DOI https://doi.org/10.32782/2220-8674-2025-15-1-25

А. А. Клєпцов[1], магістр з інженерії програмного забезпечення ORCID: 0009-0008-3147-0618
В. А. Гусєва-Божаткіна[2], старший викладач
кафедри програмного забезпечення автоматизованих систем ORCID: 0000-0002-1117-3391
[1] ТОВ «28Software»
[2] Національний університет кораблебудування імені адмірала Макарова
e-mail: mrlcrosl@gmail.com, GusevaBozh@meta.ua

АЛГОРИТМ ПОКРАЩЕННЯ ВЗАЄМОДІЇ КОРИСТУВАЧІВ У МОБІЛЬНИХ ІГРАХ ЧЕРЕЗ АДАПТИВНИЙ ПЕРЕЗАПУСК ГРИ

Анотація. Кількість випущених ігор на ринку прямо пропорційно залежить від вибагливості користувачів. Якщо ще 10–20 років тому багато гравців могли не звертати уваги на недоліки, баги або нестабільність гри, то зараз продукт, скоріш за все, буде видалений і забутий. Розробники отримають погані відгуки в маркеті, а гравець із легкістю знайде альтернативу. Звичайно, значна частина багів виправляється в процесі розроблення в тандемі спеціалістів по розробкам і Quality Assurance, але все виправити неможливо. Особливо у великих проєктах існують мільйони різних комбінацій взаємодії з грою, які можуть призвести до нестабільності, виходу з ладу основних механік або взагалі аварійному закриттю застосунку. Щоб розв'язати такі проблеми, компанією Google було розроблено додаткові інструменти під назвою Firebase Crashlytics, які інтегруються безпосередньо в гру. Пізніше такий підхід отримання звітів про помилки набув розповсюдження як загальний, який можна назвати терміном «крашлітика». Крашлітика допомагає в реальному часі бачити в програмному коді гри помилки, з якими стикаються реальні користувачі. Відповідно, розробники можуть вчасно реагувати на проблеми, які часто трапляються, і виправляти їх. Водночас крашлітика не є магічною пігулкою до успіху. Зазвичай User Experience (UX) від помилок у грі є негативним. Користувача просять надіслати скріншот, описати, що він робив, коли сталась помилка і т. ін. Як показує практика, цим будуть займатись одиниці. У статті запропоновано альтернативний підхід до взаємодії користувача з помилками у грі, розроблений на рушії Unity. Розроблено алгоритм збереження стану гри, імітації завантаження, або це ще можна назвати «втратою зв'язку» та плавного повернення гравця до точки, де відбулася помилка в логіці. Таким чином, більшість гравців навіть не зрозуміють, що в грі щось пішло не так, а можуть списати це на свій поганий інтернет, необхідність завантаження додаткового контенту в грі тощо.

*Ключові слова***:** Firebase Crashlytics, Cloud Diagnostics, крашлітика, помилка, UX, скріншот, Unity.

Постановка проблеми. Індустрія мобільних ігор стала невід'ємною частиною нашого світу. Незважаючи на те, що індустрія мобільних ігор зазнала кардинальних змін за останні роки, у 2023 році глобальний ринок ігор згенерував понад US$184 мільярди доходу, з яких саме мобільні ігри принесли близько US$90 мільярдів. Ця частка становить 49% від загального обсягу [1]. Можна з упевненістю сказати, що ринок мобільних ігор дуже великий, а проблема якості продуктів стоїть дуже гостро. Через велику кількість продуктів гравці просто не будуть звертати увагу на ігри з низьким рейтингом, відповідно, компанія-розробник може понести величезні збитки. Зазвичай провальні релізи на ігрових платформах не набувають великого розголосу, але це не означає, що їх немає.

