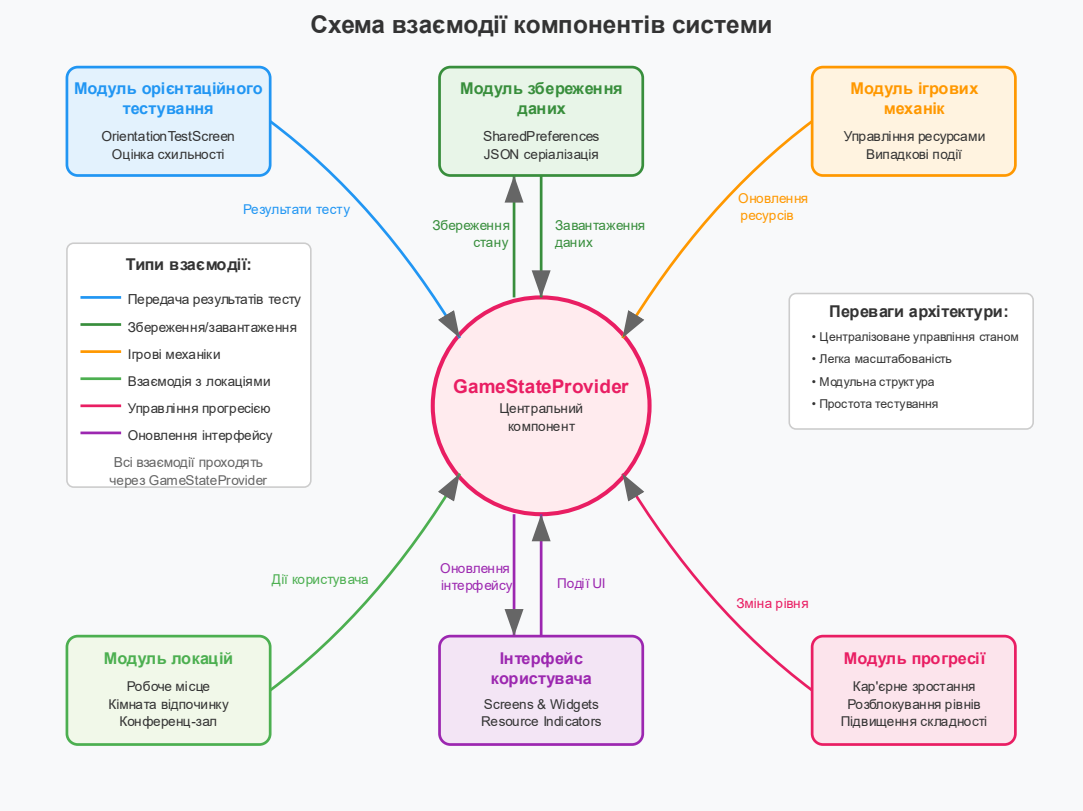


Рисунок 4.3 – Схема взаємодії компонентів системи

Модуль профорієнтаційного тестування відповідає за проведення орієнтаційного тесту, оцінку схильності користувача до професії програміста та збереження результатів. Вхідною точкою є OrientationTestScreen, який представляє інтерфейс для проходження тесту.

Модуль ігрових механік реалізує основні механіки гри. До них належать:

1) механіка керування ресурсами, що відповідає за зміну показників енергії, стресу та прогресу;

2) механіка випадкових подій і реакцій на них, що забезпечує динамічність ігрового процесу;

3) механіка написання коду та оцінки його правильності, що становить основу геймплею;

4) механіка використання підсилень, що надає додаткові можливості для подолання викликів.

Модуль локацій забезпечує логіку різних ігрових локацій та можливості взаємодії в них. У грі реалізовано три локації:

1) робоче місце, де користувач займається основною діяльністю – написанням коду;

2) кімната відпочинку, призначена для відновлення ресурсів персонажа;

3) конференц-зал, що слугує для соціальної взаємодії з віртуальними колегами.

Модуль збереження даних відповідає за персистентність даних, зберігаючи прогрес користувача між сеансами гри, код якого зображений на рисунку 4.4. Використовує бібліотеку SharedPreferences для роботи з локальним сховищем.

```
Future<void> saveState() async {
  final prefs = await SharedPreferences.getInstance();

  // Save basic game state
  prefs.setBool('testCompleted', _testCompleted);
  prefs.setInt('currentStage', _currentStage.index);
  prefs.setInt('currentDay', _currentDay);
  prefs.setDouble('energy', _energy);
  prefs.setDouble('stress', _stress);
  prefs.setDouble('codeProgress', _codeProgress);
```

Рисунок 4.4 – модуль збереження стану

Модуль прогресії керує системою кар'єрного зростання, розблокуванням рівнів та підвищенням складності з прогресом гравця.

4.3 Алгоритм роботи системи

Алгоритм роботи системи симулятора «Життя програміста» складається з двох основних частин: загального алгоритму функціонування додатку та детального алгоритму ігрового циклу, які забезпечують повний цикл взаємодії користувача з програмою.

4.3.1 Загальний алгоритм роботи системи

Загальний алгоритм роботи системи представлено на рисунку 4.5, який демонструє послідовність дій від запуску додатку до завершення роботи.

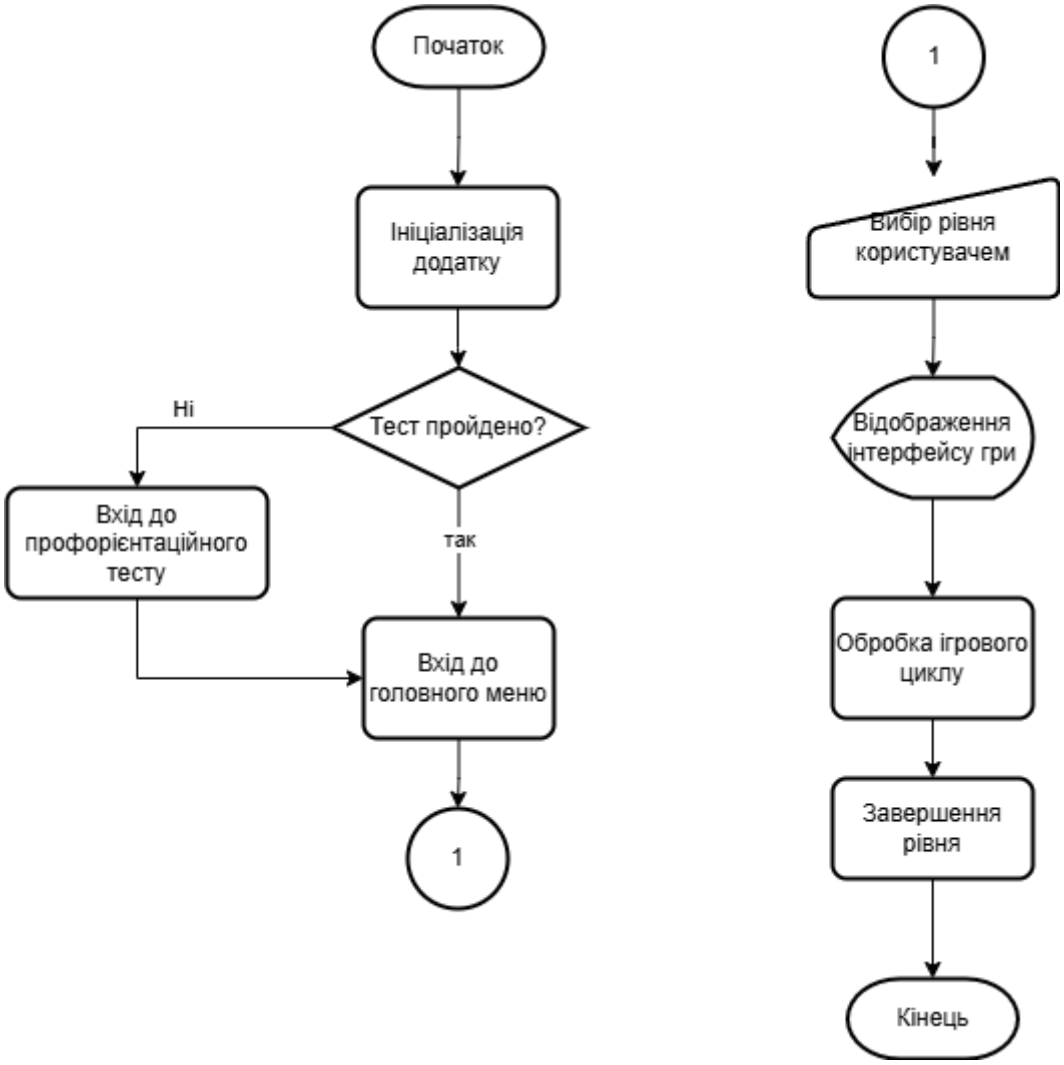


Рисунок 4.5 – Загальний алгоритм роботи системи

Алгоритм починається з ініціалізації додатку, під час якої система завантажує збережені дані користувача через механізм SharedPreferences. Далі відбувається перевірка, чи проходив користувач профорієнтаційний тест раніше.

Якщо тест не пройдено, система автоматично спрямовує користувача на екран профорієнтаційного тестування, який складається з 10 запитань для оцінки схильності до професії програміста. Результати тесту зберігаються та впливають на подальший ігровий досвід.

Після завершення тесту або якщо тест вже було пройдено раніше, користувач потрапляє до головного меню, з якого може перейти до вибору рівня. Система підтримує три етапи кар'єрного розвитку (Junior, Middle, Senior), кожен з яких містить 7 ігрових днів.

Після вибору конкретного дня відбувається ініціалізація ігрового сеансу з встановленням початкових значень ресурсів (енергія: 100%, стрес: 0%, прогрес коду: 0%) та запуском таймера робочого дня.

Центральною частиною алгоритму є ігровий цикл, який виконується безперервно до настання умов завершення дня. Цикл включає обробку дій користувача, генерацію випадкових подій та оновлення стану гри.

Завершення ігрового дня може відбутися за однією з наступних умов: досягнення 100% прогресу коду, вичерпання часу, зменшення енергії до 0% або досягнення максимального рівня стресу (100%).

Після завершення дня система оцінює результати. У випадку успішного проходження (досягнення 100% прогресу) розблоковується наступний день або рівень. При невдачі користувач може повторити спробу.

Алгоритм передбачає можливість продовження гри з новими рівнями або завершення з відображенням фінальної статистики та збереженням прогресу.

