МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Особливості автоматизованого тестування мобільних кросплатформних ігрових застосунків**

Виконав: студент 2 курсу, групи 8.1211-2іпз
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
І.В. Дідик
(підпис, ініціали та прізвище)
Керівник доцент Г.П. Коломоєць
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ Дискус П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність 121 Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т.В. Критська_____
"_12_" __вересня_____2022_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Дідик Іллі Вячеславовичу
(прізвище, ім'я, по батькові)

1.Тема роботи Особливості автоматизованого тестування мобільних кросплатформних ігрових застосунків

керівник роботи Коломоєць Генадій Павлович, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 02.06.2022 р. №597-с

2. Строк подання студентом кваліфікаційної роботи 1 грудня 2022 р.
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження особливостей автоматизованого тестування мобільних кросплатформних ігрових застосунків;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

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

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 12.09.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області.	26.10.2021	Виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником.	12.12.2021	Виконано
3	Аналіз існуючих рішень.	18.12.2021	Виконано
4	Дослідження загальної характеристики та класифікації мобільних застосунків. Особливості мобільних ігрових застосунків. Місце тестування у життєвому циклі розробки програмного забезпечення, завдання тестування. Автоматизоване тестування програмного забезпечення, види тестування.	20.12.2021	Виконано
5	Характеристика інструментів автоматизованого тестування. Інструменти White/Black-box.	01.04.2022	Виконано
6	Розробка тест-плану та тестових сценаріїв для тестування White-box інструментами.	07.10.2022	Виконано
7	Розробка тест-плану та тестових сценаріїв для тестування Black-box інструментами. Формування загальних висновків роботи.	09.10.2022	Виконано
8	Аналіз висновків з роботи існуючих рішень. Аналіз і прототипування потенційних рішень проблеми.	28.10.2022	Виконано
9	Оформлення дипломної роботи.	29.11.22	
10	Попередній захист.	01.12.22	
11	Захист дипломної роботи.	15.12.22	

Студент _____ І.В. Дідик
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г.П. Коломоєць
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Сторінок: 70

Рисунків: 37

Джерел: 15

Дідик І. В. Особливості автоматизованого тестування мобільних кросплатформних ігрових застосунків: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Г. П. Коломоєць. Запоріжжя: ЗНУ, 2022. 70 с.

Мета і завдання дослідження полягають у виявлені популярних засобів автоматизованого тестування мобільних кросплатформних ігрових застосунків, виявлення їх переваг та недоліків та визначення області найефективнішого їх застосування, зокрема для ігор, розроблених з використанням популярної платформи Unity.

У процесі дослідження були розглянуті два підходи до автоматизації. Тестування чорної скриньки — коли тестувальник не має доступу до вихідного коду програмного забезпечення, що тестується, і виконує розробку тестів за вимогами до програмного забезпечення та/або за своїм уявленням, та тестування білої скриньки — при тестуванні цим методом розробник тесту має доступ до вихідного коду програм і може писати код, який використовує бібліотеки програмного забезпечення, що тестується. Обидва підходи широко використовуються для різних типів програмного забезпечення.

За результатами дослідження був розроблений інструмент, котрий вирішує спільні проблеми різних підходів до автоматизованого тестування.

Ключові слова: *тестування чорної скриньки, тестування білої скриньки, Unity, C#.*

SUMMARY

Pages: 70

Figures: 37

Sources: 15

Didyk I.V. Features of Automation Testing of Mobile Cross-Platform Game Applications: qualification work of the master of specialty 121 "Software Engineering" / science. head G.P. Kolomoyets. Zaporizhzhia: ZNU, 2022. 70 p.

The purpose and objectives of the study are to detect popular tools for automated testing of mobile cross-platform gaming applications, identify their advantages and disadvantages and determine the area of their most effective use, in particular for games developed using the popular Unity platform.

Two approaches to automation were considered in the study. Black box testing — when the tester does not have access to the source code of the software under test and performs tests according to software requirements and / or in his imagination, and white box testing — when testing this method the test developer has access to the source code programs and can write code that uses the software libraries under test. Both approaches are widely used for different types of software.

Based on the results of the research, a tool was developed that solves the common problems of different approaches to automated testing.

Keywords: *black box testing, white box testing, Unity, C #.*

ЗМІСТ

ВСТУП

Актуальність теми

На сьогодні розробка ігрових застосунків переросла з нішевої галузі у самостійну велику індустрію, яка набула своїх технологій, критеріїв та стандартів.

Для звичайного користувача — гра є кінцевим ігровим додатком, що дає змогу отримати певний досвід шляхом інтеракції з ігровим світом. В свою чергу для розробника — гра є комплексним програмним продуктом заснованим на певній архітектурі, поділений на зовнішню оболонку та внутрішню реалізацію. В іграх багато складових, але насправді — по-справжньому якісну гру визначає її геймплей, котрий поділяється на дві частини: metagame(визначає кінцеву мету, ціль гравця, до якої він має дійти наприкінці, заради чого він грає; зумовлюється гейм дизайном) та core-gameplay(основний режим гри, котрий визначає засоби для досягнення кінцевої мети; зумовлюється жанром гри). Частіше за все metagame є не один, це робиться навмисно, щоб кожен гравець міг для себе знайти причину грати, будь то перше місце у рейтинговій таблиці, проходження гри, колекціонування «збери їх усіх», тощо.

Рис. *1 Java-гра Gravity Defied*

Мобільні телефони з 90-х років стали еволюціонувати щороку, розробники ПЗ побачили потенціал телефонів, як «КПК»(кишенькових персональних комп’ютерів) — і стали з’являтися перші мобільні ігри. З проривом технологій у розробці комп’ютерного заліза та консолей, що спеціалізуються саме на іграх — мобільні платформи не мали шансу конкурувати, мобільні ігри сприймалися як спосіб побайдикувати, не більше. Все змінилося з виходом перших сенсорних смартфонів — більш продуктивні процесори та відеочіпи, більш велика роздільна здатність та обсяг пам’яті девайсу — відкрили можливості розробляти ігри, здатні конкурувати з іншими платформами, а також відокремилися самі мобільні платформи, визначені операційними системами (IOS, Android, Windows Mobile, etc.).

У сучасному прогресивному світі — важко уявити, що в людини немає смартфона, цей багатофункціональний пристрій замінив багато електроніки, що відправилася на задвірки минулого. Доступність смартфонів дедалі зростає і наразі ми маємо повноцінний комп’ютер у кишені, що наразі є найприбутковішою ігровою платформою (Рис.1).

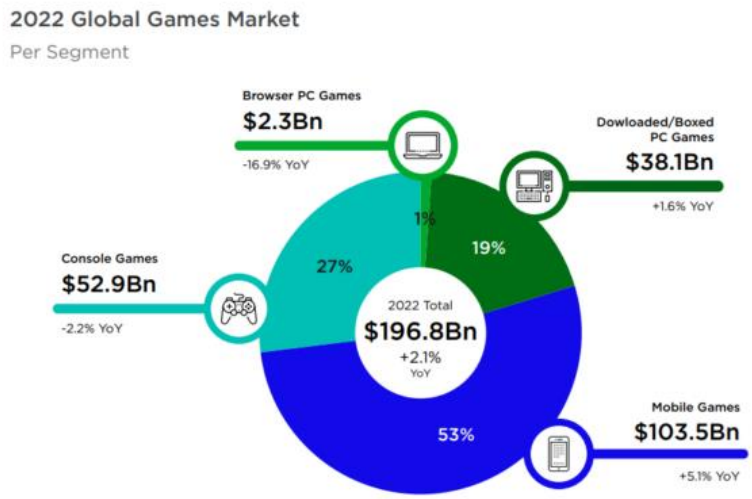

Рис. *2 Поділ ринку ігрової індустрії на 2022й рік*

У наш час ігри, зокрема мобільні, перетворились на довго підтримувані програмні продукти з інтенсивним(ітеративним) життєвим циклом — і

отримали назву «ігри-сервіси». Вимоги змінюються, доповнюються, розширюються, а іноді навпаки — від чогось доводиться відмовитись, щоб йти у ногу з часом. Тож згодом стає дедалі складніше забезпечувати прийнятну якість ігор мануальним шляхом, і, коли витрати на їх забезпечення у людиногодинах досягають певної межі, менеджери та власники продукту починають замислюватися про автоматизацію процесів тестування і зіштовхуються з низкою проблем, котрі зовсім не мають рішень(тестування симпатичного/антипатичного рівня — теорія кольору, миготіння екрану, user-experience), або мають лише тимчасові (рішення з автоматизації, які працюють доки дійсні вимоги і працюють лише на регрес).

Мета і завдання дослідження

Метою роботи є виконання дослідження популярних засобів автоматизованого тестування мобільних кросплатформних ігрових застосунків, виявлення їх переваг та недоліків та визначення області найефективнішого їх застосування, зокрема для ігор, розроблених з використанням популярної платформи Unity. Аналіз знайдених проблем та прототипування можливого рішення.

Об’єкт дослідження

Об’єктом дослідження є ігровий мобільний застосунок, в якому використовуються різні підходи до автоматизованого тестування.

Предмет дослідження

Предметом дослідження є виявлення переваг та недоліків використання популярних засобів автоматизованого тестування, визначення областей найефективнішого їх застосування.

Методи дослідження

Для вирішення поставленої задачі використовуються наступні методи дослідження:

1. Аналіз загальної характеристики мобільних застосунків.
2. Визначення особливостей мобільних ігрових застосунків.
3. Виявлення місця тестування у життєвому циклі розробки ПЗ.
4. Аналіз основних підходів до автоматизованого тестування.
5. Зважання переваг та недоліків кожного з підходів.
6. Прикладний аналіз популярних інструментів тестування.
7. Розбір алгоритмів, що використовується в інструментах.
8. Експериментальне використання інструментів автоматизованого тестування.
9. Визначення та аналіз труднощів, з якими довелось зіштовхнутись.

Наукова новизна одержаних результатів

Наукова новизна одержаних результатів дослідження полягає у тому, що за результатом дослідження буде запропоноване можливе рішення існуючих проблем для різних підходів до автоматизованого тестування мобільних кросплатформних ігрових додатків.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у тому, щоб прискорити тестування та подальшу розробку ігрових програмних продуктів, за рахунок більш якісного регресійного та ретроспективного тестування.

Апробація одержаних результатів

Результати дослідження були представлені на XIV науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2022» [1] та на II всеукраїнській

науково-практичній конференції «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України» [2].

Глосарій

Анімація (Animation) – покадрова зміна зображення для створення ефекту руху.

Ассет (Asset) – неподільний цифровий об’єкт/ресурс, котрий має певні властивості і складається з переважно однотипних даних.

Ассет бандл (Asset bundle) – це зовнішня колекція ассетів, що динамічно завантажується та відвантажується у пам’ять.

Виліт (Crash) – екстрене завершення роботи програми, викликане дефектом.

Витік пам'яті (Memory Leak) – це пам’ять, що виділена програмі, використана, але не очищена, що може привести до уповільнення роботи програми або її виліту.

Геймплей (Gameplay) – можливість взаємодіяти з грою, вирішуючи поставлені геймдизайнером задачі за допомогою ігрової механіки.

Дроп кадрів/Фреймдроп (Frame Drop) – дефект, внаслідок якого кадрова частота (фреймрейт) падає до критично низьких показників.

Зависання/Фризи (*Freeze*) – тривалий пропуск кадрів, звукових фрагментів, найчастіше пов'язаний з поганою продуктивністю гри або нестачею ОЗП.

Затримка/Лаг (*Lag*) – короткочасне зависання/запізнення виведення зображення на екран.

Ігрові механіки (Game Mechanics) – це добірка правил та петель зворотного зв'язку, які призначені створювати інтерактивний геймплей, що приносить задоволення. Вони є будівельними блоками, які застосовуються і комбінуються з ігровим і не ігровим контекстом (наприклад, квести, одиниці досвіду, досягнення).

Ігровий застосунок (Game Application) – вид програмного забезпечення, що дає користувачу певний інтерактивний досвід, організує ігровий процес, в якому гравець взаємодіє з ігровою системою через інтерфейс.

Камера (Camera) – об'єкт, який є точкою огляду користувача в іграх. У 2D іграх камера також присутня, але вона ортогональна (без перспективного спотворення) та її напрямок фіксовано.

Кат-сцена (Cut-Scene) – розповідна неінтерактивна ділянка геймплея, інтегрована окремим відеорядом або відтворена на рушії гри.

Колайдер (Collider) – об’єкти що визначають геометрію для взаємодії між елементами ігрової сцени.

Колізія (Collision) – Незаплановане зіткнення двох сутностей.

Локалізаційне тестування (Localization Testing) – перевірка правильності інтеграції та відображення текстів додатку/гри на підтримуваних мовах.

Локкіт (Lockit) – документ, що надається замовником або командою локалізаторів, в якому наведені всі тексти додатка усіма мовами, що підтримуються.

Механіка Drag-and-Drop (в перекладі з англ. означає буквально Бери-і-Кинь) – спосіб керування об'єктами в грі, який реалізується шляхом «захоплення» відображуваного на екрані об'єкта за допомогою певного контролера(миші, геймпада, пальця у випадку мобільних додатків).

Мобільний застосунок (Mobile application) – це програмне забезпечення, що розроблено для використання на операційній системі мобільного пристрою, частіше за все на смартфоні або планшеті.

Ріббон (Ribbon) - тип інтерфейсу в GUI-додатках, заснований на панелях інструментів, розділених вкладками.

Спрайт (Sprite) – графічний цілісний об’єкт у комп’ютерній графіці, що являє собою двовимірний масив пікселів.

Текстура (Texture) – зображення, що накладається на поверхню полігональної сітки моделі(мешу) для додання їй кольору, забарвлення або ілюзії рельєфу.

Тестування білої скриньки — метод тестування програмного забезпечення, який передбачає, що внутрішня структура/будова/реалізація системи відомі тестувальнику.

Тестування чорної скриньки — це метод тестування програмного забезпечення, при якому перевіряється робота програми без знання її внутрішньої реалізації та схеми роботи.

Тултип (Tooltip) – елемент графічного інтерфейсу, що служить додатковим засобом навчання користувача, що з'являється під час підведення курсору до об'єкта, що цікавить, або в разі неприпустимої дії.

Фіча (Feature) – особливість, характерна риса, функціональна вимога до програмного забезпечення.

Функціональний дефект у грі (Functional Game Defect) – дефект в роботі логіки або будь-який інший частині ігрового процесу, що не передбачений вимогами.

Хітбокси (Hitboxes) – окремі ігрові об'єкти, які накладаються поверх моделей для реалізації логік взаємодії з ними (початково використовувалися для механік нанесення шкоди). У сучасному геймдеві також хітбоксами називають колайдери.

Чіт-код (Cheat Code) – частина коду гри, невеликий додаток, який дає можливість виконувати тестування окремих частин ігор (рівнів, локацій, сцен), переходити до них без проходження великої частини гри, а також додавати необмежені ресурси персонажам (гроші, ресурси, тощо).

ADB (Android Debug Bridge) – це багатофункціональна утиліта для роботи з Android-пристроями через командний рядок, компонент Android SDK. ADB дозволяє обмінюватися даними з запущеним емулятором Android або реальним підключеним (через USB, мережа Wi-Fi) пристроєм Android (планшет, телефон).

C# — об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом

Вілтамутом та Пітером Гольде під егідою Microsoft Research (належить Microsoft). Використовується в ігровому рушії Unity.

Clipping bug – помилка у відображенні текстур. Спрайт/Полігон накладається або проходить крізь інший спрайт/полігон.

Distortion – помилка у відтворенні звуку, спотворений звук, музика та інше.

Frame rate – кількість змінюваних кадрів за одиницю часу, стандартна одиниця вимірювання – FPS (Frames per Second/Кадри за секунду).

Hardlock, Blocker – програмний дефект, внаслідок якого проходження гри далі певної точки стає неможливим. Виправляється лише повним скиданням прогресу, або перевстановленням гри.

Input Lag – затримка введення (як короткочасна, так і тривала), найчастіше пов'язана з низьким фреймрейтом, також може мати в основі проблему із затримкою пристроїв введення (миша, клавіатура, геймпад, датчик дотику смартфона і т.д.).

Invisible Wall – помилка у відображенні текстур. Додаткова геометрія без текстури, що робить її невидимою, гравець не може пройти крізь неї, що частіше за все є помилкою левел-дизайнера.

Landing page, Placement – рекламна вставка, що дозволяє користувачеві ознайомитися з іншими випущеними проектами компанії, підписуватися на розсилки. Зазвичай з'являється на початку гри перед завантаженням головного меню.

Loading Screen – це екран, що відображається гравцям під час завантаження сцени. Може бути як картинкою, так і гіф-анімацією або відео. Якщо гра потребує вторинного завантаження ресурсів – може являти собою міні-гру.

Missing Textures – помилка у відображенні текстур. Об'єкту не призначена текстура.

Pause Menu – меню, яке можна викликати безпосередньо із самого ігрового процесу і тим самим зупинити його.

Softlock – програмний дефект, внаслідок якого проходження гри далі певної точки стає неможливим. На відміну від хардлока, низка маніпуляцій з грою(fold-unfold, pause-unpause, etc.) може вивести гру з цього стану.

Splash Screen – зображення, яке з'являється під час запуску програми/гри. Зазвичай логотип або просто картинка з інформацією про гру чи розробника.

Sprite Atlas – це зображення, що містить набір (або «атлас») підзображень, кожне з яких є текстурою для деякого 2D або 3D об'єкта.