Оскільки геймдев під мобільні платформи та ПК/консолі дуже пов'язані, можна розглянути приклад таких ігор. Наприклад, гра Cyberpunk 2077 – одна з найочікуваніших ігор десятиліття – розроблена польською компанією CD Projekt RED. Анонс гри відбувся у 2012 році, й вона привертала увагу завдяки обіцянкам інноваційного геймплея, глибокої RPG-системи

та масштабного відкритого світу. На релізі гра зіткнулася із серйозною критикою, особливо на консолях попереднього покоління (PS4 та Xbox One). Основні проблеми включали: збої, низьку продуктивність, вирізаний контент, помилки в логіці. Результат такого провального релізу: гравці масово залишали негативні відгуки, гра була знята з продажу в PlayStation Store, численні повернення коштів за гру, акції компанії-розробника впали на 50% [2]. Компанії знадобилось близько 2,5 років для виправлення гри та стабілізації оцінок у маркетах, і близько 4 років – для відновлення ціни на акції.

Звичайно, цього можна було запобігти, випускаючи більш якісний продукт. Але не завжди час грає на руку. Тому в таких ситуаціях треба думати, як покращити взаємодію гравця з технічними проблемами в грі.

Аналіз останніх досліджень. Існують багато статей щодо покращення User Experience (UX) в іграх, але, на жаль, вони не стосуються саме сприйняття користувачем помилок у грі. Найчастіше вони описують UX у загальній перспективі, наприклад: плавне занурення гравця в гру, пояснення механік, зрозумілий User Interface (UI). Наприклад, Герда Губер і Давид Рюкель у своїй науковій доповіді «UX and Serious Games – A Research Agenda» описують важливість UX у іграх [3].

Німмі Рашініка Віраддана, Сарра Хабчі та Шейн Макінтош у своїй науковій статті «Crash Report Prioritization for Large-Scale Scheduled Launches» розглядають методику пріоритизації звітів про збої в програмному забезпеченні для великих запланованих запусків [4].

Деді Прасеця Крістіаді та його співавтори в доповіді «The effect of UI, UX and GX on video games» аналізують вплив дизайну інтерфейсу користувача (UI), досвіду користувача (UX) та ігрового досвіду (GX) на ефективність та сприйняття відеоігор [5].

Пейї Їн у роботі «Research on design and optimization of Game UI framework based on Unity3D» досліджує процеси проєктування та оптимізації фреймворку інтерфейсу користувача для ігор, створених на основі Unity3D [6].

Вказані роботи не охоплюють важливий аспект – сприйняття користувачем помилок у грі та можливостей їх подолання. Саме тому в цій статті розглядається підхід, який фокусується на покращенні досвіду користувача шляхом упровадження механізму адаптивного перезапуску гри. Цей підхід дозволяє зменшити рівень фрустрації у гравців, сприяючи більш позитивному сприйняттю гри загалом.

Формулювання мети статті (постановка завдання). Метою статті є:

1) дослідження наявних методів взаємодії користувача з помилками під час ігрового процесу;

2) розроблення алгоритму для покращення UX-методом збереження стану гри, підміною зображення й тихого перезавантаження гри у фоновому режимі;

3) дослідження ефективності розробленого методу.

Основна частина. Спочатку треба розглянути вже наявні методи відлову помилок під час ігрового процесу та дослідити, як вони впливають на UX.

Якщо брати до уваги ігри, розроблені на рушії Unity, – найпопулярнішому рушії для розроблення ігор під мобільні платформи, то прямо «з коробки» можна підключити Cloud Diagnostics: Game Code Debugger & Reporting [7]. Це інструмент Unity, що дозволяє розробникам ефективно виявляти, аналізувати та виправляти помилки в ігровому коді. Він автоматично збирає інформацію про краші (збої в грі, після яких вона закривається) та помилки, надаючи детальні звіти з такими ключовими даними, як стек викликів, інформація про пристрій, версію операційної системи (ОС) та інше.

Інструмент підтримує логування подій і створення власних звітів, а також групування та фільтрацію помилок для швидшого аналізу. Це дає змогу розробникам відстежувати непо-

ладки в реальному часі, зосереджуючись на найважливіших проблемах.

Cloud Diagnostics інтегрується з Unity Dashboard, де розробники можуть легко переглядати та аналізувати помилки. Завдяки цьому UX стає стабільнішим, а час на пошук і виправлення багів суттєво скорочується, що особливо важливо для великих проєктів із широкою аудиторією.