4.3.2 Алгоритм ігрового циклу

Детальний алгоритм ігрового циклу представлено на рисунку 4.6, який показує внутрішню структуру основного ігрового процесу.

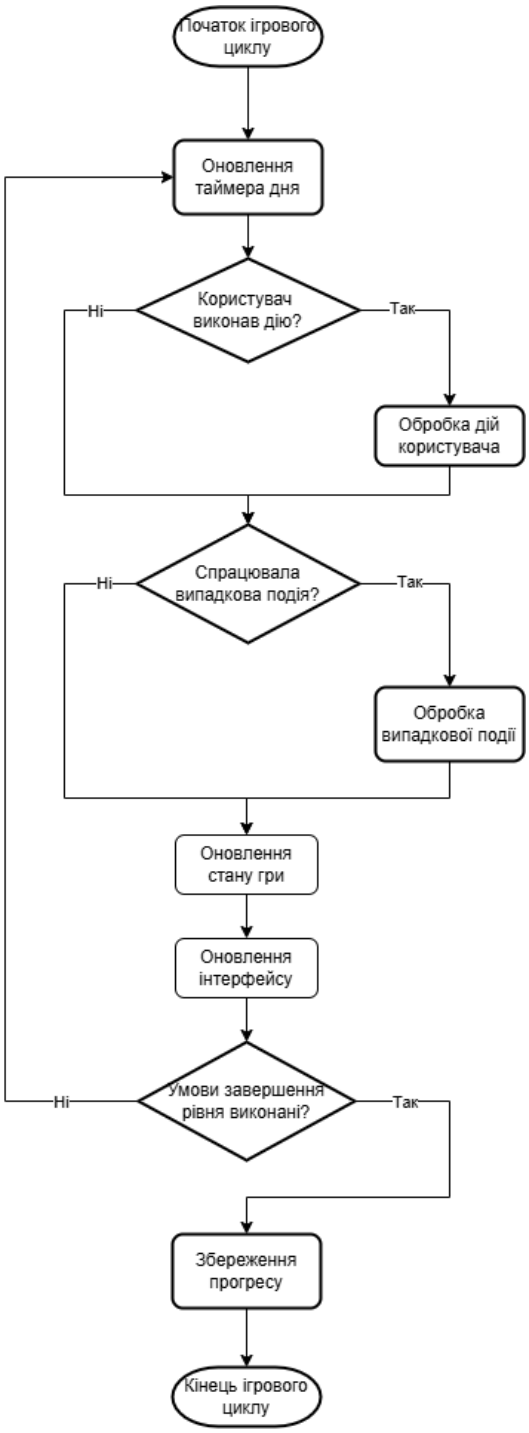


Рисунок 4.6 – Алгоритм ігрового циклу

Кожна ітерація циклу починається з оновлення таймера, який відраховує залишок часу до кінця робочого дня.

Система перевіряє наявність дій користувача, які можуть включати:

1) написання коду – збільшує прогрес та зменшує енергію;

2) дії відпочинку – відновлюють енергію та знижують стрес;

3) співпрацю з колегами – впливає на прогрес з варіаціями ресурсів.

Паралельно з обробкою дій користувача система генерує випадкові події з частотою, що залежить від рівня складності:

1) Junior рівень – події кожні 20-40 секунд;

2) Middle рівень – події кожні 15-30 секунд;

3) Senior рівень – події кожні 10-20 секунд.

Типи випадкових подій включають критичні помилки в коді (що викликають QTE міні-ігри), повідомлення про наради та нагадування про необхідність відпочинку.

Після обробки дій та подій система оновлює стан гри, включаючи всі ресурси, прогрес та внутрішні лічильники. Одночасно відбувається оновлення інтерфейсу користувача, включаючи індикатори ресурсів та журнал подій.

Наприкінці кожної ітерації циклу система перевіряє умови завершення дня. Якщо жодна з умов не виконана, цикл продовжується. При виконанні будь-якої умови завершення відбувається збереження поточного стану гри та вихід з циклу.

Ігровий цикл оптимізовано для ефективної роботи з мінімальним споживанням ресурсів системи. Автоматичне збереження відбувається кожні 30 секунд для запобігання втраті прогресу.

4.3.3 Особливості реалізації алгоритму

Алгоритм системи реалізований з урахуванням асинхронної природи Flutter фреймворку. Використовуються Timer для керування частотою подій та setState() для оновлення інтерфейсу.

Особливу увагу приділено генерації випадкових подій, які відбуваються з урахуванням поточної локації користувача та попередніх подій для забезпечення різноманітного ігрового досвіду. Алгоритм вибору події базується на імовірнісних вагах та поточному стані гри.

Система також включає механізми оптимізації продуктивності, такі як ледача ініціалізація компонентів та кешування часто використовуваних даних, що забезпечує стабільну роботу додатку на різних типах пристроїв.

Реалізований алгоритм забезпечує надійну та передбачувану роботу системи, дозволяючи користувачам отримати якісний ігровий досвід при моделюванні професійної діяльності програміста.

4.4 Реалізація основних ігрових механік

Ключові механіки, що забезпечують геймплей, реалізовані через спеціалізовані алгоритми та компоненти.

Алгоритм керування ресурсами реалізує зміну показників енергії, стресу та прогресу на основі дій користувача та подій, що відбуваються в грі. Основна логіка реалізована у методах updateEnergy(), updateStress() та updateCodeProgress() класу GameStateProvider.

Механіка випадкових подій генерує непередбачувані ситуації під час гри, що вимагають реакції користувача. Реалізована у методі _triggerRandomEvent() класу GameScreen, який викликається через певні інтервали часу залежно від рівня складності, що зображена на рисунку 4.7.

```
void _triggerRandomEvent() {
  if (!mounted) return;

  final gameState = Provider.of<GameStateProvider>(context, listen: false);

  // Don't trigger events if game is paused or on coffee break
  if (gameState.isPaused || _coffeeBreakActive) {
    _startEventTimer();
    return;
  }

  final random = Random();

  // Select random event based on probabilities
  final eventRoll = random.nextDouble();
```

Рисунок 4.7 – Метод випадкової події

Механіка QTE (Quick Time Events) реалізує міні-гру на реакцію для усунення критичних помилок. Вона включає генерацію випадкових послідовностей клавіш, відстеження введення користувача та часове обмеження.

Система прогресії забезпечує поступове збільшення складності від Junior до Senior рівня. Алгоритм включає розблокування нових днів і рівнів, а також підвищення частоти та інтенсивності випадкових подій при переході на вищі рівні складності.

Висновки до розділу 4

У четвертому розділі було детально розглянуто програмну реалізацію симулятора «Життя Програміста». Основою архітектури системи є шаблон MVVM, що забезпечує чіткий розподіл відповідальності між компонентами та полегшує підтримку коду.

Центральним компонентом системи є GameStateProvider, який керує всім ігровим процесом, зберігає стан гри та обробляє взаємодію з користувачем. Реалізовані ігрові механіки включають керування ресурсами, генерацію випадкових подій, систему QTE для реакції на критичні помилки, а також систему прогресії, що забезпечує поступове підвищення складності.

Система збереження даних, реалізована через SharedPreferences, дозволяє зберігати прогрес користувача між сеансами гри, забезпечуючи безперервність ігрового досвіду. Функціонал системи розділено на модулі, які взаємодіють між собою через GameStateProvider, що спрощує розширення та модифікацію функціональності.

Алгоритмічна складова системи забезпечує динамічний геймплей з різноманітними подіями та викликами, що залежать від рівня складності та прогресу користувача. Реалізовані механіки дозволяють користувачам отримати реалістичний досвід роботи програміста та оцінити власну схильність до цієї професії.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

5.1 Початкове знайомство та профорієнтаційний тест

Перший досвід користувача з симулятором «Життя програміста» починається з екрану завантаження (Splash Screen), який відображає назву гри та короткий опис, як показано на рисунку 5.1.

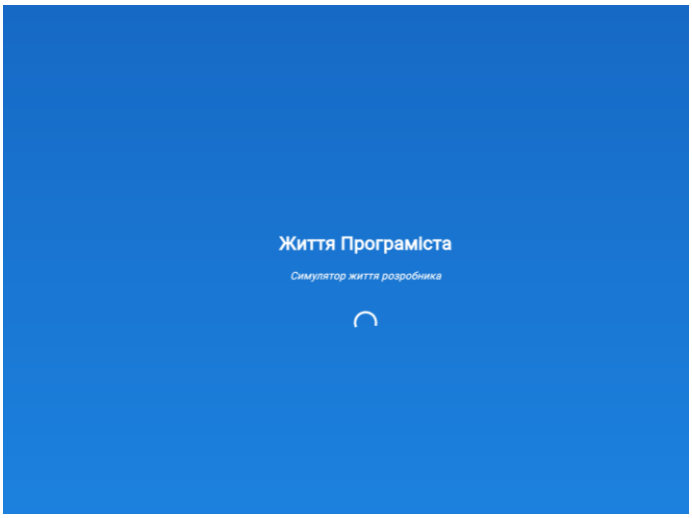


Рисунок 5.1 – Екран-заставка додатку

Після короткої анімації завантаження система перевіряє, чи проходив користувач профорієнтаційний тест раніше. Якщо тест ще не пройдено, користувач автоматично переспрямовується на екран тесту, як показано на рисунку 5.2.

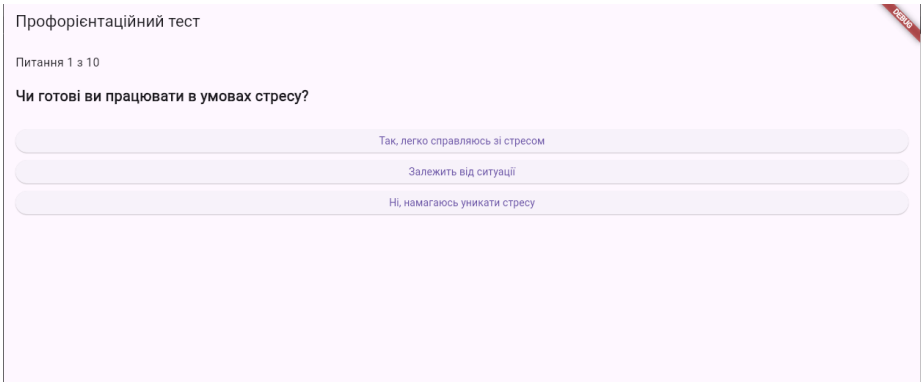


Рисунок 5.2 – Екран з питанням профорієнтаційного тесту

Профорієнтаційний тест складається з 10 запитань, які оцінюють ключові характеристики, важливі для успішної кар'єри в програмуванні. Для кожного запитання пропонується три варіанти відповіді. Користувач повинен обрати один варіант, який найкраще відповідає його особистим якостям та перевагам.

