МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»_____,

освітньо-професійної програми «Інформаційні технології проектування»_____

на тему: _Мобільний додаток розширення функціональних можливостей гри "Erenwood" з використанням технологій доповненої реальності_____

Здобувача групи ІТ.м-23 ____Шкуратова Андрійя Олександровича_____ _____
(шифр групи) (прізвище, ім'я, по батькові)

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

_____ __Андрій ШКУРАТОВ__
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник _____к.т.н., доц. Наталія ФЕДОТОВА_____ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«_____» _____ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Шкуратов Андрій Олександрович

(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи Мобільний додаток розширення функціональних можливостей гри "Erenwood" з використанням технологій доповненої реальності затверджена наказом по університету від «08» листопада 2023 р. № 1249-VI

2 Термін здачі студентом кваліфікаційної роботи «___» ___грудня___ 2023 р.

3 Вхідні дані до кваліфікаційної роботи Технічне завдання на розробку iOS додатку компаньону до настільної гри "ErenWood"

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) Аналіз аналогічних додатків, розробка технічних вимог до додатку та засобів реалізації, розробка інтерфейсу додатку, розробка алгоритмів генерації внутрішньоігрового контенту, розробка моделей та функціоналу доповненої реальності

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) Постановка задачі, аналіз аналогічних додатків, IDEF0 діаграма, перший рівень декомпозиції, діаграма варіантів використання, діаграма послідовностей, блок схема роботи додатку

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Огляд проблеми	19.09.2023 - 25.09.2023	
2	Порівняння аналогів	22.09.2023 - 24.09.2023	
3	Постановка мети та задачі	25.09.2023 - 26.09.2023	
4	Вибір технологій реалізації	26.09.2023 - 27.09.2023	
5	Структурно-функціональне можелювання	28.09.2023 - 02.10.2023	
6	Моделювання варіантів використання	02.10.2023 - 06.10.2023	
7	Створення алгоритму генерування	07.10.2023 - 21.10.2023	
8	Розробка фінального додатку	22.10.2023 - 25.11.2023	
9	Оформлення звіту	26.11.2023 - 05.12.2023	

Магістрант _____ Андрій ШКУРАТОВ

Керівник роботи _____ к.т.н., доц. Наталія ФЕДОТОВА

АНОТАЦІЯ

Тема кваліфікаційної роботи магістра «Мобільний додаток розширення функціональних можливостей гри "Erenwood" з використанням технологій доповненої реальності».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 40 найменувань, додатків. Загальний обсяг роботи − 80 сторінок, у тому числі 57 сторінок основного тексту, 4 сторінки списку використаних джерел, 21 сторінок додатків.

Актуальність роботи полягає у розширенні функціоналу гри, збільшення реграбельності та різноманітності ігрового процесу, а також додає можливість візуалізації ігрового процесу за рахунок технології доповненої реальності. Використання компаньйона спрощує рандомізацію ігрового процесу беручи на себе функцію видачі випадкових подій за винятком повторення паттернів, які неможливі в ігровому процесі.

Мета роботи: розробка мобільного додатку супутнику до настільної гри “ErenWood” для платформи iOS, з використанням мови програмування Swift, фреймворку SwiftUI та технологій доповненої реальності.

Даний додаток рекомендується використовувати в парі з настільною грою “ErenWood”.

Ключові слова: SwiftUI, Swift, iOS, настільна гра, додаток компаньон, “ErenWood”, Доповнена реальність, Augumented Reality.

ЗМІСТ

ВСТУП

У царині настільних ігор, де уява поєднується зі стратегією та товариським духом, виник яскравий ландшафт, що розвивається. Привабливість настільних ігор полягає не лише в їхніх складних правилах і захоплюючих розповідях, але й у соціальних зв'язках, які вони сприяють. Оскільки технології розвиваються та проникають у всі аспекти нашого життя, перетин традиційних настільних ігор і цифрових інновацій став благодатним ґрунтом для досліджень. Ця випускна робота розпочинає подорож у сферу розробки ігор, особливо зосереджуючись на створенні системи-супутника, призначеної для збагачення досвіду настільних ігор.

Поняття компаньйона в цьому контексті виходить за межі традиційного розуміння терміну. Замість фізичної сутності компаньйон, про який йде мова, проявляється як цифрова сутність, складно вплетена в тканину настільної гри. Цей цифровий компаньйон задуманий як інструмент, який розширює ігровий процес, сприяючи глибшому зануренню, стратегічній взаємодії та покращенню соціальної динаміки серед гравців. Оскільки межі між фізичною та цифровою сферами стираються, перспектива синергії цих елементів обіцяє змінити ландшафт настільних ігор.

У цій дипломній роботі розглядаються багатогранні виміри розробки компаньйонів, досліджуються технологічні основи, міркування щодо взаємодії з користувачем і вплив на загальну ігрову екосистему. Здійснюючи цю роботу, ми прагнемо не лише впроваджувати інновації в ігровому просторі, але й робити внесок у ширший дискурс, що оточує гармонійну інтеграцію аналогових і цифрових елементів у дозвіллі та розвагах.

У цьому дослідженні увага буде приділена викликам і можливостям, пов'язаним із розробкою компаньйона для настільної гри. Від тонкощів дизайну користувальницького інтерфейсу до потенційного впливу на динаміку гравця, ця робота спрямована на аналіз і розуміння різних аспектів, які формують роль компаньйона в настільному ігровому ландшафті. Проводячи цю

міждисциплінарну роботу, ми прагнемо внести свій внесок у знання, які не лише сприятимуть розробці ігор, але й резонуватимуть із змінними очікуваннями та бажаннями ентузіастів настільних ігор у все більш взаємопов’язаному світі.

Актуальність розробка програми-супутника для настільних ігор має значну актуальність у сучасному ігровому середовищі. Оскільки настільні ігри все більше інтегрують цифрові елементи, ця робота спрямована на зростаючий попит на інноваційні інструменти, які покращують ігровий досвід. Потенціал програми для оптимізації ігрового процесу та сприяння співпраці не лише узгоджується з уподобаннями сучасних гравців, що розвиваються, але й сприяє доступності настільних ігор. Поєднуючи традиції з технологією, ця робота стає стрижневим мостом, пропонуючи рішення, яке відповідає вимогам як досвідчених гравців, які прагнуть ефективності, так і новачків, які шукають захоплюючу точку входу у світ настільних ігор. По суті, актуальність цієї роботи полягає в її здатності модернізувати та збагатити досвід настільних ігор, забезпечуючи її продовження привабливості та доступності в епоху цифрових технологій.

Об’єктом дослідження Взаємодія та інтеграція між настільною грою "ErenWood" та додатком для iOS, що використовує технології доповненої реальності.

Предметом дослідження Інформаційна система, що представляє собою додаток до настільної гри "ErenWood", розроблений для платформи iOS та використовує технології доповненої реальності.

Мета дипломної роботи – Розробка інформаційної системи яка є доповненням до настільної гри "ErenWood" з використанням технологій доповненої реальності для платформи iOS

Методи дослідження:

– теоретичні та емпіричні;

– створення структурно-функціональної моделі додатку.

– розробка iOS додатку.

Практична цінність інформаційної системи як доповнення до настільної гри "ErenWood" з використанням технологій доповненої реальності для платформи iOS полягає в покращенні геймплею, залученні гравців, технологічних інноваціях, створенні нових можливостей гри та сприянні розвитку галузі AR-ігор, підвищенні конкурентоспроможності та загальному розширенні використання технологій доповненої реальності у світі настільних ігор.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд проблеми та актуальність розробки

Дана проблема пов’язана з необхідністю поліпшити досвід настільних ігор в епоху цифрових технологій. Традиційні настільні ігри зберігають свою популярність, отже виникає попит на інноваційні рішення, які плавно інтегрують сучасні технології без шкоди для ігрового досвіду. Основним завданням є оптимізація ігрового процесу, покращення оповідання і розвиток співпраці між гравцями та мастером гри. Актуальність розробки компаньону стає очевидною, оскільки вона має на меті забезпечити вирішення цих проблем. Використовуючи цифрову платформу, метою додатку є модернізаця ігровиого досвіду, зробивши його більш зручним для співпраці та адаптованим до різноманітних уподобань гравців. Основна мета полягає в тому, щоб подолати розрив між традиціями та інноваціями, гарантуючи, що настільні ігри залишатимуться не лише актуальними, але й дедалі популярнішими в сучасному ігровому середовищі.

Розвиток настільних ігор створює кілька проблем і можливостей, які розробка супутнього додатка намагається вирішити. Однією з основних проблем є потреба в підвищенні ефективності ігрового процесу. Традиційні методи ручки та паперу, хоч і культові, іноді можуть призвести до адміністративного тягаря, який уповільнює темп гри. Ця програма має на меті оптимізувати ці процеси, полегшуючи гравцям і майстрам гри керування персонажами, правилами та різними аспектами ігрового процесу.

Ще одна проблема – баланс між традицією та модернізацією. Хоча багато гравців цінують тактильні та соціальні аспекти традиційних настільних ігор, зростає інтерес до використання цифрових інструментів, які можуть збагатити ігровий досвід. Додаток-супутник прагне досягти цього балансу, забезпечуючи повну інтеграцію технологій, не затьмарюючи основні елементи, які роблять настільні ігри унікальними.

Крім того, оповідання є основою настільних ігор, а введення контрольованої випадковості в оповіді є делікатним, але важливим аспектом. Додаток вирішує цю проблему, пропонуючи такі функції, як генератори випадкових подій, що сприяє непередбачуваності та захоплюючості процесу оповідання, зберігаючи зв’язність оповіді.

Співпраця між гравцями та Game Master є ще одним ключовим моментом. Традиційні настільні ігри часто вимагають широкого спілкування та координації, які додаток прагне покращити за допомогою спільних даних кампанії, інструментів спілкування та спільного використання ресурсів. Цей аспект співпраці не тільки спрощує логістику, але й зміцнює соціальні зв’язки в ігровій спільноті.

Крім того, доступність та інклюзивність є ключовими проблемами. Додаток прагне зробити настільні ігри доступнішими для ширшої аудиторії, враховуючи різноманітні вподобання гравців, здібності та стилі гри. Забезпечуючи настроювані інтерфейси, віртуальні кидки кубиків та інші повноцінні функції, додаток гарантує, що кожен зможе насолоджуватися грою.

По суті, розробка додатка-супутника для настільних ігор є багатогранною відповіддю на виклики та можливості, які відкриває перетин традицій і технологій в ігровому середовищі. Він спрямований на покращення ігрового досвіду, звертаючись до ефективності, динаміки оповідання, співпраці та доступності, сприяючи подальшій актуальності та жвавості настільних ігор у цифрову епоху.

1.2 Огляд останніх досліджень і публікацій

iOS — це операційна система, розроблена Apple Inc. для своїх мобільних пристроїв, зокрема iPhone, iPad та iPod Touch. Відома своїм елегантним та інтуїтивно зрозумілим інтерфейсом користувача, iOS відома своїм фокусом на простоті, безпеці та бездоганній інтеграції з пристроями Apple. App Store,

централізований ринок програм, пропонує користувачам широкий спектр програм для різних цілей, від продуктивності до розваг.[1]

Розробка додатків для iOS використовує набір методологій для створення ефективних, зручних додатків для мобільних пристроїв Apple. Центральним у цьому процесі є використання мови програмування Swift в інтегрованому середовищі розробки (IDE) Xcode. Розробники зазвичай дотримуються архітектури Model-View-Controller (MVC) [2], яка поділяє логіку програми на окремі компоненти для покращеної організації. UIKit надає необхідні інструменти для створення графічного інтерфейсу користувача, а Auto Layout забезпечує адаптивність до різних розмірів екрана [10]. Core Data керує рівнем моделі та полегшує маніпулювання даними, а розширення програм забезпечують додаткові функції [8]. Додаткова інтеграція SwiftUI пропонує сучасний підхід до розробки інтерфейсу користувача. Розробка, керована тестуванням (TDD) із XCTest забезпечує надійність коду, а інструменти безперервної інтеграції [9] (CI) автоматизують процеси тестування та збірки, сприяючи спільній та оптимізованій робочій процедурі розробки. Анімація в контексті розробки додатків є ключовим елементом для покращення взаємодії з користувачем і оживлення інтерфейсів. Вони передбачають динамічний рух і перехід візуальних елементів, сприяючи більш привабливій та інтуїтивно зрозумілій взаємодії [3]. Ці методології разом сприяють створенню надійних, візуально привабливих і адаптивних програм для iOS.

Доповнена реальність (AR) — це трансформаційна технологія, яка накладає цифрову інформацію та віртуальні об’єкти на реальний світ, створюючи захоплюючий та інтерактивний досвід користувача [11, 12]. На відміну від віртуальної реальності (VR), AR покращує фізичне середовище, а не повністю його замінює. Програми AR використовують такі пристрої, як смартфони, планшети чи окуляри AR, щоб бездоганно поєднувати віртуальний і реальний світи. Ця технологія знаходить застосування в різних галузях, від ігор і освіти до охорони здоров’я та роздрібної торгівлі [13, 14]. У сфері мобільних додатків AR представляє такі інноваційні функції, як інтерактивні карти, розпізнавання об’єктів у реальному часі та захоплююче оповідання. Розробники

використовують такі фреймворки, як ARKit для iOS і ARCore для Android, щоб інтегрувати можливості AR у свої програми. Оскільки доповнена реальність продовжує розвиватися, вона має величезний потенціал для революції в тому, як ми сприймаємо навколишній світ і взаємодіємо з ним, відкриваючи нові можливості для розваг, освіти та практичного вирішення проблем [15].

Інтеграція доповненої реальності (AR) у настільні ігри знаменує собою революційне поєднання традиційних настільних ігор із передовими технологіями, пропонуючи гравцям покращений інтерактивний ігровий досвід. AR покращує настільні ігри, накладаючи цифрові елементи на фізичні компоненти гри, стираючи межі між віртуальним і реальним світами. Ця інтеграція представляє динамічні візуальні елементи, інтерактивну анімацію та захоплюючу історію, яка реагує на дії та рішення гравців. Наприклад, AR може оживляти ігрових персонажів на дошці, розкривати приховані елементи або вводити непередбачувані події, викликані певними ігровими ситуаціями. Використовуючи такі пристрої, як смартфони чи окуляри AR, гравці можуть спостерігати, як ігрові дошки перетворюються на яскраве тривимірне середовище. Ця інновація не лише додає до настільних ігор шар азарту та непередбачуваності, але й відкриває шляхи для нових наративів, стратегічних можливостей та спільного ігрового процесу. Оскільки технологія AR продовжує розвиватися, її інтеграція в настільні ігри обіцяє переосмислити те, як гравці залучаються до цих класичних форм розваг, пропонуючи міст між традиційними та сучасними іграми [4-7].

ARKit, фреймворк доповненої реальності (AR) від Apple, революціонізував спосіб, у який розробники передають захоплюючий досвід AR на пристрої iOS. Використовуючи розширені функції, такі як одночасна локалізація та відображення (SLAM), ARKit дозволяє точно відстежувати рух пристрою та його оточення, дозволяючи віртуальним об’єктам плавно взаємодіяти з реальним світом. Можливості розуміння сцени дозволяють ARKit розпізнавати поверхні, дозволяючи розміщувати віртуальні об’єкти на столах, підлозі чи інших фізичних структурах. Оцінка освітленості покращує реалістичність,

регулюючи освітлення віртуального об’єкта на основі навколишнього середовища [16, 17].

Розробники використали ARKit для створення різноманітних і привабливих програм. В іграх такі ігри, як Pokémon GO, використовують ARKit для більш реалістичного та інтегрованого ігрового процесу, що дозволяє віртуальним істотам з’являтися у фізичному оточенні гравця. Освітні програми, такі як інтерактивні анатомічні моделі чи астрономічні посібники, використовують ARKit, щоб надати користувачам захоплюючий досвід навчання. Додатки для роздрібної торгівлі використовують ARKit, щоб клієнти могли візуалізувати меблі чи продукти вдома перед покупкою [19, 20].

Крім того, постійні оновлення ARKit запровадили такі функції, як ARKit 2.0, який підтримує спільний досвід AR, що дозволяє кільком користувачам взаємодіяти з одним середовищем AR. Це відкриває двері для спільних ігор, дизайнерських проектів і соціальних взаємодій у сфері AR [18].

По суті, надійні можливості ARKit і постійний прогрес сприяли розвитку різноманітної екосистеми доповнених програм, збагачуючи різні галузі та пропонуючи користувачам інноваційні способи взаємодії з цифровим контентом у реальному світі.

Ігрові компаньйони — це цифрові інструменти, призначені для розширення та покращення загального досвіду гри, будь то настільні чи відеоігри. Ці компаньйони можуть приймати різні форми, починаючи від мобільних додатків і програмного забезпечення до інтерактивних веб-сайтів. У контексті настільних ігор ігровий компаньйон може пропонувати такі функції, як віртуальні аркуші персонажів, бази даних пошуку правил та інструменти керування кампаніями, спрощуючи адміністративні завдання та дозволяючи гравцям і майстрам гри більше зосереджуватися на розповіді та ігровому процесі [21, 22]. Ці інструменти часто надають гравцям централізовану платформу для організації своїх ресурсів, відстеження прогресу в грі та співпраці в спільних кампаніях [23].

Для відеоігор компаньйони можуть служити зовнішніми програмами або інтерфейсами, які доповнюють ігровий досвід. Вони можуть пропонувати карти

в реальному часі, відстеження місій або керування інвентарем, надаючи гравцям цінну інформацію, не перериваючи головний екран гри. Ігрові компаньйони також можуть розширити ігровий досвід за межі основного пристрою, дозволяючи гравцям взаємодіяти з ігровим світом, навіть коли вони знаходяться далеко від своїх консолей або ПК. Інтеграція ігрових компаньйонів відображає тенденцію до більш взаємопов'язаної та захоплюючої ігрової екосистеми, де цифрові інструменти покращують як практичні, так і розважальні аспекти ігор [24, 25]. З розвитком технологій роль компаньйонів у іграх, ймовірно, буде розвиватися, пропонуючи гравцям нові та інноваційні способи залучення до улюблених ігор.