Unity — багатоплатформове середовище розробки відеоігор, розроблене американською компанією Unity Technologies. Unity дозволяє створювати програми, що працюють на більш ніж 25 різних платформах, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-програми.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ МОБІЛЬНИХ КРОСПЛАТФОРМНИХ ІГРОВИХ ЗАСТОСУНКІВ

1.1 Огляд проблеми

На даний момент існує два підходи до автоматизації тестування. Тестування чорної скриньки — коли тестувальник не має доступу до вихідного коду програмного забезпечення, що тестується, і виконує розробку тестів за вимогами до програмного забезпечення та/або за своїм уявленням, та тестування білої скриньки – при тестуванні цим методом розробник тесту має доступ до вихідного коду програм і може писати код, який використовує бібліотеки програмного забезпечення, що тестується. Обидва підходи широко використовуються для різних типів програмного забезпечення.

Складнощі з саме ігровими програмами виникають через те, що вони спираються на візуальну взаємодію користувача з інтерфейсом, а логіка, в свою чергу – є прихованою і не завжди очевидною. Також, у іграх дуже часто використовується, так званий, фактор випадковості, що виникає через процедурну генерацію, тощо.

Щоб підтвердити якість ігрового застосунку, потрібно провести тестування за основними методами, та визначити які з них можна автоматизувати.

Функціональне тестування. Як випливає з назви, за допомогою функціонального тестування тестувальники перевіряють працездатність програми відповідно до заданих специфікацій.

Під час процесу тестування QA виявляє загальні проблеми, такі як виконання певної логіки згідно вимогам, цілісність асетів, аудіо-відео файли, масштабованість, графіку і т.д.

Для автоматизованого тестування виникає проблема валідації даних, тому що, залежачи від оптимізації – аудіо-відео контент може бути навмисно стиснений, від чого виникне невідповідність фактичного та очікуваного

результату. Функціональне автоматизоване тестування може бути здійснене на рівні коду, у вигляді Unit-тестів, або з додатковими інструментами (чіт-кодами, консольними командами, котрі реалізовані спеціально для тесту окремого функціоналу).

Тестування сумісності. Основна мета цього виду тестування – виявити проблеми сумісності. Існують різні версії пристроїв навіть у рамках однієї технології, тому обов'язково необхідно проводити цей тип тестування.

Ігровий додаток повинен однаково працювати на всіх передбачуваних вимогами мобільних пристроях і операційних системах.

Отже, тестувальники перевіряють поведінку програми на всіх мобільних пристроях, враховуючи основні функції, такі як роздільна здатність, дозволи, спеціальні можливості тощо.

Для автоматизованого тестування це є простою задачею, тому що створення нового екземпляру емулятора, або одночасний запуск застосунку на підключених девайсах, або онлайн-фермах, таких як AWS, Samsung Remote Test Lab, та зняття логів – є не складними задачами.

Play-тестування. Як випливає з назви, у цьому методі тестування тестувальник повинен грати як гравець. QA грає від імені гравця та перевіряє, чи з'являється помилка.

Поряд із цим тестувальник також оцінює рівень розважальності ігрової програми, може оцінювати її здатність до проходження, складність рівнів, тощо. В цілому, цей метод тестування допомагає зробити додаток інноваційним, цікавим та орієнтованим на гравців.

Ця задача абсолютно не підлягає автоматизації, тому що спирається на людський фактор і сприйняття.

Регресійне тестування. Один із найважливіших методів, який слід використовувати під час тестування. Воно гарантує роботу старих фіч програми при додаванні нових змін, коректне оновлення на нову версію.

Крім того, регресійне тестування повторно перевіряє всю функціональність програми з нуля та відловлює нові помилки що могли виникнути під час реалізації нового контенту.

Ніхто не може дозволити собі йти на компроміс з якістю, а отже, кожен етап тестування має свою цінність. Після регресійного тестування ігрова програма стає більш продуктивною.

Цей тип тестування якнайкраще підлягає під автоматизацію, тому що можна провести валідацію асетів, спираючись на попередню ітерацію програмного застосунку, можна автоматизувати функціональні тести незмінних фіч, та створити тест що може повністю проходити гру від початку до кінця.

Тестування продуктивності. Цей тип тестування визначає ступінь грабельності(playability) з точки зору комфорту гравця і можливостей девайсу.

Основна увага приділяється тому, щоб гра мала прийнятний фреймрейт (framerate), що на разі є показником у 30 кадрів на секунду(fps). Також треба звертати увагу на фрізи, лаги, фреймдропи.

Окремим стовпом стоїть час завантаження застосунку, бо яким би гарним він не був, ніхто не буде чекати надто довгу загрузку, особливо якщо вона представлена у вигляді статичного лоадеру. Прийнятний час завантаження вираховується окремими метриками, котрі залежать від статичності/динамічності/інтерактивності лоадеру, а останнім часом – від нагород за очікування.

1.2 Класифікація мобільних застосунків

Мобільні застосунки є різноплановим програмним забезпеченням, тож для розуміння механізмів його роботи та засобів розробки і тестування – їх поділяють за різними ознаками.

1.2.1 Класифікація за застосуванням

Промо-застосунки. Такі програми робляться на замовлення бізнесу, слугують за для просування бренду, вони слугують для агрегації усієї інформації щодо послуг, акцій та відгуків. Прикладом є сервіси з доставки їжі, виклику таксі, купівлі квитків у кіно.

Контентні сервіси. Ці застосунки створені за для швидкого доступу до певного контенту: агрегатори новин, курси валют, погода, плани тренувань. Дуже часто у них використовуються рекламні модулі як засіб монетизації, котрий можна відключити за додаткову платню.

Соціальні мережі. Кожна популярна соцмережа має такий застосунок, що дозволяє використовувати нативну оболонку для перебування у соціальній мережі. Прикладами є Instagram, Facebook, та ін. Зазвичай такі програми є заздалегідь встановленими у оболонку девайсу.

Ігрові застосунки. До цієї категорії належать мобільні ігри усіх жанрів. Головна цільова аудиторія – люди, що молодше 27 років, проте з кожним роком вибудовується тенденція на збільшення популярності ігор казуального сегменту і для старшої вікової категорії. Кожна конкретна гра вирішує засіб монетизації, проте останнім часом сформувався тренд на мобільні ігри-сервіси, що довго підтримуються та включають мікротранзакції.

1.2.2 Класифікація за типами

Нативні застосунки. Це програми, що цілком знаходяться у пам’яті девайсу, написані певною мовою під певну платформу(Java для Android, objective-C або Swift для iOS). Розповсюджуються через магазин додатків(Play Market, App Store та ін.). Явною ознакою таких застосунків є те, що вони можуть працювати у штатному режимі без доступу в Інтернет.

Мобільні веб-застосунки. Це розподілена програма, котра має клієнт у вигляді браузера, який з’єднується з віддаленим сервером. Як правило написані засобами HTML5 – вони обов’язково повинні мати вихід в інтернет за для

своєї працездатності. Перевагою є те, що фактично вони займають місце на девайсі не більше, ніж потрібно для ярлика на робочому столі.

Гібридні застосунки. Цей тип застосунків є симбіозом нативних та веб застосунків, частіш за все оснований на клієнт-серверній мікросервісній архітектурі, частіш за все кросплатформні. Можуть працювати без Інтернет-з'єднання лише частково. Займають місця більш ніж веб-застосунки, але менш ніж нативні, бо частково підвантажують інформацію з сервера.

Що стосується місця ігрових програм – вони можуть належати до кожного з цих типів. Нативні Java-ігри, браузерні ігри що належать до веб-застосунків, та останнім часом ігри все більш почали відноситися саме до категорії гібридних застосунків.

1.3 Аналіз типових багів у ігрових застосунках

Для кожного програмного забезпечення характерні певні типові проблеми, зо виникають під час розробки. Для мобільних ігрових застосунків характерні наступні помилки.

1.3.1 Баги продуктивності. Фреймдропи

Продуктивність – є визначальною рисою того чи буде цільовий користувач взагалі грати у гру.

Продуктивність виміряється у FPS(Frames per seconds) за принципом більше-краще. Оптимальною кадровою частотою у екшн-іграх є 60 FPS, в свою чергу для інших жанрів нормою є показник у 30 FPS.

Зазвичай, досвідчені розробники інтегрують у ігри власні інструменти для замірів продуктивності. В іншому випадку, тестувальник використовує сторонні утиліти.

Рис. *3 Гра з інтегрованою системою замірів продуктивності*

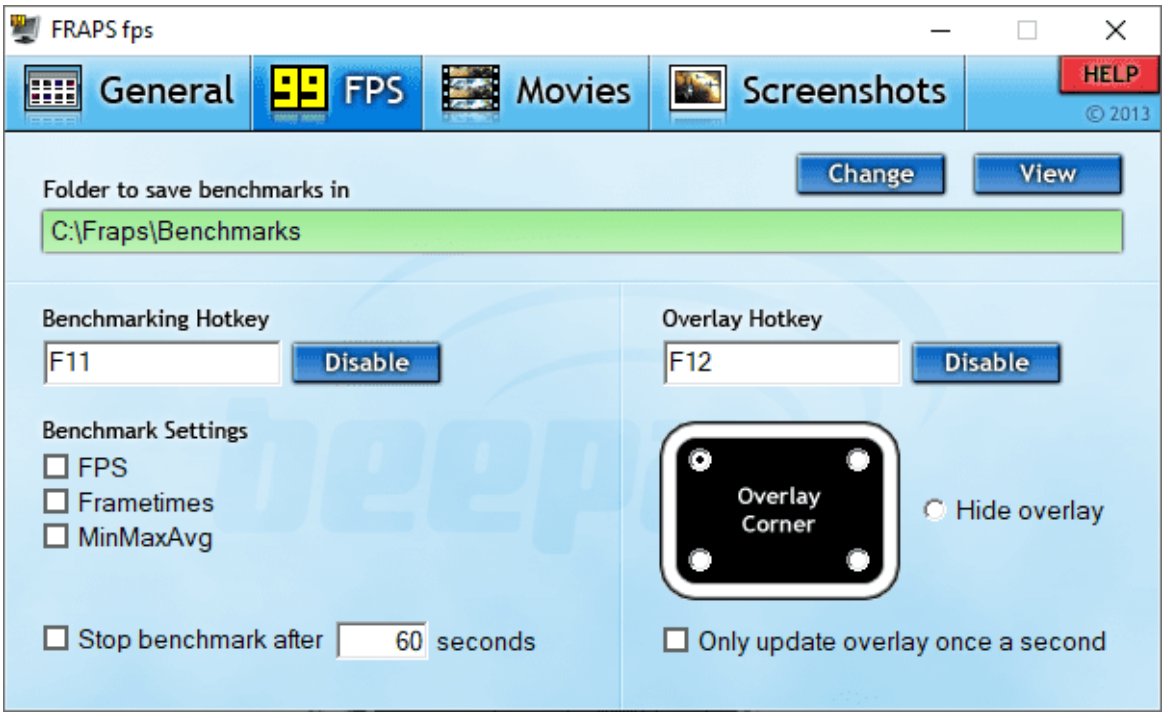

Рис. *4 FrapsFPS – Сторонній інструмент для вимірювання кадрової частоти у грі*

Також, під час тестування продуктивності, важливо перевіряти застосунок на предмет витоків пам’яті. З цим допоможуть інструменти, що класифікуються як системні монітори, або інструменти телеметрії. Вони дозволяють отримати дані щодо навантаження процесора, відеоядра, оперативної пам’яті, швидкого розряджання батареї або переповнення зайнятого місця у довгостроковій пам’яті.

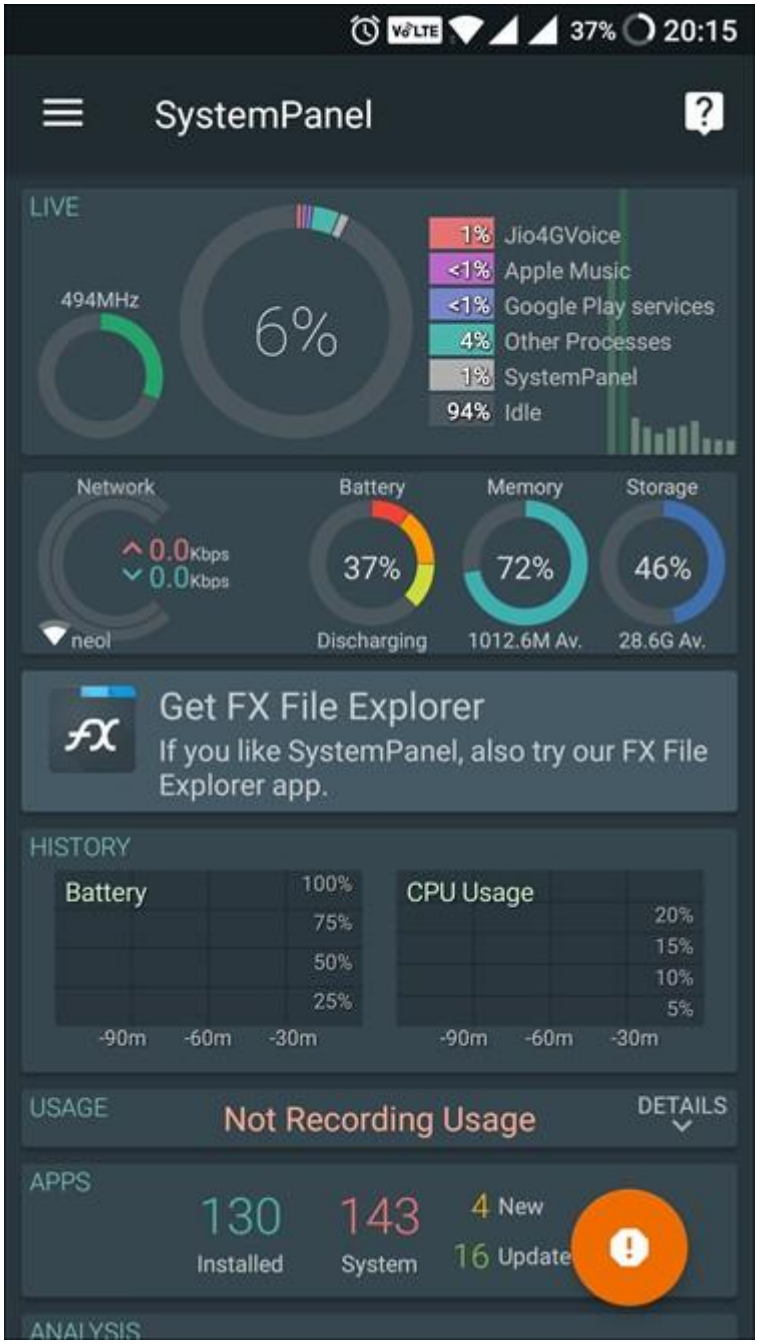

Рис. *5 Інструмент моніторингу статусу мобільного телефона*

1.3.2 Функціональні баги. Баги інтеграції, скриптів

Під час тестування ігор — функціональні баги є одними з найрозповсюджених і складають більшу частку від знайдених проблем. Вони можуть бути дуже різними у своєму зовнішньому прояві, та мати як дуже значні так и навпаки — майже не мати наслідків. Будь то некоректно інтегрований арт,

колізія попапів, або блокери і софтлоки — вони тим чи іншим чином впливають на досвід користувача, тому варто від них позбуватися першочергово.