Запитання тесту охоплюють різні аспекти, такі як:

- готовність працювати в умовах стресу;
- перевага роботи в команді чи самостійно;
- важливість зарплати порівняно з цікавими задачами;
- готовність вчитися та опановувати нові технології;
- підхід до вирішення проблем (проактивний чи реактивний);
- ставлення до змін дедлайнів;
- важливість віддаленої роботи;
- ставлення до балансу роботи та особистого життя.

Після відповіді на всі запитання система аналізує результати та відображає детальний звіт, як показано на рисунку 5.3.

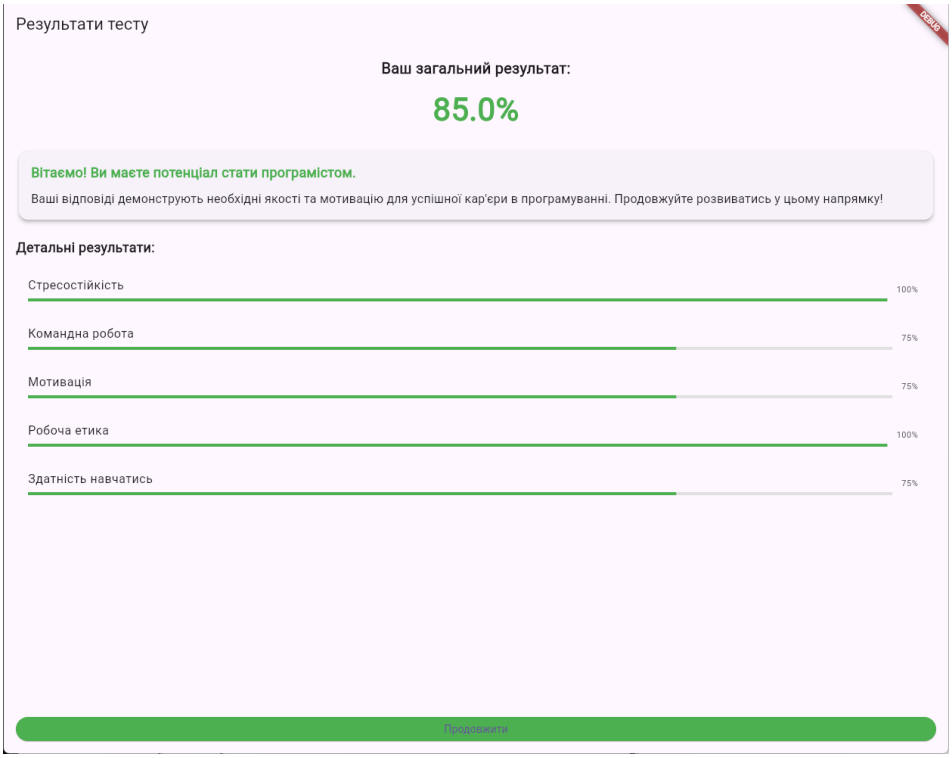


Рисунок 5.3 – Екран з результатами профорієнтаційного тесту

Звіт включає:

- загальну оцінку схильності до професії програміста (у відсотках);
- оцінки за п'ятьма ключовими показниками (стресостійкість, командна робота, мотивація, робоча етика, здатність до навчання);
- текстові рекомендації базуючись на отриманих результатах.

Якщо загальна оцінка перевищує поріг (зазвичай 60%), користувач отримує позитивний відгук, який підтверджує його потенціал у програмуванні та заохочує продовжити знайомство з професією через гру. Якщо оцінка нижча за поріг, система пропонує користувачу або спробувати пройти тест ще раз, або все одно продовжити гру, звертаючи увагу на області, які потребують розвитку.

5.2 Навігація головним меню та вибір рівня

Після проходження профорієнтаційного тесту користувач потрапляє до головного меню симулятора «Життя програміста», яке зображено на рисунку 5.4.

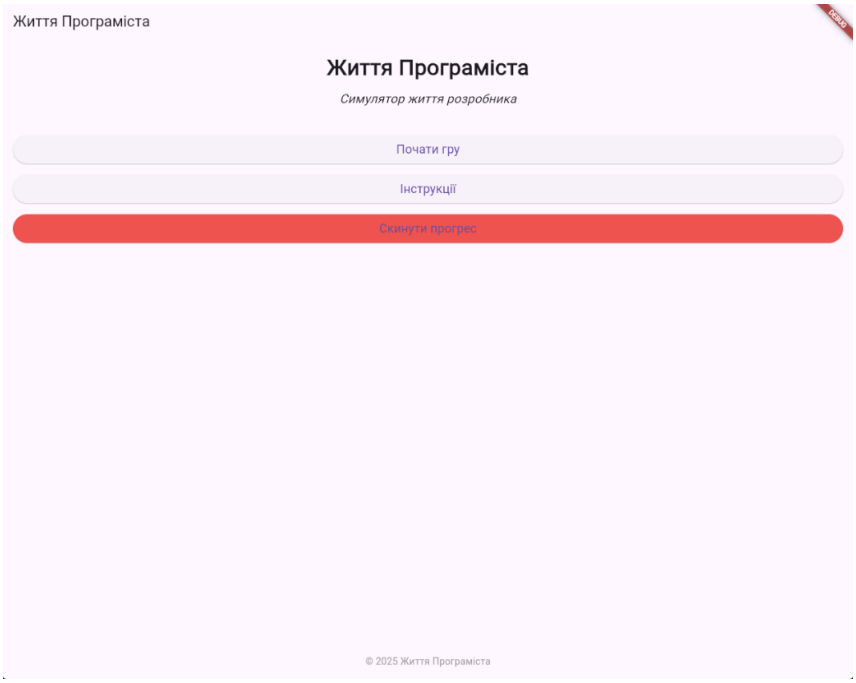


Рисунок 5.4 – Головне меню додатку

Головне меню являє собою центральний хаб навігації, з якого користувач може отримати доступ до різних розділів гри. Екран головного меню містить:

1) заголовок гри – розташований у верхній частині екрана, відображає назву «Життя Програміста» та підзаголовок «Симулятор життя розробника»;

2) кнопка «Почати гру» – основна кнопка, що веде до екрана вибору рівня, де користувач може обрати стадію кар'єри та день для проходження;

3) кнопка «Інструкції» – відкриває діалогове вікно з детальними інструкціями щодо ігрового процесу, включаючи пояснення основних механік, системи ресурсів та цілей гри;

4) кнопка «Скинути прогрес» – дозволяє користувачу повністю скинути весь ігровий прогрес та почати гру з початку. Перед скиданням прогресу система запитує підтвердження, щоб запобігти випадковій втраті даних.

При натисканні на кнопку «Почати гру» користувач переходить до екрана вибору рівня, який зображено на рисунку 5.5.

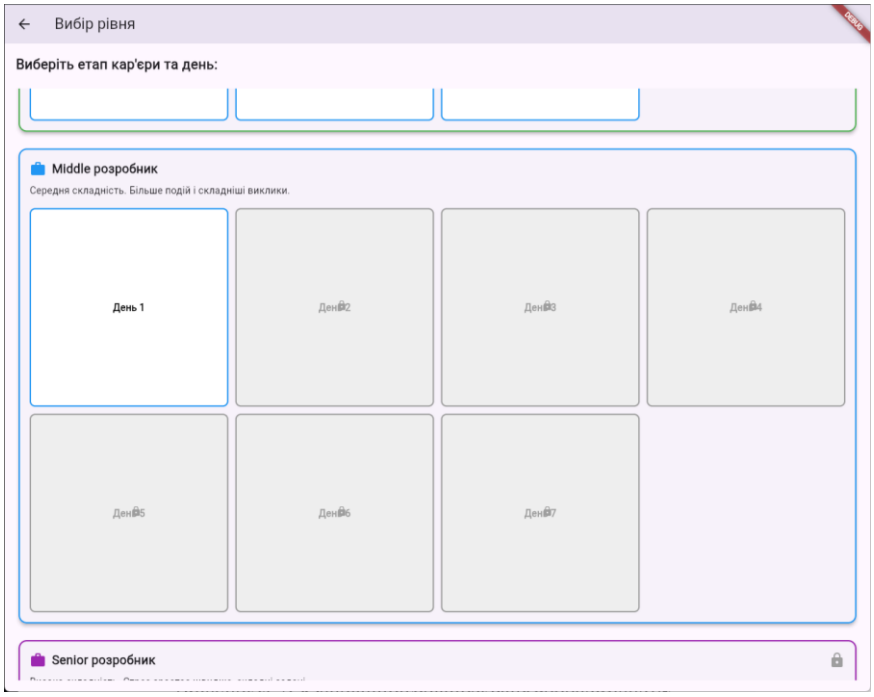


Рисунок 5.5 – Екран вибору рівня

Екран вибору рівня організований за стадіями кар'єри програміста:

1) junior-розробник – початковий етап кар'єри з найнижчим рівнем складності. Ця стадія завжди доступна з самого початку гри і складається з 7 днів. Кожен день являє собою окремий рівень з унікальними завданнями та викликами;

2) middle-розробник – середній етап кар'єри з підвищеною складністю. Ця стадія розблоковується після успішного проходження всіх 7 днів junior-рівня і також складається з 7 днів;

3) senior-розробник – найвищий етап кар'єри з максимальною складністю. Розблоковується після проходження всіх днів middle-рівня і також містить 7 днів, але з найскладнішими завданнями та інтенсивнішими випадковими подіями.

Доступні дні відображаються у вигляді сітки з 7 елементів для кожної стадії кар'єри. Дні, які ще не розблоковані, позначаються іконкою замка. Користувач може обрати будь-який доступний день, натиснувши на відповідний елемент сітки.

5.3 Ігровий процес та керування ресурсами

Основний ігровий процес симулятора «Життя програміста» відбувається на ігровому екрані, який включає три основні локації та систему керування ресурсами. Головний ігровий екран представлено на рисунку 5.6.