1.3 Аналіз аналогічних додатків

Для ліпшого розуміння функціоналу, що потенційно можна додати до додатку необхідно проаналізувати декілька схожих додатків, найліпше підходять наступні: D&D 5th Edition [41] і The Lord of the Rings: Journeys in Middle-Earth [31]

Dungeons & Dragons (D&D) 5th Edition Companion App:

1. Допомога в правилах і керування персонажами: Додаток-супутник для D&D 5th Edition є комплексним інструментом для гравців і Dungeon Masters. Він допомагає шукати правила, описувати заклинання та керувати таблицею персонажів, забезпечуючи оновлення в реальному часі та автоматизацію різноманітних ігрових механік.

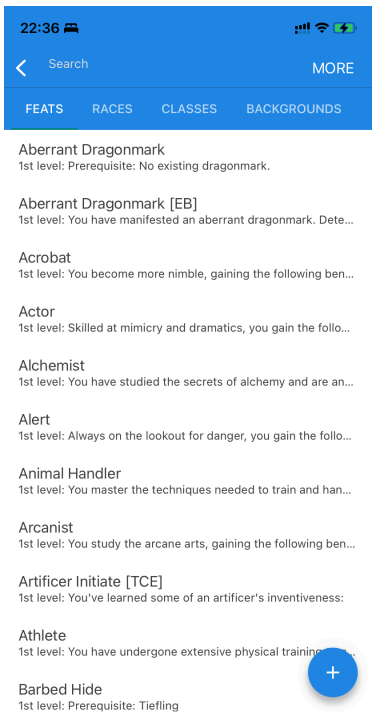

Рисунок 1.1 - Перегляд правил та відомостей ігрових елементів

Джерело: Зроблено автором

2. Динамічні таблиці символів та інтеграція: Гравці можуть безперешкодно керувати своїми персонажами за допомогою програми, відстежуючи хіти, слоти для заклинань та інвентар. Інтеграція з цифровими аркушами символів спрощує ігровий процес, зменшуючи потребу в ручних розрахунках і паперовій роботі.

3. Інтерактивний віртуальний кидок кубиків:

Включення віртуального кидка кубиків покращує ігровий процес, надаючи гравцям швидкий і доступний інструмент для кидання кубиків, не перериваючи хід гри. Він підтримує різні типи кубиків і забезпечує справедливе й точне вирішення дій у грі.

4. Керування кампанією:

Dungeon Masters користуються перевагами функцій керування кампанією, що дозволяє їм упорядковувати нотатки, відстежувати інформацію NPC і динамічно коригувати зустрічі на основі вибору гравця. Додаток підтримує змінний характер кампаній, сприяючи індивідуальному та захоплюючому досвіду оповідання.

Рисунок 1.2 - Вікно підключення та керування компанією

Джерело: Зроблено автором

5. Підключення плеєра:

Додаток полегшує підключення гравців, дозволяючи ділитися даними кампанії, забезпечуючи спільну розповідь і координацію. Він покращує соціальний аспект настільних ігор, надаючи платформу для спілкування та обміну ресурсами між гравцями.

6. Підтримка розширення та інтеграції:

Додаток-супутник легко вміщує розширення та додатковий вміст, гарантуючи, що ігровий досвід може розвиватися з новими наборами правил або додатковими матеріалами. Ця гнучкість сприяє довговічності та адаптивності системи D&D 5th Edition.

Додаток-супутник «The Lord of the Rings: Journeys in Middle-Earth»:

1. Наративи, керовані програмою:

Додаток-супутник для подорожей у Середзем’я використовує підхід, орієнтований на розповідь. Він виконує роль майстра гри, генеруючи динамічні сюжетні лінії, квести та несподівані події на основі вибору гравця. Ця розповідь, керована додатком, покращує відтворення та створює персоналізований ігровий досвід.

2. Дослідження карти в реальному часі:

Важливою особливістю програми є дослідження карти в реальному часі. У міру того, як гравці рухаються віртуальним зображенням Середзем’я, додаток відкриває нові локації, ініціює зустрічі та динамічно коригує історію, що розгортається. Ця інтеграція технології додає візуальний і захоплюючий рівень традиційного настільного дослідження.

Рисунок 1.3 - Перегляд ігрового поля в додатку

Джерело: Додаток The Lord of the Rings: Journeys in Middle-Earth [31]

3. Події та зустрічі в грі:

Додаток представляє внутрішньоігрові події та зустрічі, включаючи навколишні звуки, тематичну музику та інтерактивні елементи. Ця аудіовізуальна інтеграція покращує атмосферу гри, занурюючи гравців у багату історію Середзем’я.

4. Хід кампанії та можливості для розблокування:

Додаток підтримує хід кампанії, записуючи вибір гравців і результати. Він також представляє контент, який можна розблокувати на основі успіху кампанії, створюючи відчуття безперервності та винагороди. Ця функція заохочує довгострокову взаємодію та інвестиції в загальний наратив.

Рисунок 1.4 - Відображення фізичної карти в додатку

Джерело: Додаток The Lord of the Rings: Journeys in Middle-Earth [31]

5. Адаптивна складність і масштабування:

Додаток динамічно регулює складність зіткнень залежно від продуктивності гравця, забезпечуючи складний, але збалансований досвід. Ця адаптивна складність сприяє задоволенню гравців і враховує різні рівні навичок у ігровій групі.

Загалом, супутні додатки D&D 5th Edition і The Lord of the Rings: Journeys in Middle-Earth демонструють потенціал цифрової інтеграції в настільні ігри. У той час як програма D&D зосереджена на допомозі з правилами, управлінні персонажами та організації кампанії, програма The Lord of the Rings наголошує на наративному грі, дослідженні в реальному часі та захоплюючій оповіді. Кожна програма задовольняє конкретні потреби та філософію дизайну

відповідної ігрової системи, демонструючи універсальність супутніх програм для покращення настільних ігор.

Таблиця 1.1 - порівняльний аналіз додатків компаньонів

	D&D 5th Edition	**LOTR: Journeys in Middle-Earth**
Доступність	+	-
Дизайн	+	+
Покращення оповідання	-	+
Кросплатформна підтримка	+	+
Автономний режим	+	-
Інтеграція з фізичними компонентами	-	+
Пошук правил	+	+

Джерело: побудовано автором

Виходячи з аналізу приходимо до висновку, шо основний функціонал додатку повинен містити комплексну інтеграцію в ігровий процес, та мати справочну інформацію, що допоможе гравцю ліпше зрозуміти ігровий процес гри.

2. ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Значення розробки компаньйона для настільних ігор виходить за межі простої конвергенції аналогових і цифрових сфер; це говорить про еволюцію ігрового досвіду в епоху розвитку технологій. Ця робота є важливою з кількох причин:

- Покращене залучення гравців:

Представлення добре продуманого компаньйона для настільних ігор має потенціал для підвищення залучення гравців. Завдяки плавній інтеграції цифрових елементів компаньйон може надавати динамічні оповіді, зворотній зв’язок у реальному часі та інтерактивні компоненти, які захоплюють гравців і занурюють їх у світ ігор.

- Адаптація до мінливих вимог:

Ландшафт ігрових уподобань постійно змінюється. Визнаючи попит на більш динамічні, вдосконалені технології, розробка компаньйона для настільних ігор відповідає мінливим очікуванням сучасних геймерів, які шукають гармонійне поєднання традиційних і цифрових елементів у своєму дозвіллі.

- Розширення соціальних зв’язків:

Настільні ігри за своєю суттю є соціальними, часто покладаються на особисту взаємодію. Представлення компаньйона, призначеного для покращення, а не заміни цих взаємодій, може сприяти новим рівням соціального зв’язку. Він діє як фасилітатор, зміцнюючи зв’язки між гравцями та створюючи спільний досвід, який виходить за межі фізичного настільного комп’ютера.

- Інноваційна динаміка ігрового процесу:

Супутня система представляє інноваційну динаміку ігрового процесу, кидаючи виклик традиційним уявленням про механіку настільних ігор.

Використовуючи цифрові елементи, компаньйон може представляти динамічні події, приховані квести та сюжетні лінії, що розвиваються, перетворюючи статичні настільні ігри на живі, дихальні враження, які розвиваються з кожним проходженням.

- Освітні можливості:

Окрім розваг, розробка компаньйона для настільної гри надає освітні можливості. Включаючи такі елементи, як стратегічні посібники, історичний контекст або інтерактивні навчальні модулі, компаньйон може служити цінним інструментом як для звичайних, так і для серйозних геймерів, розширюючи потенційну аудиторію та збагачуючи загальний досвід гри.

- Розвиток ринку та конкурентоспроможність:

У міру того, як ігрова індустрія продовжує розвиватися, інновації стають ключовим фактором конкурентоспроможності. Розробка компаньйона для настільної гри ставить дизайнерів і розробників ігор в авангард цієї еволюції, пропонуючи унікальний продукт, який задовольняє бажання різноманітної та більш розширеної ігрової спільноти.

- Дослідження та розробки:

Ця робота сприяє поточним дослідженням і розробкам у ширшій сфері взаємодії людини з комп’ютером та дизайну ігор. Статті, отримані в процесі розробки, відгуки користувачів і вплив на досвід гравців, можуть стати основою для майбутніх починань, формуючи траєкторію настільних і цифрових ігор.

По суті, важливість цієї роботи полягає в її потенціалі переосмислити та збагатити досвід настільних ігор, адаптуватись до сучасних очікувань і зробити внесок у ширший дискурс щодо злиття традиційних і цифрових форм розваг. Заглиблюючись у це міждисциплінарне дослідження, ми досліджуємо незвідані території ігрового дизайну, технологічної інтеграції та взаємодії з користувачем, щоб прокласти шлях до нової ери настільних ігор.

Мета дипломної роботи – полягає в створенні інформаційної системи доповнення до настільної гри "ErenWood" з використанням технологій доповненої реальності для платформи iOS. А саме, надати розроблену програму-супутник, яка доповнює сильні сторони настільної гри, одночасно

використовуючи переваги цифрових інструментів, зрештою збагачуючи загальний ігровий досвід.

Задачі розглянуті в вирішенні цієї мети наступні:

- Аналіз існуючих додатків, публікацій та технології реалізації.
- Провести ретельне планування, та визначити мету та завдання проекту, створити план розробки та провести оцінку ризиків.
- Спроектувати моделі додатку
- Моделі персонажів
- Моделі ігрових предметів
- Розробка інтерфейсу додатку
- Розробка логіки додатку
- Розробка бази даних додатку

2.2 Техгнології та інструменти реалізації

UIKit і SwiftUI — це фреймворки, розроблені Apple для побудови користувальницьких інтерфейсів у додатках iOS, але вони суттєво відрізняються за підходом до дизайну та складністю. UIKit, старший з двох, дотримується імперативної парадигми та покладається на більш докладний синтаксис за допомогою Objective-C або Swift. Розробники, які використовують UIKit, повинні чітко визначити, як інтерфейс має змінюватися у відповідь на дані або взаємодію користувача. Незважаючи на те, що UIKit є потужним і гнучким, він часто передбачає написання додаткового коду для досягнення певних макетів інтерфейсу користувача та поведінки [29].

З іншого боку, SwiftUI, представлений нещодавно, використовує декларативний підхід. Це дозволяє розробникам описати бажаний інтерфейс і поведінку, а фреймворк автоматично визначає, як цього досягти. SwiftUI спрощує кодову базу, роблячи її більш стислою та читабельною. Він також забезпечує попередній перегляд у режимі реального часу під час розробки,

дозволяючи розробникам миттєво бачити внесені ними зміни. SwiftUI вважається більш зручним для початківців і заохочує швидший цикл розробки, хоча він може ще не охоплювати всі можливості UIKit, особливо для складних і дуже налаштованих інтерфейсів. Підсумовуючи, хоча UIKit пропонує глибокі налаштування та гнучкість, SwiftUI спрощує процес розробки завдяки своєму декларативному синтаксису та попередньому перегляду в реальному часі, що робить його привабливим вибором для розробки сучасних додатків для iOS.

Таблиця 2.1 - порівняльний аналіз фреймворків UIKit та SwiftUI

	UIKit	**SwiftUI**
Парадигма програмування	Імперативна	Декларативна
Синтаксис	Більш багатослівний, часто вимагає додаткового коду для макетів	Лаконічний і читабельний код, що зменшує шаблонний код
Опис інтерфейсу користувача	Програмний	Декларативний
Перегляд в реальному часі	Обмежена підтримка попереднього перегляду в реальному часі	Великі попередні перегляди в реальному часі, що дозволяють візуалізувати в реальному часі
Гнучкість	Дуже гнучкий, підходить для складного інтерфейсу користувача та налаштування	Висока гнучкість, хоча може мати обмеження для дуже специфічних конструкцій
Анімації	Широкий контроль над анімаціями на основі CoreAnimation	Вбудовані та прості у використанні функції анімації

Джерело: побудовано автором

Виходячи з усього вищезазначеного SwiftUI краще підходить для розробок iOS додатку. Розробка програми для iOS з використанням SwiftUI передбачає низку кроків і методологій, які відповідають декларативному характеру фреймворку:

Ініціалізація проекту:

Створення нового проекту SwiftUI в Xcode. Обрання відповідного шаблону проекту на основі структури та мети програми, як-от окреме подання або програма з основними деталями.

Розуміння представлень і модифікаторів:

SwiftUI обертається навколо переглядів, основних компонентів інтерфейсу користувача. Перегляди можна налаштувати за допомогою модифікаторів, щоб змінити їх вигляд і поведінку. Ознайомлення з різними режимами перегляду, такими як текст, зображення та кнопка.

Декларативний синтаксис:

SwiftUI підтримує декларативний підхід, коли розробник визначає, як має виглядати користувальницький інтерфейс, без явних деталей, як цього досягти. Цей підхід спрощує розробку інтерфейсу користувача, дозволяючи SwiftUI обробляти базові складності.

Управління станами:

Використання оболонки властивості @State для керування станом у представленнях. Це дозволяє інтерфейсу користувача автоматично реагувати на зміни в базових даних, створюючи реактивний і динамічний досвід користувача.

Навігація та презентація:

Реалізація навігації між представленнями за допомогою таких конструкцій, як NavigationView та NavigationLink. Модальне представлення нових видів може бути досягнуто за допомогою таких механізмів, як аркуш або повний екран.

Списки та форми:

Використання List для відображення списків елементів, які можна прокручувати, і Form для створення структурованих форм із згрупованими розділами. SwiftUI надає простий синтаксис для визначення динамічних списків і форм.

Комбінована інтеграція фреймворку:

Включення фреймворку Combine для обробки асинхронних та керованих подій коду. Combine представляє концепції реактивного програмування, що дозволяє ефективно керувати асинхронними операціями.

Розпізнавання жестів:

Застосування засобів розпізнавання жестів для взаємодії користувачів за допомогою вбудованих модифікаторів SwiftUI, таких як onTapGesture, onLongPressGesture тощо. Ці жести покращують інтерактивність інтерфейсу користувача.

Доступність:

Надання пріоритету доступності, використовуючи модифікатори доступності SwiftUI для покращення взаємодії з людьми з обмеженими можливостями. Для покращення доступності можна вказати такі елементи, як мітки, підказки та характеристики.

Тестування та налагодження:

Регулярне тестування та налагодження програми SwiftUI. SwiftUI пропонує такі інструменти, як попередній перегляд полотна для попереднього перегляду в реальному часі, інтерактивні функції налагодження та тестові рамки для забезпечення стабільності та правильності програми.

Для розробки функціоналу доповненої реальності в iOS використовується фреймворк ARKit.

ARKit — це потужний фреймворк доповненої реальності (AR), розроблений Apple для пристроїв iOS, призначений для бездоганної інтеграції доповненої реальності в програми. Представлений разом з iOS 11, ARKit з тих пір розвивався через кілька версій, кожна з яких принесла вдосконалення та нові функції. Цей фреймворк використовує вдосконалене апаратне забезпечення та датчики, наявні в iPhone та iPad, щоб забезпечити точну та захоплюючу взаємодію AR [30].

Ключові особливості:

1. Відстеження руху: ARKit дозволяє пристроям точно відстежувати своє положення та орієнтацію в режимі реального часу, дозволяючи віртуальним об’єктам бути закріпленими та безперебійно взаємодіяти з реальним світом.

2. Розуміння сцени: Фреймворк може розпізнавати поверхні та об’єкти в навколишньому середовищі, дозволяючи розробникам розміщувати віртуальний вміст на поверхнях реального світу.

3. Оцінка освітленості: ARKit враховує умови навколишнього освітлення, коригуючи зовнішній вигляд віртуальних об’єктів відповідно до освітлення в реальному світі, підвищуючи загальну реалістичність доповненої реальності.

4. Відстеження та розпізнавання облич: ARKit містить інструменти для відстеження та розпізнавання облич, що дозволяє програмам накладати на обличчя користувача вміст доповненої реальності, наприклад маски чи анімованих персонажів.

5. World Mapping: ARKit підтримує створення постійної карти фізичного середовища, полегшуючи спільний досвід AR, де кілька користувачів можуть взаємодіяти з тим самим віртуальним вмістом у спільному просторі.

6. Виявлення горизонтальної та вертикальної площини: ARKit може виявляти горизонтальні поверхні, як-от підлоги та столи, а також вертикальні поверхні, як-от стіни, що дозволяє розробникам точніше розміщувати віртуальні об’єкти в середовищі.

7. Розпізнавання зображень: Фреймворк дозволяє розпізнавати 2D-зображення, запускаючи певний досвід доповненої реальності, коли камера пристрою визначає попередньо визначене зображення.

Розробка з ARKit:

Розробники використовують ARKit з мовою програмування Swift і Xcode IDE. ARKit спрощує реалізацію функцій доповненої реальності, надаючи абстракції високого рівня для типових завдань, таких як рендеринг сцени та введення користувачами. Він підтримує різноманітні пристрої, від iPhone до iPad, забезпечуючи широку базу користувачів для програм AR.

Програми:

ARKit використовується в різних галузях, включаючи ігри, освіту, роздрібну торгівлю та охорону здоров’я. Від інтерактивних ігор до освітніх програм, які оживляють предмети, ARKit дає змогу розробникам створювати інноваційний та захоплюючий AR-контент для користувачів iOS.

Підсумовуючи, ARKit зіграв ключову роль у розвитку доповненої реальності на пристроях iOS. Його безперервна еволюція в поєднанні з надійним набором функцій позиціонує ARKit як провідну структуру для розробників, які прагнуть бездоганно включити доповнену реальність у свої програми для iOS.

3. МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДОПОВНЕННЯ

3.1 Структурно-функціональне моделювання процесу

Для структурно-функціонального проєктування використовується Structured Analysis and Design Technique або SADT.