Рис. *6 Функціональний баг «помилка скрипта рендерера»*

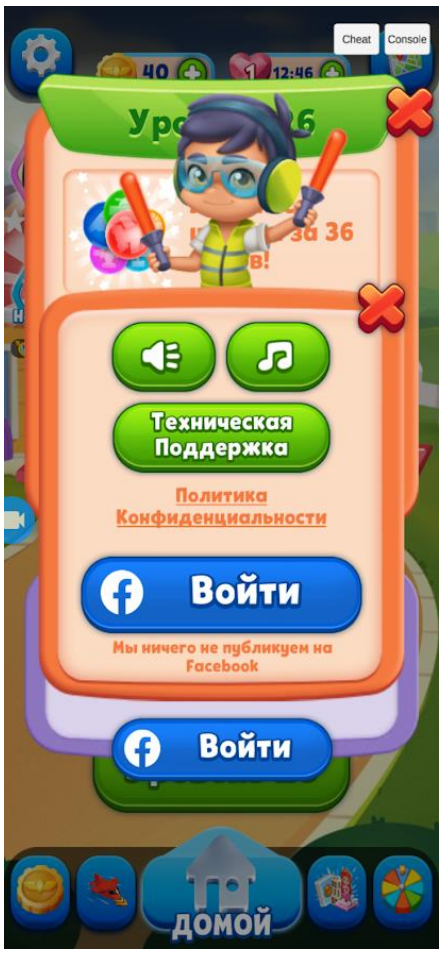

Рис. *7 Функціональний баг «колізія попапів»*

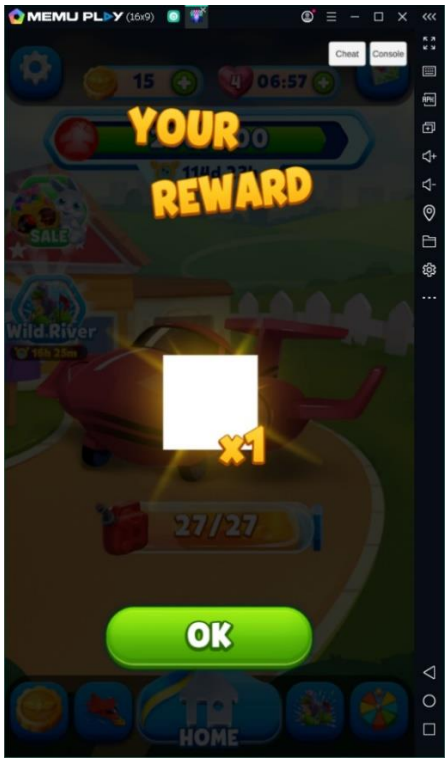

Рис. *8 Функціональний баг інтеграції арту*

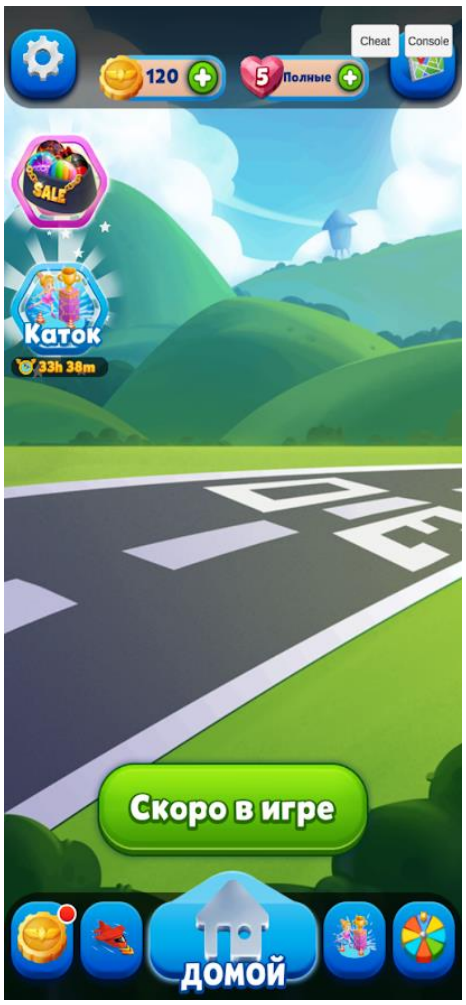

Рис. *9 Функціональний баг «зникнення літака»*

1.3.3 Баги локалізації. Некоректна інтеграція тексту

Отримавши від команди локалізатора або замовника особливий документ — локкіт, тестувальник має виконати перевірку локалізації гри. Проблеми можуть бути як на рівні коду, а саме: не співпадання пар ключ-значення, мова може використовувати не звичну кодову систему, а також можуть бути проблеми на рівні інтерфейсу, коли певний елемент UI не був розрахований на текстовку певної довжини.

RefillNotification	Нотификация восстановления всех жизней, настройки задаются кодом(NotificationType - FULL_REFILL, delay Notification = времени когда будут все жизни восстановлены).	Lives restored! Let's play!	Leben wiederhergestellt! Ran ans Spiel!	¡Vidas restauradas! ¡A jugar!
SingleRefillNotification	Нотификация восстановления одной жизни, настройки задаются кодом(NotificationType - SINGLE_REFILL, delay Notification = времени когда будет одна жизнь восстановлена). PlayerOfflineNotification - нотификация связана периодом	+1 life! Enjoy the adventure!	1 Leben mehr! Starte ins Abenteuer!	¡+1 vida! ¡Disfruta la aventura!
PlayerOfflineNotification	Нотификация связана периодом пребывания юзера в режиме оффлайн,в коде планируется сразу три типа нотифы(OFFLINE_FOR_DAY,OFFLINE_FOR_THREE_DAYS, OFFLINE_FOR_WEEK) Т.е. нацелена на возврат юзера по истечению периода 1, 3, 7 дней	Let's play! Rewards await!	Ran ans Spiel! Es gibt Belohnungen!	¡A jugar! ¡Tiene premio!
FreeSpinNotification	Нотификация восстановления состояния бесплатного вращения колеса фортуны, настройки задаются кодом(NotificationType - FREE_SPIN, delay Notification = времени когда будет восстановлено состояние бесплатного вращения).	Let's play! Spin the Wheel and collect rewards!	Ran ans Spiel! Dreh am Rad und hol dir Belohnungen!	¡A jugar! ¡Tira la ruleta y recoge recompensas!
UnlockLocations	Отсылается в случае: - Юзеру осталось 3 или менее лвл до анлока следующей локации нев мейин ветке - Юзер находится вне игры - С момента вывода с игры, прошло 3 часа	NEW EPISODE ahead! Let's play a few levels and unlock it!	NEUE EPISODE voraus! Spielen wir ein paar Level zum Freischalten!	¡Un NUEVO EPISODIO se avecina! ¡Juega a algunos niveles y desbloquéalo!
BoardTargetNotification	Отсылается в ммоент когда приложение свернуто. Правило отправки: Юзер находится на текущем неактивном уровней, таргет которого задач в конфиге. Пример: на всех уровня со скорой помощью	Need your help! Taxi the planes on the board!	Deine Hilfe wird gebraucht! Leite die Flugzeuge übers Spielfeld!	¡Necesitan tu ayuda! ¡Desplaza los aviones por el tablero!
StartClientEvent	Отправка нотификации, в момент когда юзер находился вне игры, при этом стартовал клиентский ивент (ярмарка) Т.к. условия были соблюдены для ее старта при закрытом приложении	NEW MISSION! Beat new levels and win rewards!	NEUE MISSION! Meistere neue Level und lass dich reich belohnen!	¡UNA NUEVA MISIÓN! ¡Supera nuevos niveles y gana recompensas!
ClientEventEndTime	Отсылается в случе если: - Была запущена ярмарка на N таймер - Данная ярмарка не была пройдена и юзер вышел с игры - В момент когда до завершения ярмарки остается 3 часа, срабатывает триггер отправки нотификации Применимо и для откатных и для ярмарок с энергией	MISSION almost expired! Complete it to earn rewards!	MISSION fast abgelaufen! Schnell zu Ende bringen und Belohnungen einsacken!	¡La MISIÓN está a punto de caducar! ¡Complétala y gana premios!

Рис. *10 Локкіт*

Рис. *11 Баг локалізації «Не інтегрований текст»*

1.4 Інструменти, що були використані для дослідження

AltUnity — це інструмент автоматизації тестування на основі інтерфейсу користувача з відкритим вихідним кодом, який допомагає знаходити об'єкти у вашій грі Unity і взаємодіяти з ними за допомогою тестів, написаних на C #, Python або Java.

Unium — це бібліотека для Unity, що забезпечує автоматизацію. Вона надає HTTP API, який можна використовувати для інструментів тестування.

SikuliX — відкрите крос-платформне візуальне середовище створення сценаріїв-скриптів, яке орієнтоване на програмування графічного інтерфейсу за допомогою зображень. Як скриптова мова в Sikuli використовується Jython.

AirTest — це image recognition крос-платформний фреймворк для мобільних ігор та програм в основі якого лежить image thersholding.

1.5 Процес дослідження

Дана робота була виконана за допомогою ігрового рушія Unity, скрипти були написані у середовищі розробки Microsoft Visual Studio 2019 високофункціональною мовою C#. C# — є об'єктно-орієнтованою мовою програмування і аналогічно іншим сучасним мовам групує пов'язані поля, методи, властивості і події в структури даних, які називаються класами. А також програмними системами SikuliX IDE та AirTest IDE, емулятор MEMU та браузер Google Chrome.

1.5.1 Прикладне використання інструментів білої скриньки

Згідно з ISTQB, тестування білої скриньки має два основних принципових визначення. По-перше, це комплекс заходів з тестування програмного забезпечення, засноване на аналізі вихідного коду ПЗ, структури системи вцілому, або його окремих компонентів. По-друге, це метод тест-дизайну, процес,

котрий визначає обрані тест-кейси, засновуючись на внутрішній реалізації системи, доступу до компонентів.

До переваг тестування за методом білого ящика відноситься можливість працювати з застосунком на ранніх етапах розробки, коли ще не був інтегрований UI.

Серед недоліків можна виділити високий рівень входу спеціаліста, а також складність належної підтримки автоматизації тестових сценаріїв, за рахунок динамічних змін чи правок.

При досліджені методів білої скриньки до ігрового рушія Unity були підключені модулі AltUnity Tester та Unium.

Після підключення AltUnity Tester, у рамках знайомства з інструментом був написаний тестовий метод що валідує поля ігрової сцени та симулює натискання певного об’єкта, з подальшими змінами значення ігрової валюти. Після чого тест було виконано та отримано позитивну відповідь.

Лістинг *1 AltUnity Test*

```
[Test]
public void Test()
{
    //Download game scene
    altUnityDriver.LoadScene("GameScene");
    //Validate start values
    StringAssert.EndsWith(altUnityDriver.FindOb-
ject(By.NAME, "HealthLabel").GetText(), "HEALTH: 5");
    StringAssert.EndsWith(altUnityDriver.FindOb-
ject(By.NAME, "GoldLabel").GetText(), "GOLD: 1000");
    StringAssert.EndsWith(altUnityDriver.FindOb-
ject(By.NAME, "WaveLabel").GetText(), "WAVE: 1");
    //Spend coins test
    altUnityDriver.FindObject(By.NAME, "Openspot
(4)").Click();
    StringAssert.EndsWith(altUnityDriver.FindOb-
ject(By.NAME, "GoldLabel").GetText(), "GOLD: 800");
}
```

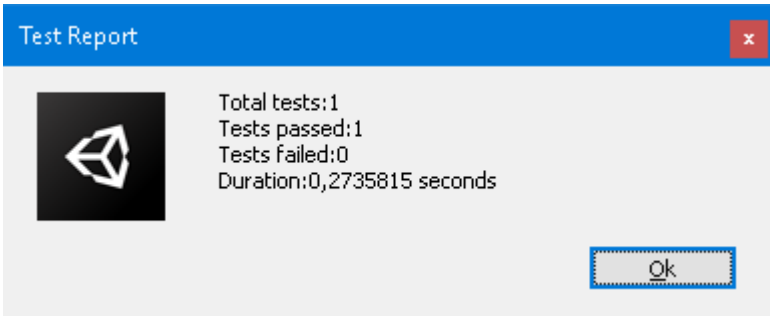

Рис.*1 Результат виконання тесту інструментом AltUnity*

Після підключення Unium, у рамках знайомства з інструментом , було налаштоване підключення до браузера Google Chrome та відкрито порти, щоб він мав доступ до сцен ігрового рушія Unity.

Лістинг *2 Результат виведення ігрової сцени інструментом Unium у форматі JSON*


```
[[{"name":"Main Camera","tag":"MainCamera","activeInHierar-
chy":true,"components":["Transform","Cam-
era","FlareLayer","AudioListener","CameraShake","Audi-
oSource"],"children":[]},{"name":"Openspot","tag":"Un-
tagged","activeInHierarchy":true,"components":["Trans-
form","SpriteRenderer","BoxCollider2D","AudioSource","Plac-
eMonster"],"children":[]},{"name":"Background","tag":"Un-
tagged","activeInHierarchy":true,"components":["Trans-
form","SpriteRenderer"],"children":[]},{"name":"Openspot
(1)","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["Transform","SpriteRenderer","BoxCollider2D","Audi-
oSource","PlaceMonster"],"children":[]},{"name":"Can-
vas","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["RectTransform","Canvas","Can-
vasScaler","GraphicRaycaster"],"children":["Event-
System","HealthLabel","GoldLabel","WaveLabel","GameOverLa-
bel","GameWonLabel","NextWaveTop","NextWaveBottom","Tutori-
alScreen","TutorialButton"]},{"name":"Openspot
(2)","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["Transform","SpriteRenderer","BoxCollider2D","Audi-
oSource","PlaceMonster"],"chil-
dren":[]},{"name":"Cookie","tag":"Untagged","activeInHier-
archy":true,"components":["Transform","SpriteRenderer","Au-
dioSource"],"children":["HealthIndicator1","HealthIndica-
tor2","HealthIndicator3","HealthIndicator4","HealthIndica-
tor5"]},{"name":"Openspot
```

(3)","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["Transform","SpriteRenderer","BoxCollider2D","Audi-
oSource","PlaceMonster"],"children":[]},{"name":"Openspot
(4)","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["Transform","SpriteRenderer","BoxCollider2D","Audi-
oSource","PlaceMonster"],"children":[]},{"name":"Openspot
(5)","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["Transform","SpriteRenderer","BoxCollider2D","Audi-
oSource","PlaceMonster"],"children":[]},{"name":"Openspot
(6)","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["Transform","SpriteRenderer","BoxCollider2D","Audi-
oSource","PlaceMonster"],"children":[]},{"name":"Openspot
(7)","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["Transform","SpriteRenderer","BoxCollider2D","Audi-
oSource","PlaceMonster"],"children":[]},{"name":"Openspot
(8)","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["Transform","SpriteRenderer","BoxCollider2D","Audi-
oSource","PlaceMonster"],"children":[]},{"name":"Openspot
(9)","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["Transform","SpriteRenderer","BoxCollider2D","Audi-
oSource","PlaceMonster"],"children":[]},{"name":"Openspot
(10)","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["Transform","SpriteRenderer","BoxCollider2D","Audi-
oSource","PlaceMonster"],"children":[]},{"name":"Openspot
(11)","tag":"Untagged","activeInHierarchy":true,"compo-
nents":["Transform","SpriteRenderer","BoxCollider2D","Audi-
oSource","PlaceMonster"],"children":[]},{"name":"Ga-
meManager","tag":"Untagged","activeInHierarchy":true,"com-
ponents":["Transform","GameManagerBehavior"],"chil-
dren":[]},{"name":"Road","tag":"Untagged","activeInHierar-
chy":true,"components":["Transform","SpawnEnemy"],"chil-
dren":["Waypoint0","Waypoint1","Waypoint2","Way-
point3","Waypoint4","Waypoint5"]},{"name":"En-
emy(Clone)","tag":"Enemy","activeInHierarchy":true,"compo-
nents":["Transform","AudioSource","MoveEnemy","Rigid-
body2D","CircleCollider2D","EnemyDestructionDele-
gate"],"children":["Sprite","HealthBarBackground","Health-
Bar"]},{"name":"Enemy(Clone)","tag":"Enemy","activeInHier-
archy":true,"components":["Transform","Audi-
oSource","MoveEnemy","Rigidbody2D","CircleCollider2D","Ene-
myDestructionDelegate"],"children":["Sprite","HealthBar-
Background","HealthBar"]},{"name":"Enemy(Clone)","tag":"En-
emy","activeInHierarchy":true,"components":["Trans-
form","AudioSource","MoveEnemy","Rigidbody2D","CircleCol-
lider2D","EnemyDestructionDelegate"],"chil-
dren":["Sprite","HealthBarBackground","Health-
Bar"]},{"name":"GameObject","tag":"Untagged","activeInHier-
archy":true,"components":["Transform","UniumCompo-
nent","UniumSimulate"],"children":[]}]]

З кожним компонентом можна взаємодіяти, та приписувати до нього певний алгоритм дій. Але, такий підхід не враховує користувацьку взаємодію, і, у разі, коли тестовий метод звертається до певного компоненту, викликом методу компонента не може бути симульовано процес натискання кнопки, свайпу, тощо. У ситуації коли тест може натиснути на клавішу, а людина — ні, виникає неспівпадання фактичного результату. Як приклад — взаємодія з затімняючою маскою: користувач не може натиснути на кнопку, котра знаходиться під маскою, але тестування програмним методом, що напряму звертається до компонента кнопки і методу click() — запросто це робить.

1.5.2 Прикладне використання інструментів чорної скриньки

Згідно з ISTQB, тестування чорної скриньки має два основних принципових визначення. По-перше, це перевірка застосунку на функціональні і нефункціональні вимоги без доступу до вихідного коду розроблюваної системи, її компонентів, архітектури. По-друге, це метод тест-дизайну, що включає у себе процедури написання тестових сценаріїв, засновуючись на специфікаціях, власного досвіду, або логічному сприйнятті.

Значною перевагою над інструментами білої скриньки є те, що тестування є максимально наближеним до кінцевого користувача, так як напряму взаємодіє з інтерфейсом, а не кодом. Також важливим є те, що потрібний рівень знань від спеціаліста є не таким високим і досягається за відносно невеликий час при певному рівні мотивації.

Недоліками є те, що без знань коду буває важко визначитись з тестовими сценаріями, а сценарії user-flow, що передбачують певну поведінку користувачів — можуть не повністю покривати систему. Також, у певних випадках — тест кейси можуть пересікатись і викликати надмірність тестування у певних аспектах, де не вдається застосовувати практики pairwise testing`у.

Для використання інструменту SikuliX було встановлене програмне забезпечення SikuliX IDE, емулятор MEMU. Та написаний код тесту, що

автоматизує перевірку завантаження гри, проходження першого рівня, та коректність зміни ігрової сцени, а також створює файл, що повідомляє про успіх/провал тесту та вказує на якому етапі було виявлено проблему. Піддослідною програмою став developer-build мобільної гри Bubble Planes, в котрому реалізовано чіт-панель та консоль Lunar.