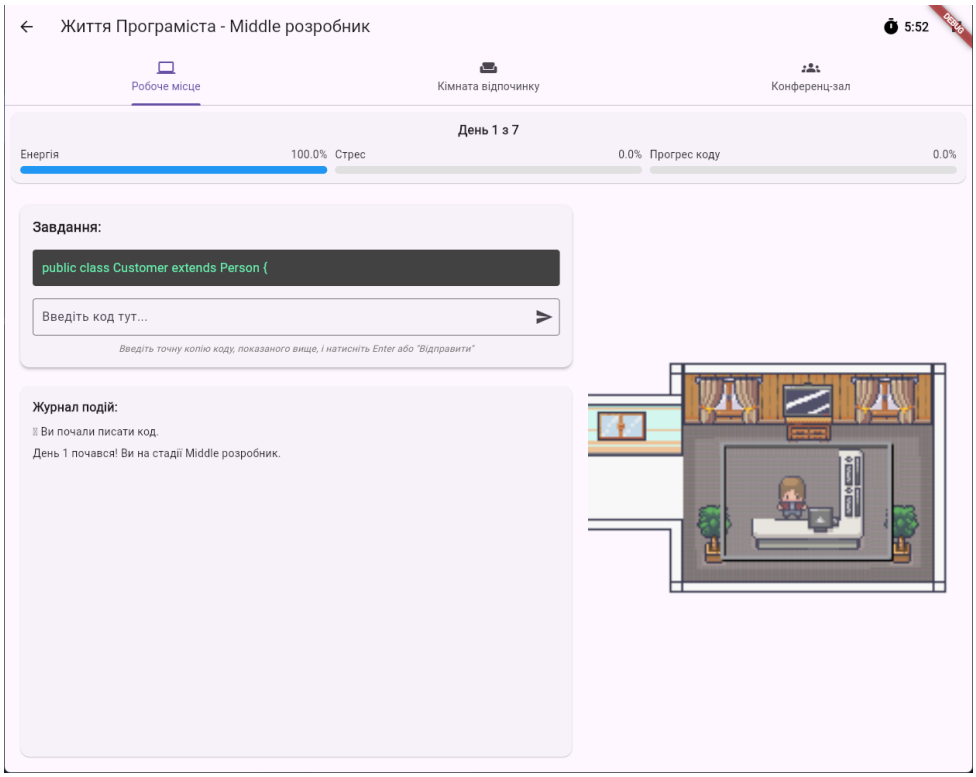


Рисунок 5.6 – Основний ігровий екран (Робоче місце)

У верхній частині екрану розташована панель ресурсів, яка відображає поточний стан трьох ключових показників:

- енергія (синій індикатор) – відображає фізичний та ментальний стан персонажа;
- стрес (червоний індикатор) – показує рівень психологічного напруження;
- прогрес коду (зелений індикатор) – відображає прогрес у виконанні поточного завдання.

Також на верхній панелі відображається поточний день (наприклад, «День 3 з 7») та таймер, що показує залишок часу до кінця робочого дня (початкове значення – 6:00, що відповідає 6 хвилинам реального часу).

Під панеллю ресурсів розташовані вкладки для переключення між трьома локаціями. Користувач може в будь-який момент перейти до іншої локації, натиснувши на відповідну вкладку.

На локації «Робоче місце» користувач займається основною діяльністю – написанням коду. Інтерфейс цієї локації включає:

- панель з відображенням фрагменту коду, який потрібно відтворити;
- поле для введення коду;
- кнопка для відправки коду;
- журнал подій, що відображає повідомлення про успіхи, помилки та випадкові події.

Для написання коду користувач повинен точно відтворити запропонований фрагмент коду в полі введення та натиснути кнопку «Відправити» або клавішу Enter. Якщо код введено правильно, прогрес збільшується, але витрачається невелика кількість енергії. Якщо допущено помилку, рівень стресу збільшується.

Локація «Кімната відпочинку», представлена на рисунку 5.7, призначена для відновлення ресурсів персонажа.

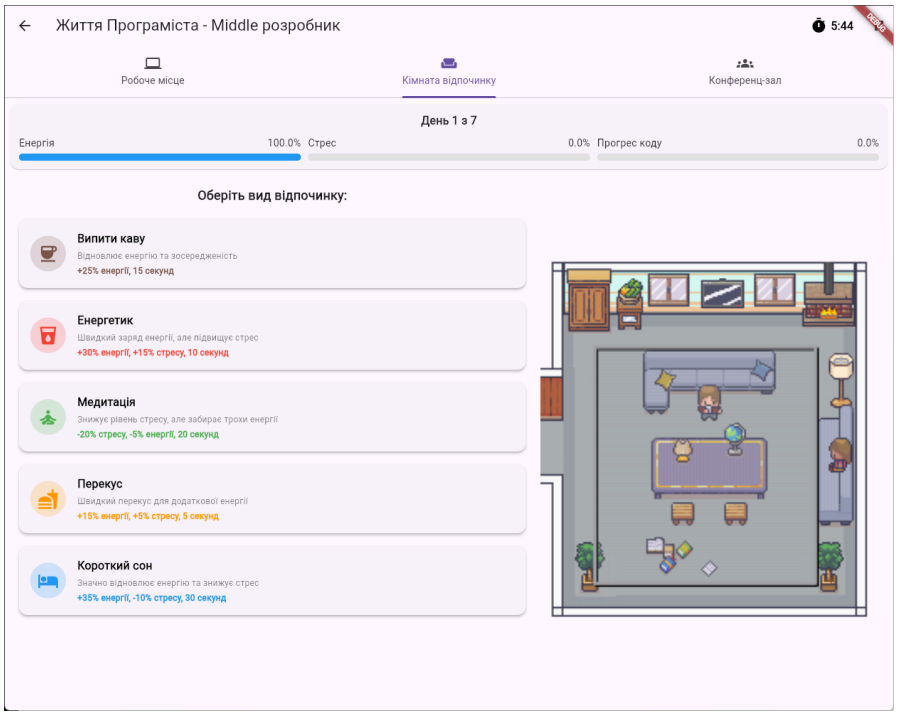


Рисунок 5.7 – Вкладка «Кімната відпочинку»

Тут користувач може виконати наступні дії:

- випити каву (відновлює 25% енергії, потребує 15 секунд);
- випити енергетик (відновлює 30% енергії, збільшує стрес на 15%, потребує 10 секунд);
- медитувати (знижує стрес на 20%, витрачає 5% енергії, потребує 20 секунд);
- перекусити (відновлює 15% енергії, збільшує стрес на 5%, потребує 5 секунд);
- короткий сон (відновлює 35% енергії, знижує стрес на 10%, потребує 30 секунд).

Кожна дія має період охолодження, протягом якого її не можна використати повторно. Під час виконання дії користувач не може писати код, але таймер продовжує відлік часу.

Локація «Конференц-зал», представлена на рисунку 5.8, дозволяє взаємодіяти з віртуальними колегами.

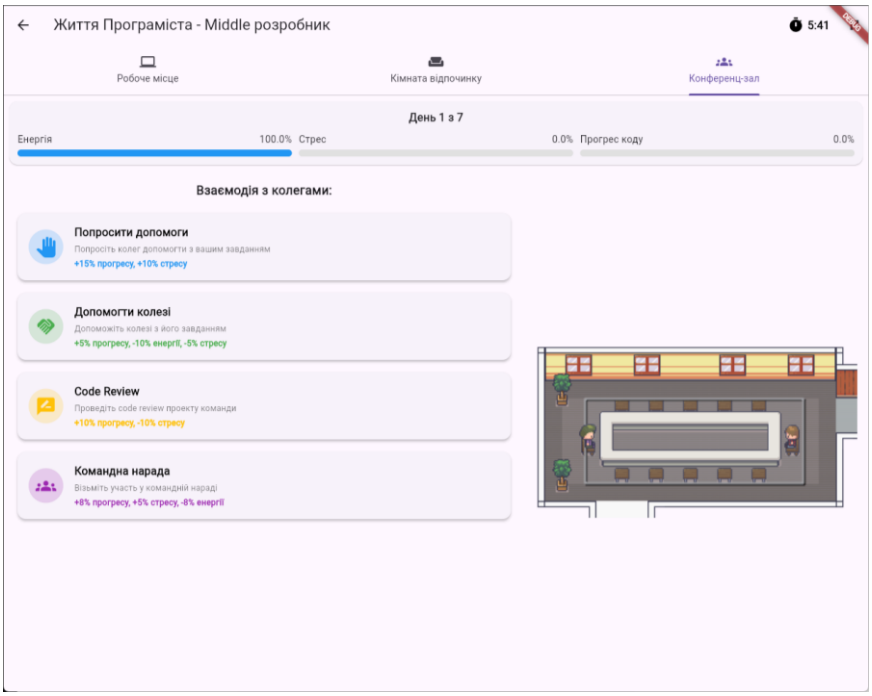


Рисунок 5.8 – Вкладка «Конференц-зал»

Доступні дії включають:

- попросити допомоги (збільшує прогрес на 15%, збільшує стрес на 10%);
- допомогти колезі (збільшує прогрес на 5%, знижує стрес на 5%, витрачає 10% енергії);
- code review (збільшує прогрес на 10%, знижує стрес на 10%);
- командна нарада (збільшує прогрес на 8%, збільшує стрес на 5%, витрачає 8% енергії).

Як і дії в кімнаті відпочинку, ці взаємодії мають періоди охолодження.

5.4 Випадкові події та міні-ігри

Важливою складовою симулятора «Життя програміста» є система випадкових подій та міні-ігор, які додають елемент непередбачуваності та динамізму в ігровий процес, моделюючи реальні ситуації, з якими стикаються програмісти у своїй роботі.

Події в симуляторі виникають з певною періодичністю, яка залежить від поточної стадії кар'єри:

- junior: події виникають рідше (кожні 20-40 секунд) та мають менший вплив;
- middle: середня частота подій (кожні 15-30 секунд) з помірним впливом;
- senior: події виникають часто (кожні 10-20 секунд) та мають значний вплив.

Вибір події відбувається з урахуванням поточної локації та імовірнісних ваг для різних типів подій. Наприклад, критичні помилки в коді виникають з імовірністю 40%, повідомлення про наради – 30%, нагадування про перерву – 30%.

Критичні помилки в коді – одна з найпоширеніших подій, що моделює виявлення серйозної проблеми в коді, яка вимагає негайного вирішення. При виникненні такої події:

- рівень стресу збільшується на 20%;
- користувач тимчасово не може писати код (блокується на 10 секунд);
- для junior-рівня: код автоматично виправляється через 10 секунд;
- для middle та senior-рівнів: з'являється міні-гра QTE (quick time event), яка вимагає швидкого натискання певної послідовності клавіш для «виправлення помилки».