Техніка структурованого аналізу та проектування (SADT) — це систематична методологія, розроблена для аналізу та проектування складних систем. В основі SADT використовується графічне моделювання для представлення процесів, функцій і потоків даних в організації чи системі. Це графічне позначення, яке часто зображується за допомогою рамок, стрілок і символів, дає змогу візуально та ієрархічно розбити систему, підкреслюючи чіткий і структурований підхід. Методологія передбачає декомпозицію процесу, коли загальна система систематично розбивається на більш дрібні, більш керовані компоненти. Діаграми потоку даних (DFD) відіграють ключову роль в ілюструванні потоку даних між різними процесами, надаючи комплексне уявлення про рух інформації всередині організації. SADT також використовує дерева функцій для представлення ієрархії функцій у системі, допомагаючи зацікавленим сторонам зрозуміти взаємозв’язки між різними функціями та операціями.

Однією з ключових сильних сторін SADT є його здатність підвищувати ясність і комунікацію між зацікавленими сторонами. Графічне представлення структури та процесів системи робить її доступною для людей із різними технічними знаннями, сприяючи ефективній комунікації між аналітиками, дизайнерами та іншими зацікавленими сторонами. Структурований підхід методології сприяє логічному розподілу системи, полегшуючи виявлення потенційних проблем, розуміння функціональності системи та всебічне документування процесів. Хоча SADT знайшов значне застосування в 1980-х і 1990-х роках, його принципи продовжують впливати на сучасну системну

інженерію та методології проектування, служачи основою для розуміння та моделювання складних систем.

Розробимо IDEF0 діаграму для кращого розуміння функціональних процесів додатку рисунок 3.1.

Рисунок 3.1 - Контекстна діаграма компаньону до гри “ErenWood”

Джерело: побудовано автором

Стрілки, «Ігровий контент» та «правила гри», ілюструють можливості та рамки користування додатком. Стрілки «користувач», «програмне забезпечення» і «технічне забезпечення» - це інструменти, завдяки яким користувач взаємодіє з грою.

Далі більш детально про ці компоненти:

- правила гри – це компонент який визначає можливий контент та правила генерації цього контенту;

- користувач – це гравець;

- програмне забезпечення – це готовий продукт, який гравець повинен запустити для генерування контенту. Сюди можуть входити як операційна система гаджету, так і сам додаток;

- технічне забезпечення – це гаджет з необхідними системними вимогами, які дозволять користуватись додатком;

- збережені дані гравця – це кінцева фіксація прогресу досягнутого гравцем по мірі проходження гри.

Для ліпшого розуміння можна зробити декомпозицію діаграми (рисунок 3.2), вона складається з наступних елементів:

- Початок гри
- Генерація рівня
- Проходження рівня
- Збереження рузультатів

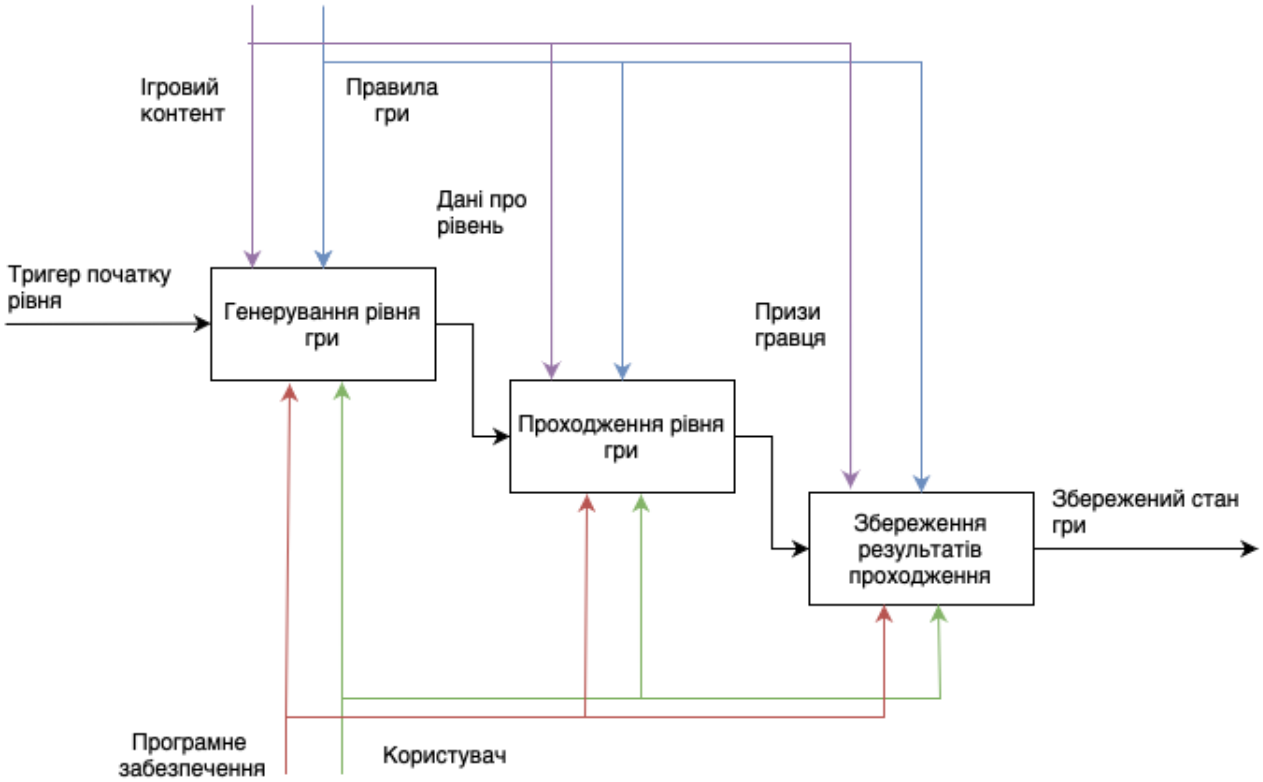

Рисунок 3.2 - Перший рівень декомпозиції

Джерело: побудовано автором

Після початку гри, гравець починає переміщатися по карті і при потраплянні на підземелля (рівень) гравець може обрати чи проходити його, після проходження рівня гравець отримує нагороду та зберігає її.

3.2 Моделювання варіантів використання

Діаграми варіантів використання є важливими при розробці програмного забезпечення, оскільки вони забезпечують візуальне представлення функціональних можливостей системи з точки зору її користувачів. Ці діаграми пропонують чіткий і стислий спосіб зрозуміти, як користувачі взаємодіють із системою, ілюструючи різні сценарії та дії користувача [26, 27]. Відображаючи варіанти використання, учасників та їхні стосунки, діаграми варіантів використання сприяють ефективній комунікації між зацікавленими сторонами, включаючи розробників, дизайнерів і клієнтів. Вони служать схемою для розробки функцій програмного забезпечення, керуючи процесом розробки шляхом визначення вимог користувача та поведінки системи. Крім того, діаграми варіантів використання сприяють ідентифікації потенційних помилок, забезпечуючи більш комплексний і орієнтований на користувача підхід до проектування та розробки програмного забезпечення. По суті, вони є найважливішим інструментом для фіксації, організації та передачі функціональних вимог системи, сприяючи спільному розумінню між усіма сторонами, залученими до життєвого циклу розробки програмного забезпечення [28].

На рисунку 3.2 представлена діаграма варіантів використання компаньона до гри “ErenWood”

На зображеній діаграмі присутній актор, який є гравцем, та можливі варіанти використання додатку:

- Початок гри;
- Перегляд правил;

- Проходження рівня;
- Генерування рандомної секції;
- Вікно доповненої реальності;
- Генерування рандомної пастки;
- Генерування рандомного персонажу;
- Генерування рандомної події;
- Генерування рандомної нагороди;

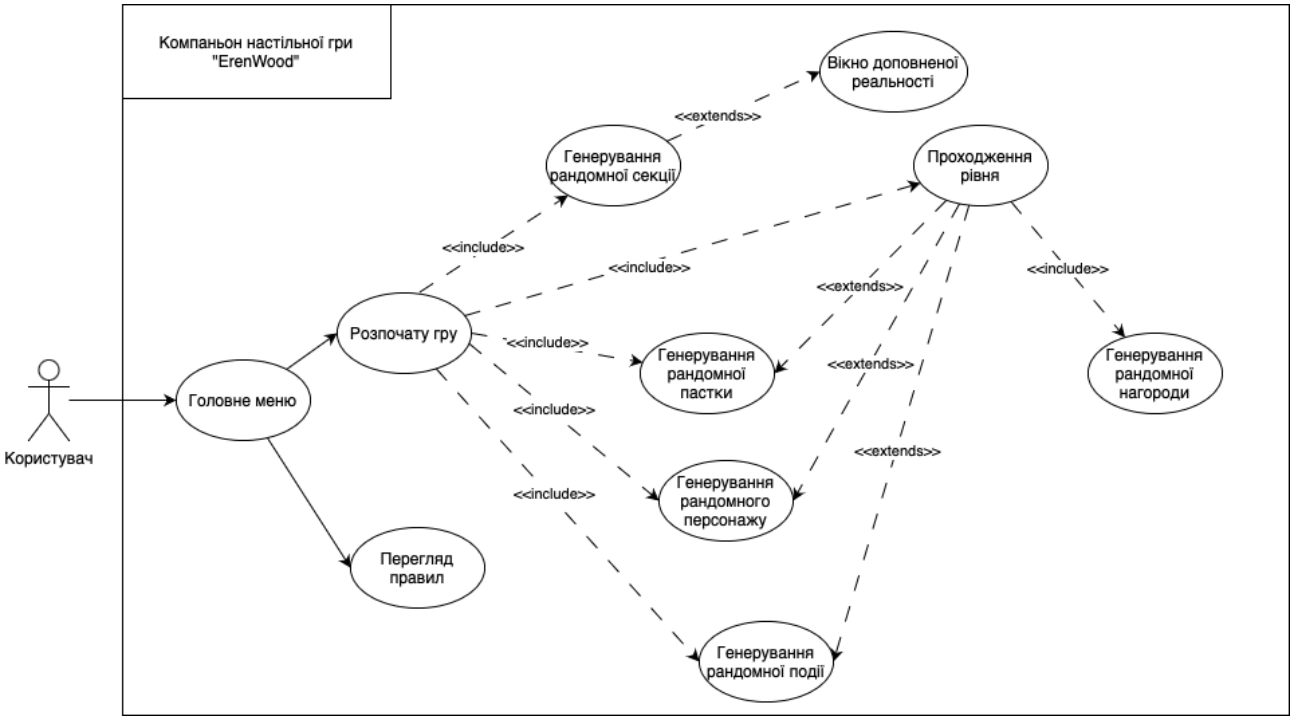

Рисунок 3.3 - Діаграма варіантів використання компаньона до гри “ErenWood”

Джерело: побудовано автором

Деякі варіанти мають <<include>> зв’язки між собою. Зв'язок <<include>> ілюструє звичайний варіант використання. Також деякі варіанти мають зв’язки <<extends>>, який демонструє можливості нестандартного варіанту використання.

4. ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Створення інтерфейсу додатку

Для створення інтерфейсу додатку використаємо додаток Figma [32. 33], що дозволяє створювати дизайн як для мобільних так і для десктопних додатків.

Наш майбутній додаток буде складатися з 4-х екранів:

- Головний екран
- Екран правил
- Екран гри
- Екран доповненої реальності

Додаток буде розроблено для платформи iOS, отже як базовий девайс будемо використовувати iPhone 11.

Оскільки останній екран складається з камери, для нього дизайн можемо не розробляти.

Головний екран повинен бути простим та складатися з двох кнопок:

- Розпочати гру
- Переглянути правила

Тож перейдемо до фігми та розробимо дизайн цього додатку (Рисунок 4.1).

Перша кнопка переправляє нас на екран гри, інша переправляє користувача на екран правил.

Для символів на кнопках виберемо системні картинки, що надають Apple безкоштовно в своєму каталозі SF Symbols [34. 35].

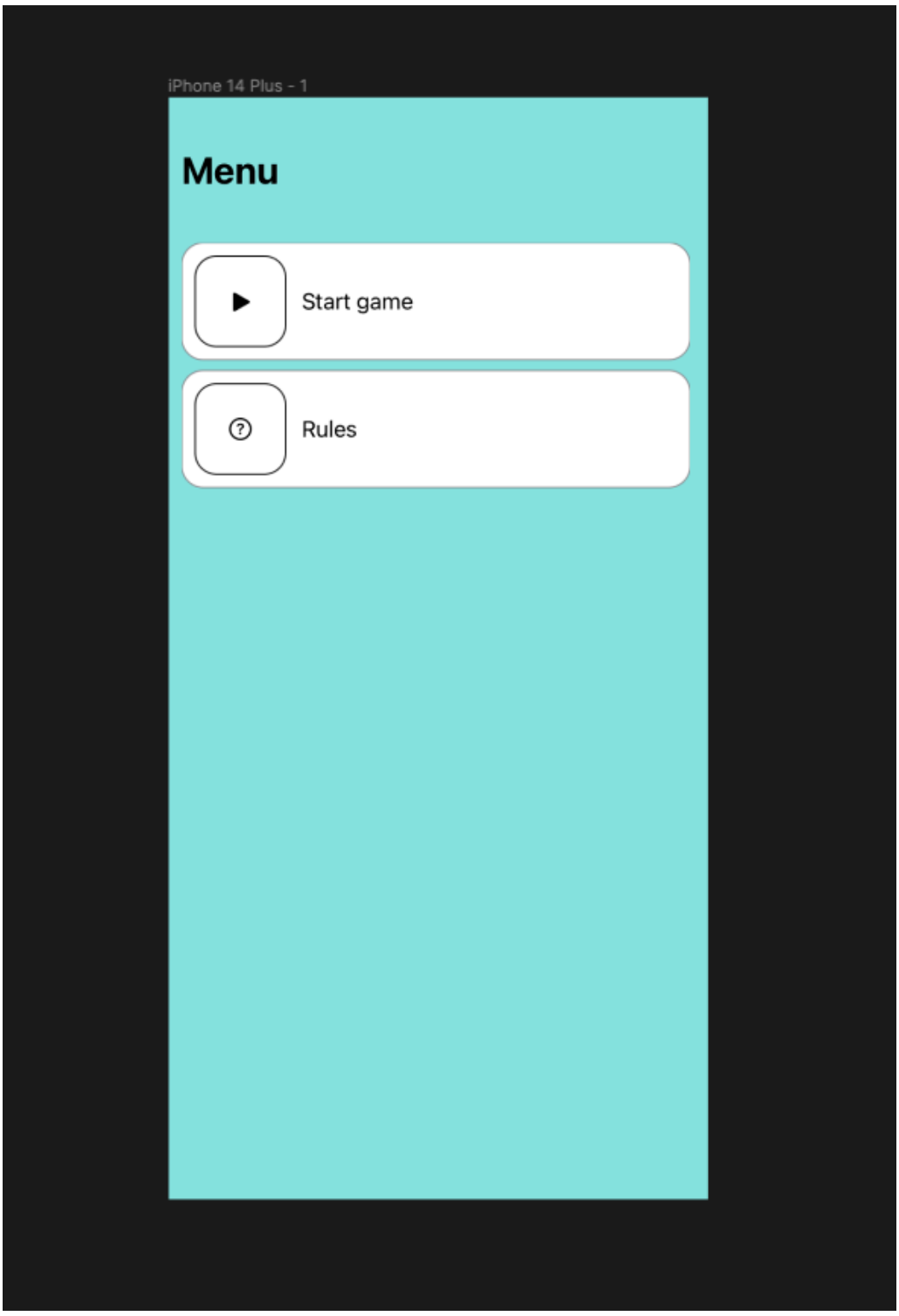

Рисунок 4.1 - Прототип дизайну головного екрану гри

Джерело: Зроблено автором

Наступним екраном розробимо екран правил, це простий текстовий скрін, цо містить усю інформацію необхідну для початку гри (Рисунок 4.2).

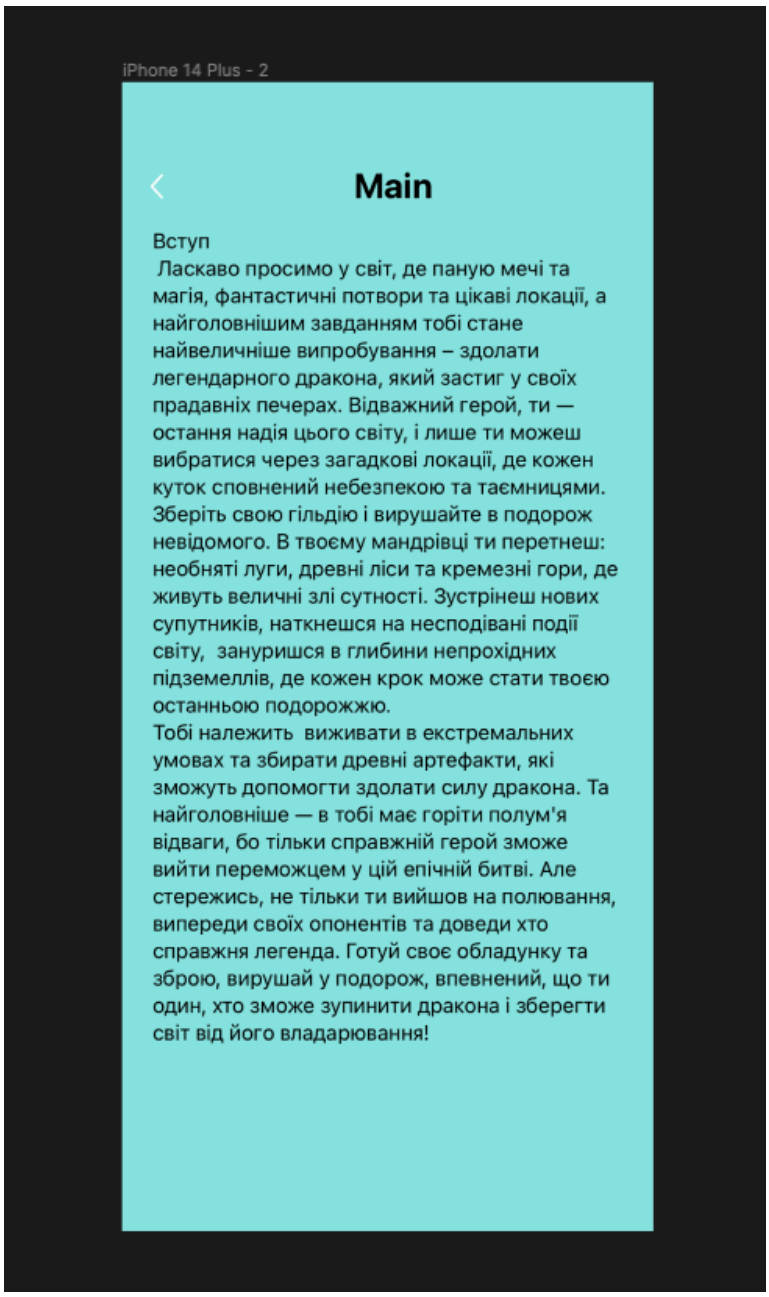

Рисунок 4.2 - Прототип екрану правил гри

Джерело: Зроблено автором

Наступним екраном розробимо екран гри. Екран має містити наступне:

- Кнопка назад на головне меню
- Кнопка отримати випадковий івент
- Кнопка отримати випадкове створіння
- Кнопка отримати випадкову пастку
- Кнопку проходження рівня
- Кнопку генерування секції гри

- Кнопку початку гри

Нижче представлено дизайн даного екрану (Рисунок 4.3), також додатково необхідно розробити екран проходження пілземелля, що буде поверх даного екрану (Рисунок 4.4 - 4.5).