Другий інструмент, котрий також користується попитом, – Firebase Crashlytics [8]. Цей інструмент для моніторингу та діагностики помилок у мобільних застосунках та іграх, який є частиною платформи Firebase від Google. Він автоматично фіксує збої, надає детальні звіти в реальному часі та допомагає розробникам швидко визначати й усувати критичні проблеми.

Crashlytics створює звіти про помилки з ключовою інформацією, як-от стек викликів, параметри пристрою, версія застосунку та ОС. Інструмент автоматично групує схожі помилки, визначаючи їх вплив на користувачів, і дозволяє додавати власні логи для кращого розуміння контексту подій перед крашем.

Проста інтеграція SDK Firebase робить Crashlytics зручним для використання в застосунках на iOS, Android та Unity. Це дозволяє розробникам швидко реагувати на помилки, покращуючи стабільність і загальний досвід користувачів, а також заощаджує час на діагностику та розв'язання проблем.

На основі багаторічного досвіду роботи з обома інструментами можна зробити висновок, що вони обидва забезпечують високий рівень функціональності. Firebase пропонує широкий спектр корисної інформації та модульність, яка дозволяє видаляти або додавати потрібні функції через пакети. Проте в багатьох випадках, зокрема в контексті відлову винятків (Exceptions), пакет Firebase Crashlytics поступається можливостям Cloud Diagnostics. Багато винятків просто не логуються пакетом. Також стек викликів (Stack trace) не дає корисної інформації на відміну від Cloud Diagnostics. Отже, під час дослідження вирішено використовувати інструмент Cloud Diagnostics від Unity.

На цьому етапі також варто зазначити, що обидві системи вміють працювати з двома типами помилок у коді: виняток (Exception) та збій (Crash). Дослідимо різницю між ними.

Exception – це програмна помилка, яка виникає під час виконання коду, але не призводить до повного завершення роботи програми. Наприклад, це може бути спроба доступу до неіснуючого індексу масиву або виклик методу для null-об'єкта. У таких випадках програма може продовжувати працювати, якщо виняток буде належним чином оброблено.

Crash, на відміну від Exception, є критичною помилкою, яка призводить до аварійного завершення роботи програми. Це може відбуватися, наприклад, через неправильну роботу пам'яті або критичні збої у взаємодії із системними компонентами. Такі помилки потребують негайної уваги розробників, оскільки вони суттєво впливають на користувацький досвід.

Розуміння різниці між винятками та збоями дозволяє розробникам ефективніше діагностувати та виправляти проблеми, використовуючи відповідні інструменти, як-от Firebase Crashlytics або Cloud Diagnostics.

На перший погляд, може здатися, що Exception – це щось неважливе і його можна ігнорувати, але це не так. Розглянемо простий приклад.

Припустимо, в грі є економічна система. Розробник урахував, що кількість грошей у гравця не може бути від'ємною. Однак у певній ситуації це правило не оброблено належним чином, що призводить до виникнення винятка.

Короткий приклад коду на C# наведено у [9]. У цьому прикладі, якщо метод SpendMoney викликається із сумою, яка перевищує поточний баланс гравця, тоді генерується виняток *InvalidOperationException*. Якщо цей виняток не буде оброблено, гра може зупинитися, подальші дії користувача будуть ігноруватись програмним кодом або призводити то інших винятків, тим самим генеруючі нескінченну чергу помилок. З огляду на це винятки мають пря-

мий вплив на користувацький досвід, навіть якщо вони не є фатальними для всієї програми. Їх оброблення дозволяє створювати стабільніші й надійніші продукти. Якщо під час збоїв (crash) розробник ніяк не може вплинути на UX, бо гра банально закриється, то з винятками можна щось зробити.

Зазвичай розробники роблять подібні універсальні вікна (рис. 1), які просто відображають помилку та мають одну або декілька кнопок. Іноді також додається текстове поле для опису проблеми, яка виникла у гравця.