Повідомлення про наради – подія, що сповіщає про збори команди в конференц-залі. Користувачу пропонується перейти до конференц-залу для участі в нараді, яка може підвищити прогрес коду, але вимагає витрат енергії.

Нагадування про перерву – подія, що рекомендує користувачу відпочити через високий рівень втоми. Пропонується перейти до кімнати відпочинку для відновлення енергії.

QTE – це міні-гра, що вимагає від користувача швидкої реакції та точності, представлена на рисунку 5.9.

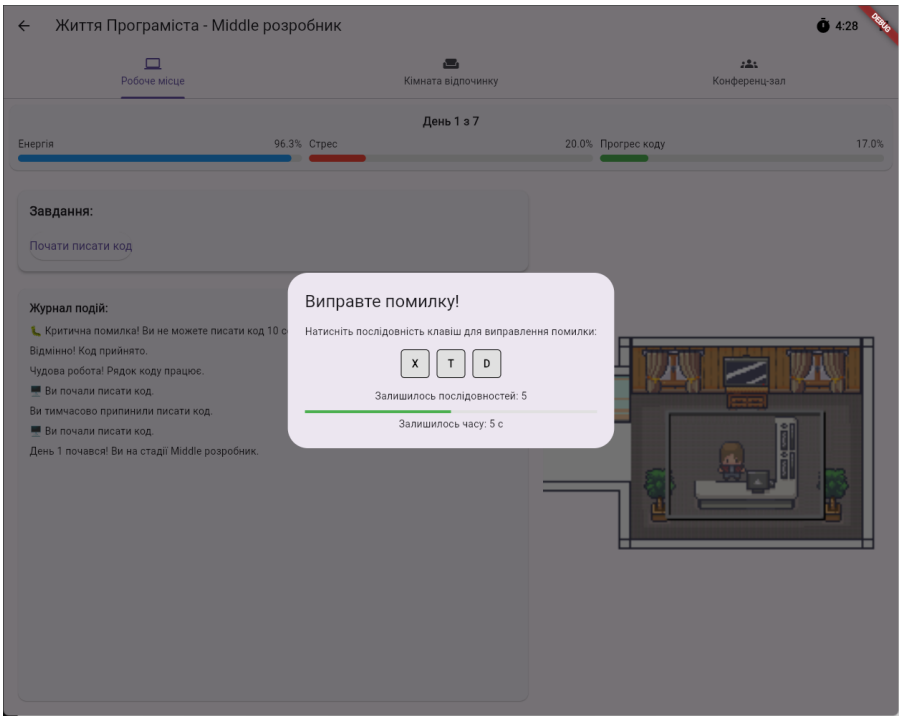


Рисунок 5.9 – Діалог QTE для виправлення помилки

Вона використовується для імітації процесу виправлення критичних помилок у коді і включає в собі 6 етапів:

1) відображається діалогове вікно з заголовком «виправте помилку!»;

2) генерується випадкова послідовність клавіш, яку користувач повинен натиснути;

3) запускається таймер (8-10 секунд залежно від рівня складності);

4) користувач повинен послідовно натиснути всі клавіші у вказаному порядку;

5) якщо послідовність виконана правильно, зменшується кількість залишкових послідовностей;

6) для повного завершення QTE потрібно виконати 5 послідовностей.

Результати міні-гри:

- успішне завершення: стрес зменшується на 10%, виводиться повідомлення про успішне виправлення помилки, користувач може продовжити писати код;

- невдале завершення: стрес збільшується на 10%, виводиться повідомлення про невдалу спробу, користувач не може писати код протягом наступних 10 секунд.

Особливим типом міні-гри є «кавові перерви» та інші дії в кімнаті відпочинку. При виборі такої дії:

- користувач не може писати код або виконувати інші дії;
- запускається таймер (тривалість залежить від обраної дії, від 5 до 30 секунд);
- після завершення таймера автоматично застосовуються ефекти дії (відновлення енергії, зміна рівня стресу);

Для запобігання постійного використання одних і тих самих дій в симуляторі реалізована система охолодження. Після використання певної дії (наприклад, випити каву або допомогти колезі) вона стає недоступною на певний період:

- кава: 15 хвилин ігрового часу;
- енергетик: 30 хвилин ігрового часу;
- медитація: 20 хвилин ігрового часу;
- допомога колезі: 10 хвилин ігрового часу;
- участь у нараді: 20 хвилин ігрового часу.

Дії, що знаходяться на охолодженні, відображаються в інтерфейсі з сірим кольором та іконкою замка.

В будь-який момент гри користувач може натиснути кнопку паузи у верхньому правому куті екрану, щоб відкрити меню паузи, представлене на рисунку 5.10.

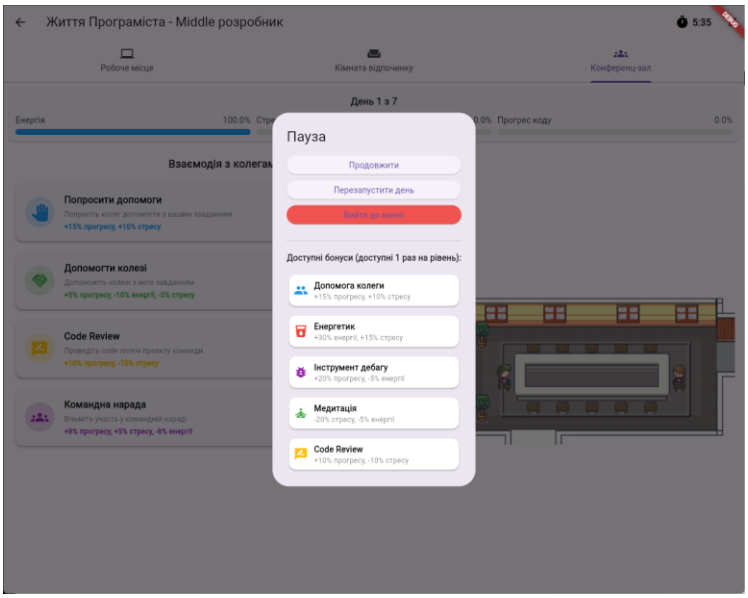


Рисунок 5.10 – Меню паузи з пауерапами

Це меню дозволяє продовжити гру, перезапустити поточний день, вийти до меню вибору рівня, використати доступні підсилення (powerups). Підсилення доступні один раз на рівень і надають значні бонуси для подолання складних ситуацій.

5.5 Система прогресії

Система прогресії в симуляторі «Життя програміста» побудована таким чином, щоб поступово підвищувати складність, одночасно забезпечуючи відчуття досягнення та розвитку.

Гра моделює кар'єрне зростання програміста через три послідовні етапи: Junior розробник із фокусом на базових навичках та адаптації, Middle розробник із підвищеною складністю та відповідальністю, і Senior розробник, що характеризується максимальною складністю та автономією в роботі.

Кожен етап складається з 7 ігрових днів, кожен з яких представляє окремий рівень з унікальними завданнями та викликами.

З кожним наступним етапом кар'єри та днем в рамках етапу зростає складність гри:

- складність кодових фрагментів підвищується (від простих операторів на Junior до складних патернів на Senior);
- частота випадкових подій збільшується (від рідкісних на Junior до частих на Senior);
- інтенсивність впливу подій зростає (більший вплив на ресурси);
- швидкість витрачання енергії при написанні коду підвищується;
- складність міні-ігор QTE збільшується (більше клавіш, менше часу).

Система розблокування рівнів працює наступним чином:

- junior-рівень день 1 доступний з самого початку гри;
- кожен наступний день розблоковується після успішного проходження попереднього;
- для розблокування першого дня наступного етапу кар'єри необхідно успішно пройти всі 7 днів попереднього етапу;
- розблокований прогрес зберігається навіть при невдалих спробах проходження.

Після успішного завершення ігрового дня система показує діалог з результатами, як представлено на рисунку 5.11.

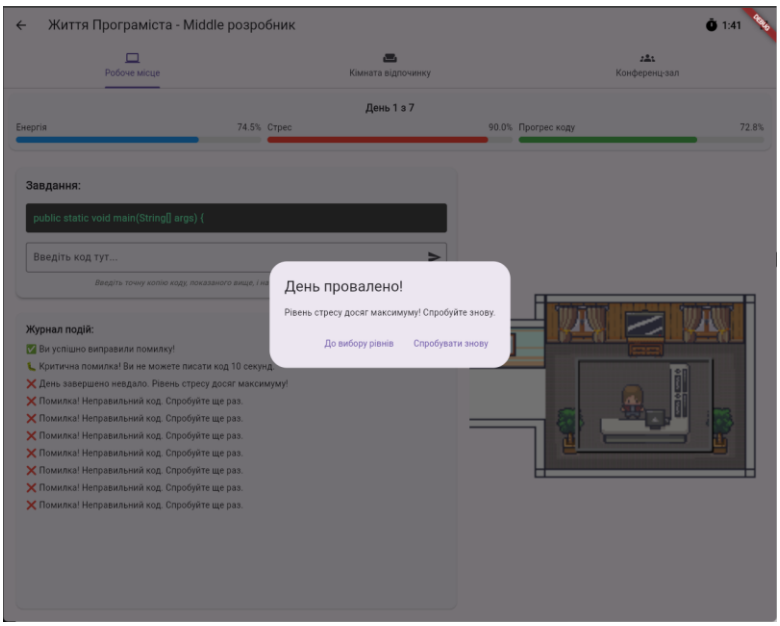


Рисунок 5.11 – Діалог з результатами дня

Такій система забезпечує поступове зростання складності та надає користувачу відчуття прогресу та досягнення, моделюючи реальний кар'єрний шлях програміста.

Висновки до розділу 5

У п'ятому розділі було детально розглянуто взаємодію користувача з програмною системою симулятора «Життя програміста». Початковий досвід користувача включає проходження профорієнтаційного тесту, який оцінює схильність до професії програміста за ключовими показниками, такими як стресостійкість, командна робота, мотивація, робоча етика та здатність до навчання.

Головне меню системи забезпечує інтуїтивно зрозумілу навігацію, надаючи користувачу доступ до початку гри, інструкцій та можливості скинути прогрес. Екран вибору рівня організовано за стадіями кар'єри програміста (Junior, Middle, Senior), де кожен етап містить 7 днів з поступовим зростанням складності.