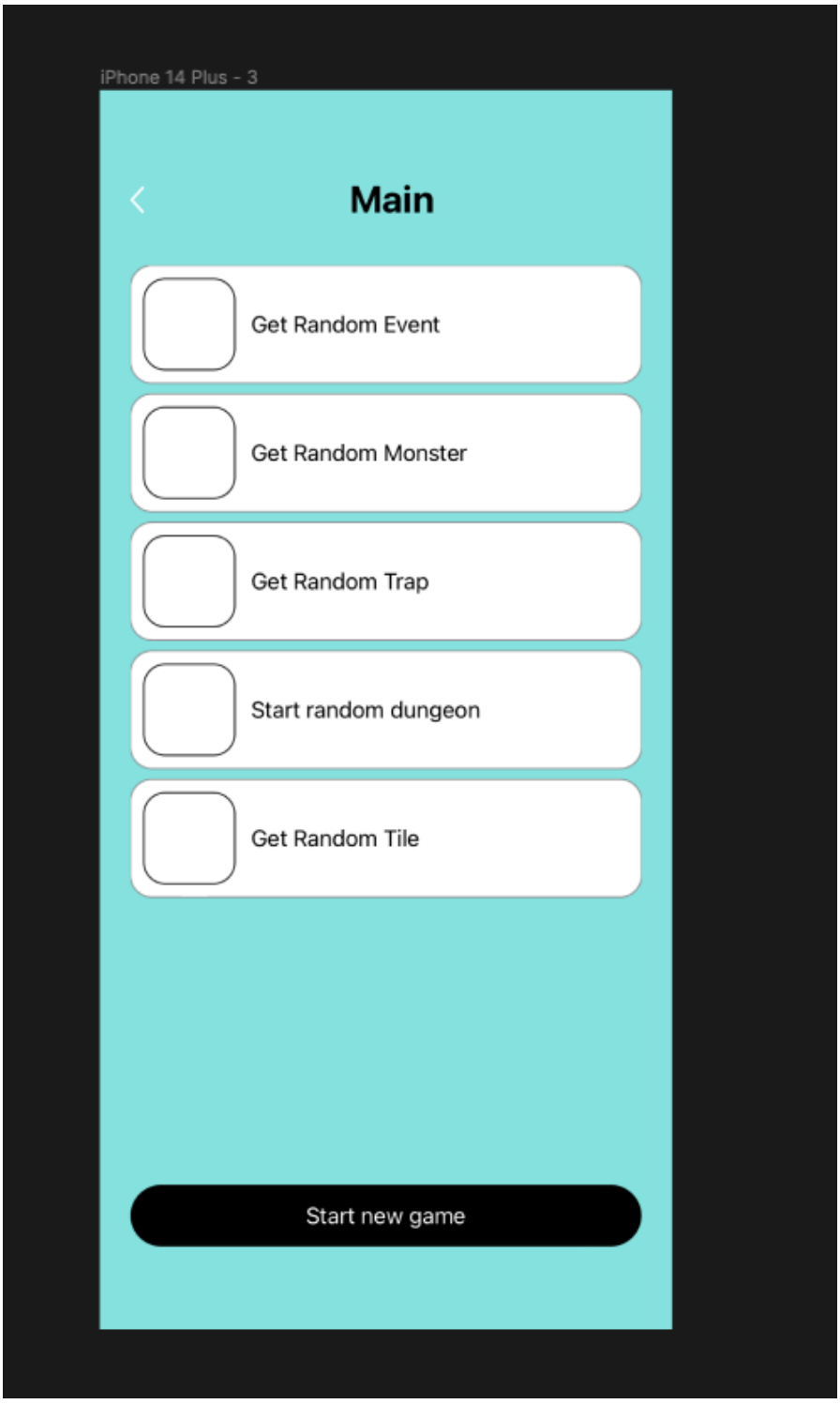

Рисунок 4.3 - Екран гри

Джерело: Зроблено автором

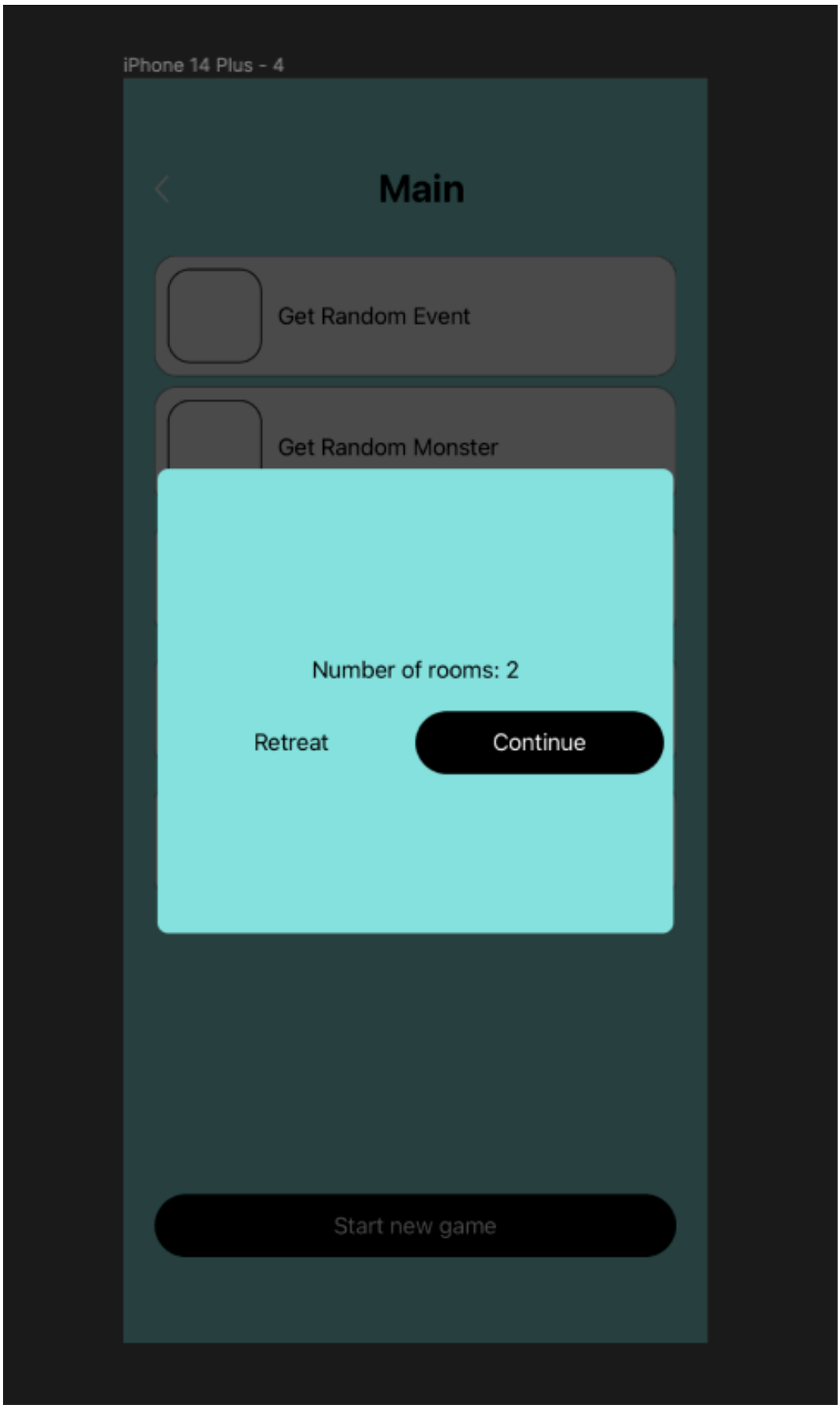

Рисунок 4.4 - Екран початку проходження підземелля

Джерело: Зроблено автором

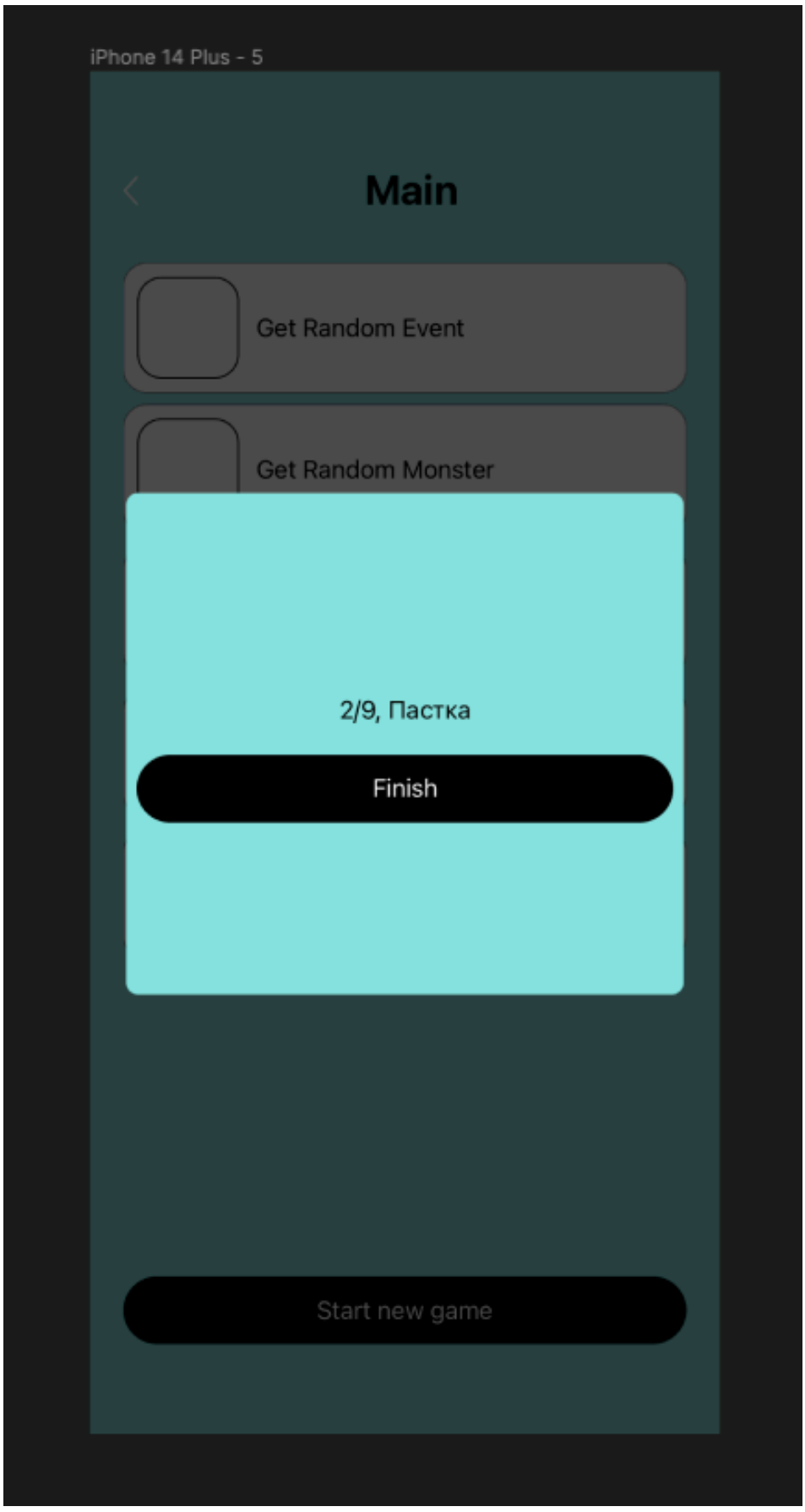

Рисунок 4.5- Екран завершення проходження підземелля

Джерело: Зроблено автором

4.2 Створення алгоритму генерування внутрішньоігрового контенту

Основним функціоналом додатку є генерування контенту, тож розробимо діаграму послідовності [36] для кращого розуміння послідовності виконання подій в додатку (Рисунок 4.6).

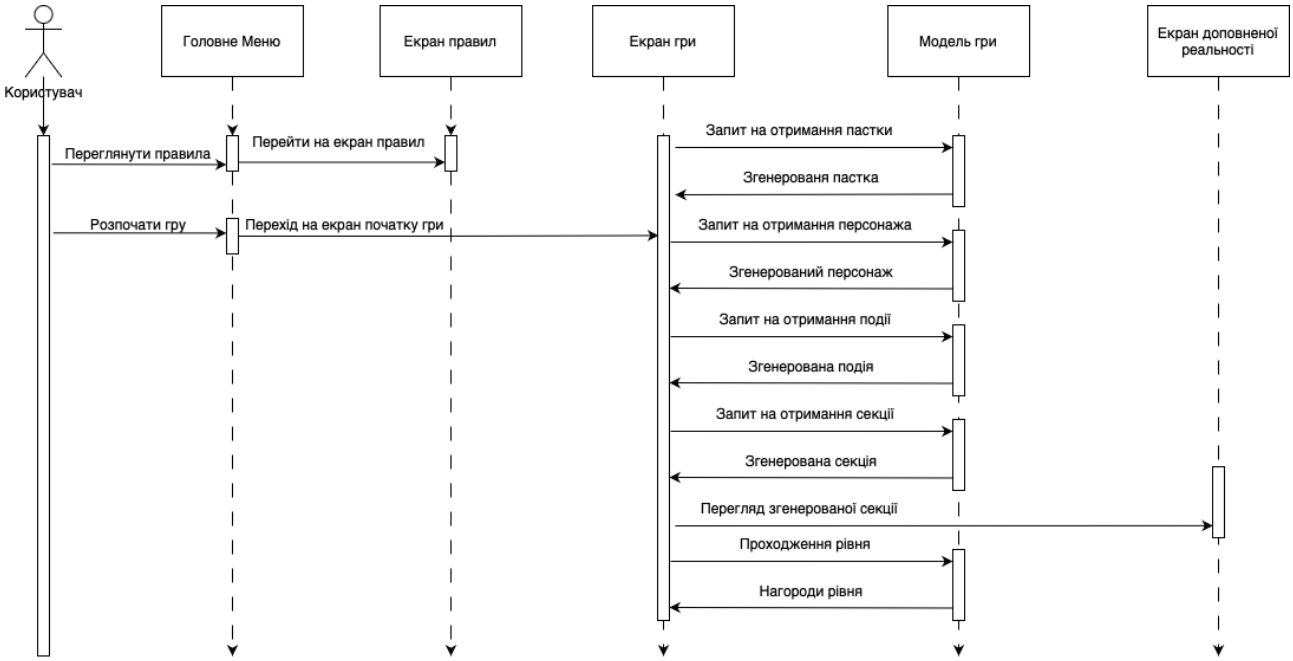

Рисунок 4.6 - Діаграма послідовності процесу гри в додатку

Джерело: Побудовано автором

Екран гри містить в собі кнопки, що генерують внутрішньоігровий контент. Найскладнішим з цих алгоритмів є проходження рівня, оскільки він потребує взаємодії з інтерфейсом користувача та містить в собі інші алгоритми додатку. Розробимо Блок схему [37] цього алгоритму (Рисунок 4.7).

З діаграми ми бачимо, що додаток складається з 4-х екранів:

- Головний екран
- Екран з правилами
- Екран гри
- Екран доповненої реальності

Рисунок 4.7 - Блок схема роботи алгоритму проходження рівня

Джерело: Побудовано автором

З блок схеми ми можемо зрозуміти як повинен працювати алгоритм і можемо переходити до його програмної реалізації (Рисунок 4.8).

```swift
    func continueDungeon() {
        if self.dungeonSize == self.dungeonIteration {
            self.finishDungeon()
            return
        }
        let randomIndex = Int(arc4random_uniform(UInt32(self.dungeonEvents.count)))
        let randomElement = self.dungeonEvents[randomIndex]
        var result = ""
        switch randomElement {
        case .monster:
            result = self.getRandomUnusedMonster()?.monsterName ?? ""
        case .trap:
            result = self.getRandomTrap()?.trapName ?? ""
            self.replaceOccurrences(of: .trap)
        case .event:
            result = self.getRandomUnusedEvent()?.eventName ?? ""
            self.replaceOccurrences(of: .event)
        case .treasure:
            self.replaceOccurrences(of: .treasure)
        }
        self.dungeonIteration += 1
        self.dungeonMessage = result
        self.dungeonRewards.append(result)

        if self.dungeonIteration == self.dungeonSize {
            self.dungeonMessage = dungeonRewards.joined(separator: ", ")
        }
    }
```

Рисунок 4.8 - Алгоритм проходження підземелля

Джерело: Зроблено автором

Для взаємодії з інтерфейсом додатку використовуються можливості фреймворку combine, а саме змінні позначені як @Published, @State та @Binding [38].

4.3 Створення сцену доповненої реальності

Для створення функціоналу доповненої реальності створимо 3д модель секції, що буде інтегруватися з фізичним полем. Модель секції представлена нижче (Рисунок 4.9).