Лістинг *3 Вихідний код тестового сценарію SikuliX*

```
#Start definition scene
startGameIcon = "startGameIcon.png"
preloaderScreen = "preloaderScreen.png"
loaderScreen = "loaderScreen.png"
pptos = "pptos.png"
acceptPPTOS = "acceptPPTOS.png"
tutorShoot = Pattern("tutorShoot.png").targetOffset(-
52,126)
firstMassiveClick = "firstMassiveClick.png"
blueMassiveClick = "blueMassiveClick.png"
redMassiveClick = "redMassiveClick.png"
orangeMassiveClick = "orangeMassiveClick.png"
cheatButton = "cheatButton.png"
completeGameButton = "completeGameButton.png"
completeGamePopup = "completeGamePopup.png"
nextButton = "nextButton.png"
xmark = "xmark.png"
lobbyScene = "lobbyScene.png"
#Start logic scene
import logging
FORMAT='%(asctime)-15s %(message)s'
logging.basicConfig(format=FORMAT)
logger=logging.getLogger('')

def test_scene(scene, name):
```

```
        counter = 20;
        while (counter > 0):
            try:
                if exists(scene):
                    logger.warning(name +'is loaded')
                    break
                elif (counter>0):
                    counter -= 1
                if (counter == 0):
                    logger.warning('ERROR:'+name+' is NOT
loaded')
                    my_file = open("result.txt", "w+")
                    my_file.write("Failed"+name)
                    my_file.close()
                    popError("Uuups, this did not work")
                    break
            except:
                pass

    def test_interaction(scene, name):
        counter = 20;
        while (counter > 0):
            try:
                if exists(scene):
                    logger.warning(name+' is finded')
                    click(scene)
                    logger.warning(name+' clicked')
                    break
                elif (counter>0):
                    counter -= 1
                if (counter == 0):
                    logger.warning('ERROR: '+name+' is NOT
exist')
```

```
                    my_file = open("result.txt", "w+")
                    my_file.write("Failed"+name)
                    my_file.close()
                    popError("Uuups, this did not work")
                    break
            except:
                pass
    #Start usage scene
    test_interaction(startGameIcon, 'startGameIcon')
    test_scene(preloaderScreen, 'preloaderScreen')
    test_scene(loaderScreen, 'loaderScreen')
    test_scene(pptos, 'pptos')
    test_interaction(acceptPPTOS, 'acceptPPTOS')
    sleep(2)
    test_interaction(tutorShoot, 'tutorShoot')
    sleep(2)
    test_interaction(firstMassiveClick,
'firstMassiveClick')
    sleep(2)
    test_interaction(blueMassiveClick, 'blueMassiveClick')
    sleep(2)
    test_interaction(redMassiveClick, 'redMassiveClick')
    sleep(2)
    test_interaction(orangeMassiveClick,
'orangeMassiveClick')
    sleep(2)
    test_scene(completeGamePopup, 'completeGamePopup')
    test_interaction(nextButton, 'nextButton')
    sleep(2)
    test_interaction(xmark, 'xmark')
    sleep(2)
    test_scene(lobbyScene, 'lobbyScene')
```

```
my_file = open("result.txt", "w+")
my_file.write("Passed")
my_file.close()
```

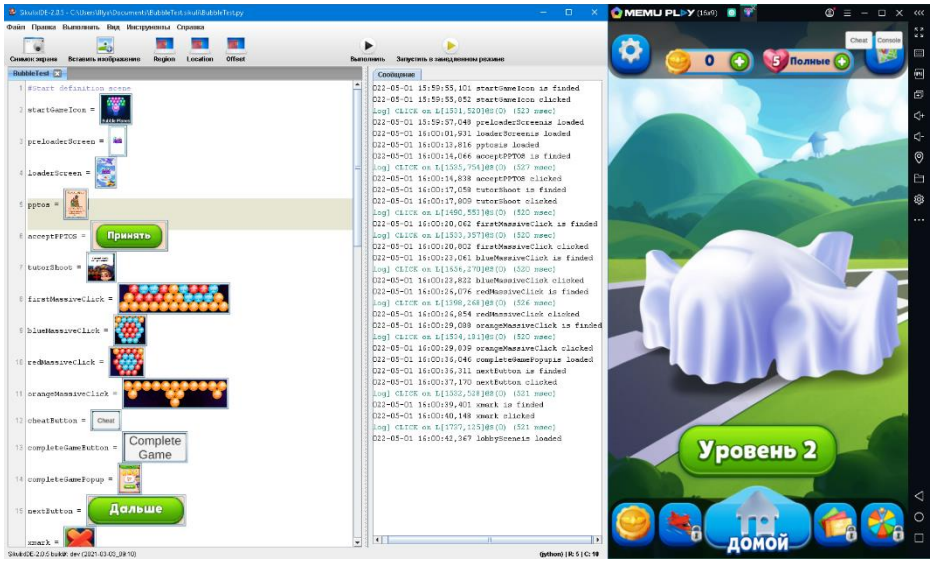

Рис. *12 Результат виконання тесту інструментом SikuliX*

Для використання інструменту AirTest було встановлене програмне забезпечення AirTest IDE, емулятор MEMU. Та написаний код тесту, що автоматизує перевірку завантаження гри, проходження першого рівня, та коректність зміни ігрової сцени. Програма, що тестувалась — незмінна.

Лістинг *4 Вихідний код тестового сценарію AirTest*

```
# -*- encoding=utf8 -*-
__author__ = "Illya"

from airtest.core.api import *

auto_setup(__file__)

touch(Template(r"tpl1649764833939.png",
record_pos=(0.282, -0.281), resolution=(418, 703)))
```

```
    wait(Template(r"tpl1649764891852.png", record_pos=(-
0.053, 0.011), resolution=(418, 703)))
    assert_exists(Template(r"tpl1649764891852.png",
record_pos=(-0.053, 0.011), resolution=(418, 703)),
"Preloader is imaged")
    wait(Template(r"tpl1649765604555.png", record_pos=(-
0.033, 0.394), resolution=(418, 703)))

    assert_exists(Template(r"tpl1649764900660.png",
record_pos=(-0.053, 0.059), resolution=(418, 703)), "Loader
is imaged")
    sleep(10)
    wait(Template(r"tpl1649764915420.png", record_pos=(-
0.038, -0.083), resolution=(418, 703)))
    assert_exists(Template(r"tpl1649764915420.png",
record_pos=(-0.038, -0.083), resolution=(418, 703)), "PPTOS
is imaged")

    touch(Template(r"tpl1649764929688.png", record_pos=(-
0.038, 0.456), resolution=(418, 703)))

    wait(Template(r"tpl1649765249498.png", record_pos=(-
0.041, -0.283), resolution=(418, 703)))

    touch(Template(r"tpl1649765300306.png", record_pos=(-
0.349, 0.322), resolution=(418, 703)))
    sleep(3)

    wait(Template(r"tpl1649765325056.png", record_pos=(-
0.043, -0.339), resolution=(418, 703)))

    touch(Template(r"tpl1649765342884.png",
record_pos=(0.16, -0.374), resolution=(418, 703)))
```

```
    sleep(3)
    wait(Template(r"tpl1649765357512.png", record_pos=(-
0.041, -0.401), resolution=(418, 703)))
    touch(Template(r"tpl1649765376239.png", record_pos=(-
0.258, -0.372), resolution=(418, 703)))
    sleep(3)
    wait(Template(r"tpl1649765385464.png", record_pos=(-
0.043, -0.513), resolution=(418, 703)))
    touch(Template(r"tpl1649765393340.png", record_pos=(-
0.041, -0.492), resolution=(418, 703)))
    wait(Template(r"tpl1649765415377.png", record_pos=(-
0.033, -0.205), resolution=(418, 703)))
    touch(Template(r"tpl1649765425002.png", record_pos=(-
0.045, 0.071), resolution=(418, 703)))
    wait(Template(r"tpl1649765435857.png", record_pos=(-
0.033, -0.279), resolution=(418, 703)))
```

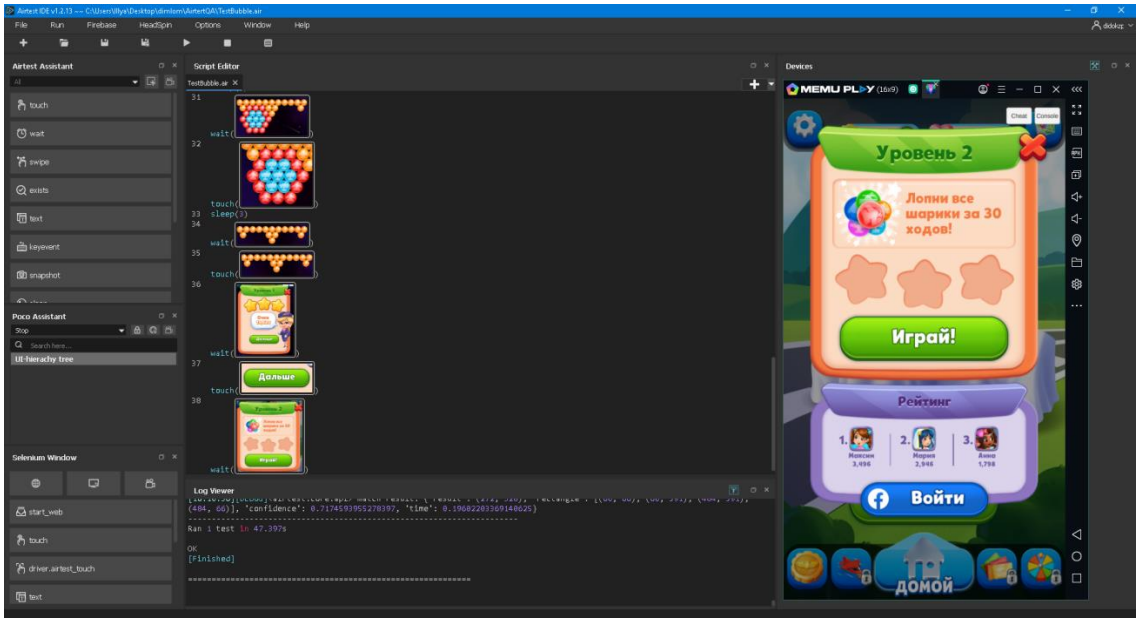

Рис. *13 Результат виконання тесту інструментом AirTest*

Обидві ці програми демонструють методи, у яких середовище симулює користувацьку взаємодію з інтерфейсом. Аналізуються скріншоти

компонентів, котрі потім програмна система відшукує на екрані девайсу або емульованого тестового середовища. Даний метод автоматизованого тестування більш наближений до симуляції поведінки реального гравця.

1.6 Результати дослідження засобів автоматизованого тестування

Спочатку були розглянуті засоби, що використовують підхід "білої скриньки" — інструменти автоматизації тестування AltUnity та Unium. Першою проблемою, з якою довелося зіткнутися — система повинна мати властивість бути тестованою. Це накладає певні обмеження на архітектуру та тестову систему — у випадку коли вона інтегрується до готового проекту. Ці інструменти дозволяють заглибитись у функціонал програмного застосунку, оцінити швидкість роботи алгоритмів, перевірити відповідність полів типу ключ-значення на рівні виконуваного коду тощо. Проте, оперуючи лише об’єктами на сцені ігрового рушія, ми не можемо у повній мірі зімітувати реальну поведінку гравця, а також провести оцінювання якості front-end частини.

Наступним кроком було дослідження інструментів автоматизації, що використовують підхід "чорної скриньки" — SikuliX та AirTest, які передбачають розробку тестових сценаріїв та верифікацію отриманих результатів засобами, що працюють на основі алгоритмів комп’ютерного зору(OpenCV та Thresholding). Перевагою даних інструментів є максимальне наближення тестових сценаріїв до ігрового досвіду реального гравця, а саме — взаємодія із ігровим застосунком за допомогою інтерфейсу. Проте значним недоліком є те, що комп’ютерний зір можна обманути, тож іноді будуть траплятись небажані тестові сценарії, коли проблема насправді є, але комп’ютерний зір її не змозі виявити, в першу чергу це стосується верифікації кольорів, бо Thresholding аналізує чорно-білу інтерпретацію вхідного зображення за рівнем «сірості», де деякі кольори можуть мати однаковий відтінок з точки зору «балансу білого» і відображатися однаково в чорно-білому забарвленні.

Частково цю проблему вирішує методологія "тестування скріншотами", що є поширеною при тестуванні front-end частини Веб-застосунків і може бути задіяна для ігрових програмних застосунків. Проте на сьогодні не існує універсального інструменту для такого типу тестування, тож його можна розробити індивідуально, під власні потреби, використовуючи алгоритм попіксельного порівняння. Також методологія "тестування скріншотами" може допомогти інструментам "білої скриньки" вирішити проблему з верифікацією front-end частини.

1.7 Висновки з розділу 1

1. Були визначені типи мобільних застосунків за різними критеріями.
2. Були оглянуті проблемні моменти автоматизованого тестування мобільних кросплатформних ігрових застосунків.
3. Були розглянуті методи тестування, та виявлені ти з них, що підлягають автоматизації.
4. Були обрані інструменти, для дослідження.
5. Виявлені проблеми автоматизованого тестування мобільних кросплатформних ігрових застосунків.
6. Досліджені сучасні методи автоматизованого тестування мобільних застосунків.
7. Проведений аналіз отриманих результатів дослідження.
8. Запропоноване можливе рішення проблеми.
9. Був визначений концепт майбутнього програмного застосунку для рішення проблем автоматизованого тестування мобільних кросплатформних ігрових застосунків.

РОЗДІЛ 2 ПРОЕКТ ПРОГРАМНОЇ СИСТЕМИ

2.1 Концепція основного застосунку

Головною ідеєю розроблюваного застосунку — є інструмент, котрий підходить для методології тестування скріншотами, буде реалізований алгоритм попіксельного порівняння очікуваного та фактичного результату, котрий, на відміну від Image Thresholding — враховує кольори, також система регулювання рівню допуску. Програма прийматиме масив скриншотів або динамічно підвантажуватиме їх під час регресійного тестування. Наприкінці роботи програми — повинна створюватися тека із файлами, що містять у собі : очікуваний результат, фактичний результат, та відмінність між ними (сірий колір — співпадання, синій колір — допуск, червоний колір — розбіжність), та лог, для мануального відтворення багу або проводити ітеративне тестування та відправляти листа з результатами на пошту тестувальника.

2.2 Огляд існуючих рішень

При дослідженні існуючих рішень — довелось зіштовхнутися з двома типами програм: ті, що підходять для автоматичного тестування у браузері, та ті, що підходять для мануального порівняння скриншотів та не мають API для доступу ззовні.

2.2.1 DiffChecker

Це веб-інструмент, котрий може порівнювати зображення, текст, файлову структуру, тощо. Він має власне API та може використовуватися у сторонніх застосунках. Перевагою є різні типи представлення результату порівняння: через прозорість, слайдер, різницю, що виділена іншим кольором. Недоліком є те, що інструмент не дозволяє використовувати рівні допуску и

формувати зворотній зв’язок з тестувальником, для цього потребується дороблятиви внутрішню логіку застосунку.

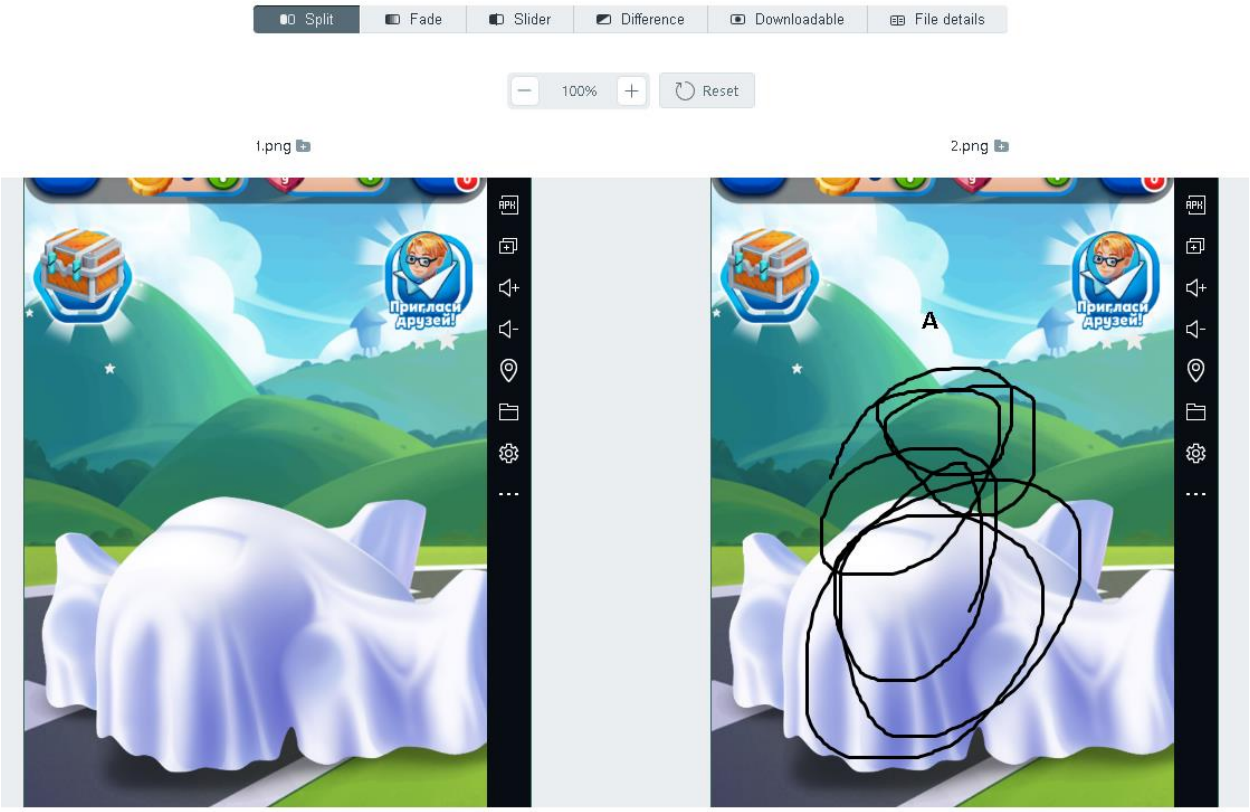

Рис. *14 Результат роботиDiffChecker(баг – візуальні артефакти)*

2.2.2 Beyond Compare 4

Це десктопний інструмент, що може порівнювати зображення, текст, JSON, файлову структуру, тощо. Він не має власного API і може використовуватися тільки як окремий інструмент. Перевагою є те, що порівняння зображення має функцію динамічного допуску, і наглядно демонструє різницю порівняння. Недоліком є те, що програма є платною, та не може формувати зворотній зв’язок для тестувальника і автоматизувати процес перевірки.