Основний ігровий процес відбувається на ігровому екрані, який містить три локації: робоче місце для написання коду, кімнату відпочинку для відновлення ресурсів та конференц-зал для соціальної взаємодії. Система керування ресурсами включає показники енергії, стресу та прогресу коду, які динамічно змінюються залежно від дій користувача та випадкових подій.

Випадкові події, такі як критичні помилки в коді, повідомлення про наради та нагадування про перерву, додають динамізму та непередбачуваності в ігровий процес. Міні-гра QTE для виправлення критичних помилок вимагає від користувача швидкої реакції та точності, моделюючи реальні стресові ситуації в роботі програміста.

Система прогресії забезпечує поступове зростання складності від Junior до Senior рівня, збільшуючи частоту та інтенсивність випадкових подій, складність кодових фрагментів та швидкість витрачання ресурсів. Така структура дозволяє користувачу відчути реалістичний шлях кар'єрного зростання програміста.

Розроблений інтерфейс є інтуїтивно зрозумілим та адаптивним, забезпечуючи комфортний досвід взаємодії з симулятором. Система надає користувачу детальний зворотний зв'язок щодо його дій та прогресу, допомагаючи краще зрозуміти особливості професії програміста та оцінити власну схильність до цієї сфери діяльності.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено симулятор «Життя програміста», що представляє собою повноцінний інтерактивний інструмент для професійної орієнтації абітурієнтів та майбутніх студентів. Симулятор дозволяє користувачам в ігровій формі ознайомитися з реаліями роботи програміста та оцінити власну схильність до цієї професії.

Основними результатами роботи є розробка комплексного профорієнтаційного тесту, який оцінює ключові характеристики, важливі для успішної кар'єри в програмуванні, такі як стресостійкість, командна робота, мотивація, робоча етика та здатність до навчання. Також було створено багаторівневу систему прогресії, що моделює три основні етапи кар'єри програміста (Junior, Middle, Senior) з поступовим підвищенням складності та відповідальності. Важливим досягненням стала реалізація системи керування ресурсами, включаючи енергію, стрес і прогрес коду, що відображає реальні виклики, з якими стикаються програмісти у повсякденній роботі. Було розроблено три унікальні локації – робоче місце, кімнату відпочинку та конференц-зал, кожна з яких пропонує специфічні механіки взаємодії та можливості для керування ресурсами. Крім того, впроваджено систему випадкових подій та міні-ігор, що створюють динамічний та непередбачуваний ігровий процес, моделюючи реальні ситуації з професійного життя програміста.

У подальшому планується розширення функціоналу симулятора через додавання нових типів завдань, подій та локацій, а також інтеграцію більш складних механік, таких як командні проекти та професійне зростання через освоєння нових технологій.

Таким чином, розроблений симулятор «Життя програміста» є цінним інструментом для профорієнтації, що допомагає абітурієнтам та майбутнім студентам зрозуміти специфіку роботи програміста та оцінити власну схильність до цієї професії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GameDev Tycoon: Офіційний сайт. URL: https://www.greenheartgames.com/app/game-dev-tycoon/. (Дата звернення: 19 05 2025).

2. Software Inc.: Офіційний сайт. URL: https://softwareinc.coredumping.com/. (Дата звернення: 19 05 2025).

3. CodeCombat: Офіційний сайт. URL: https://codecombat.com. (Дата звернення: 19 05 2025).

4. CheckiO: Офіційний сайт. URL: https://checkio.org. (Дата звернення: 19 05 2025).

5. Flutter: Документація. URL: https://docs.flutter.dev. (Дата звернення: 19 05 2025).

6. Flame: Документація. URL: https://flame-engine.org/docs/. (Дата звернення: 19 05 2025).

7. Dart: Документація. URL: https://dart.dev/guides. (Дата звернення: 19 05 2025).

8. Flutter Provider Package: Документація. URL: https://pub.dev/packages/provider. (Дата звернення: 19 05 2025).

9. Flutter Shared Preferences Package: Документація. URL: https://pub.dev/packages/shared_preferences. (Дата звернення: 19 05 2025).