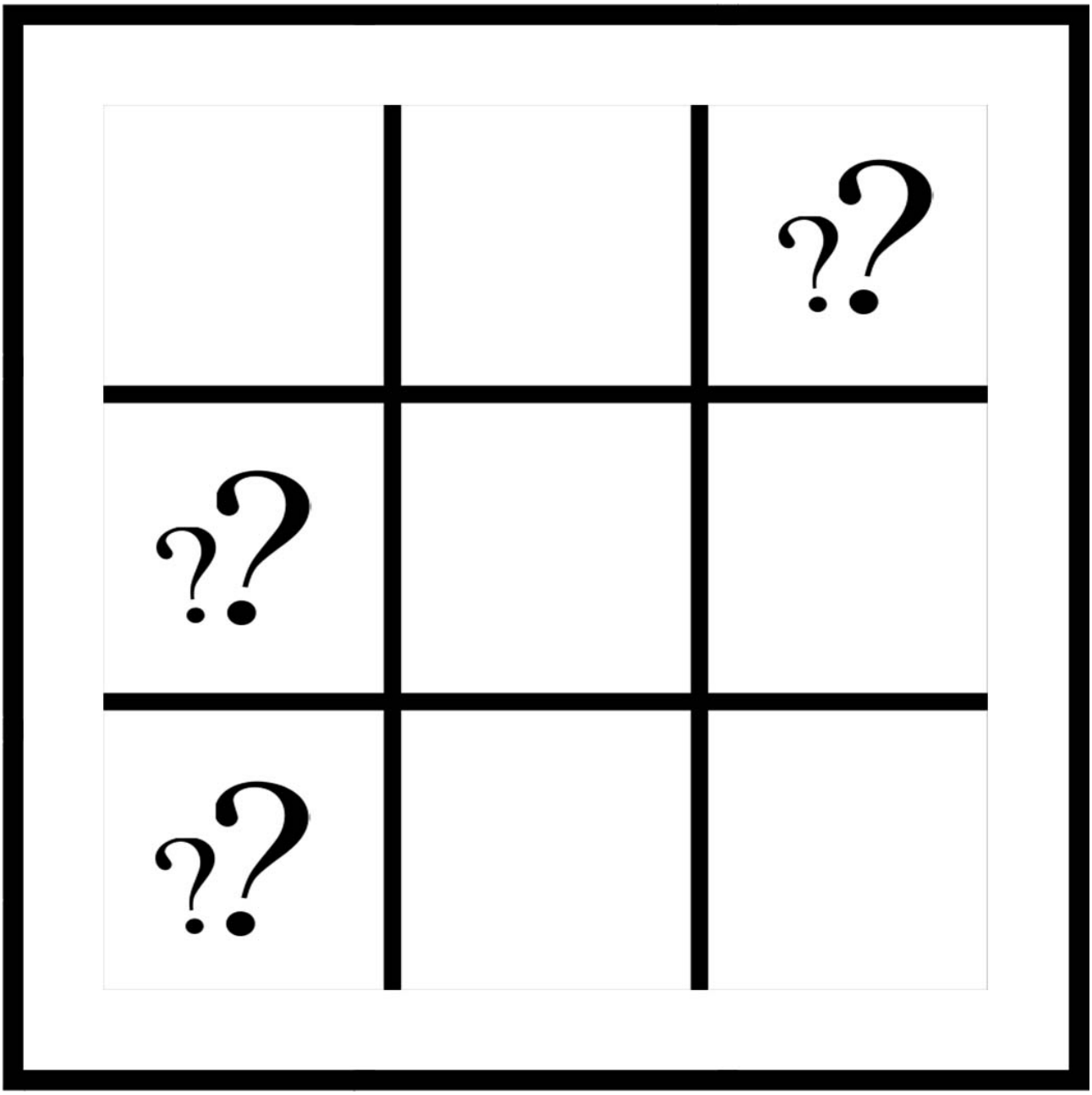

Рисунок 4.9 - Секція з подіями на полі

Джерело: Автор Кіптенко Богдан

Для відтворення даної секції використаємо наявні елементи, що надаються компанією Apple [39]. Замість знаку питання використаємо шахову фігуру коня. Розроблена модель представлена нижче (Рисунок 4.10).

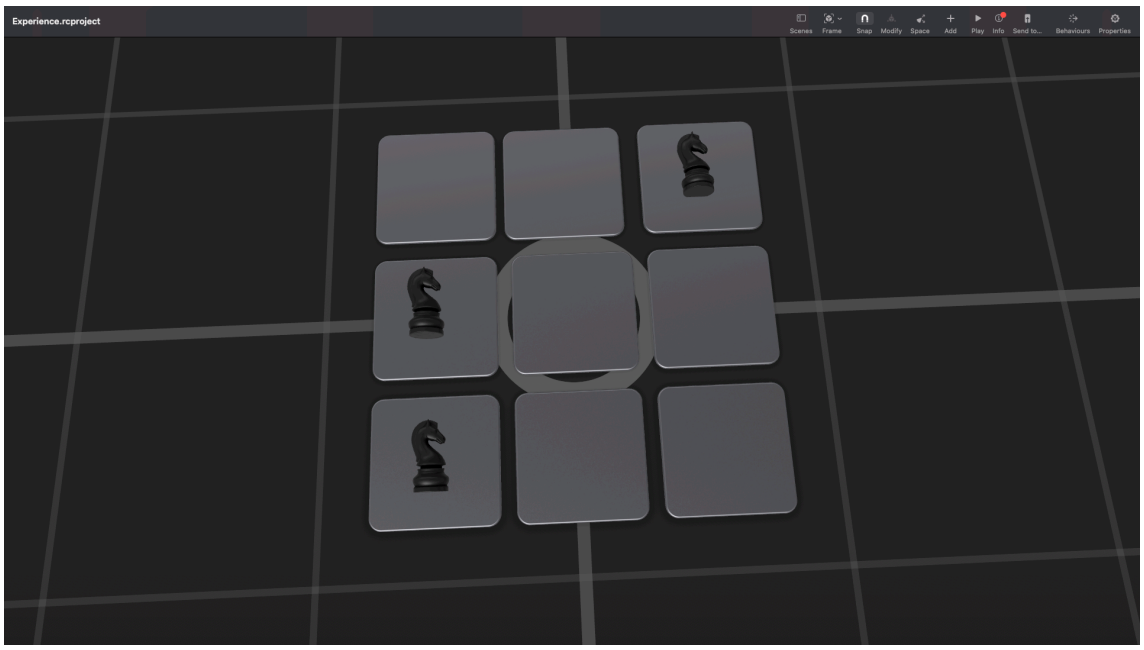

Рисунок 4.10 -Вікно розробки 3д моделей в додатку Reality Composer

Джерело: Зроблено автором

Далі створюємо сцену, що буде містити дану фігуру. Оскільки SwiftUI не підтримує ARKit використаємо протокол UIViewRepresentable [40] щоб портувати код UIKit на SwiftUI (Рисунок 4.11).

```
48  struct ARViewContainer: UIViewRepresentable {
49      func makeUIView(context: Context) -> ARView {
50          let arView = ARView(frame: .zero)
51
52          let boxAnchor = try! Experience.loadScene()
53
54          // Add the box anchor to the scene
55          arView.scene.anchors.append(boxAnchor)
56
57          return arView
58      }
59
60      func updateUIView(_ uiView: ARView, context: Context) { }
61  }
```

Рисунок 4.11 - Додаємо сцену на екран

Джерело: Зроблено автором

Приклад роботи даного екран в додатку можна побачити на прикладі нижче (Рисунок 4.12).

Рисунок 4.12 - Екран генерації секцій додатку

Джерело: Зроблено автором

Для проєкції моделі на поверхню використовуються якорі, зараз якор виставлений на горизонтальну поверхню, для інтеграції з фізичним полем додаємо якір, який є фізичною секцією поля. Нижче можна побачити фізичну секцію поля (Рисунок 4.13), доданий якір до сцени (Рисунок 4.14) та приклад його використання (Рисунок 4.15).

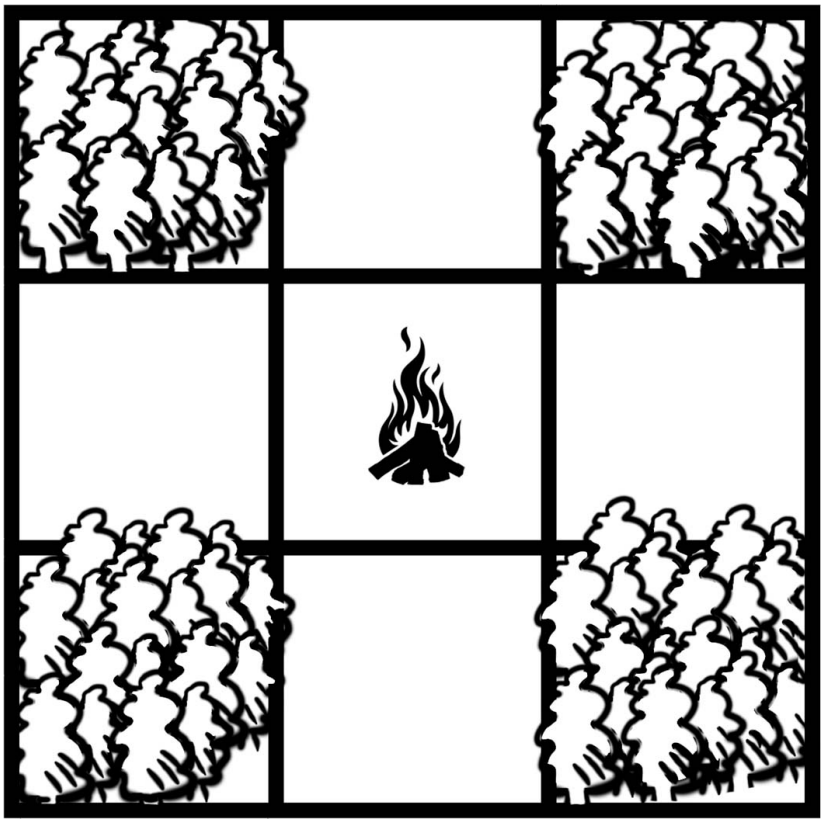

Рисунок 4.13 - Фізична секція ігрового поля

Джерело: Автор Кіптенко Богдан

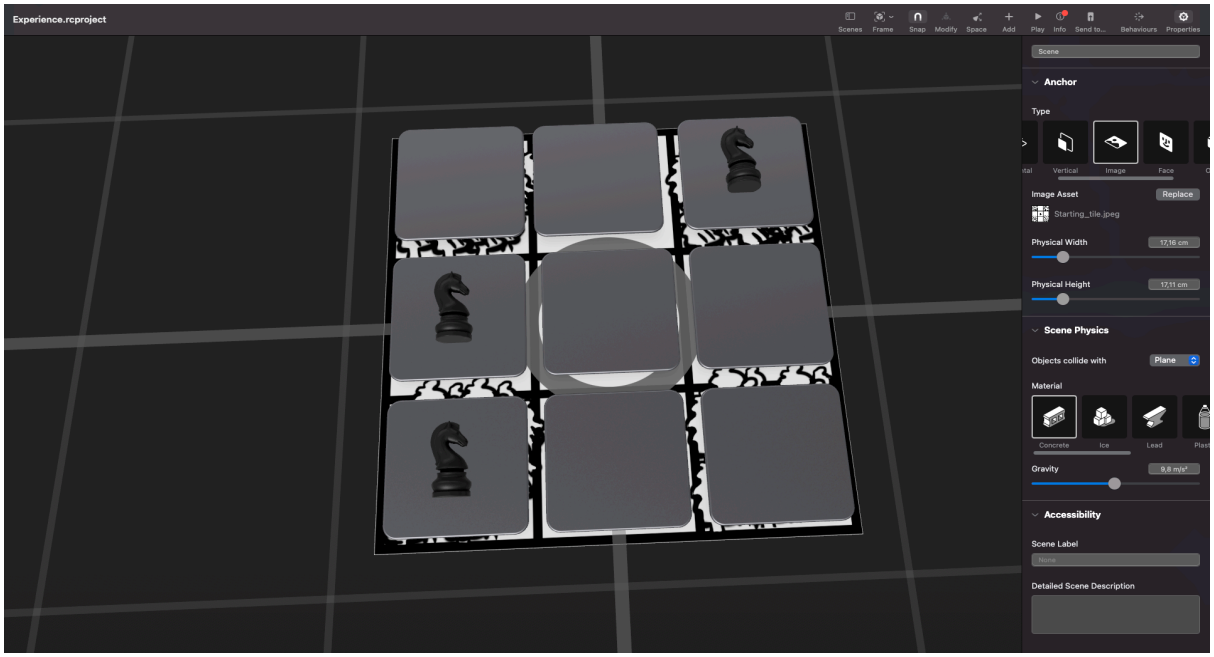

Рисунок 4.14 - Додаємо секцію поля як якір сцени

Джерело: Зроблено автором

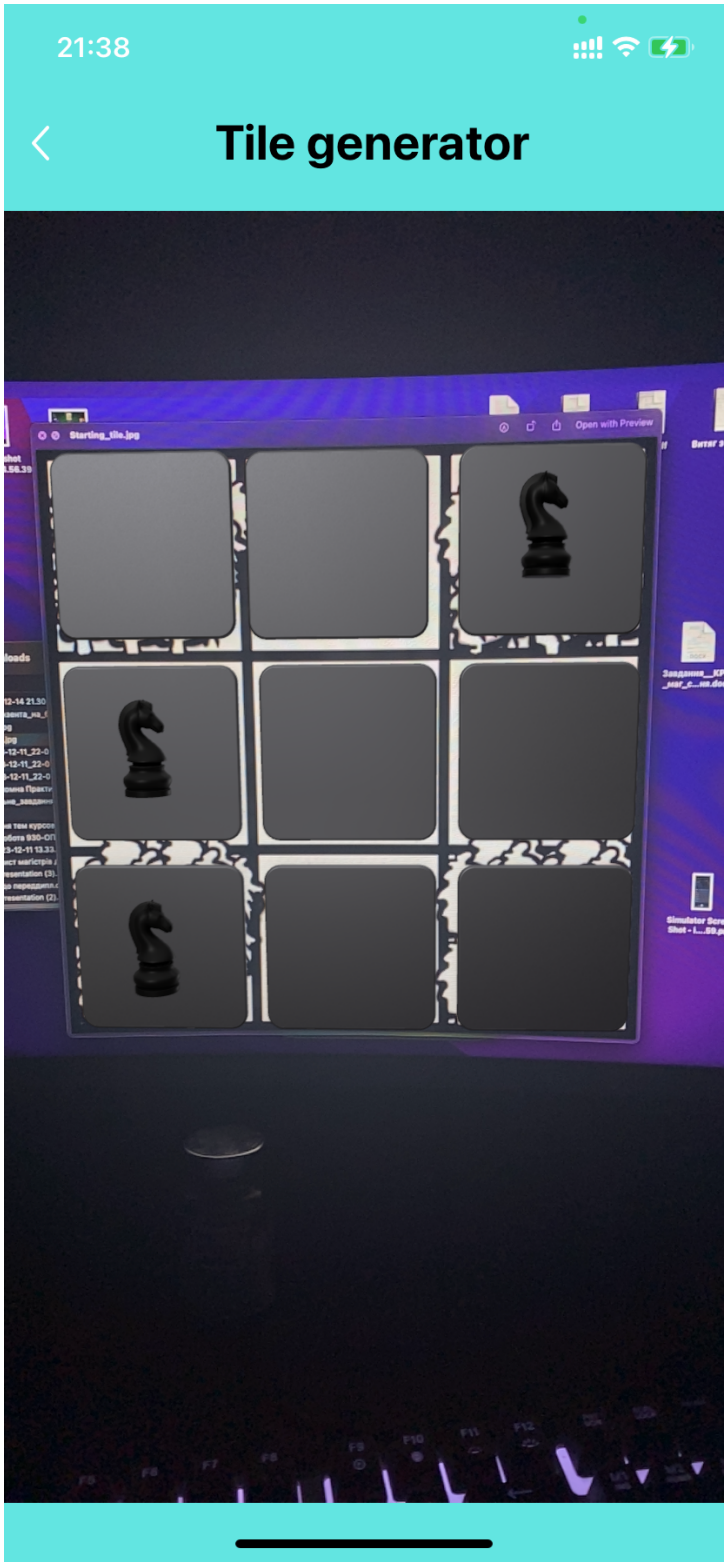

Рисунок 4.15 - Приклад роботи якора

Джерело: Зроблено автором

4.4 Тестування проєкту

Основними функціями для тестування є методи генерації контенту та проходження рівнів. Для тесту будемо використовувати інструменти для процесу виявлення, аналізу та виправлення помилок або неполадок в програмному коді вбудовані в XCode.

Спочатку протестуємо функціонал генерування івенту. Через спрощену систему генерації подій, деякі події треба перегенерувати на інших генераторах, до прикладу - при випадінні події “Пастка” потрібно запускати скрипт генерації пастки (Рисунок 4.16).

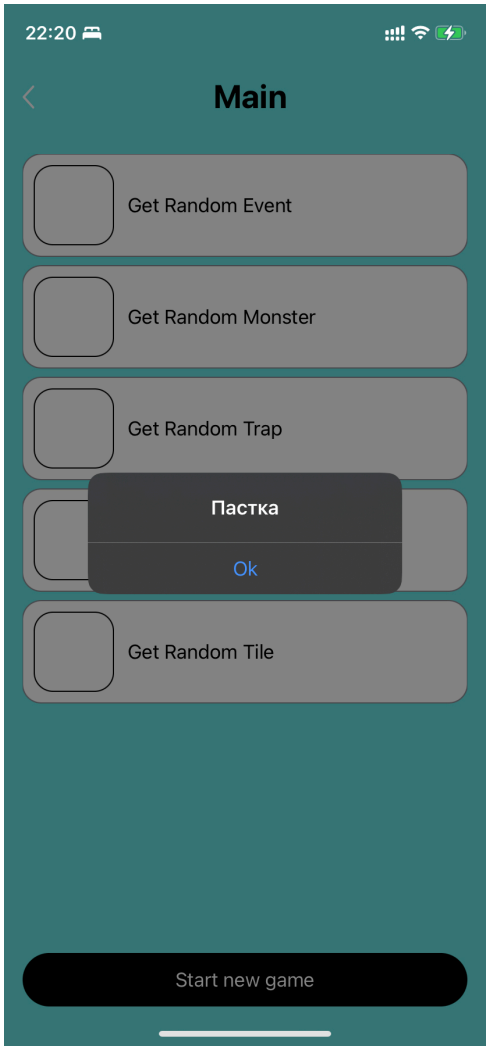

Рисунок 4.16 - Генерація випадкової події

Джерело: Зроблено автором

Цю проблему можна усунути в подальшому за рахунок класифікування подій за типами.