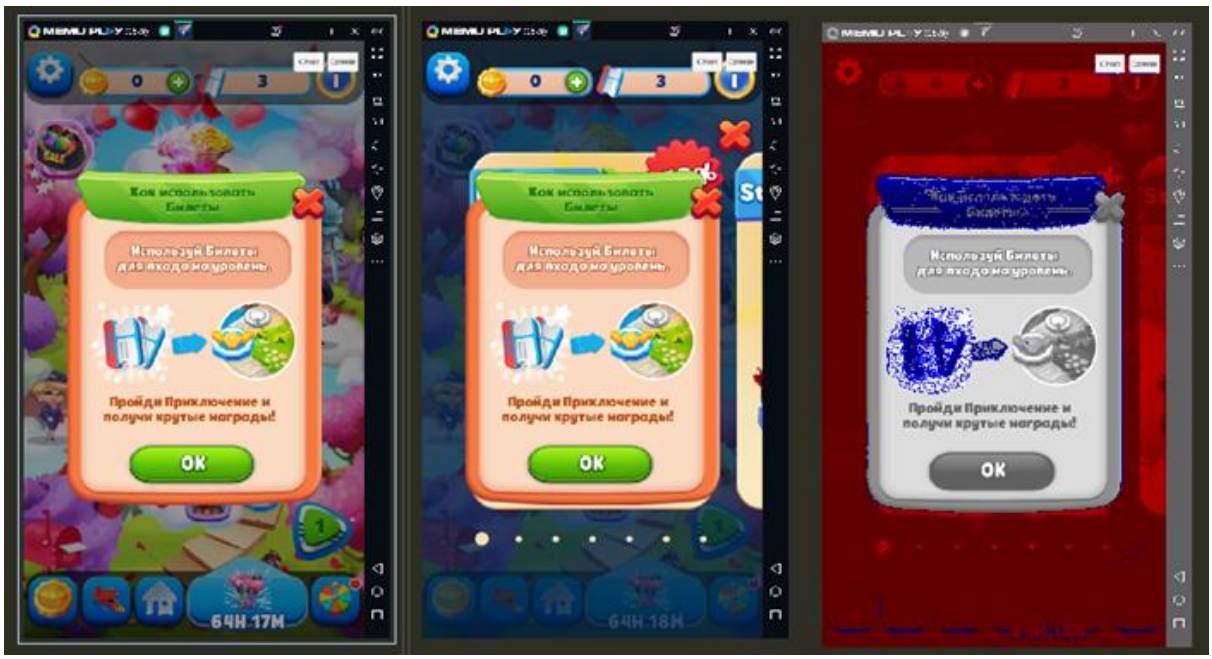

Рис. *15 Результат роботи Beyond Compare 4(баг – колізія попапів)*

2.2.3 IMGonline

Це веб-застосунок для порівння зображень, що має мінімалістичний інтерфейс і є дуже дружнім для користування ззовні, за допомогою Selenium.

Застосунок швидко виконує роботу і дозволяє одразу завантажити результат, також є можливість вибору кольору що відображається у місцях неспівпадання, ступінь стиснення файлу. Недоліком є те, що результат не виводиться одразу, не можна задати рівень допуску, та не виводиться інформація щодо процентажу відмінності.

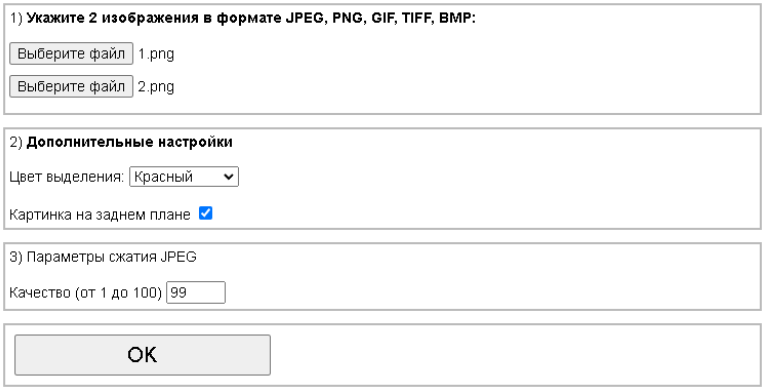

Рис. *16 Інтерфейс веб-застосунку IMGonline*

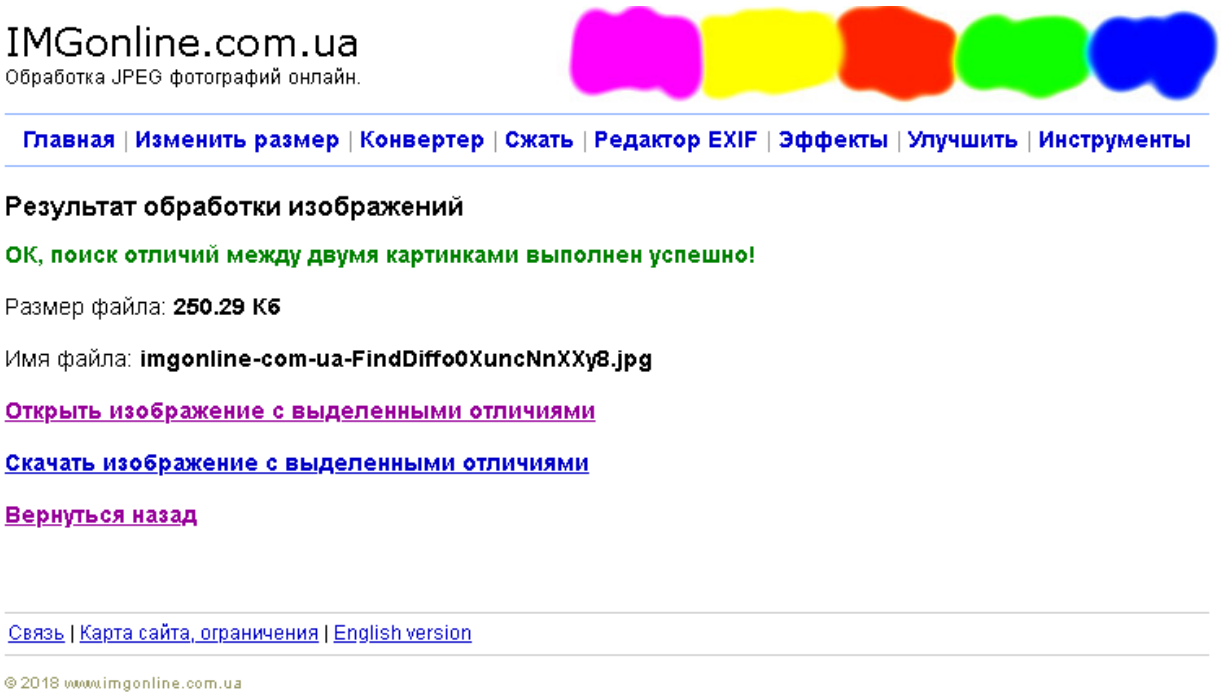

Рис. *17 Результат роботи веб-застосунку IMGonline*

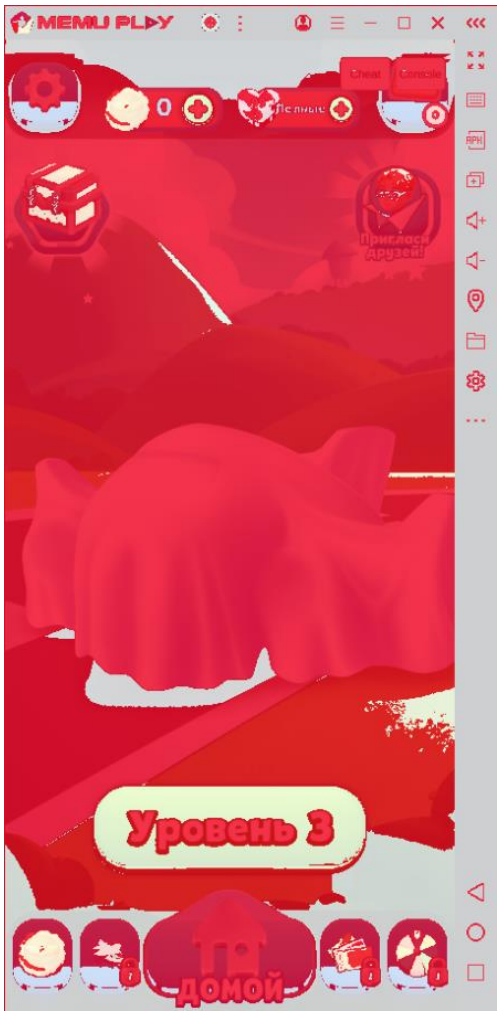

Рис. *18 Файл вихідного результату веб-застосунку IMGonline*

Підсумовуючи, кожен застосунок є або спеціалізованим інструментом під конкретну мову програмування, або не у повній мірі задовільняє потреби тестувальника з точки зору приймального тестування та зворотного зв’язку.

2.3 Проект тестової системи

Щоб покривати вимоги до тестової системи буде використовуватись декілька програмних засобів: інструмент тестування чорної скрині AirtestIDE, за допомогою якого будуть створені тестові сценарії, бо він є більш розвинутою версією SikuliX, а також власноруч створений інструмент регресійного тестування на основі алгоритму порівняння скриншотів.

Було розроблено 2 сценарії. Перший робить еталонні скріншоти, з релізної версії додатку. Другий за тим самим флоу – робить фактичні скріншоти, з тестованої версії додатку.

Під кінець роботи програм-сценаріїв — буде запускатись розроблений інструмент порівняння, за результатами котрого — надсилатися зворотній зв’язок на пошту тестувальника.

Рис. *19 Діаграма проекту автоматичної тестової системи*

Також буде можливість користуватися програмою мануально. Таким чином тестувальник, використовуючи засіб порівняння, зможе наглядно побачити результат порівняння за допомогою User Interface.

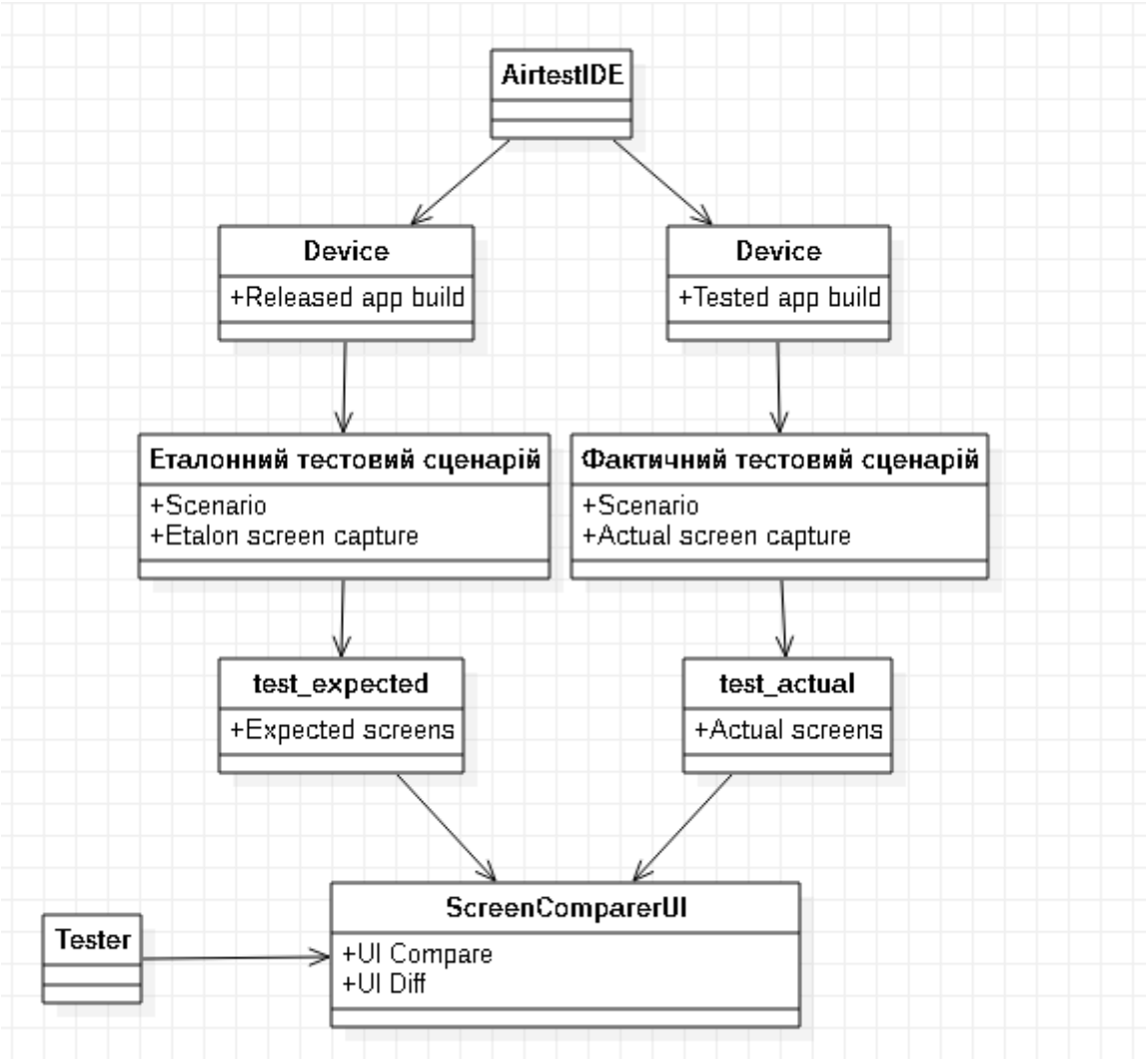

Рис. *20 Діаграма проекту мануальної тестової системи*

РОЗДІЛ 3 РЕАЛІЗАЦІЯ І ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Розробка програмного застосунку порівняння скриншотів

Для розробки інструмента, що буде порівнювати скріншоти було використане середовище розробки Visual Studio 2019, мова програмування C#. Програма являє собою інструмент що порівнює 2 зображення і видає результат на основі рівня допуску(процент відмінності, котрий можна списати на динамічну idle-анімацію, стиснення скріншота). Результати виконаних тестів програма відправляє на пошту тестувальнику.

Також, програму можна використовувати у мануальному режимі, де можна побачити фактичну відмінність між фактичним і очікуваним результатом.

3.1.1 Розробка алгоритму порівняння

Під час розробки алгоритму було задіяне попіксельне порівняння зображень, котрі являють собою Bitmap у компоненті Image.

Лістинг *5 Метод Difference*

```
            Bitmap Bmp1 = (Bitmap)Img1;
            Bitmap Bmp2 = (Bitmap)Img2;
            Bitmap Bmp3 = (Bitmap)Img1;

            double counter = 0;

            for (int i = 0; i < Bmp1.Width; i++)
            {
                for (int j = 0; j < Bmp1.Height; j++)
                {
                    if (Bmp1.GetPixel(i, j) !=
Bmp2.GetPixel(i, j))
                    {
                        Bmp3.SetPixel(i, j, Color.Fro-
mArgb(128, 255, 0, 0));
                        counter++;
                    }
                }}
            pictureBox3.Image = Bmp3;
```

Після порівняння, алгоритм запам’ятовує розбіжності за допомогою змінної counter, що знадобиться для подальшого визначення рівня допуску.

3.1.2 Логування даних

Шляхом випробувань, спираючись на активні idle-анімації та рівень стиснення, було задано наступні критерії апробації результатів:

- **Diff <= 0% - Definitely good**
- **Diff <=15% - Potential good**
- **Diff <=49% - Potential problem**
- **Diff >=50% - Absolute problem**

Для того, щоб запам’ятати результати тестів – було зроблене логування результатів порівняння:

Лістинг *6 Метод LogData*

```
        void LogData(string _text)
        {
            string filePath = "testlogger.txt";
            string text = _text;
            label1.Text = text;
            FileStream fileStream = null;
            if (!File.Exists(filePath))
                fileStream = File.Create(filePath);
            else
                fileStream = File.Open(filePath, File-
Mode.Append);
            StreamWriter output = new Stream-
Writer(fileStream);
            output.WriteLine(text);
            output.Close();
        }
```

Параметром методу є поступаючий рядок, у котрому визначено що саме необхідно залогувати, у нашому випадку це результати порівняння:

Лістинг *7 Логування методом LogData*

```
            if (Math.Round(counter /
(pictureBox3.Image.Width * pictureBox3.Image.Height) * 100)
<= 0)
```

```
            {
                LogData("Definitely Good, diff = " + Con-
vert.ToString(Math.Round(counter / (pictureBox3.Image.Width
* pictureBox3.Image.Height) * 100)) + "%");
            }
            else if (Math.Round(counter / (pictureBox3.Im-
age.Width * pictureBox3.Image.Height) * 100) <= 15)
            {
                LogData("Potential Good, diff = " + Con-
vert.ToString(Math.Round(counter / (pictureBox3.Image.Width
* pictureBox3.Image.Height) * 100)) + "%");
            }
            else if (Math.Round(counter / (pictureBox3.Im-
age.Width * pictureBox3.Image.Height) * 100) > 15 &&
                Math.Round(counter / (pictureBox3.Im-
age.Width * pictureBox3.Image.Height) * 100) <= 49)
            {
                LogData("Potential Problem, diff = " + Con-
vert.ToString(Math.Round(counter / (pictureBox3.Image.Width
* pictureBox3.Image.Height) * 100)) + "%");
            }
            else if (Math.Round(counter / (pictureBox3.Im-
age.Width * pictureBox3.Image.Height) * 100) >= 50)
            {
                LogData("Absolute Problem, diff = " + Con-
vert.ToString(Math.Round(counter / (pictureBox3.Image.Width
* pictureBox3.Image.Height) * 100)) + "%");
            }
```

3.1.3 Реалізація зворотнього зв’язку з тестувальником

Для реалізації зворотнього зв’язку з тестувальником було написано метод SendFeedback, котрий слугує для надсилання тестувальнику записаного лог-файлу на електронну пошту, щоб він міг подивитись як завершились тести, якщо система виконує їх автоматично.