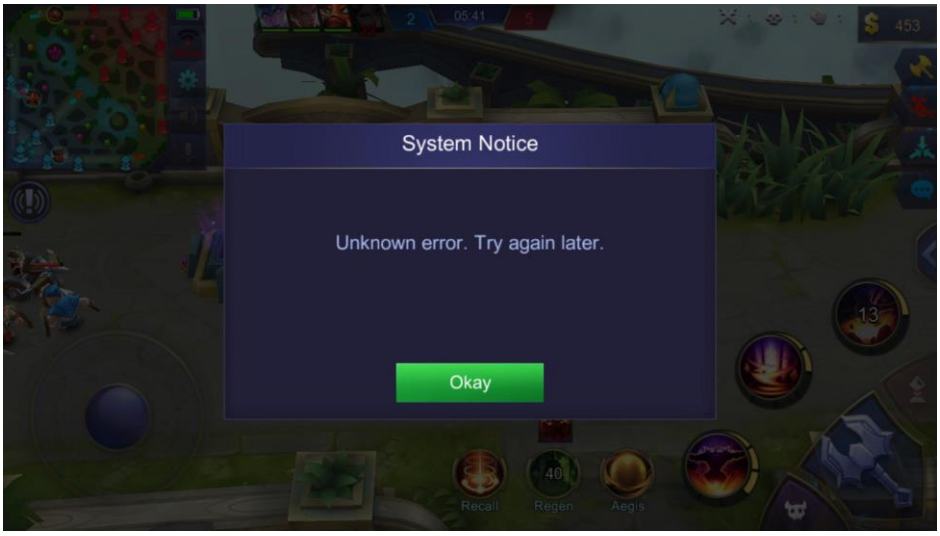

Рис. 1. Приклад UI, який сповіщає гравця про помилку

Як показує практика, це поганий UX. Гравців відлякують незрозумілі вікна, а також додаткові прохання описати проблему. Це напряму впливає на статистику та оцінку гри в маркеті.

Отже, для реалізації було створено тестовий проєкт Unity. Проєкт складається з кількох сцен і скриптів, що реалізують функціонал оброблення помилок у грі з покращеним UX. Ключова ідея полягає у створенні механізму, який обробляє помилки, затемнює екран, робить скриншот поточного стану, відображає його користувачу, перезавантажує основну сцену і водночас зберігає та відновлює стан об'єктів.

Сутності, які використано в проєкті, та алгоритм роботи:

1. Сцени

1.1. MainScene: Основна сцена гри, яка містить об'єкти, що змінюють свій стан у реальному часі, наприклад, змінюють колір, рухаються або відображають різні спрайти. Спрайт – це двовимірне растрове зображення в комп'ютерній графіці, яке можна відобразити на екрані.

1.2. ExceptionHandler: Сцена для оброблення помилок. Вона містить логіку, що перехоплює винятки та відображає спеціальне затемнення й скріншот стану гри.

1.3. Bootstrap: Початкова сцена, яка ініціалізує необхідні ресурси та відповідає за завантаження основних сцен.

2. Основні скрипти

2.1. ExceptionsHandler.cs: Реалізує перехоплення помилок за допомогою *Application.logMessageReceived*.

У разі помилки: Робить скріншот поточного стану. Відображає скріншот на UI Image. Потім активує затемнення і запускає перезавантаження основної сцени.

2.2. SaveManager.cs: Зберігає та відновлює стан об'єктів через PlayerPrefs. Використовує

інтерфейс ISavable, щоб визначити об’єкти, які можуть зберігати свій стан.

2.3. ISavable.cs: Інтерфейс для збереження та відновлення стану об’єктів. Реалізований у таких класах:

2.3.1. SavableImage: Зберігає стан спрайту (назву файлу).

2.3.2. SavableMover: Зберігає позицію об’єкта.

2.3.3. SavableColor: Зберігає колір об’єкта.

2.4. FakeException.cs:

2.4.1. Скрипт для тестування оброблення винятків.

2.4.2. Викликає штучну помилку для демонстрації механізму.

2.5. Bootstrap.cs: Скрипт для завантаження сцен. Перевіряє, чи завантажена сцена ExceptionHandler, і завантажує її лише за необхідності.

3. Логіка роботи

3.1. Ініціалізація: Bootstrap завантажує ExceptionHandler і MainScene.