ДОДАТОК А Лістинг розробленої системи

```
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'dart:convert';

enum CareerStage { Junior, Middle, Senior }

enum PowerupType {
  askColleague,  // +15% code progress, +10% stress
  energyDrink,   // +30% energy, +15% stress
  debuggingTool, // +20% code progress, -5% energy
  meditation,    // -20% stress, -5% energy
  codeReview,    // +10% code progress, -10% stress
}

enum Location {
  workplace,
  breakRoom,
  conferenceRoom,
}

class GameStateProvider with ChangeNotifier {
  // Resources
  double _energy = 100.0;
  double _stress = 0.0;
  double _codeProgress = 0.0;

  // Game state
  CareerStage _currentStage = CareerStage.Junior;
  int _currentDay = 1;
  bool _isGameActive = false;
  bool _isPaused = false;
  DateTime? _levelStartTime;
  int _remainingSeconds = 360; // 6 minutes
  Location _currentLocation = Location.workplace;

  // Career orientation test results
  Map<String, dynamic> _testResults = {};
  bool _testCompleted = false;

  // Level progression
  CareerStage _highestUnlockedStage = CareerStage.Junior;
  Map<String, int> _highestUnlockedDay = {
    CareerStage.Junior.toString(): 1,
    CareerStage.Middle.toString(): 0,
    CareerStage.Senior.toString(): 0,
  };

  // Coding stats
  int _totalLinesOfCodeWritten = 0;
  int _codeErrorsCount = 0;
  double _typingAccuracy = 100.0; // % of typing accuracy

  // Powerups and interactions
  Map<PowerupType, bool> _availablePowerups = {
    PowerupType.askColleague: true,
    PowerupType.energyDrink: true,
    PowerupType.debuggingTool: true,
```

```
  PowerupType.meditation: true,
  PowerupType.codeReview: true,
};

// Cooldowns for location interactions
Map<String, DateTime> _interactionCooldowns = {};

// Getters
double get energy => _energy;
double get stress => _stress;
double get codeProgress => _codeProgress;
CareerStage get currentStage => _currentStage;
int get currentDay => _currentDay;
bool get isGameActive => _isGameActive;
bool get isPaused => _isPaused;
bool get testCompleted => _testCompleted;
Map<String, dynamic> get testResults => _testResults;
CareerStage get highestUnlockedStage => _highestUnlockedStage;
int get remainingSeconds => _remainingSeconds;
Map<PowerupType, bool> get availablePowerups => _availablePowerups;
Location get currentLocation => _currentLocation;
int get totalLinesOfCodeWritten => _totalLinesOfCodeWritten;
int get codeErrorsCount => _codeErrorsCount;
double get typingAccuracy => _typingAccuracy;

// Initialize game state from shared preferences
Future<void> initialize() async {
  final prefs = await SharedPreferences.getInstance();
  _testCompleted = prefs.getBool('testCompleted') ?? false;

  if (_testCompleted) {
    // Load basic game state
    _currentStage = CareerStage.values[prefs.getInt('currentStage') ?? 0];
    _currentDay = prefs.getInt('currentDay') ?? 1;
    _energy = prefs.getDouble('energy') ?? 100.0;
    _stress = prefs.getDouble('stress') ?? 0.0;
    _codeProgress = prefs.getDouble('codeProgress') ?? 0.0;

    // Load progression data
    final highestStageIndex = prefs.getInt('highestUnlockedStage') ?? 0;
    _highestUnlockedStage = CareerStage.values[highestStageIndex];

    // Load highest unlocked day for each stage
    final juniorDay = prefs.getInt('highestDay_${CareerStage.Junior}') ?? 1;
    final middleDay = prefs.getInt('highestDay_${CareerStage.Middle}') ?? 0;
    final seniorDay = prefs.getInt('highestDay_${CareerStage.Senior}') ?? 0;

    _highestUnlockedDay = {
      CareerStage.Junior.toString(): juniorDay,
      CareerStage.Middle.toString(): middleDay,
      CareerStage.Senior.toString(): seniorDay,
    };

    // Load powerups
    final powerupsJson = prefs.getString('availablePowerups');
    if (powerupsJson != null) {
      final Map<String, dynamic> powerupsMap = jsonDecode(powerupsJson);
      _availablePowerups = {
        for (var entry in powerupsMap.entries)
          PowerupType.values.firstWhere((e) => e.toString() == entry.key): entry.value as bool
```

```
      };
    }

    // Load test results
    final testResultsJson = prefs.getString('testResults');
    if (testResultsJson != null) {
      _testResults = jsonDecode(testResultsJson);
    }

    // Load interaction cooldowns
    final cooldownsJson = prefs.getString('interactionCooldowns');
    if (cooldownsJson != null) {
      final Map<String, dynamic> cooldownsMap = jsonDecode(cooldownsJson);
      _interactionCooldowns = {
        for (var entry in cooldownsMap.entries)
          entry.key: DateTime.parse(entry.value as String)
      };
    }

    // Load coding stats
    _totalLinesOfCodeWritten = prefs.getInt('totalLinesOfCodeWritten') ?? 0;
    _codeErrorsCount = prefs.getInt('codeErrorsCount') ?? 0;
    _typingAccuracy = prefs.getDouble('typingAccuracy') ?? 100.0;
  }

  notifyListeners();
}

// Save current state to shared preferences
Future<void> saveState() async {
  final prefs = await SharedPreferences.getInstance();

  // Save basic game state
  prefs.setBool('testCompleted', _testCompleted);
  prefs.setInt('currentStage', _currentStage.index);
  prefs.setInt('currentDay', _currentDay);
  prefs.setDouble('energy', _energy);
  prefs.setDouble('stress', _stress);
  prefs.setDouble('codeProgress', _codeProgress);

  // Save progression data
  prefs.setInt('highestUnlockedStage', _highestUnlockedStage.index);
  prefs.setInt('highestDay_${CareerStage.Junior}', _highestUnlockedDay[CareerStage.Junior.toString()] ?? 1);
  prefs.setInt('highestDay_${CareerStage.Middle}', _highestUnlockedDay[CareerStage.Middle.toString()] ?? 0);
  prefs.setInt('highestDay_${CareerStage.Senior}', _highestUnlockedDay[CareerStage.Senior.toString()] ?? 0);

  // Save powerups
  final powerupsMap = {
    for (var entry in _availablePowerups.entries)
      entry.key.toString(): entry.value
  };
  prefs.setString('availablePowerups', jsonEncode(powerupsMap));

  // Save test results
  prefs.setString('testResults', jsonEncode(_testResults));

  // Save interaction cooldowns
  final cooldownsMap = {
    for (var entry in _interactionCooldowns.entries)
      entry.key: entry.value.toIso8601String()
```

```
  };
  prefs.setString('interactionCooldowns', jsonEncode(cooldownsMap));

  // Save coding stats
  prefs.setInt('totalLinesOfCodeWritten', _totalLinesOfCodeWritten);
  prefs.setInt('codeErrorsCount', _codeErrorsCount);
  prefs.setDouble('typingAccuracy', _typingAccuracy);
}

// Check if a specific level is unlocked
bool isLevelUnlocked(CareerStage stage, int day) {
  if (stage.index < _highestUnlockedStage.index) {
    // All days in previous stages are unlocked
    return true;
  } else if (stage.index > _highestUnlockedStage.index) {
    // No days in future stages are unlocked
    return false;
  } else {
    // For current stage, check day against highest unlocked day
    return day <= (_highestUnlockedDay[stage.toString()] ?? 0);
  }
}

// Unlock a specific level
void unlockLevel(CareerStage stage, int day) {
  // Update highest unlocked stage if needed
  if (stage.index > _highestUnlockedStage.index) {
    _highestUnlockedStage = stage;
  }

  // Update highest unlocked day for this stage if needed
  final currentHighest = _highestUnlockedDay[stage.toString()] ?? 0;
  if (day > currentHighest) {
    _highestUnlockedDay[stage.toString()] = day;
  }

  saveState();
  notifyListeners();
}

// Set current stage
void setCurrentStage(CareerStage stage) {
  _currentStage = stage;
  saveState();
  notifyListeners();
}

// Set current day
void setCurrentDay(int day) {
  _currentDay = day;
  saveState();
  notifyListeners();
}

// Set current location
void setCurrentLocation(Location location) {
  _currentLocation = location;
  notifyListeners();
}
```

```
// Check if an interaction is on cooldown
bool isInteractionOnCooldown(String interactionId) {
  final cooldownEnd = _interactionCooldowns[interactionId];
  if (cooldownEnd == null) {
    return false;
  }

  return DateTime.now().isBefore(cooldownEnd);
}

// Set interaction cooldown
void setInteractionCooldown(String interactionId, int cooldownMinutes) {
  _interactionCooldowns[interactionId] = DateTime.now().add(Duration(minutes: cooldownMinutes));
  saveState();
  notifyListeners();
}

// Game mechanics

// Update code progress based on the code submitted
void updateCodeProgress(double amount) {
  _codeProgress += amount;

  // Ensure progress doesn't exceed 100%
  if (_codeProgress > 100) _codeProgress = 100;

  saveState();
  notifyListeners();
}

// Record a successfully written line of code
void recordCodeWritten(int lineLength) {
  _totalLinesOfCodeWritten += 1;

  // Typing longer code lines is more tiring
  double energyDecrease = (lineLength / 20.0) * 0.5;
  _energy -= energyDecrease;
  if (_energy < 0) _energy = 0;

  saveState();
  notifyListeners();
}

// Record a code error
void recordCodeError() {
  _codeErrorsCount += 1;

  // Update typing accuracy
  if (_totalLinesOfCodeWritten > 0) {
    int totalAttempts = _totalLinesOfCodeWritten + _codeErrorsCount;
    _typingAccuracy = (_totalLinesOfCodeWritten / totalAttempts) * 100;
  }

  saveState();
  notifyListeners();
}

void updateStress(double amount) {
  _stress += amount;
```

```
  // Ensure stress stays within bounds
  if (_stress < 0) _stress = 0;
  if (_stress > 100) _stress = 100;

  saveState();
  notifyListeners();
 }

 void updateEnergy(double amount) {
  _energy += amount;

  // Ensure energy stays within bounds
  if (_energy < 0) _energy = 0;
  if (_energy > 100) _energy = 100;

  saveState();
  notifyListeners();
 }

 void startNewDay() {
  _energy = 100.0;
  _stress = 0.0;
  _codeProgress = 0.0;
  _isGameActive = true;
  _isPaused = false;
  _remainingSeconds = 360; // 6 minutes
  _levelStartTime = DateTime.now();
  _currentLocation = Location.workplace;

  // Reset powerups for the day
  _availablePowerups = {
   PowerupType.askColleague: true,
   PowerupType.energyDrink: true,
   PowerupType.debuggingTool: true,
   PowerupType.meditation: true,
   PowerupType.codeReview: true,
  };

  // Reset interaction cooldowns
  _interactionCooldowns = {};

  saveState();
  notifyListeners();
 }

 void pauseGame() {
  if (_isGameActive && !_isPaused) {
   _isPaused = true;

   // Calculate time remaining
   if (_levelStartTime != null) {
    final elapsedSeconds = DateTime.now().difference(_levelStartTime!).inSeconds;
    _remainingSeconds = 360 – elapsedSeconds;
    if (_remainingSeconds < 0) _remainingSeconds = 0;
   }

   saveState();
   notifyListeners();
  }
 }
```

```
void resumeGame() {
  if (_isGameActive && _isPaused) {
    _isPaused = false;
    _levelStartTime = DateTime.now().subtract(Duration(seconds: 360 – _remainingSeconds));

    saveState();
    notifyListeners();
  }
}

void updateRemainingTime() {
  if (_isGameActive && !_isPaused && _levelStartTime != null) {
    final elapsedSeconds = DateTime.now().difference(_levelStartTime!).inSeconds;
    _remainingSeconds = 360 – elapsedSeconds;

    // Ensure time doesn't go negative
    if (_remainingSeconds < 0) _remainingSeconds = 0;

    // Check for time's up condition
    if (_remainingSeconds <= 0) {
      _isGameActive = false;
      notifyListeners();
      return;
    }

    notifyListeners();
  }
}

void completeDay(bool success) {
  _isGameActive = false;

  if (success) {
    // Unlock next day
    if (_currentDay < 7) {
      unlockLevel(_currentStage, _currentDay + 1);
    } else if (_currentStage != CareerStage.Senior) {
      // If completed all days in current stage, unlock first day of next stage
      final nextStageIndex = _currentStage.index + 1;
      if (nextStageIndex < CareerStage.values.length) {
        final nextStage = CareerStage.values[nextStageIndex];
        unlockLevel(nextStage, 1);
      }
    }
  }

  saveState();
  notifyListeners();
}

void usePowerup(PowerupType powerup) {
  if (_availablePowerups[powerup] != true) return;

  switch (powerup) {
    case PowerupType.askColleague:
      _codeProgress += 15;
      updateStress(10);
      break;
    case PowerupType.energyDrink:
```

```
      updateEnergy(30);
      updateStress(15);
      break;
    case PowerupType.debuggingTool:
      _codeProgress += 20;
      updateEnergy(-5);
      break;
    case PowerupType.meditation:
      updateStress(-20);
      updateEnergy(-5);
      break;
    case PowerupType.codeReview:
      _codeProgress += 10;
      updateStress(-10);
      break;
  }

  // Ensure code progress doesn't exceed 100%
  if (_codeProgress > 100) _codeProgress = 100;

  // Mark powerup as used
  _availablePowerups[powerup] = false;

  saveState();
  notifyListeners();
}

// Location-specific functions

 void drinkCoffee() {
  if (!isInteractionOnCooldown('coffee')) {
    updateEnergy(25);
    setInteractionCooldown('coffee', 15); // 15 minutes cooldown
    saveState();
    notifyListeners();
  }
}

void drinkEnergyDrink() {
  if (!isInteractionOnCooldown('energyDrink')) {
    updateEnergy(30);
    updateStress(15);
    setInteractionCooldown('energyDrink', 30); // 30 minutes cooldown
    saveState();
    notifyListeners();
  }
}

void doMeditation() {
  if (!isInteractionOnCooldown('meditation')) {
    updateStress(-20);
    updateEnergy(-5);
    setInteractionCooldown('meditation', 20); // 20 minutes cooldown
    saveState();
    notifyListeners();
  }
}

void helpColleague() {
  if (!isInteractionOnCooldown('helpColleague')) {
```

```
    _codeProgress += 5;
    updateEnergy(-10);
    updateStress(-5);
    setInteractionCooldown('helpColleague', 10); // 10 minutes cooldown
    saveState();
    notifyListeners();
  }
}

void attendMeeting() {
  if (!isInteractionOnCooldown('meeting')) {
    _codeProgress += 8;
    updateStress(5);
    updateEnergy(-8);
    setInteractionCooldown('meeting', 20); // 20 minutes cooldown
    saveState();
    notifyListeners();
  }
}

void setTestCompleted(Map<String, dynamic> results) {
  _testCompleted = true;
  _testResults = results;
  saveState();
  notifyListeners();
}

void resetAllProgress() {
  _energy = 100.0;
  _stress = 0.0;
  _codeProgress = 0.0;
  _currentStage = CareerStage.Junior;
  _currentDay = 1;
  _isGameActive = false;
  _isPaused = false;
  _highestUnlockedStage = CareerStage.Junior;
  _highestUnlockedDay = {
    CareerStage.Junior.toString(): 1,
    CareerStage.Middle.toString(): 0,
    CareerStage.Senior.toString(): 0,
  };
  _testCompleted = false;
  _testResults = {};
  _availablePowerups = {
    PowerupType.askColleague: true,
    PowerupType.energyDrink: true,
    PowerupType.debuggingTool: true,
    PowerupType.meditation: true,
    PowerupType.codeReview: true,
  };
  _interactionCooldowns = {};
  _totalLinesOfCodeWritten = 0;
  _codeErrorsCount = 0;
  _typingAccuracy = 100.0;

  saveState();
  notifyListeners();
}

// Get player statistics for current game
```

```
  Map<String, dynamic> getGameStats() {
    return {
      'stage': _currentStage.toString().split('.').last,
      'day': _currentDay,
      'totalLinesOfCode': _totalLinesOfCodeWritten,
      'codeErrors': _codeErrorsCount,
      'typingAccuracy': _typingAccuracy.toStringAsFixed(1) + '%',
      'remainingTime': '${(_remainingSeconds / 60).floor()}:${(_remainingSeconds % 60).toString().padLeft(2, '0')}',
      'energy': _energy.toStringAsFixed(1) + '%',
      'stress': _stress.toStringAsFixed(1) + '%',
      'codeProgress': _codeProgress.toStringAsFixed(1) + '%',
    };
  }
}
```