У зв’язку зі зміною політики безпеки Google – змінився спосіб надсилання повідомлення через протокол SMTP. Тепер недостатньо мати логін та пароль від пошти-відправника. Зараз кожний сторонній застосунок має генерувати токен для відправки листа, для цього на аканті гугл повинна бути увімкнена двофакторна аутентифікація. Також, поштова скринька має бути відкритою до взаємодії зі сторонніми застосунками, тому під час написання методу довелось згенерувати токен для застосунку.

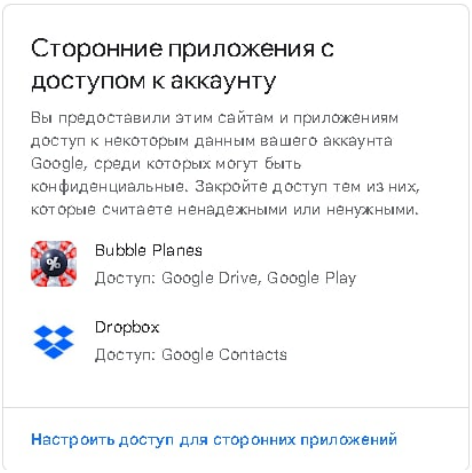

Рис. *21 Дозвіл на доступ стороннім застосункам*

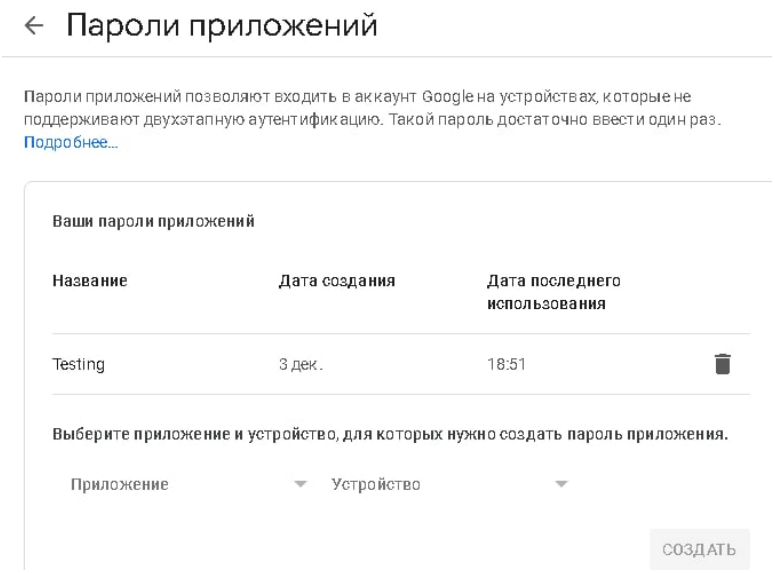

Рис. *22 Генерація токена для застосунку*

Лістинг *8 Метод SendFeedback*

```
        void SendFeedback()
        {
            string dateFileName = "";
            dateFileName = DateTime.Now.ToLongDate-
String()+DateTime.Now.Hour+"_"+DateTime.Now.Mi-
nute+"_"+DateTime.Now.Second;
            File.Copy("testlogger.txt", dateFile-
Name+".txt");

            //njmnwvglxlrkwqeg

           MailMessage mail = new MailMessage();
```

```
            mail.To.Add("ilya.didyk@gmail.com");
            mail.From = new MailAd-
dress("didtwin2@gmail.com");
            mail.Subject = "Test results Bubble Planes";
            mail.Body = "View Logs to get feedback";

            Attachment attachment;
            attachment = new Attachment(dateFileName +
".txt");
            mail.Attachments.Add(attachment);

            SmtpClient smtp = new SmtpClient();
            smtp.Port = 587;
            smtp.EnableSsl = true;
            smtp.UseDefaultCredentials = false;
            smtp.Host = "smtp.gmail.com";
            smtp.Credentials = new System.Net.NetworkCreden-
tial("didtwin2@gmail.com", "njmnwvglxlrkwqeg");
            smtp.Send(mail);

        }
```

Після виконання даного метода тестувальник отримує листа, в якому відображається файл, що має назву, що співпадає з датою та часом виконання тесту. Також можна вказати тему листа, що визначає до якого саме застосунка був виконаний тест. У нашому разі це був Bubble Planes. Також є можливість не відправляти фідбек, якщо всі тести пройшли успішно.

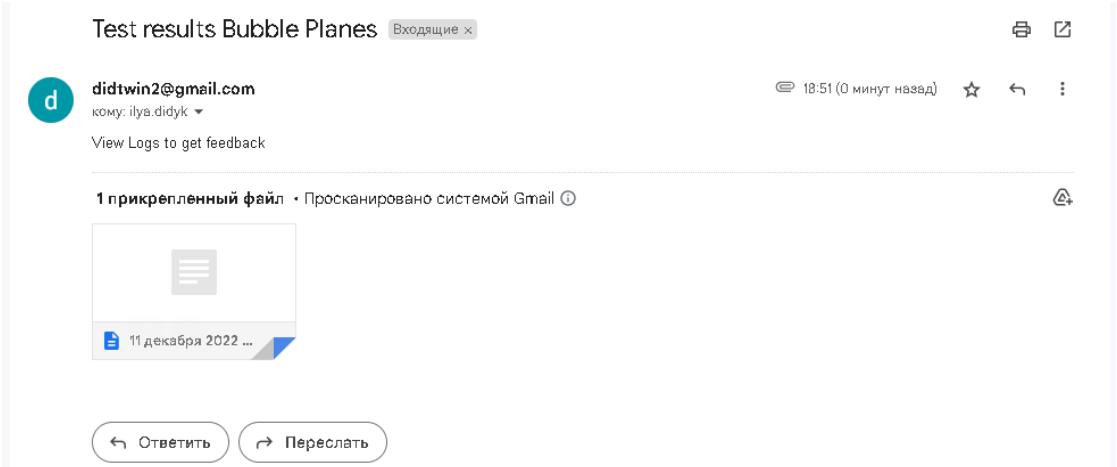

Рис. *23 Лист з результатами тестування*

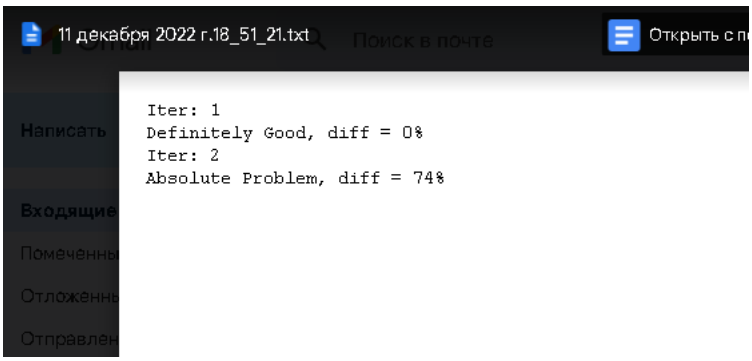

Рис. *24 Файл зворотного зв'язку з проміжними результатами тестування*

3.2 Реалізація застосунку мануального використання

Для того, щоб побачити наглядний результат порівняння, був розроблений User Interface для Screenshot Comparer`a. Якщо тестувальник після отримання результатів тестування захоче окремо подивитись на певний етап, з незадовільним результатом – він може використати іншу версію додатку, котра має в собі форму WinForms, на якій розміщено очікуваний, актуальний скріншоти та результат їх порівняння з наглядно відображеним Diff`ом.

Різниця з основним додатком лише в тому, що користувачу відображається Windows Form з клавішею Compare. Користувач може самостійно завантажити бажані зображення у теку result_expected або result_actual, що лежать у корені (\bin), але важливо, щоб вони мали однакову назву.

Рис. *25 Приклад використання Screenshot Comparer з UI(баг кольорів емулятора)*

Розроблений додаток має шляхи для вдосконалення з точки зору користувацького інтерфейсу та розширення функціоналу. Проте зараз він надає можливість користуватися ним у цілях тестування додатків, використовуючи будь-які засоби для тестових сценаріїв та зняття скріншотів.

Тим не менш, інструмент придатний лише для регресійного тесту і не може використовуватися під час перевірки нового арту навіть за наявності макета. Цю проблему можна частково вирішити, якщо використати певні алгоритми штучного інтелекту, проте розробка і підтримка, навчання та супровід такого ПЗ не має сенсу з точки зору продуктових затрат, і, навіть якщо використовується алгоритм, що апроксимує макет і фактичний результат, це не дасть чіткого розуміння щодо коректності інтеграції того чи іншого арту, бо ігрові рушії і рендерери можуть накладати певні обмеження, а якістю зображення іноді жертвують навмисно задля оптимізації пам’яті.

РОЗДІЛ 4 РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ МОБІЛЬНИХ КРОСПЛАТФОРМНИХ ІГРОВИХ ЗАСТОСУНКІВ

За результатами дослідження можна побачити, що галузь тестування ігор зараз знаходиться на стадії зародження. Ігрові програми на відміну від інших, мають низку особливостей, котрі не дозволяють повноцінно покрити тестами програмну систему, і це виражається на декількох рівнях: рівень людського сприйняття, рівень доцільності витрат, рівень технічних можливостей.

4.1 Особливості тестування на рівні людського сприйняття

Що стосується таких комплексних програмних систем як ігри, на рівні людського сприйняття є певні особливості відносно інших програм. Різні типи тестування стикаються з різними проблемами, виділяючи основні, такі як:

Функціональне тестування. Функціонал застосунку може змінюватись, замінюватись, автоматизоване тестування не може оцінити доцільність того чи іншого функціоналу у проекті, на відміну від гравців. Мануальний тестувальник є саме тою людиною, яка повністю знає як і за якими принципами працює застосунок і може відсіяти вдалі і невдалі функціональні рішення, виступає провідником між гравцем та командою-розробником. Наприклад, на технічному рівні функціонал працює без нарікань, але гравці почали масово йти з гри, а все тому що у гру добавили ту функцію, яка їм не подобається. Тоді функцію потрібно видалити, а тестувальнику — провалідувати, що проект працює коректно.

Тестування локалізації. При інтеграції локалізації за принципом ключ-значення, автоматизоване тестування може лише підтвердити, що значення та ключі співпадають. Але не завжди увесь текст, що використовується у грі — є у коді гри безпосередньо. Як приклад — монетизація через придбання певних товарів інтегрується через API Google Play Store. Важливо перевірити яким

саме символом відображається та або інша валюта на кирилиці, латиниці, яке кодування використовує та чи інша бібліотека. Людське сприйняття швидко виявить і провалідує проблему, на відміну від автоматизованого інструмента.

Тестування UI. На рівні людського сприйняття — тестування UI полягає у виявленні недоліків у роботі як художника, так і інтегратора. Тестувальник має спиратися на декілька факторів, які не властиві комп'ютеру, а саме: знання основних принципів графічного дизайну(людино-машинна взаємодія, теорія кольору, тощо), слідування загальній стилістиці застосунку, власне сприйняття краси. Також, за власним досвідом, машині не властиві певні фобії, наприклад, вона не визначить, що іконка, припустимо, що має багато цяток – може викликати приступ трипофобії.

Тестування UX. Користувацький досвід є невід'ємною частиною гарної гри. Наскільки користувачу зручно виконувати ті чи певні дії (переходити по меню, взаємодіяти з тултипами та попапами, тощо). Користувач може втомитися закривати рекламні попапи, може бути збентежений кількістю нотифікацій, що приходить від застосунку, в свою чергу програма не є здатною оцінити такі аспекти, на відміну від експерта в області якості ігрових застосунків.

Тестування вимог. Ігровим застосункам притаманна динамічність розвитку. Вимоги можуть змінюватись під час розробки, між версіями, після випуску застосунку в реліз. Задача тестувальника у цьому випадку — передбачити можливі, а що найважливіше — неможливі зміни, котрі можуть знадобитись. Потрібно враховувати можливості функціональних контролерів і методи обробки інформації. Якщо менеджер, що ставить задачу – не достатньо глибоко занурений у проект, або у інші його аспекти, такі як правила сервісів, платформ розповсюдження, тощо — задача тестувальника – повідомити про це. З власного досвіду — у гру має додатися функціонал що має у собі певні риси азартних ігор, тож за правилами магазину — отримує певну вікову відмітку і може втратити рекламних партнерів, просування платформою тримачем та ін. Тож задача досвідченого тестувальника — повідомити про існуючу проблему, яку точно не зможе виявити автоматизований інструмент.

Плейтестування. Особливий тип тестування для відеоігор, що набирає цільову, нецільову, нейтральну аудиторії гри, та QA. Під час плейтестування визначається рівень складності гри в цілому, або її певних аспектів. Звичайно можна написати інструмент, котрий буде автоматично проходити той чи інший рівень, проте, результат завжди буде один і той самий, на відміну від результату, що зумовлений людським фактором.

Регресійне тестування. Цей тип тестування однозначно підходить для автоматизації, тому що з точки зору людського сприйняття — це є рутина, котра швидко виснажує. Там де машина буде слідкувати інструкціям — людина може щось прогледіти, тому що на перший погляд функціонал, котрий раніше вже тестувався не один раз — працює як треба, виникає проблема «перенавчання» котра властива нейромережам, коли замість того, щоб проводити тестування у вузьких кейсах — людина обійдеться лише перевіркою у стандартному user-flow.

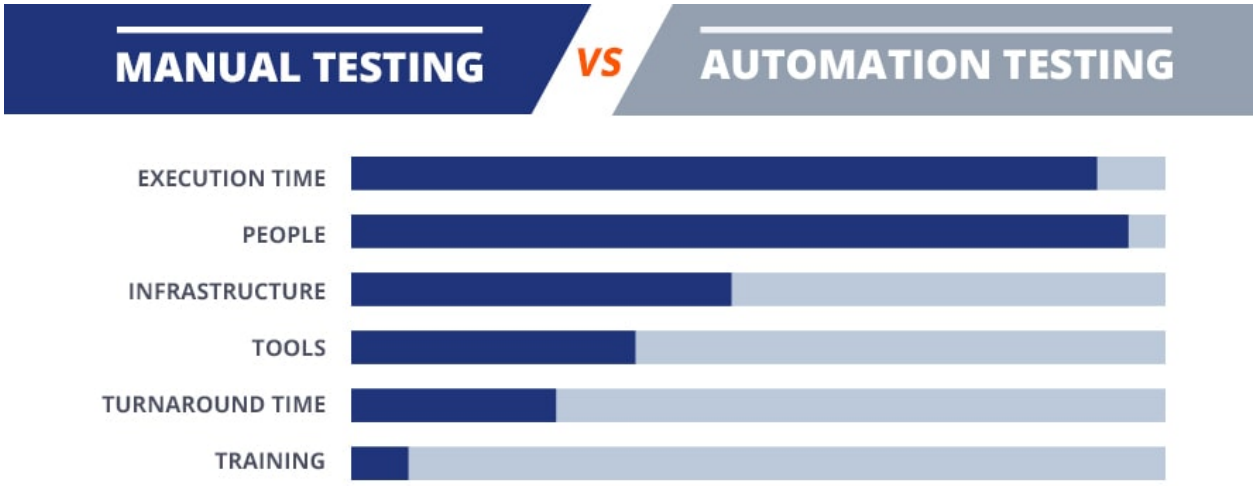

Рис. *26 Статистика за підходами до тестування*

4.2 Особливості тестування на рівні доцільності витрат

З точки зору доцільності витрат — галузь тестування ігор, нажаль йде по принципу: найкраще-найдешевше. На сьогоднішній день розробники ігор прибігають до розповсюдженої практики «тестуй граючи», а саме — Green

Light, де звичайним гравцям безкоштовно(у форматі відкритого альфа/бета тесту), або за гроші(у форматі закритого альфа/бета тесту) дають пограти у сирий продукт, де розробники сподіваються виявити якісь проблеми за допомогою відповідальних гравців, що пишуть баг репорти, або за допомогою інструментів трекінгу дій гравця, також таке явище є одним з проявів краудсорсингу. Таким чином відбувається економія на штаті внутрішньо-компанійського відділу QA, або аутсорсі.