3.2. Робота в грі:

3.2.1. Гравець взаємодіє з об’єктами, які змінюють свій стан.

3.2.2. Об’єкти зберігають свої стани через SaveManager.

4. Оброблення помилки:

У разі винятку ExceptionsHandler:

4.1 Перехоплює повідомлення через *Application.logMessageReceived*.

4.2 Робить скріншот екрана.

4.3 Відображає скріншот на UI Image в сцені ExceptionHandler.

4.4 Потім запускає затемнення і викликає перезавантаження MainScene.

5. Перезавантаження:

5.1. SaveManager зберігає стани об’єктів перед вивантаженням сцени.

5.2. Після перезавантаження MainScene об’єкти відновлюють свої стани.

5.2.1. **Ключові елементи UI:**

6. Canvas для ExceptionHandler:

6.1. Містить Image, на який накладається скріншот.

6.2. Має затемнення, яке відображає прогрес перезавантаження.

7. Затемнення:

7.1. Створюється на Canvas у вигляді чорного напівпрозорого зображення.

7.2. Анімується через DOTween для плавного затемнення.

Роботу алгоритму представлено на рис. 2.

Для ознайомлення проєкт викладений у вільний доступ на платформі GihHub [10].

Для оцінювання ефективності імплементованого алгоритму адаптивного перезапуску в разі помилок було проведено експеримент на демоверсії мобільної гри для Android. В експерименті взяли участь 1418 унікальних користувачів. Середня щоденна аудиторія (Daily Active Users, DAU) становила приблизно 200 користувачів на день.

Основною метрикою ефективності вибрано One Day Retention (1-day retention) – показник, що вимірює частку користувачів, які повернулися до гри через 24 години після першої взаємодії. Чим вищий цей показник, тим краще гра утримує гравців, що може свідчити про зменшення фрустрації через можливі технічні проблеми та покращення користувацького досвіду.

Для коректного аналізу впливу алгоритму було проведено A/B тест:

1) Група A (контрольна) – стандартний механізм оброблення помилок без алгоритму адаптивного перезапуску;

2) Група B (експериментальна) – алгоритм адаптивного перезапуску активовано.

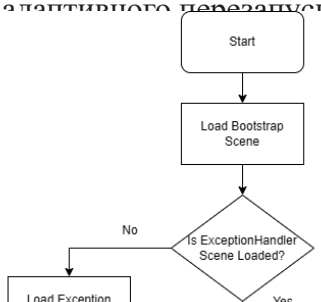

Рис. 2. Блок-схема роботи адаптивного перезапуску

Експеримент тривав 7 днів, і за цей період було зібрано дані щодо повернення користувачів та оброблення помилок. Для збору статистики використовувалися Unity Dashboard, Unity Crashlytics та Unity A/B Testing.

Для тестування механізму оброблення помилок у грі були навмисно введені тестові ексепшени під час дії купівлі предмета в магазині з ймовірністю 15%. Це дозволило проаналізувати, як алгоритм адаптивного перезапуску впливає на користувацький досвід у критичних сценаріях.

Таблиця 1

Порівняння 1-day retention між групами

Група	Кількість гравців	One Day Retention, %
A (контрольна)	703	35,4
B (експериментальна)	715	47,2

Таблиця 2

Кількість крашів та ексепшенів у кожній групі

Група	Загальна кількість крашів	Загальна кількість ексепшенів	Тестові ексепшени (під час купівлі)
A (контрольна)	15	102	50
B (експериментальна)	13	98	48

З отриманих даних видно, що кількість крашів та ексепшенів у двох групах залишалася приблизно однаковою, що свідчить про коректність умов тестування. Однак ключова метрика One Day Retention у групі з імплементованим алгоритмом адаптивного перезапуску значно покращилася, що підтверджує позитивний вплив цього рішення на користувацький досвід.

Висновки. Упровадження адаптивної системи значно покращує UX гравця, що підтверджено результатами експерименту. У результаті роботи було розроблено і протестовано алгоритм на базі рушія Unity, який дозволяє ефективно мінімізувати вплив технічних збоїв на користувацький досвід.