ДОДАТОК Б Презентація

Національний технічний університет України
«Київський політехнічний інститутімені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ГРА-СИМУЛЯТОР «ЖИТТЯ ПРОГРАМІСТА»

Ушаков Віктор Віталійович, ТВ-13

Старший викладач Дацюк Оксана Антонівна

2025

Актуальність теми

Проблематика:
- Абітурієнти не мають реалістичного уявлення про специфіку роботи програміста
- Традиційні методи профорієнтації не передають повсякденні реалії програмування

Значення для галузі інженерії ПЗ:
- Поліпшення якості підготовки майбутніх IT-фахівців через усвідомлений вибір професії
- Формування реалістичних очікувань від професії програміста

Мета:
Розробка інтерактивного симулятора для професійної орієнтації у сфері програмування з реалістичним моделюванням робочих процесів

Постановка задачі

Основна задача:
Розробити інтерактивний симулятор «Життя програміста», який дозволить абітурієнтам та майбутнім студентам у реалістичній ігровій формі ознайомитися з основними аспектами роботи програміста, оцінити власні схильності до професії та зробити усвідомлений вибір кар'єрного шляху в IT-сфері

Перелік основних задач:
- Створити систему прогресії
- Зробити систему вибору рівнів
- Розробити простий зрозумілий для кожного ігровий процес, який відображає життя програміста
- Розробка основної концепції гри зрозумілої для користувача із будьяким досвідом у програмуванні
- Реалізація у найбільш доступним чином, щоб охопити якомога більшебажаючих

Аналіз предметної області

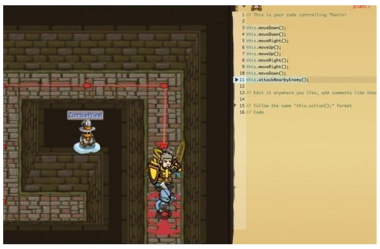
Огляд існуючих рішень та їх недоліків

Інтерфейс гри GameDev Tycoon

Робочий процес у Software Inc.

Аналіз предметної області

Огляд існуючих рішень та їх недоліків

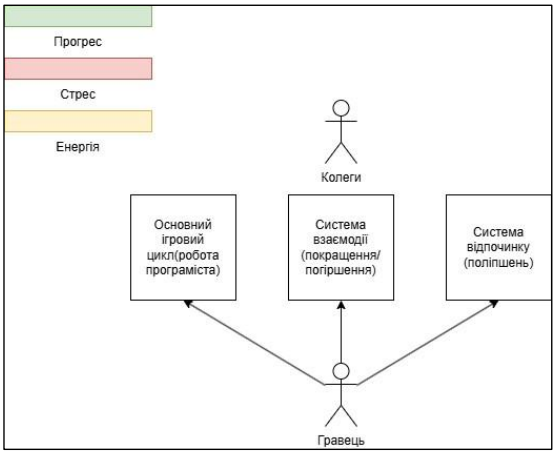


Ігровий інтерфейс CodeCombat

Платформа CheckiO для вивчення Python

ОСНОВНА КОНЦЕПЦІЯ ГРИ

На основі проведеного аналізу аналогів було створено таку початкову концепцію гри



Проєктування системи

Архітектура програмного застосунку

Керуючий модуль (центральний)

•Управління станом • Координація модулів • Ігрова логіка • Цілісність даних

7 основних модулів:

1.UI модуль - інтерфейс користувача та навігація

2.Конфігурація - налаштування та параметри системи

3.Обробка подій - випадкові події та взаємодії

4.Управління даними - збереження, бекап та безпека

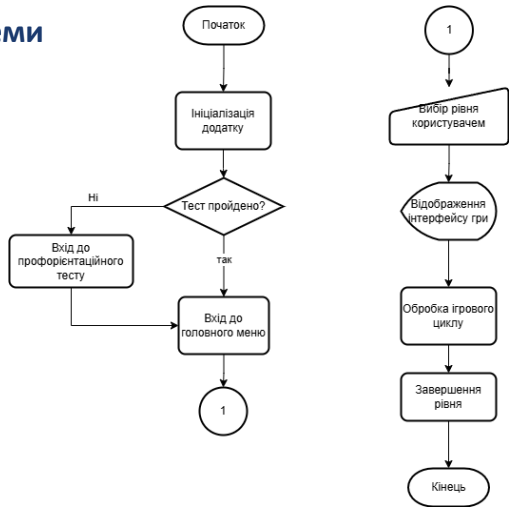
5.Оцінка і тести - профорієнтаційне тестування

6.Статистика - аналіз результатів та метрики

7.Ігрові механіки - симуляція та управління ресурсами

Реалізація системи

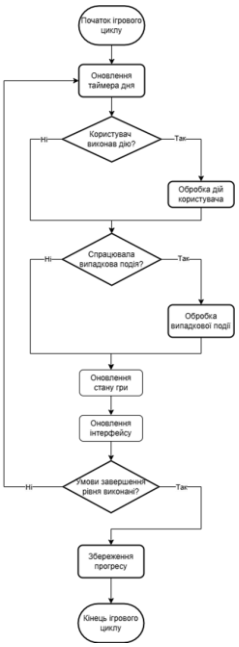
Основний алгоритм системи



Реалізація системи

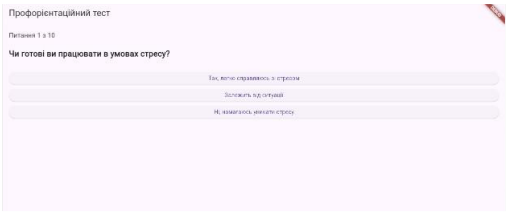
Основний алгоритм ігрового циклу

На даній ілюстрації зображений основний ігровий цикл. Цей цикл наглядно відображає послідовність роботи системи саме у момент гри у вибраному гравцем рівні.

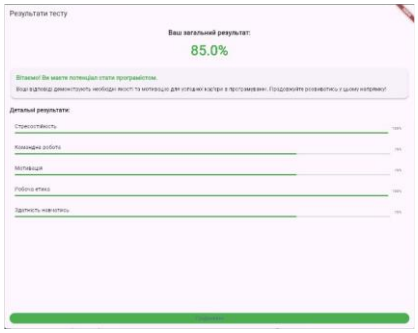


Реалізація системи

Профорієнтаційний тест



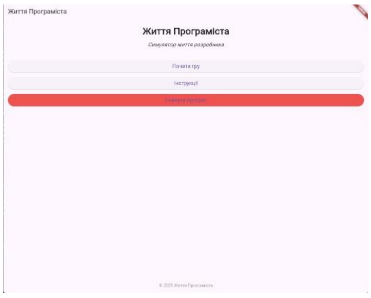
Питання з переліку



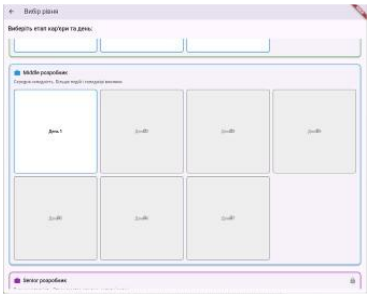
Екран з результатами профорієнтаційноготесту

Реалізація системи

Інтерфейс меню



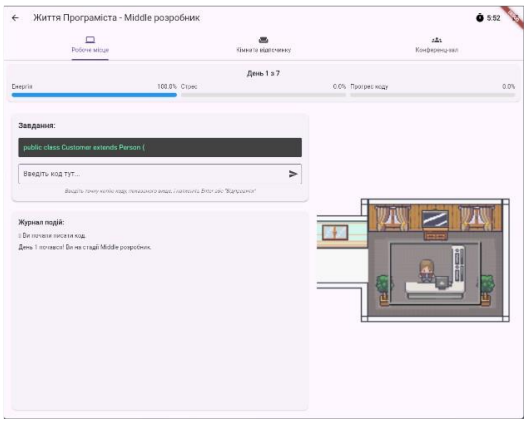
Головне меню додатку



Екран вибору рівня

Реалізація системи

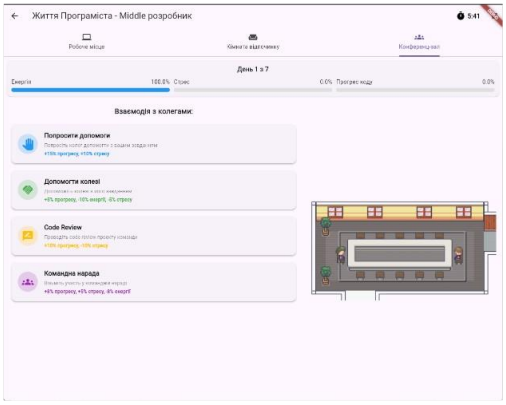
Реалізація гри



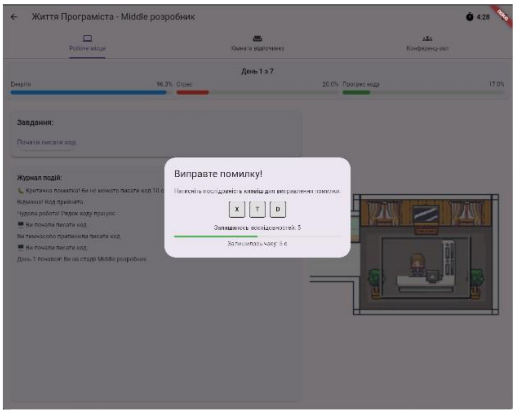
Основний екран та поле
для введення коду

Реалізація системи

Реалізація гри



Екран конференц-зали



Діалог QTE для виправлення помилки

Впровадження та супровід

Система готова для впровадження для профорієнтаційної роботи на нашій кафедрі.

Впровадження:

- Розгортання гри у якості веб додатку для більшого обсягу потенційних користувачів
- Найбільш оптимальний проміжок часу для запуску додатка- період днів відкритих дверей факультету для усіх охочих абітурієнтів

Супровід:

- Регулярне оновлення системи з урахуванням відгуків користувачів
- Можливість масштабування на інші кафедри/факультети

Висновки

Основні результати роботи:

- Розроблено профорієнтаційний тест для користувача
- Було розроблена основна концепція гри зрозуміла для користувача із будь-яким досвідом у програмуванні

- Розроблена концепція включає у собі такі аспекти:
  - Система рівнів та складностей(від Junior до Senior)
  - Система завдань
  - Система спрощень та ускладнень
  - Система випадкових подій