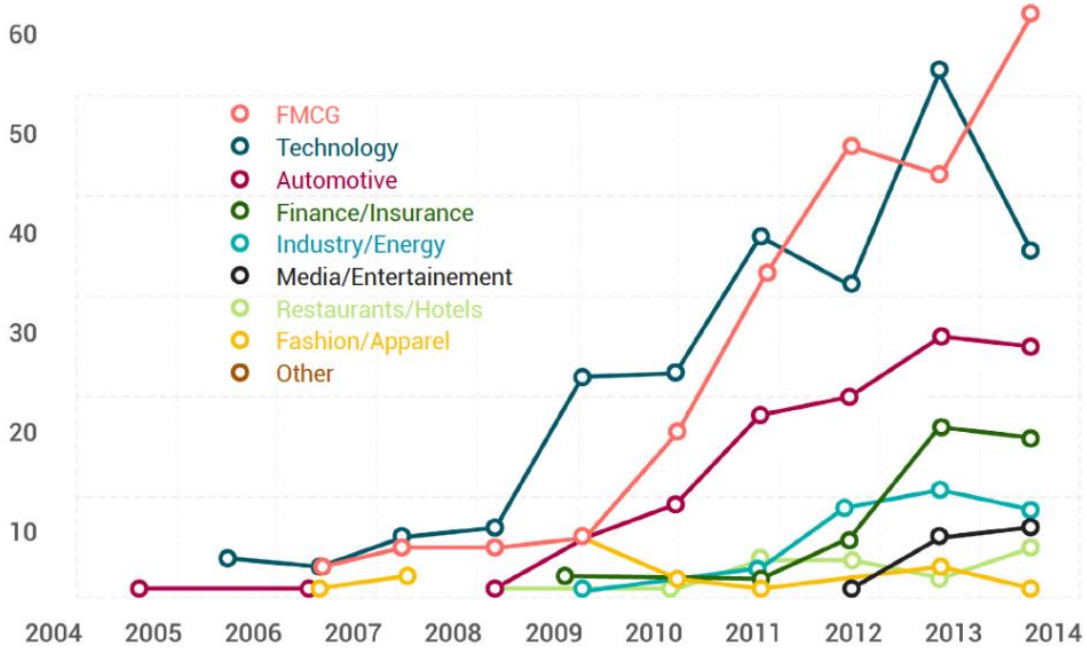

Рис. *27 Використання краудсорсингу у різних галузях*

Зазвичай, економія на спеціалістах забезпечення якості програмного забезпечення, особливо QA, що тестують проект під час розробки — проблеми можуть виявлятися занадто пізно та не можуть бути вирішеними швидко чи взагалі. Тому розробники мають свій штат QA, зазвичай manual, навчають їх та виховують спеціалістів у галузі менеджменту або автоматизації.

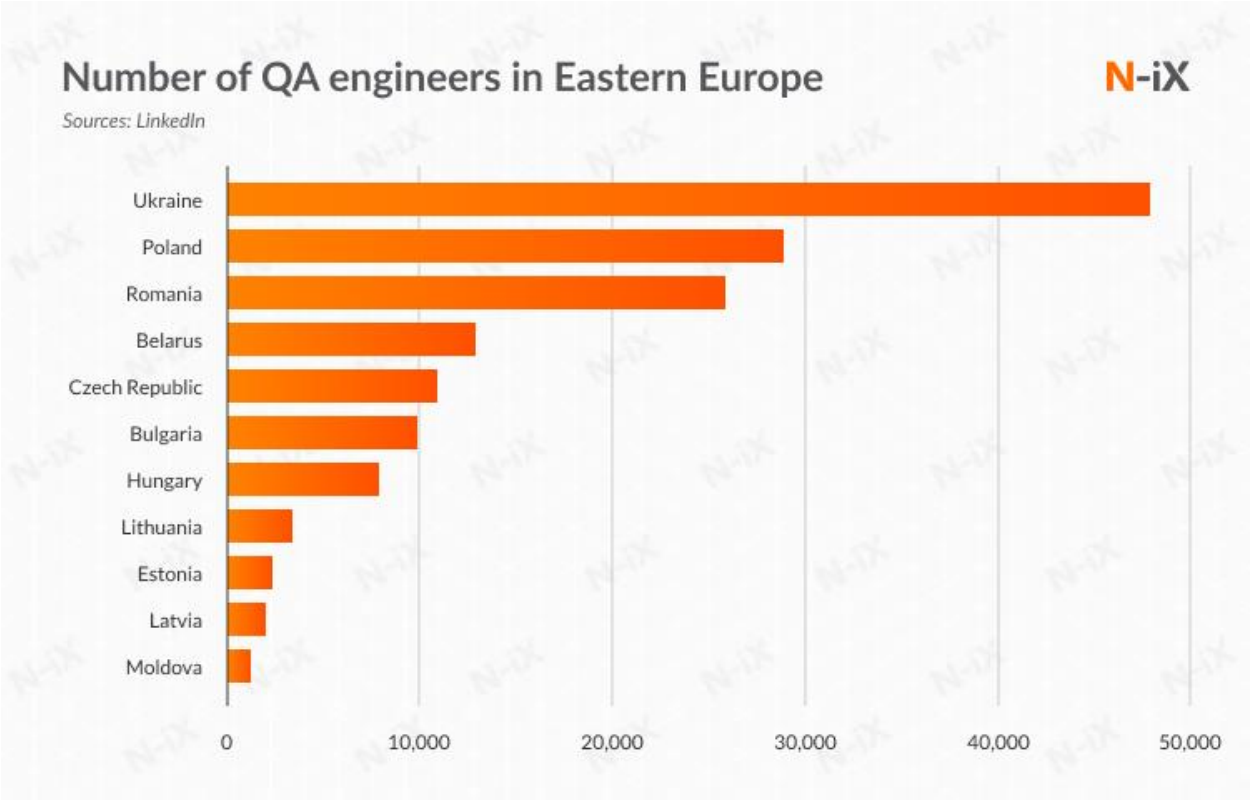

Рис. *28 Ринок QA інженерів у Східній Європі*

Проте на автоматизацію може не вистачити ресурсу, бо зміни відбуваються ледь не щотижня, особливо якщо розроблюється гра-сервіс, а це потребує певних навичок і вміння не тільки продумати тест-кейси, а й інтегрувати їх у автоматизовану тестову систему, актуалізуючи її під нові реалії.

Також, можна придумати і розробити певні системи автоматизованого тестування на основі штучного інтелекту (прогонщики), котрі вже використовуються такими крупними компаніями, як, наприклад, Amazon. Але кількість витрат на розробку, навчання, підтримування нейромереж — не є доцільною з фінансової точки зору, і, якщо такі речі стають частиною автоматизованого тестування – це або ентузіазм у малих масштабах, або тимчасова оренда вже існуючих сервісів(AWS Device Farm, AWS X-Ray).

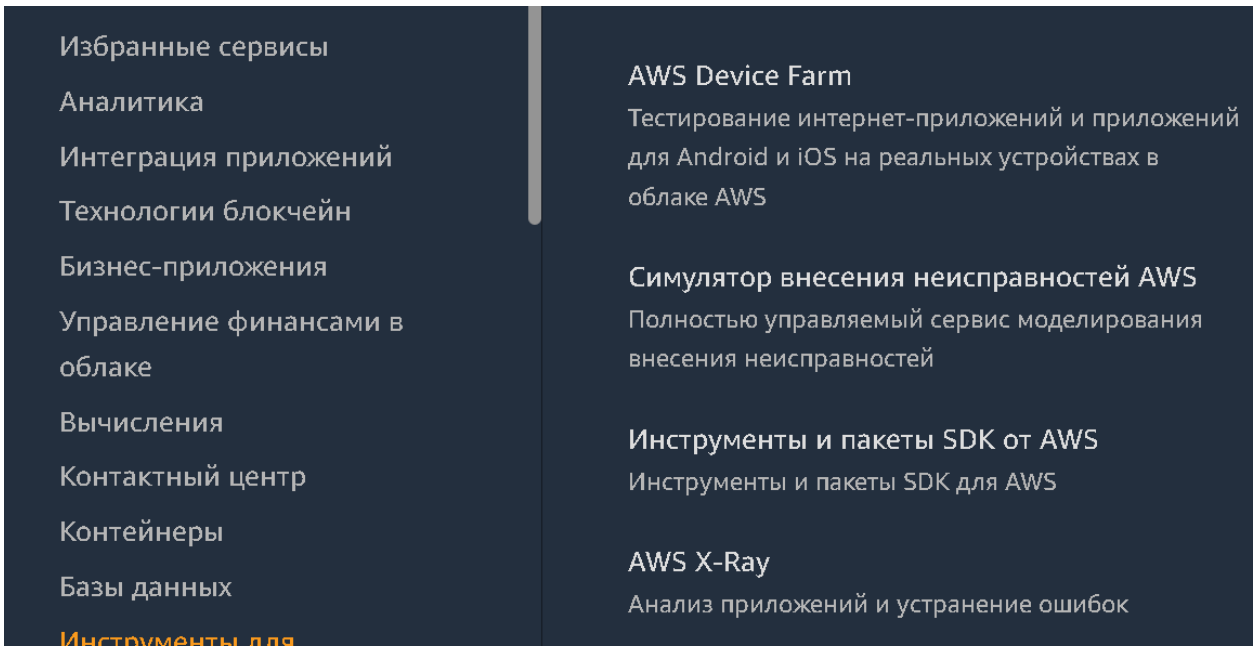

Рис. *29 Сервіси для автоматичної прогонки застосунку на предмет помилок (Smoke, Sanity testing)*

4.3 Особливості тестування на рівні технічних можливостей

На рівні технічних можливостей автоматизоване тестування ігор наразі набуває своєї форми, з’являються певні тестові фреймворки, притаманні для конкретного жанру, або платформи.

Щодо кросплатформних застосунків, то на різних стадіях розробки — для автоматизованого тестування підходять різні інструменти, якщо продукт знаходиться на рівні проектування, або рівні початкової реалізації, можливо буде доцільним у цей момент інтегрувати інструменти білої скриньки, або навіть прибігти до методології розробки TDD, звичайно ж можна інтегрувати інструменти білої скриньки і на останніх етапах розробки, але за умови, що це зумовлено архітектурою.

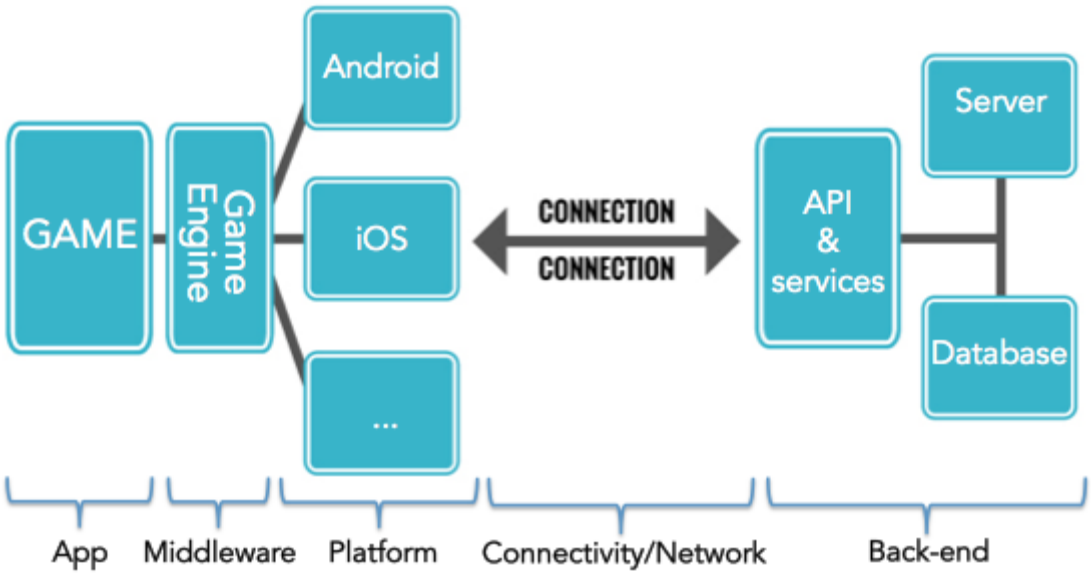

Рис. *30 Середньостатистична архітектура сучасних мобільних кросплатформних ігор*

Якщо ігровий проект вже вийшов в реліз, або знаходиться на фінальній стадії розробки або підтримується, доцільним буде інтегрувати автоматизовану тестову систему, засновану на інструментах чорної скриньки. Це не тільки не навантажить код, але й буде максимально наближено до поведінки реального гравця.

Як показало дослідження — автоматизація тестування ігрових застосунків поки що не розповсюджена саме по тих причинах, що вона не може покрити усі ті вимоги, котрі потрібні для якості продукту, тож розробники поки що бачать розвиток у екстенсивному методі, розширюючи штат менш кваліфікованими співробітниками або взагалі — звичайними гравцями.

4.4 Результати розробки Image Comparer`a

Розроблений програмний інструмент Image Comparer дозволяє зменшити навантаження на процеси ручного тестування у сегменті регресійних перевірок. Наразі він є тимчасовим рішенням у моїй компанії та згодом буде покращуватися під масове застосування. Зручним є те, що результати приходять

на пошту, де тестувальник може побачити у якому саме місці є розбіжність очікуваного та фактичного результату, і, використовуючи той самий інструмент, може самостійно подивитись у чому саме полягає проблема.

Інструмент не має обмежень щодо методів його використання у автоматизації тестування, тобто підходять як методи білої, так і чорної скриньки.

4.5 Вектори потенційного розвитку автоматизації тестування ігор

Дослідження показало, що автоматизувати можна не лише регресійні функціональні та приймальні тести. Під час роботи доводиться стикатися з іншими задачами, що є не менш придатними до автоматизованого тестування.

4.5.1 Автоматизація тестування веб-запитів

Наразі, переважна кількість мобільних ігрових застосунків в тому чи іншому ступені мають певну залежність від інтернет з’єднання, це може бути зумовлено архітектурою застосунку (клієнт-серверна), або необхідними умовами для функціонування не застосунку, а його обвісу, сторонніх бібліотек, що відповідають за аналітику, рекламу, платіжні системи, трекінг, тощо. Зазвичай ці бібліотеки відправляють певні запити до сервера, і отримують конкретну відповід, в залежності від стану системи. Таким чином, можна автоматизувати перевірку веб-запитів. На примітивному рівні, автоматизований аналіз трафіку вже реалізованих у програмах-сніферах(Fiddler і його спеціальна скриптова мова FiddlerScript чи інструмент Auto Responder), що мають доступ до пакетів мережі, проте спеціалізованого ПЗ під таку задачу, поки що не знайшлося, або ж воно не популярне.

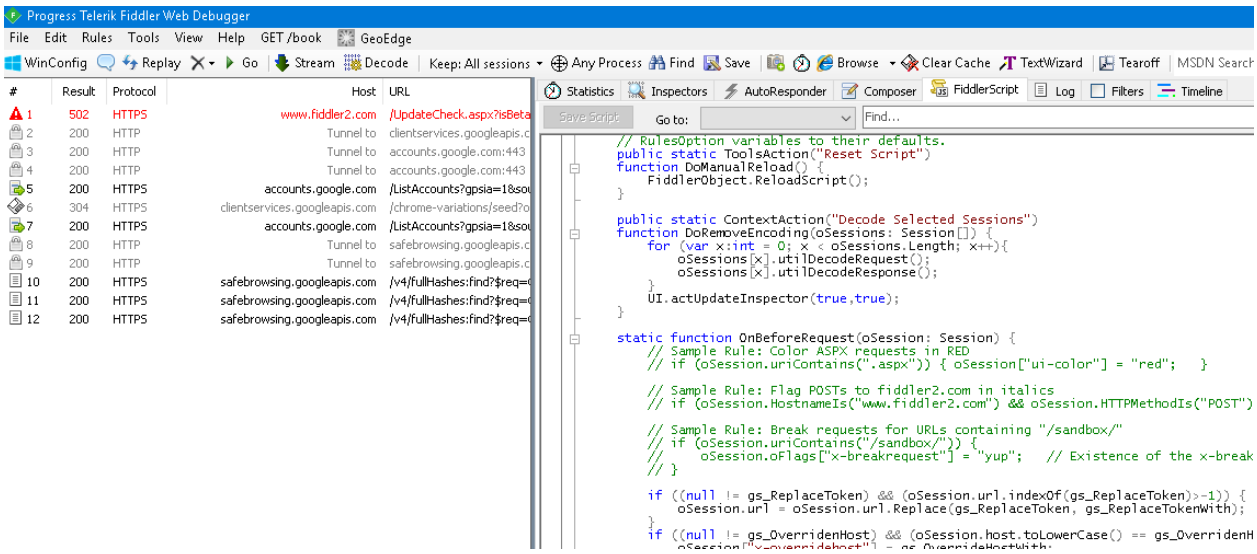

Рис. *31 FiddlerScript*

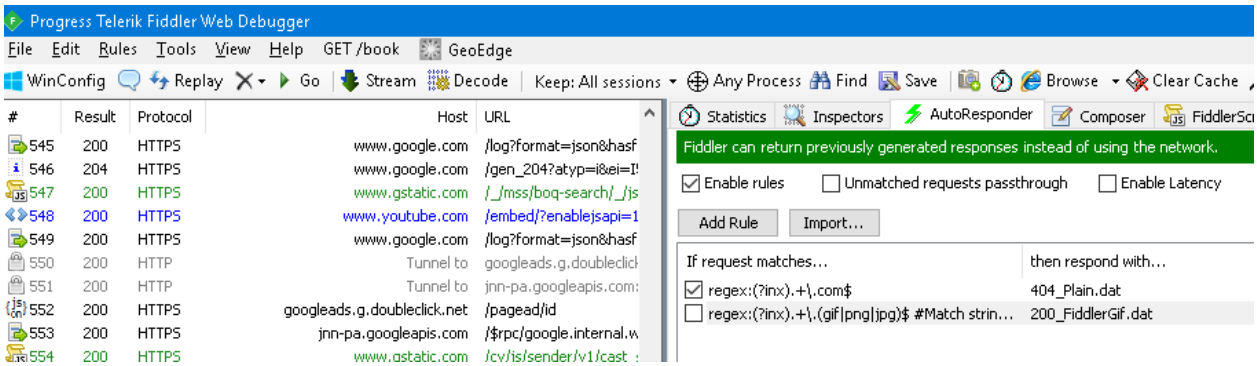

Рис. 32 *Fiddler Auto Responder*

4.5.2 Автоматизація тестування безпеки

Як правило, ігрові компанії тестують безпеку, за допомогою існуючих популярних методів зламу, що знаходять на спеціалізованих форумах з пен-тестінгу, ігрових форумах, хакерських форумах і т.д.