При тестуванні генерації створінь та пасток додаток веде себе коректно, та відпрацьовує наданий сценарій (Рисунок 4.17 - 4.18).

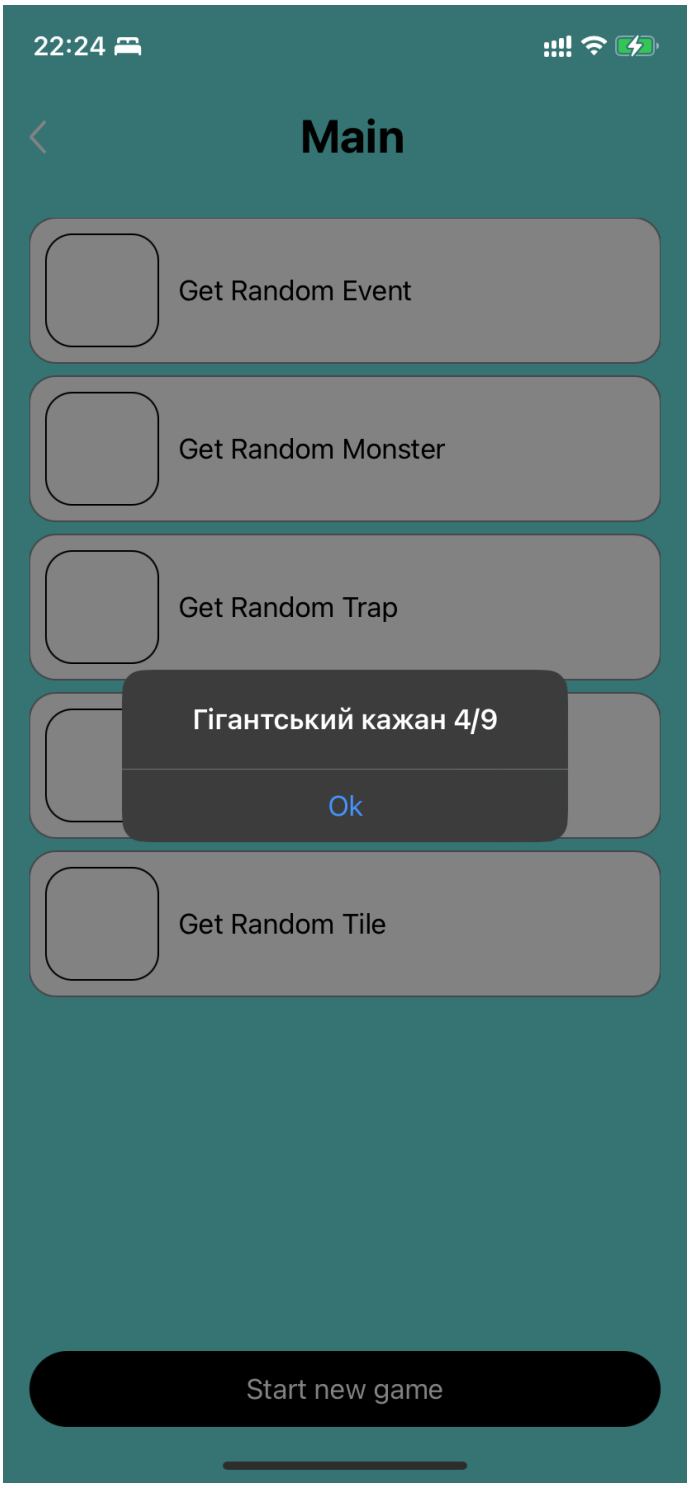

Рисунок 4.17 - Генерація випадкового створіння

Джерело: Зроблено автором

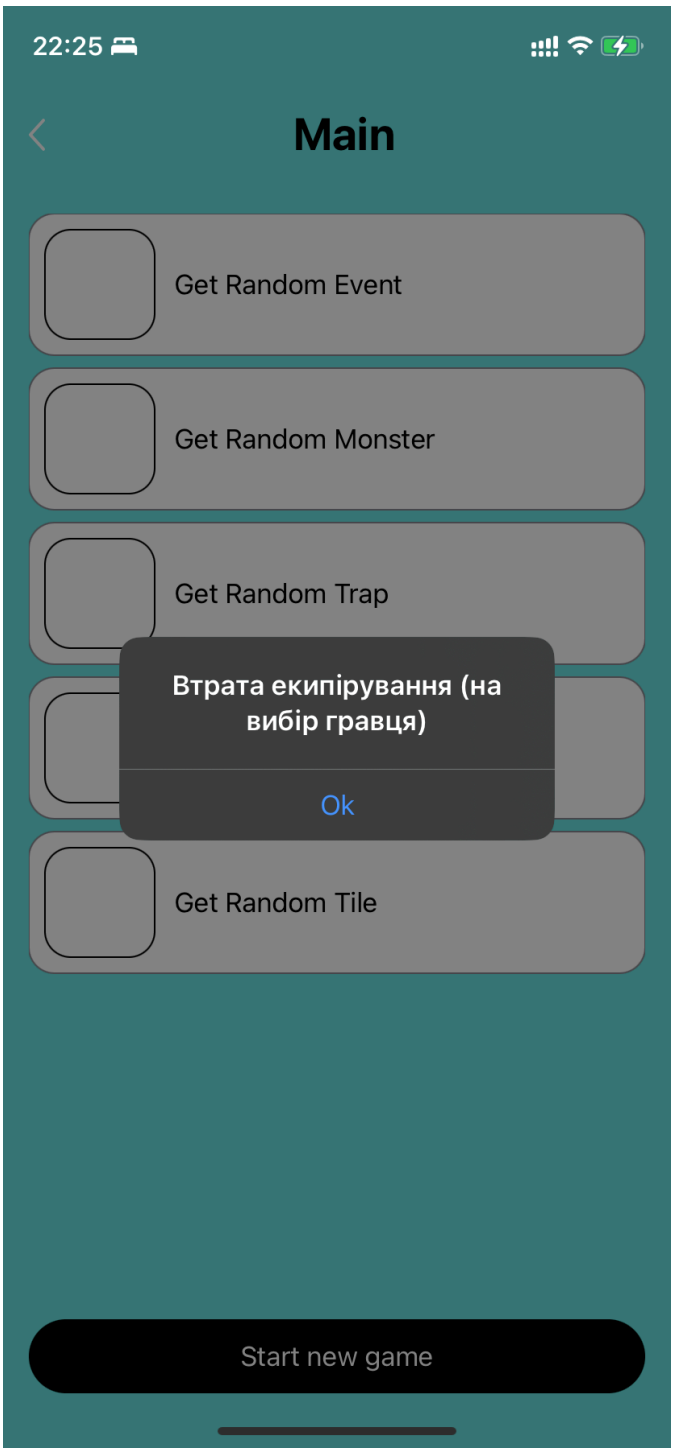

Рисунок 4.18 - Генерація випадкової пастки

Джерело: Зроблено автором

Останнім було протестоване алгоритм проходження підземель, він має схожі недоліки до першого тест кейсу, оскільки з певною долею вірогідності може випасти подія, які не є типізованими, що створює певні труднощі при проходженні. Інший функціонал підземель працює згідно з блок схемою

зазначеною в пункті 4.2. Приклад роботи підземель (Рисунок 4.19 - 4.20). Також під час тестування було помічено візуальний баг (Рисунок 4.21), текст виходив за рамки контейнеру, в подальшому ця помилка була виправлена (Рисунок 4.22).