Даний підхід надає розробникам можливість збирати статистику, відстежувати помилки та уникати перевантаження користувача негативними сценаріями. Експеримент показав, що в разі збереження рівня технічних помилок основна метрика утримання користувачів (One Day Retention) значно покращується. Це дозволяє розробникам і компаніям отримувати більше часу на виправлення критичних помилок у продукті, тоді як більшість гравців будуть менш критично ставитися до проєкту, сприймаючи можливі збої у грі як проблеми зі своїм інтернет-з'єднанням або пристроєм.

Список використаних джерел

1. Udonis. Статистика мобільного геймінгу URL: https://www.blog.udonis.co/mobile-marketing/mobile-games/ mobile-gaming-statistics

2. The Gamer. Cyberpunk 2077 Ultimate Edition для PS5 та Xbox: історія багів та помилок від CDPR URL: https://www.thegamer.com/cyberpunk-2077-ultimate-edition-ps5-xbox-bugs-glitches-cdpr-history/

3. Guber G., Rukel D. UX and Serious Games–A Research Agenda Games and Learning Alliance (GaLA 2023). Lecture Notes in Computer Science (LNCS), vol. 14475. Cham : Springer, 2023. C. 214–222. https://doi.org/10.1007/978-3-031-49065-1_21

4. Weeraddana N.R., Habchi S., McIntosh S. Crash Report Prioritization for Large-Scale Scheduled Launches.

URL: https://rebels.cs.uwaterloo.ca/papers/icse2025seip_weeraddana.pdf

5. Kristiadi D. P. et al. The effect of UI, UX and GX on video games 2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom). IEEE, 2017. C. 158–163. https://doi.org/10.1109/CYBERNETICSCOM.2017.8311702

6. Yin P. Research on design and optimization of Game UI framework based on Unity3D // 2019 International Conference on Electronic Engineering and Informatics (EEI). – IEEE, 2019. – C. 221–223. https://doi.org/10.1109/EEI48997.2019.00055

7. Unity. Cloud Diagnostics URL: https://unity.com/products/cloud-diagnostics

8. Firebase. Документація Crashlytics. URL: https://firebase.google.com/docs/crashlytics

9. GitHub. Скрипт Economy.cs. URL: https://github.com/MrlCrosl/Economy/blob/main/Economy.cs

10. GitHub. Проєкт ScreenshotReload. URL: https://github.com/MrlCrosl/ScreenshotReload

Стаття надійшла до редакції 25.03.2025 р.

A. Klieptsov[1], V. Guseva-Bozhatkina[2]
[1]28Software
[2]Admiral Makarov National University of Shipbuilding

ALGORITHM FOR IMPROVING USER INTERACTION IN MOBILE GAMES THROUGH ADAPTIVE GAME RESTART

Summary

Abstract. The growing number of games on the market raises user expectations. Ten to twenty years ago, players tolerated flaws, bugs, or instability. Today, such issues often lead to uninstalls and poor app store reviews, as alternatives are easy to find. Although many bugs are addressed during development through collaboration with QA specialists, it's impossible to catch them all. Especially in large projects, millions of different interaction combinations can lead to instability, malfunction of core mechanics, or even app crashes. Additional tools known as Crashlytics were developed and integrated directly into games to address such issues. These tools enable developers to monitor errors in the game's code in real time, encountered by real users. This allows developers to address recurring problems and fix them promptly. At the same time, crashlytics is not a magic pill for success. The user experience (UX) of encountering errors in a game is generally negative. Users are often asked to send a screenshot, to describe their actions when the error occurred, and so on. In practice, very few users engage in such activities. This article proposes an alternative approach to user interaction with in-game errors, developed for the Unity engine. The proposed solution includes an algorithm for saving the game state, simulating a loading process – also referred to as a "connection loss" – and seamlessly returning the player to the point where the logical error occurred. This way, most players will not even realize that something went wrong in the game and may attribute the interruption to a poor internet connection or the need to load additional game content.

Keywords: Firebase Crashlytics, Cloud Diagnostics, error, UX, screenshot, Unity.