Зазвичай такі програми мають придатний до автоматизації інтерфейс, однакові принципи і сценарії роботи, та мають чітко визначений очікуваний результат, тому проблем з валідацією перевірки не виникне.

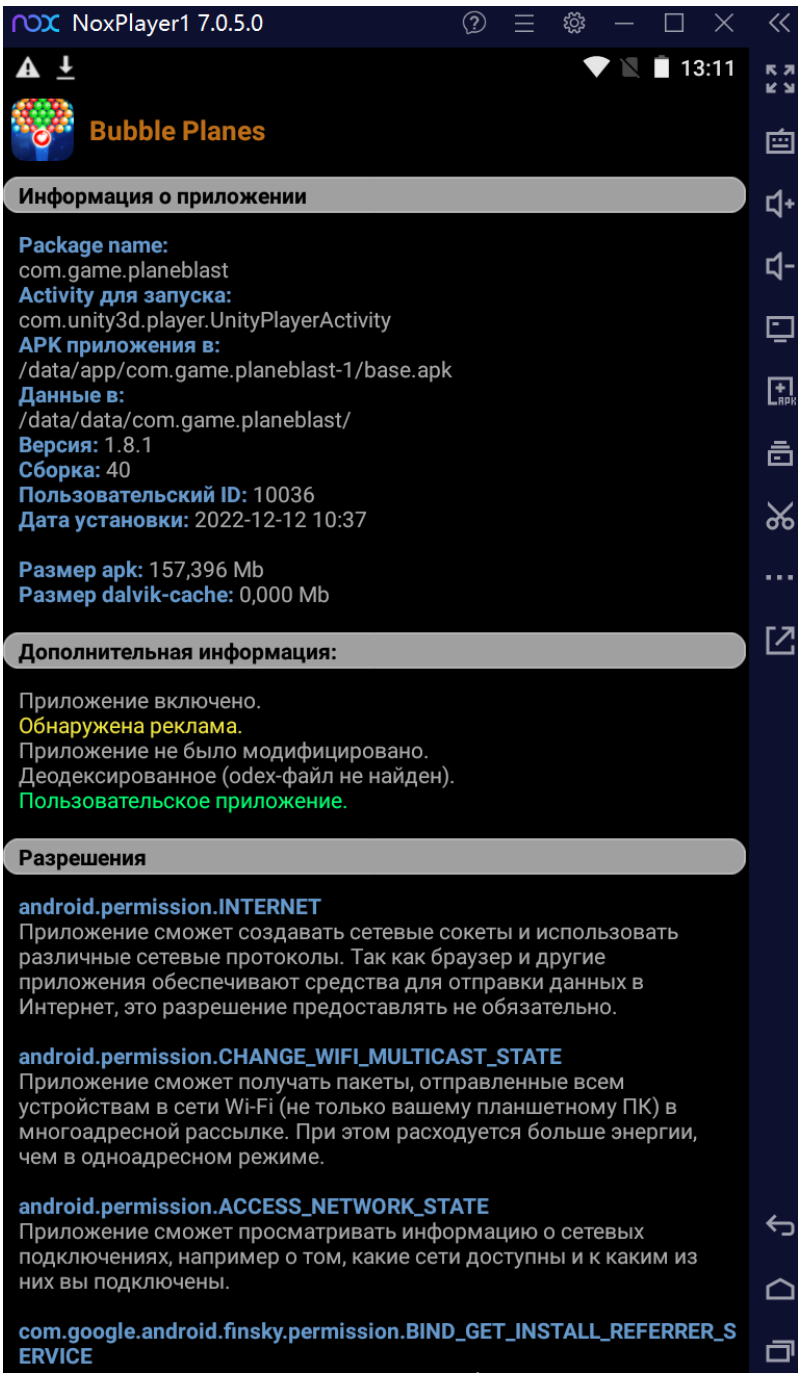

Рис. *33 Lucky Patcher*

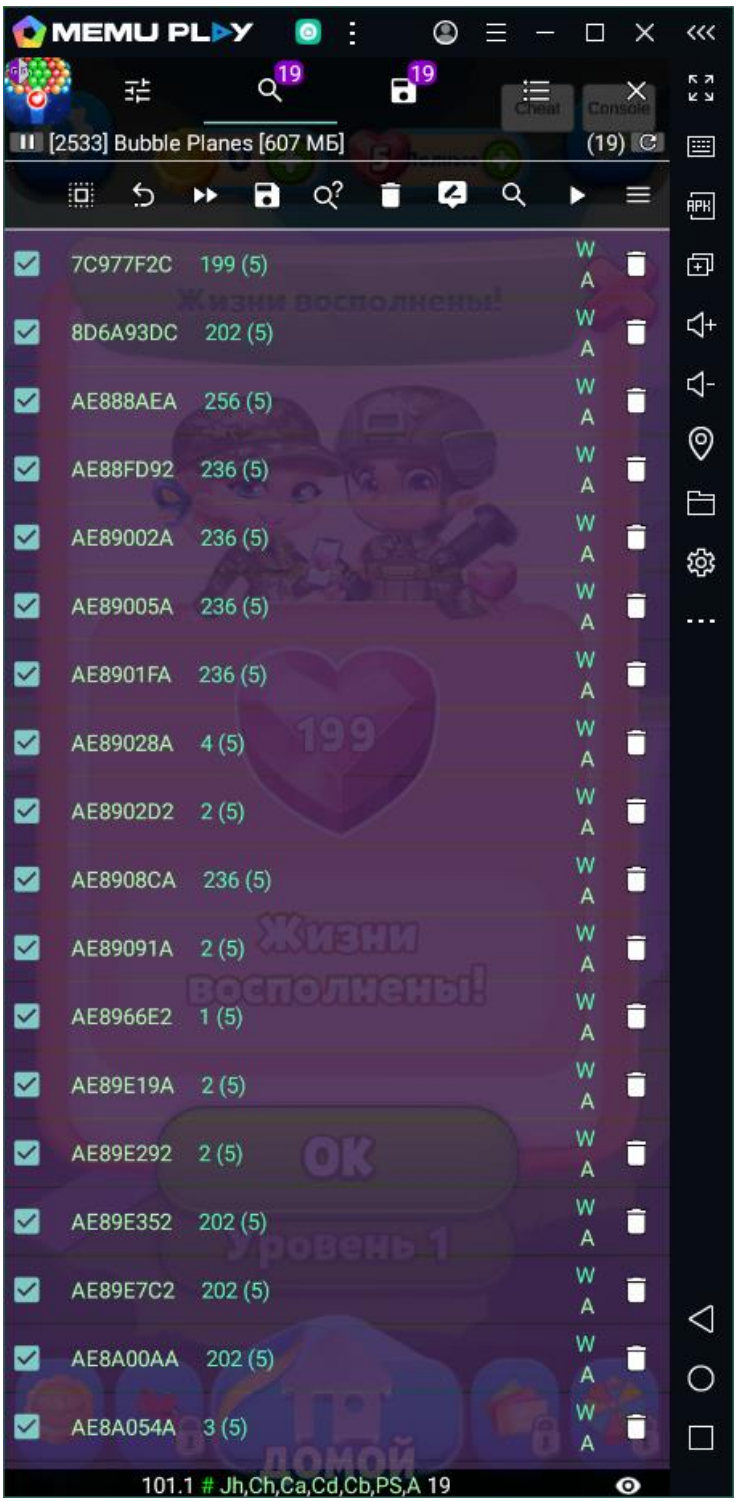

Рис. *34 Game Guardian*

Можна використати інструменти чорної скриньки, щоб створювати тестові сценарії, що враховують використання сторонніх чіт-програм, таких як Lucky Patcher, або Game Guardian.

4.5.3 Автоматизація конфігураційного тестування

Мобільні кросплатформні ігрові застосунки повинні працювати однаково добре на максимально можливій кількості платформ та їх конфігурацій с точки зору апаратного та програмного забезпечення. Тому важливо, щоб кожна ітерація перевірялась регулярно на усіх доступних видах платформ і девайсів.

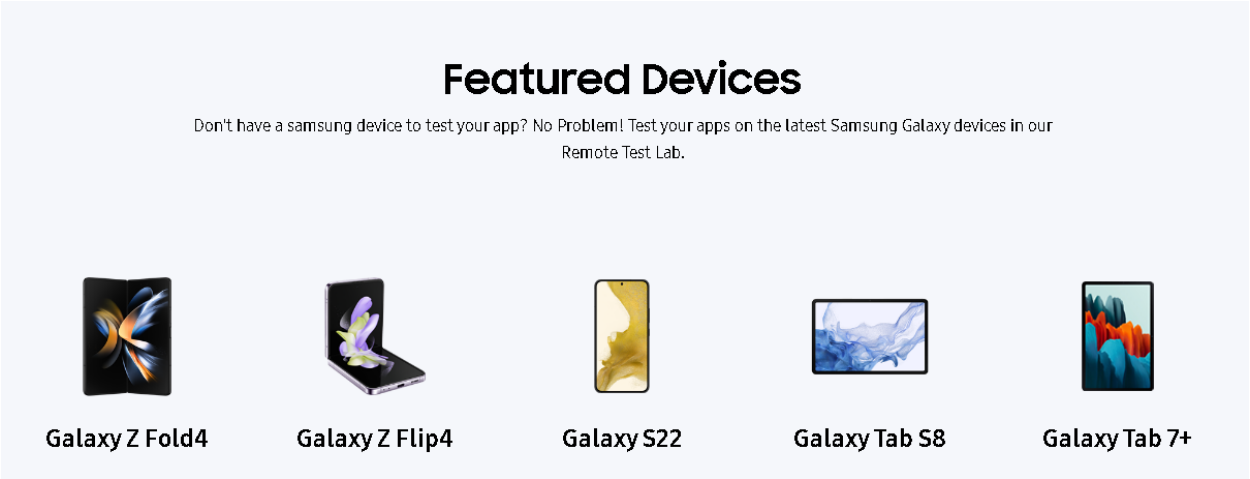

Рис. *35 Доступні девайси на Samsung Test Lab*

Рис. *36 Доступні девайси на ATOMP.IO Device Farm*

Тож було гарно розробити інструмент-автоматизатор, котрий агрегує декілька сервісів-ферм та збирає результати Sanity тестування кожного разу як застосунок відправляється у реліз. Прикладами ферм є Samsung Remote Test Lab, Xamarin Test Cloud, AWS Device Farm, тощо.

4.5.4 Автоматизація тестування продуктивності

Дуже важливо у сучасному швидкому світі мати інструмент, котрий автоматично тестує швидкість завантаження програми та виводить результати, кореляції, залежності, динаміку росту або спаду продуктивності. На заміну ручним таблицям, можуть прийти інструменти, котрі заміряють швидкість завантаження застосунку, збереженням даних і т.д. Що дозволить виявлять проблему на більшій кількості етапів розробки застосунку такий інструмент можна вже зараз створити для певної платформи, засновуючись на вже готових інструментах моніторингу.

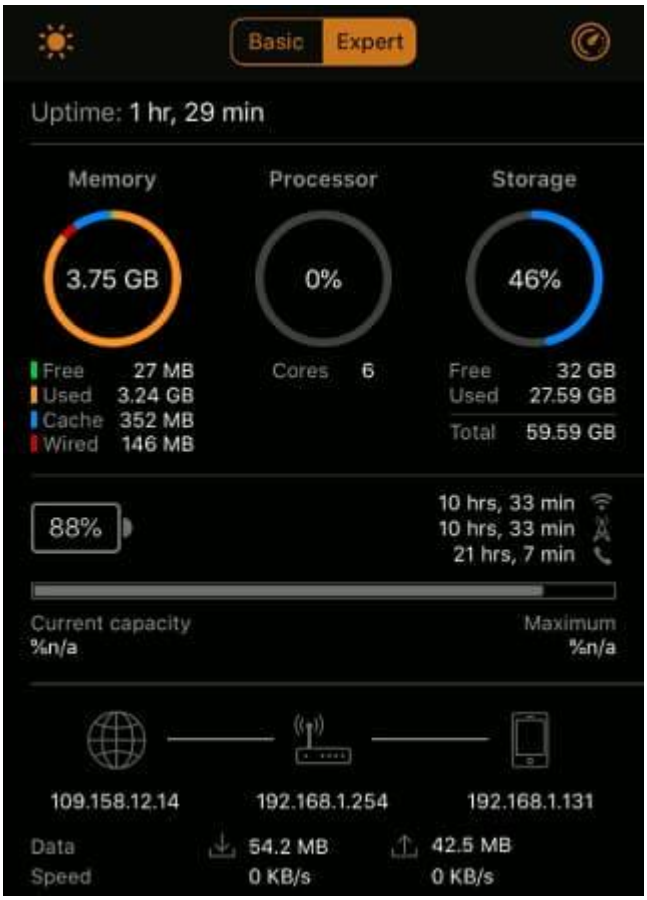

Рис. *37 Інструмент моніторингу продуктивності для iOS*

ВИСНОВКИ

1. Досліджена проблема автоматизованого тестування мобільних кросплатформних ігрових застосунків.

2. Досліджені сучасні методи автоматизованого тестування мобільних застосунків.

3. Проведений аналіз отриманих результатів дослідження.

4. Запропоноване можливе рішення проблеми.

5. Був визначений концепт програмного застосунку для рішення проблем автоматизованого тестування мобільних кросплатформних ігрових застосунків.

6. Був розроблений інструмент автоматизованого тестування скріншотами Image Comparer.

7. Був проведений аналіз можливих шляхів для впровадження автоматизованого тестування і розробки необхідних інструментів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дідик І. В., магістрант 1 курсу, Коломоєць Г. П., доцент — науковий керівник. Дослідження проблеми автоматизованого тестування мобільних кросплатформових ігрових застосунків. Молода наука-2022 : зб. наук. праць студентів, аспірантів і молодих вчених. Запоріжжя : ЗНУ, 2022.

2. Дідик І. В., магістрант 2 курсу, Коломоєць Г. П., доцент — науковий керівник. Доцільність автоматизованого тестування ігрових застосунків. АКТУАЛЬНІ ПИТАННЯ СТАЛОГО НАУКОГО-ТЕХНІЧНОГО ТА СОЦІАЛЬНО-ЕКОНОМІЧНОГО РОЗВИТКУ РЕГІОНІВ УКРАЇНИ : зб. наук. праць студентів, аспірантів і молодих вчених. Запоріжжя : ЗНУ, 2022.

3. Lars Doucet, Anthony Pecorella Game engines on Steam: The definitive breakdown, 2021. URL: https://www.gamedeveloper.com/business/game-engines-on-steam-the-definitive-breakdown (дата звернення 14.04.2021р.).

4. Oleg Gulsom, Докладніше про Sikuli у автоматизації тестування, 2012.URL: https://habr.com/ru/post/163883/ (дата звернення 13.04.2021р.).

5. Олексій Сімкін, Airtest IDE — новий шлях в автоматизації тестування мобільних ігор? , 2019.URL: https://habr.com/ru/post/461773/ (дата звернення 13.04.2021р.).

6. Олексій Сімкін, AirTest IDE и Image Recognition — автоматизація тестування мобільних ігор основуючись на розпізнаванні зображень, 2019.URL: https://habr.com/ru/post/462587/ (дата звернення 14.04.2021р.).

7. Журнал «Хакер», Візуальні скрипти. Sikuli: проста автоматизація через скріншоти та Python, 2011.URL: https://xakep.ru/2011/06/11/57412/ (дата звернення 14.04.2021р.).

8. Viaccess-Orca, Why and how to automatize testing in unity projects?, 2020.URL: https://vrtogether.eu/2020/11/23/automatize-testing-unity-projects/ (дата звернення 14.04.2021р.).

9. QA TestLab, Матеріали лекції №7, Тестування ігор, 2021.URL: https://training.qatestlab.com/blog/course-materials/lecture-game-testing/ (дата звернення 12.11.2021р.).

10. Cem Kaner, Hung Q. Nguyen, Jack Falk, Testing Computer Software, P.480, 1999.

11. Mark Fewster, Dorothy Graham, Software Testing Automation: Effective Use of Test Execution Tools, P.592, 1999.

12. Tim Riley, Adam Goucher, Beautiful Testing: Leading Professionals Reveal How They Improve Software (Theory in Practice) 1st Edition, P.350, 2009.

13. Daniel Knott, Hands-On Mobile App Testing: A Guide for Mobile Testers and Anyone Involved in the Mobile App Business 1st Edition, P.256, 2015.

14. Jesse Schell, The Art of Game Design: A Book of Lenses 1st Edition, P.520, 2008.

15. Lee Copeland, A Practitioner's Guide to Software Test Design Illustrated Edition, P.312, 2004.

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Дідик Ілля Вячеславович , студент 2 курсу магістратури, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz17bd-9@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Особливості автоматизованого тестування мобільних кросплатформних ігрових застосунків**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 30.11.2022 _____ Дідик Ілля Вячеславович
(студент)

Дата 30.11.2022_____ Коломоєць Геннадій Павлович
(науковий керівник)