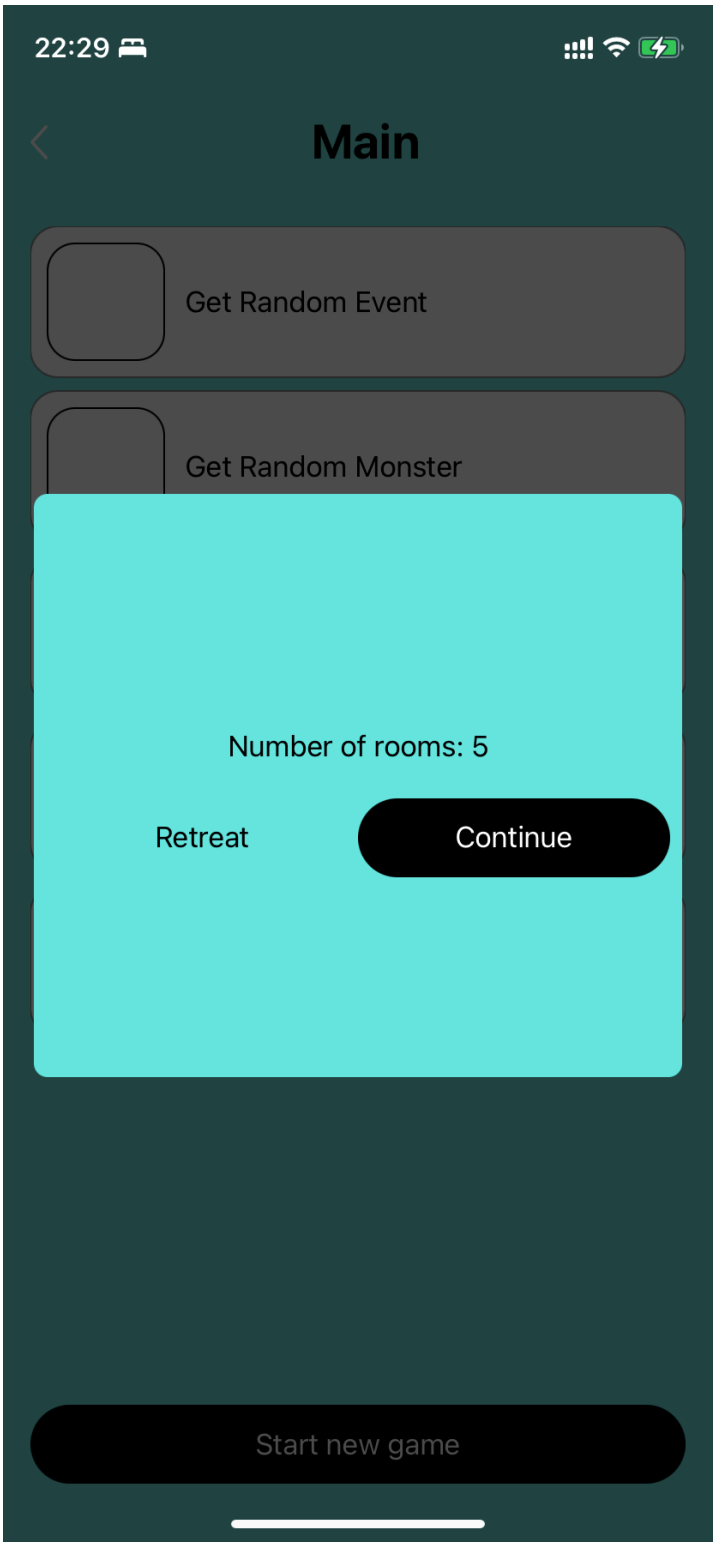

Рисунок 4.19 - Кількість кімнат в підземеллі

Джерело: Зроблено автором

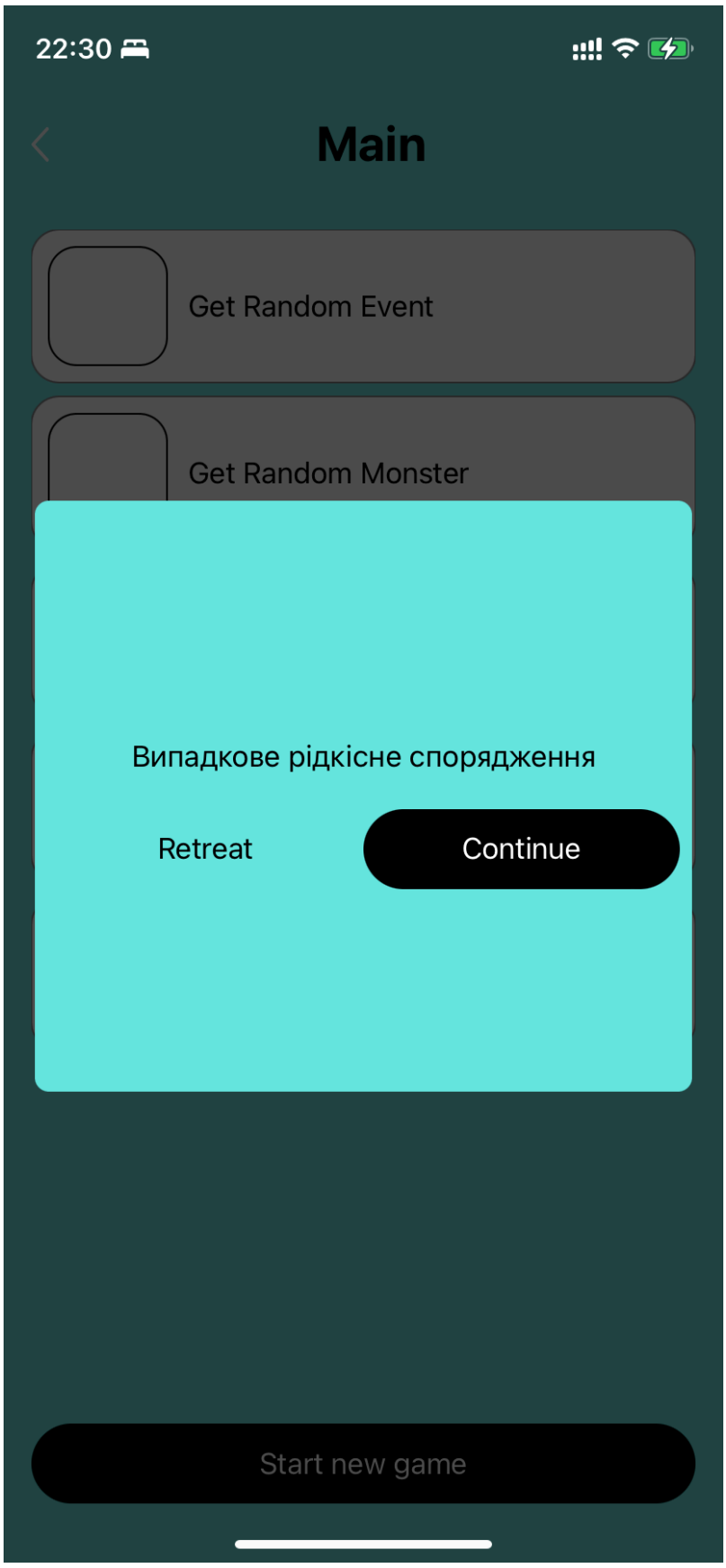

Рисунок 4.20 - Нагорода в першій кімнаті

Джерело: Зроблено автором

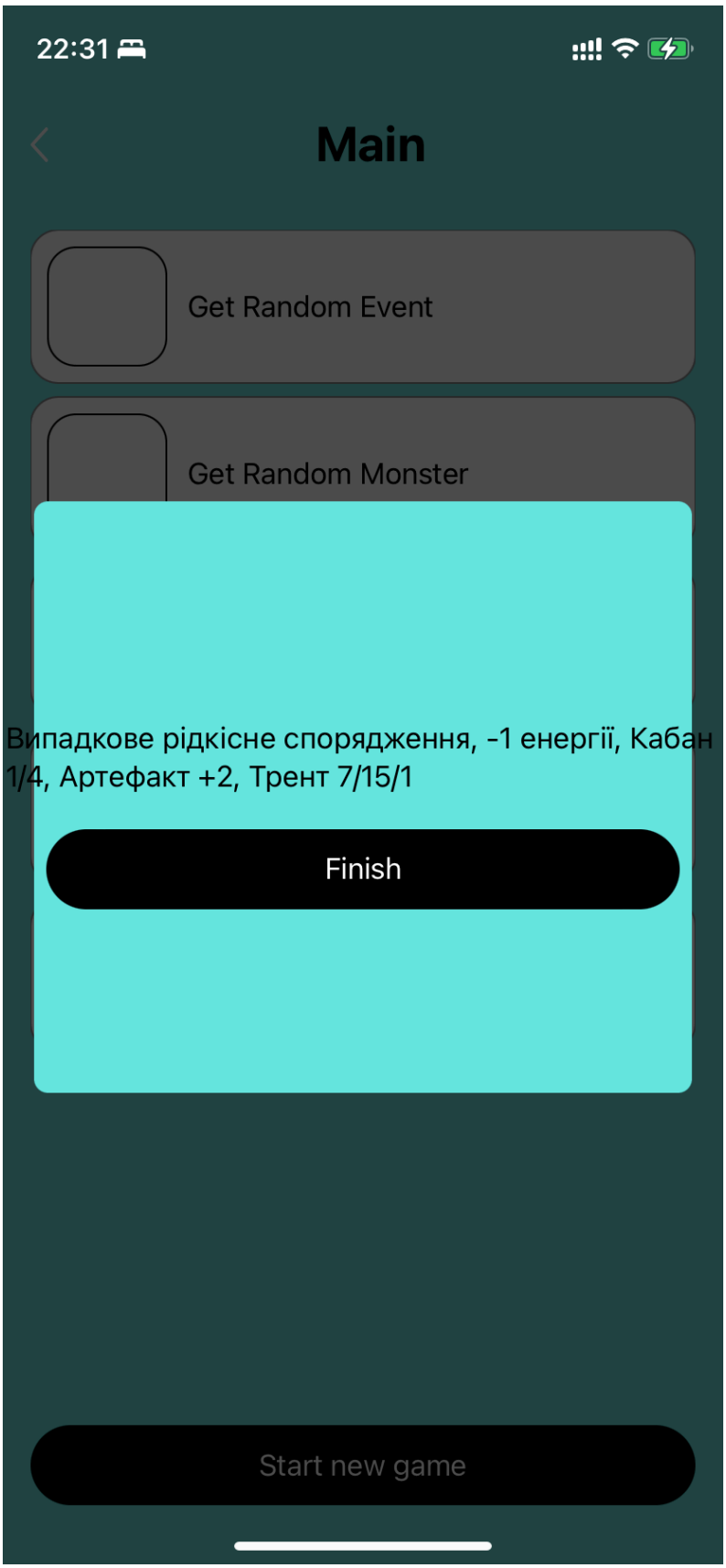

Рисунок 4.21 - Візуальний баг, текст виходить за рамки контейнеру

Джерело: Зроблено автором

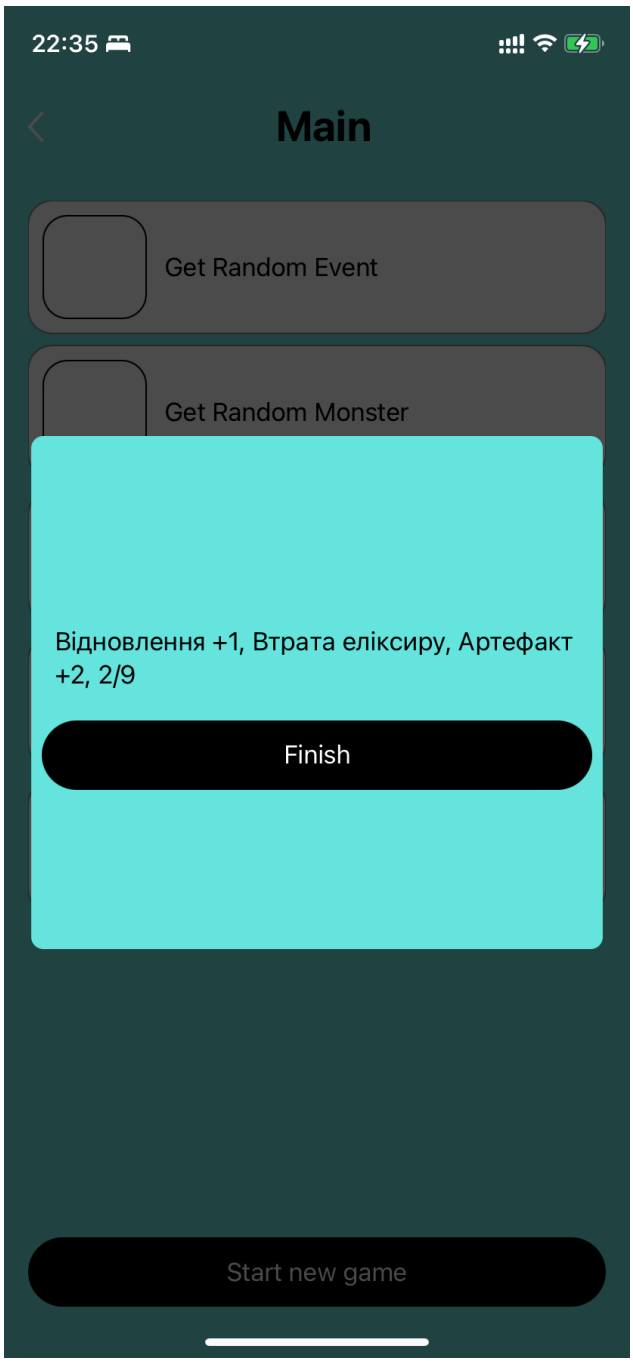

Рисунок 4.22 - Баг виправлено

Джерело: Зроблено автором

При подальшому тестуванні додатку помилок не було виявлено.

ВИСНОВКИ

В результаті виконання дипломного проекту було досягнуто мету у створенні інформаційної системи для настільної гри "ErenWood" з використанням технологій доповненої реальності для платформи iOS. Проведений аналіз існуючих додатків та технологій надав засади для розробки ефективного рішення.

Ретельне планування та визначення мети проекту, а також створення плану розробки з оцінкою ризиків, забезпечило систематичний підхід до роботи, що сприяло успішній реалізації завдань. Отримані результати створюють основу для подальшого розвитку та вдосконалення інформаційної системи.

Розроблені моделі персонажів, ігрових предметів та інтерфейсу додатку визначають вигляд та функціонал системи, забезпечуючи користувачам високоякісний ігровий досвід.

Логіки, використані в додатку, були розроблені із застосуванням сучасних підходів та найкращих практик, забезпечуючи стабільність та ефективність системи. Завдяки виконанню цих етапів, інформаційна система стає не тільки доповненням до настільної гри, але і самостійним продуктом, що збагачує ігровий простір.

Проведене тестування дозволило заздалегідь виправити можливі проблеми та знайти шляхи їх вирішення, що є важливим аспектом для успішної завершення проекту.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. SwiftUI Cookbook: A guide to solving the most common problems and learning best practices while building SwiftUI apps - Year: 2021 | Book | Publisher: Packt Publishing
2. A Quantitative Performance Benchmark of Different Navigation Patterns and User Interface Design Frameworks for an Enhanced iOS Experience - Year: 2023 | Conference Paper | Publisher: IEEE
3. Animating SwiftUI Applications: Create visually stunning and engaging animations for iOS with SwiftUI - Year: 2023 | Book | Publisher: Packt Publishing
4. LonelyScape: Increasing Attractiveness of Escape Room Game Using Augmented Reality Technology - Year: 2023 | Conference Paper | Publisher: IEEE
5. Improvement in the fun of the board game by A.R. introduction (In the case of Japanese Board Game “Sugoroku”) - Year: 2013 | Conference Paper | Publisher: IEEE
6. The Effects of Integrating Digital Board Game into Prime Factorization Learning on Elementary Students’ Flow Experience - Year: 2022 | Conference Paper | Publisher: IEEE
7. Design and realization of a two-armed multifunctional companion robot with electronic game board - Year: 2009 | Conference Paper | Publisher: IEEE
8. Analysis of iOS SQLite Schema Evolution for Updating Forensic Data Extraction Tools - Year: 2020 | Conference Paper | Publisher: IEEE
9. Creating our first unit test using XCTest [Електронний ресурс] – Режим доступу до ресурсу: https://www.hackingwithswift.com/read/39/2/creating-our-first-unit-test-using-xctest
10. Understanding Auto Layout [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/index.html

11. Augmented Reality in Education Learning and Training - Year: 2018 | Conference Paper | Publisher: IEEE

12. What is augmented reality (AR)? [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/whatis/definition/augmented-reality-AR

13. Understanding virtual reality and augmented reality [Електронний ресурс] – Режим доступу до ресурсу: https://edu.gcfglobal.org/en/thenow/understanding-virtual-reality-and-augmented-reality/1/

14. How Does Augmented Reality Work? [Електронний ресурс] – Режим доступу до ресурсу: https://hbr.org/2017/11/how-does-augmented-reality-work

15. Augmented Reality (AR) vs. Virtual Reality (VR): What's the Difference? [Електронний ресурс] – Режим доступу до ресурсу: https://www.pcmag.com/news/augmented-reality-ar-vs-virtual-reality-vr-whats-the-difference

16. ARKit and ARCore in serve to augmented reality - Year: 2020 | Conference Paper | Publisher: IEEE

17. ARKit as indoor positioning system - Year: 2019 | Conference Paper | Publisher: IEEE

18. Utilizing Apple’s ARKit 2.0 for Augmented Reality Application Development - Year: 2019 | Conference Paper | Publisher: IEEE

19. SwiftShot: Creating a Game for Augmented Reality [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/arkit/arkit_in_ios/swiftshot_creating_a_game_for_augmented_reality

20. The 29 best AR apps for iOS that you need try [Електронний ресурс] – Режим доступу до ресурсу: https://www.cnet.com/pictures/best-ar-apps-for-ios-that-you-need-to-try/

21. Design and implementation of the companion app. service for scene-based product advertisement - Year: 2017 | Conference Paper | Publisher: IEEE

22. Companion Apps: How They Can Add Value to Your Game [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/pulse/companion-apps-how-can-add-value-your-game-sam-marsh-cpe2f

23. What is a companion app and what is its function? [Електронний ресурс] – Режим доступу до ресурсу: https://www.sir-apfelot.de/en/companion-app-20836/

24. 10 Video Games That Have Companion Apps [Електронний ресурс] – Режим доступу до ресурсу: https://www.thegamer.com/best-video-games-companion-apps/

25. Companion Apps are Taking Video Games to the Next Level Електронний ресурс] – Режим доступу до ресурсу: https://brainhub.eu/library/companion-apps

26. Effective Label Matching for Automatic Evaluation of Use -- Case Diagrams - Year: 2012 | Conference Paper | Publisher: IEEE

27. Designing A Use Case Diagram For Developing An Electricity Consumption (EC) System - Year: 2021 | Conference Paper | Publisher: IEEE

28. Qualifying use case diagram associations - Year: 2006 | Magazine Article | Publisher: IEEE

29. SwiftUI vs. UIKit: The Evolution of iOS App Development [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@nareshkukkala/swiftui-vs-uikit-the-evolution-of-ios-app-development-f87dcdfa9ea3

30. Augmented reality by Apple [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/augmented-reality/

31. Journeys in Middle-earth [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/ua/app/journeys-in-middle-earth/id1446642892

32. What is Figma? - Режим доступу до ресурсу: www.nobledesktop.com/learn/figma/what-is-figma

33. Figma Tech Stack: 10 Reasons Why Designer Need To Use - Режим доступу до ресурсу: https://www.techmagic.co/blog/figma-design-tools/

34. SF Symbols 5 - Режим доступу до ресурсу: https://developer.apple.com/sf-symbols/

35. SF Symbols: The benefits and how to use them guide - Режим доступу до ресурсу: https://www.avanderlee.com/swift/sf-symbols-guide/

36. Sequence diagrams - Режим доступу до ресурсу: https://www.ibm.com/docs/en/rsm/7.5.0?topic=uml-sequence-diagrams

37. What is a Flowchart - Режим доступу до ресурсу: https://www.lucidchart.com/pages/what-is-a-flowchart-tutorial

38. Data flow in SwiftUI: State, Binding, StateObject, ObservedObject - Режим доступу до ресурсу: https://en.proft.me/2021/05/22/data-flow-swiftui-state-binding-stateobject/

39. Creation tools for spatial apps - Режим доступу до ресурсу: https://developer.apple.com/augmented-reality/tools/

40. UIViewRepresentable - Режим доступу до ресурсу: https://developer.apple.com/documentation/swiftui/uiviewrepresentable

ДОДАТОК А

А.1 Ідентифікація мети IT-проекту

Готовим продуктом проекту є мобільний додаток компаньон до гри «ErenWood». Для ілюстрування результату, буде використано метод SMART (Specific, Measurable, Achievable, Relevant, Time-framed) у таблиці А.1.

Таблиця А.1 – Деталізація мети проекту методом SMART

Specific (деталізована)	Розробити інформаційну систему, інтегровану в настільну гру
Measurable (вимірювана)	Покращити та пришвидшити ігровий процес на 50%
Achievable (досяжна)	Існує багато прикладів аналогічних додатків до інших ігор, що полегшує розробку даного додатку
Relevant (реалістична)	Дослідження поставленої проблеми допоможить оптимізовувати настільні ігри в майбутньому
Time-framed (обмежена в часі)	Поставлену задачу можна виконати за декілька тижнів

А.2 Планування змісту структури робіт інформаційної системи

Для ефективнішого розподілу часу необхідно розбити роботу на етапи, для цього розробимо WBS структуру проєкту (рисунок А.1). Також створимо OBS структуру (рисунок А.2) для розуміння розподілення задач між виконавцями.

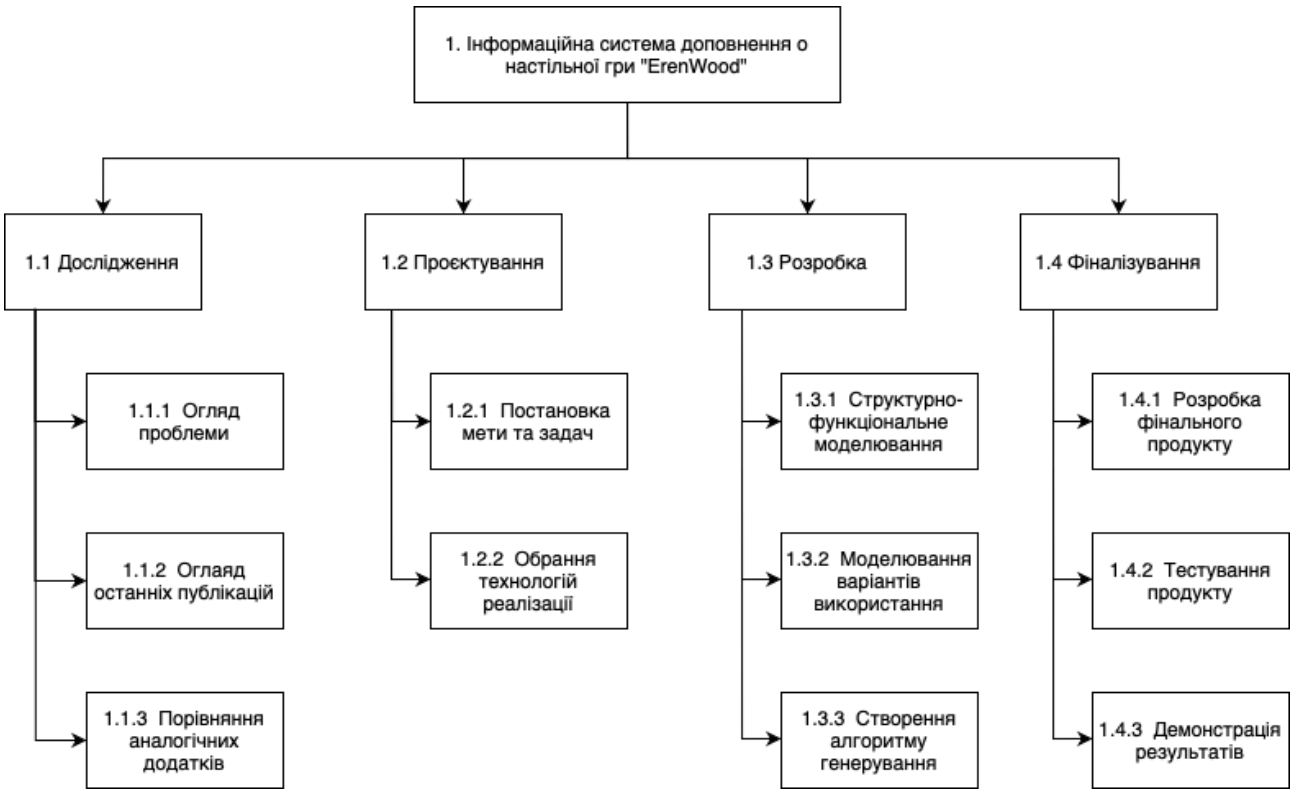

Рисунок А.1 – WBS структура проекту

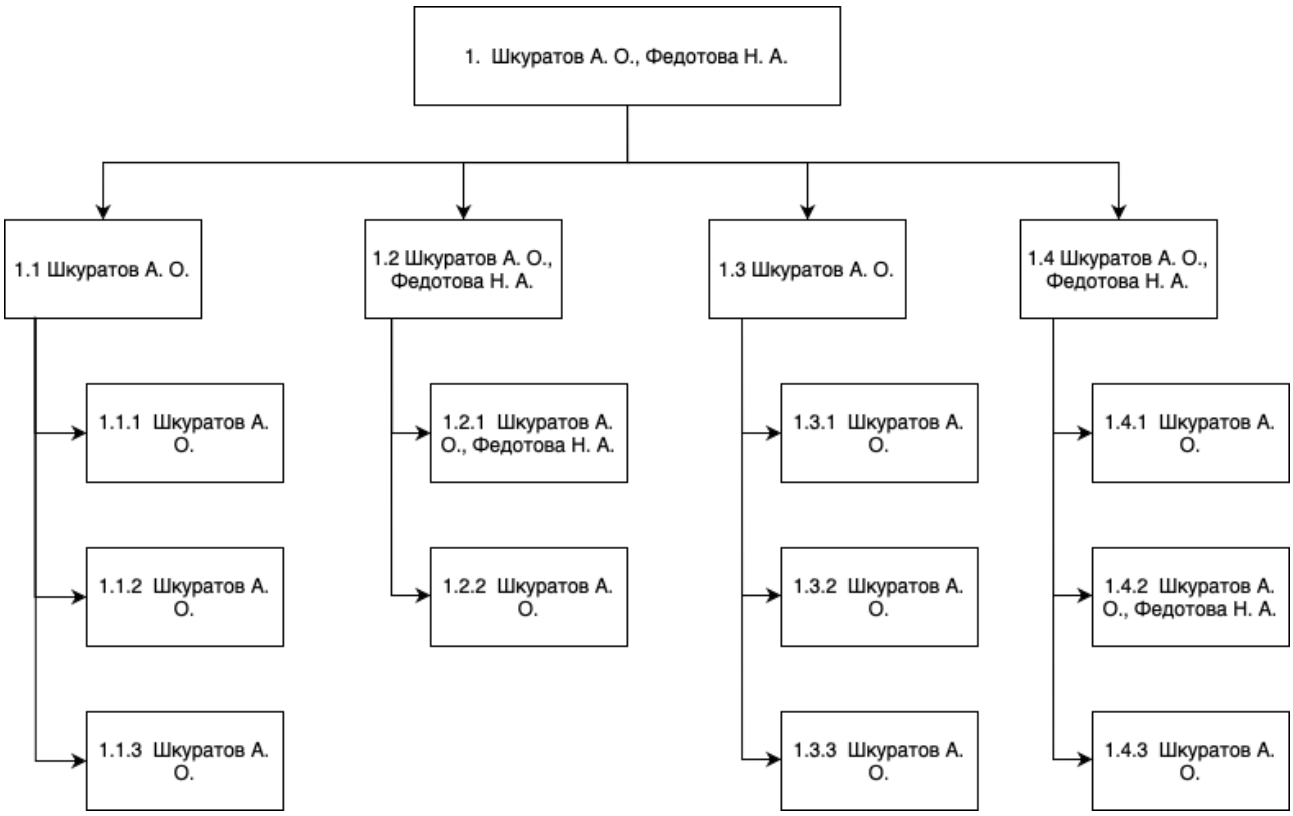

Рисунок А.2 – OBS структура проекту

А.3 Побудова календарного графіку виконання інформаційної системи

Для відображення завдань та запланованого часу на їх виконання розробимо діаграму Ганту. Побудована діаграма Ганта зображена на рисунках А.3 та А.4.

1	Задача	Дата початку	Дата закінчення
2	1. Огляд проблеми	19.09.2023	25.09.2023
3	2. Порівняння аналогів	22.09.2023	24.09.2023
4	3. Постановка мети та задачі	25.09.2023	26.09.2023
5	4. Вибір технологій реалізації	26.09.2023	27.09.2023
6	5. Структурно-функціональне моделювання	28.09.2023	02.10.2023
7	2. Моделювання варіантів використання	02.10.2023	06.10.2023
8	7. Створення алгоритму генерування	07.10.2023	21.10.2023
9	8. Розробка фінального додатку	22.10.2023	25.11.2023
10	9. Оформлення звіту	26.11.2023	05.12.2023

Рисунок А.3 – Діаграма Ганта проекту

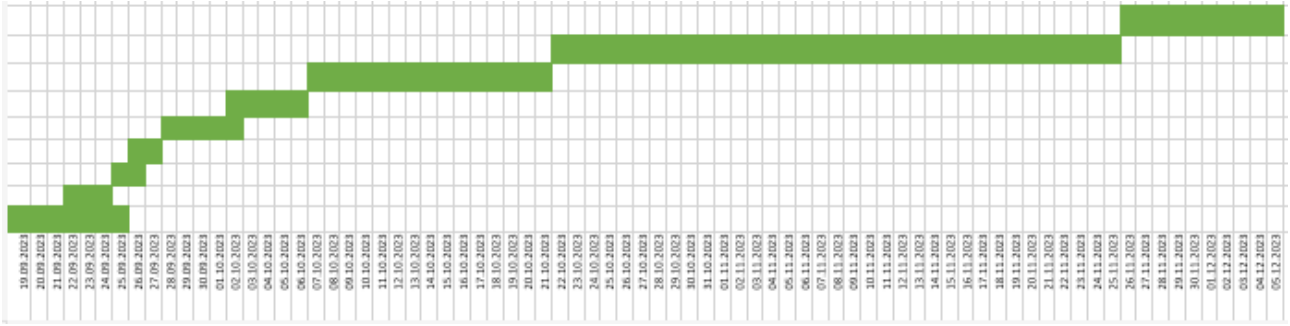

Рисунок А.4 – Діаграма Ганта проекту

А.4 Планування ризиків проекту

Проведемо оцінку ризиків (табл. А.2 та А.3). Для зручності введемо критерії та поділимо їх на категорії які в свою чергу поділимо не градації по значущості:

Ймовірність виникнення:

1. Низька ймовірність
2. Середня ймовірність
3. Висока ймовірність

Ступінь впливу:

1. Низький
2. Середній

3. Високий

Рівень ризику:

1. Прийнятні
2. Виправдні
3. Недопустимі

Таблиця А.2 – Шкала оцінювання ризиків

Оцінка	Ймовірність виникнення	Вплив ризику	Рівень ризику
	Низька	Низький	Прийнятні
	Середня	Середній	Виправдані
	Висока	Високий	Недопустимі

Таблиця А.3 – Матриця оцінки ризиків

Ризик	Оцінка		
	Ймовірність виникнення	Вплив ризику	Тип ризику
Недоліки планування робіт			
Поява додаткових завдань/вимог			
Недотримання календарного плану			
Відсутність інтернет-зв'язку			

ДОДАТОК Б

```
// ErenwoodCompanionApp.swift
// Main додатку
import SwiftUI
@main
struct ErenwoodCompanionApp: App {
    var body: some Scene {
        WindowGroup {
            MenuView()
        }
    }
}
RulesView.swift
struct RulesView: View {
    @Environment(\.dismiss) var dismiss
   @State var rules = """
   …
    """
   var body: some View {
        ZStack {
            Color.mint.ignoresSafeArea()
           VStack {
                HStack(alignment: .center) {
                    Image(systemName: "chevron.left")
                        .resizable()
                        .frame(width: 10, height: 20)
                        .onTapGesture {
                            dismiss()
                        }
                    Spacer()
                    Text("Main")
```

```
                .font(.title)
                .fontWeight(.bold)
                .foregroundColor(.black)
              Spacer()
            }
            .padding(.horizontal, 16)
            .padding(.top, 15)
           ScrollView {
              Text(rules)
                .foregroundColor(.black)
                .padding(.horizontal, 16)
            }
         }
      }
   }
}
struct RulesView_Previews: PreviewProvider {
   static var previews: some View {
      RulesView()
   }
}

// CustomARView.swift
// Екран доповненої реальності
import Foundation
import SwiftUI
import UIKit
import RealityKit
struct RealityKitView: View {
   @Environment(\.dismiss) var dismiss
  var body: some View {
```

```
        ZStack {
            Color.mint.ignoresSafeArea(.all)
            VStack {
                HStack(alignment: .center) {
                    Image(systemName: "chevron.left")
                        .resizable()
                        .frame(width: 10, height: 20)
                        .onTapGesture {
                            dismiss()
                        }
                    Spacer()
                    Text("Tile generator")
                        .font(.title)
                        .fontWeight(.bold)
                        .foregroundColor(.black)
                    Spacer()
                }
                .padding(.top, 15)
                .padding(.horizontal, 16)
                ARViewContainer()
                    .padding(.top, 5)
            }
        }
    }
}
struct ARViewContainer: UIViewRepresentable {
    func makeUIView(context: Context) -> ARView {
        let arView = ARView(frame: .zero)
        let boxAnchor = try! Experience.loadScene()
        arView.scene.anchors.append(boxAnchor)
        return arView
```

```
    }
  func updateUIView(_ uiView: ARView, context: Context) { }
}

//PopoverView.swift
// Екран проходження рівня
import Foundation
import SwiftUI
struct PopoverView: View {
    @Binding var show: Bool
    @Binding var title: String
    @Binding var amountOfRooms: Int
    @Binding var currentRoom: Int
   let retreatCompletion: (() -> Void)?
    let continueCompletion: (() -> Void)?
    let finishDungeonCompletion: (() -> Void)?
   var body: some View {
        ZStack {
            Color.black.opacity(0.7)
                .allowsHitTesting(true)
                .ignoresSafeArea(.all)
                .edgesIgnoringSafeArea(.all)
            VStack(alignment: .center) {
                ZStack(alignment: Alignment(horizontal: .center, vertical: .center)) {
                    Rectangle()
                        .cornerRadius(8)
                        .foregroundColor(.mint)
                        .padding(.horizontal, 18)
                    VStack(spacing: 20) {
                        Text(title)
                            .foregroundColor(.black)
```

```
                    .font(.body)
                    .padding(.horizontal, 30)
                VStack(spacing: 16) {
                    if amountOfRooms == currentRoom {
                        MainButtonView(type: .fill, title: "Finish") {
                            show.toggle()
                            finishDungeonCompletion?()
                        }
                    } else {
                        HStack {
                            MainButtonView(type: .border, title: "Retreat") {
                                show.toggle()
                                retreatCompletion?()
                            }
                            MainButtonView(type: .fill, title: "Continue") {
                                continueCompletion?()
                            }
                        }
                    }
                }
                .padding(.horizontal, 25)
            }.padding(.top, 20)
        }
        .frame(height: 340, alignment: .center)
    }
  }
}
struct PopoverView_Preview: PreviewProvider {
  static var previews: some View {
```

```
                PopoverView(show: .constant(true), title: .constant("Monster 5 / 5"),
amountOfRooms: .constant(5), currentRoom: .constant(5),
                    retreatCompletion: {
            }, continueCompletion: {
            }, finishDungeonCompletion: {
            })
        }
    }

    //ContentView.swift
    // Екран гри
    import SwiftUI
    struct MainView: View {
       @Environment(\.dismiss) var dismiss
      @ObservedObject var viewModel = MainViewModel()
      @State var randomValue = ""
       @State var isShowAlert = false
      @State var isShowDungeonAlert = false
      @State var showTileScreen = false
      var body: some View {
          ZStack {
             VStack {
                HStack(alignment: .center) {
                   Image(systemName: "chevron.left")
                      .resizable()
                      .frame(width: 10, height: 20)
                      .onTapGesture {
                         dismiss()
                      }
                  Spacer()
                  Text("Main")
```

```
                .font(.title)
                .fontWeight(.bold)
                .foregroundColor(.black)
            Spacer()
        }
        .padding(.top, 15)
        ScrollView(showsIndicators: false) {
            VStack(spacing: 8) {
                ListItemView(title: "Get Random Event").onTapGesture {
    randomValue = viewModel.getRandomUnusedEvent()?.eventName ?? ""
                    isShowAlert.toggle()
                }
                ListItemView(title: "Get Random Monster").onTapGesture {
                                                        randomValue =
viewModel.getRandomUnusedMonster()?.monsterName ?? ""
                    isShowAlert.toggle()
                }
                ListItemView(title: "Get Random Trap").onTapGesture {
                    randomValue = viewModel.getRandomTrap()?.trapName ?? ""
                    isShowAlert.toggle()
                }
                ListItemView(title: "Start random dungeon").onTapGesture {
                    viewModel.startDungeon()
                    self.isShowDungeonAlert.toggle()
                }
                ListItemView(title: "Get Random Tile").onTapGesture {
                    self.showTileScreen.toggle()
                }
            }
        }
        .padding(.top, 15)
```

```
            MainButtonView(title: "Start new game") {
                viewModel.startNewGame()
            }
            .padding(.top, 20)
        }
        .padding(.horizontal, 16)
        if isShowDungeonAlert {
            PopoverView(show: $isShowDungeonAlert,
                        title: $viewModel.dungeonMessage,
                        amountOfRooms: $viewModel.dungeonSize,
                        currentRoom: $viewModel.dungeonIteration) {
                viewModel.finishDungeon()
            } continueCompletion: {
                viewModel.continueDungeon()
            } finishDungeonCompletion: {
                viewModel.finishDungeon()
            }
        }
    }
    .background(Color.mint)
    .navigate(to: RealityKitView(), when: $showTileScreen)
    .alert(randomValue, isPresented: $isShowAlert) {
        Button("Ok") {
            isShowAlert.toggle()
        }
    }
  }
}
struct ContentView_Previews: PreviewProvider {
  static var previews: some View {
    MainView()
```

```
    }
}

//MainViewModel.swift
// Модель головного екрану
import Foundation
enum DungeonResults {
    case monster
    case trap
    case event
    case treasure
}
class MainViewModel: ObservableObject {
    private var events: [ErEvent] = []
    private var monsters: [ErMonster] = []
    private var traps: [ErTrap] = []
   @Published var dungeonMessage: String = ""
    @Published var dungeonSize = 0
    @Published var dungeonIteration = 0
    private var dungeonRewards: [String] = []
    var dungeonEvents: [DungeonResults] = []
   init() {
        self.startNewGame()
    }
   func startNewGame() {
        self.events = Constants.events
        self.monsters = Constants.monsters
        self.traps = Constants.traps
    }
   func getRandomUnusedEvent() -> ErEvent? {
        let unusedEvents = events.filter { !$0.isUsed }
```

```
            guard !unusedEvents.isEmpty else {
                return nil
            }
            let randomIndex = Int(arc4random_uniform(UInt32(unusedEvents.count)))
            var randomEvent = unusedEvents[randomIndex]
            randomEvent.isUsed = true
            if let index = events.firstIndex(where: { $0.id == randomEvent.id }) {
                events[index] = randomEvent
            }
            print(events)
            return randomEvent
        }
        func getRandomUnusedMonster() -> ErMonster? {
            let unusedNonReusableMonsters = monsters.filter { !$0.isUsed }
            guard !unusedNonReusableMonsters.isEmpty else {
                return nil
            }
                                                    let randomIndex =
Int(arc4random_uniform(UInt32(unusedNonReusableMonsters.count)))
            var randomMonster = unusedNonReusableMonsters[randomIndex]
            if !randomMonster.isReusable {
                randomMonster.isUsed = true
              if let index = monsters.firstIndex(where: { $0.id == randomMonster.id}) {
                    monsters[index] = randomMonster
                }
            }
            return randomMonster
        }
        func startDungeon() {
            self.dungeonSize = Int.random(in: 2...5)
```

```
        self.dungeonIteration = 0
        self.dungeonMessage = "Number of rooms: \(self.dungeonSize)"
       if dungeonSize == 2 {
           self.dungeonEvents = [.monster, .event, .event, .event, .event, .treasure]
        } else if dungeonSize == 3 {
           self.dungeonEvents = [.trap, .monster, .event, .event, .event, .treasure]
        } else if dungeonSize == 4 {
           self.dungeonEvents = [.monster, .trap, .trap, .event, .event, .treasure]
        } else {
           self.dungeonEvents = [.monster, .monster, .trap, .event, .event, .treasure]
        }
    }
    func continueDungeon() {
        if self.dungeonSize == self.dungeonIteration {
           self.finishDungeon()
           return
        }
                                                    let randomIndex =
Int(arc4random_uniform(UInt32(self.dungeonEvents.count)))
        let randomElement = self.dungeonEvents[randomIndex]
        var result = ""
        switch randomElement {
        case .monster:
           result = self.getRandomUnusedMonster()?.monsterName ?? ""
        case .trap:
           result = self.getRandomTrap()?.trapName ?? ""
           self.replaceOccurrences(of: .trap)
        case .event:
           result = self.getRandomUnusedEvent()?.eventName ?? ""
           self.replaceOccurrences(of: .event)
        case .treasure:
```

```
            self.replaceOccurrences(of: .treasure)
        }
        self.dungeonIteration += 1
        self.dungeonMessage = result
        self.dungeonRewards.append(result)
       if self.dungeonIteration == self.dungeonSize {
            self.dungeonMessage = dungeonRewards.joined(separator: ", ")
        }
    }
   private func replaceOccurrences(of item: DungeonResults) {
        for index in 0..<dungeonEvents.count {
            if self.dungeonEvents[index] == item {
                self.dungeonEvents[index] = .monster
            }
        }
    }
   func finishDungeon() {
        self.dungeonSize = 0
        self.dungeonIteration = 0
        self.dungeonEvents = []
        self.dungeonRewards = []
        self.dungeonMessage = ""
    }
   func getRandomTrap() -> ErTrap? {
        return traps.randomElement()
    }
}

//MenuView.swift
//Головний екран додатку
import SwiftUI
```

```
struct MenuView: View {
    @State var startGame: Bool = false
    @State var showRules: Bool = false
    var body: some View {
        VStack {
            HStack {
                Text("Menu")
                    .font(.title)
                    .fontWeight(.bold)
                    .foregroundColor(.black)
                    .padding(.top, 24)
                Spacer()
            }
            ScrollView(showsIndicators: false) {
                VStack(spacing: 8) {
                    ListItemView(title: "Start game", image: "play.fill").onTapGesture {
                        startGame.toggle()
                    }
                                                        ListItemView(title: "Rules", image:
"questionmark.circle").onTapGesture {
                        showRules.toggle()
                    }
                }
            }
            .padding(.top, 20)
        }
        .padding(.horizontal, 16)
        .background(Color.mint)
        .navigate(to: MainView(), when: $startGame)
        .navigate(to: RulesView(), when: $showRules)
```

```
    }
}
struct MenuView_Previews: PreviewProvider {
    static var previews: some View {
        MenuView()
    }
}

//Models.swift
//Файл х моделями гри
import Foundation
struct ErEvent {
    let id = UUID()
   var eventName: String
   var isUsed: Bool
}
struct ErMonster {
    let id = UUID()
   var monsterName: String
   var isUsed: Bool
    var isReusable: Bool
}
struct ErTrap {
    let id = UUID()
   var trapName: String
}

//Constants.swift
//Дані гри
import Foundation
struct Constants {
```

```
    static let events: [ErEvent] = [
        …
    ]
    static let monsters: [ErMonster] = [
        …
    ]
    static let traps: [ErTrap] = [
        …
    ]
}

//MainButtonView.swift
//Представлення кнопки
import Foundation
import SwiftUI
enum MainButtonViewType {
    case fill
    case border
    var foregroundColor: Color {
        switch self {
        case .fill: return .init(.white)
        case .border: return .init(.black)
        }
    }
    var backgroundColor: Color {
        switch self {
        case .fill: return .init(.black)
        case .border: return .clear
        }
    }
}
```

```
struct MainButtonView: View {
  var type: MainButtonViewType = .fill
   var title: String
   var icon: Image?
   var action: () -> Void
  var body: some View {
      Button(action: action, label: {
         HStack(spacing: 10) {
            Spacer()
            Text(title)
               .frame(height: 46)
            if let icon {
               icon
            }
            Spacer()
         }
      }).buttonStyle(MainButtonStyle(type: type))
   }
}
private struct MainButtonStyle: ButtonStyle {
  let type: MainButtonViewType
   @Environment(\.isEnabled) private var isEnabled: Bool
  func makeBody(configuration: Self.Configuration) -> some View {
            let currentForegroundColor = !isEnabled || configuration.isPressed ?
type.foregroundColor.opacity(0.3) : type.foregroundColor
      return configuration.label
         .foregroundColor(currentForegroundColor)
                              .background(!isEnabled || configuration.isPressed ?
type.backgroundColor.opacity(0.3) : type.backgroundColor)
         .cornerRadius(24)
   }
```

```
}

//ListItemView.swift
//Представлення кнопки меню
import Foundation
import SwiftUI
struct ListItemView: View {
    @State var title: String
    @State var image: String? = nil
   var body: some View {
        HStack(spacing: 12) {
            ZStack {
                Rectangle()
                    .foregroundColor(.clear)
                    .frame(width: 68, height: 68)
                    .background(Color.clear)
                    .border(.black, width: 1, cornerRadius: 16)
                if image != nil {
                    Image(systemName: image!)
                        .foregroundColor(.black)
                }
            }.padding([.top, .bottom, .leading], 10)
            VStack(alignment: .leading, spacing: 4) {
                Text(title)
                    .foregroundColor(.black)
            }
            Spacer()
        }.overlay(
            RoundedRectangle(cornerRadius: 16)
                .stroke(Color.gray, lineWidth: 1)
        ).background(RoundedRectangle(cornerRadius: 16).fill(Color(.white)))
```

```
    }
}
//View+Ext.swift
//Розширення основного протоколу View
import Foundation
import SwiftUI
extension View {
    func border(_ color: Color, width: CGFloat, cornerRadius: CGFloat) -> some
View {
        overlay(RoundedRectangle(cornerRadius: cornerRadius).stroke(color,
lineWidth: width))
    }
    func navigate<NewView: View>(to view: NewView, when binding:
Binding<Bool>) -> some View {
        NavigationView {
            ZStack {
                self.navigationBarTitle("")
                NavigationLink(
                    destination: view
                        .navigationBarTitle("")
                        .navigationBarBackButtonHidden(),
                    isActive: binding
                ) {
                    EmptyView()
                }
                .isDetailLink(false)
            }
        }
        .navigationViewStyle(.stack)
    }
}
```