Міністерство освіти і науки України
Кам'янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп'ютерних наук

Дипломна робота
магістра

з теми: **«РОЗРОБКА МОБІЛЬНОГО КРОСПЛАТФОРМНОГО ІГРОВОГО ЗАСТОСУНКУ»**

Виконав: студент групи KN1-M21
спеціальності 122 Комп'ютерні науки
Олійник Микола Віталійович

Керівник:
Щирба В.С., кандидат фізико-математичних наук,
доцент кафедри комп'ютерних наук

Рецензент:
Сморжевський Ю.Л., кандидат педагогічних наук, доцент кафедри математики

Кам'янець-Подільський – 2022

ЗМІСТ

ВСТУП

Користуватися технікою щороку стає все легше, а кількість техніки яка допомагає та полегшує життя людям стає все більшою.

Потужній пристрій може дозволити собі будь яка людина. Із-за глобальної діджиталізації існуючі ринки різко збільшили свій дохід та спричинили розвиток нових. Користувачі нових технологій зростає що секунди. Гарними прикладом є Індія після реформ по зменшенню ціни за Інтернет у операторів зв'язку користувачів Інтернету збільшилось в декілька разів.

Мобільні пристрої на даний час є найбільш доступна платформа у всьому світі. Завдяки мобільним платформам люди почали економити час якій вони витрачали на рутину працю або бюрократичні справи наприклад оплата різних платежів.

Розробники комп'ютерних ігор одні із перших почали переносити свої проекти на мобільні платформи. Завдяки цьому почався розвиток всіх напрямків на мобільних платформ. Телефонні ігри в наш час призвели до того що люди хочуть скоротити умовну годину за захоплюючою грою в телефоні.

Метою створення дипломної роботи є розробка та введення в обіг мобільної гри за допомогою кросплатформеної технології Flutter яка дозволяє в зручний спосіб реалізувати програмні продукти.

Методи розробки базуються на інструментах розробки технології Flutter. Платформа Flutter є рівень абстракції, який забезпечує управління взаємодією між загальним кодом і кодом базової платформи. В результаті роботи розглянуто методи різні методи та технології розробки, обґрунтовано доцільність використання вибраної технології, описано предметну область. Розроблено серверну частину програми та клієнтську частину для середовища Android та Windows.

Об'єкт досліджень – процес розробки мобільних додатків для операційних систем iOS та Android та практичне використання засобу Flutter з метою підвищення ефективності розробки мобільних додатків.

Предмет досліджень – методи та підходи засобу Flutter у розробці програмного забезпечення для мобільних пристроїв.

Мета роботи – вдосконалення та пришвидшення процесу розробки якісних мобільних додатків під популярні мобільні операційні системи за допомогою засобу Flutter.

Досягнення поставленої мети передбачає вирішення таких *завдань*:

1. Здійснити аналіз існуючих методів та інструментів, які дозволяють займатися одночасною розробкою під Android та iOS.
2. Проаналізувати нове рішення від Google – фреймворк Flutter як засіб для розробки мобільних ігрових додатків.
3. Розробити мобільний кросплатформений ігровий застосунок.

Методи дослідження. Для розв'язання поставлених завдань використано: теорія системного аналізу, принципи об'єктно-орієнтованого програмування, принципи декларативного програмування.

Наукова новизна роботи:

Наукова новизна полягає у тому, що вдосконалено методи визначення ефективності мобільних додатків з використанням засобу Flutter, отримали подальший розвиток процеси ефективної розробки мобільних додатків за допомогою засобу Flutter та декларативного підходу до побудови мобільних *інтерфейсів.*

Структура та обсяг дипломної роботи. Дипломна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатку.

Повний обсяг дипломної роботи становить 65 сторінок, у тому числі: 54 сторінок основного тексту,25 рисунків, 2 таблиць, список використаних джерел із 17 найменувань та 1 додатку.

РОЗДІЛ 1. ТЕХНОЛОГІЇ РОЗРОБКИ КРОСПЛАТФОРМНИХ ДОДАТКІВ

1.1. Розробка кросплатформних додатків

FLUTTER досить молода платформа, яка приваблює розробників своєю простотою. Швидкість її роботи і висока продуктивність досягається за рахунок застосування декількох технік. По-перше, Flutter не використовує JavaScript, його творці вважали кращою мову програмування Dart, яка легко компілюється в двійковий код. Завдяки цьому швидкість виконання операцій нічим не поступається Swift, Kotlin, Java. Також платформа не використовує нативні компоненти, відмальовуючи інтерфейс у графічному движку в міру необхідності – тільки у випадку, якщо до нього внесено зміни.

В ОС Linux, iOS, Android, Windows Flutter працює за допомогою віртуальної машини Dart з JIT-компілятором. Одне з головних переваг цього SDK (software development kit) – реалізація функції «гарячого перезавантаження», завдяки чому зміна коду може бути застосована в уже запущеному додатку, і його перезавантаження не буде потрібно. Віджети Flutter оснащені вбудованими елементами – скролінг, навігація, шрифти та іконки. Код, написаний на Flutter, компілюється з використанням нативного компілятора Dart.

Xamarin – це кросплатформна технологія, частина платформи .NET, призначена для створення мобільних і веб-додатків. Основною ідеєю є сумісність служб, які написані на різних мовах програмування. На сьогоднішній день реалізована для платформ Windows, FreeBSD, а також в варіанті для ОС Linux (проєкт Mono). Розділяється на дві основні частини – це середовище виконання, свого роду віртуальна машина, а також інструменти розробника.

Як середовищ розробки виступають VisualStudio, C ++, C #, SharpDevelop. Як і Java, середа .NET створює байт-код, який виконується віртуальною машиною. Код створюється на мові CIL, common intermediate language. Використання байт-коду дозволяє реалізувати крос-платформеність

на рівні вже скомпільованого проєкту. Перед запуском байт-код перетворюється JIT-компілятором в машинний код.

REACT NATIVE – це платформа для розробки мобільних додатків, створена Facebook і має відкритий код. Дозволяє розробляти програми для iOS, Android, UWP і Web. React Native не використовує CSS або HTML дозволяє створювати код на мовах Swift і Objective-C для iOS, а також на Java для Android. React Native дозволяє створювати Кросплатформені додатки, компоненти платформи взаємодіють з власними API-інтерфейсами за допомогою декларативної парадигми інтерфейсу React і Java Script. Завдяки цьому стає можливим створювати додатки для груп розробників.

Компоненти React обертають існуючий власний код і взаємодіють з власними API-інтерфейсами через декларативну парадигму призначеного для користувача інтерфейсу React і JavaScript. Це дозволяє створювати власні додатки для цілих нових груп розробників і дозволяє існуючим власним командам працювати набагато швидше. В основному принципи роботи React Native ідентичні таким у React, але, на відміну від останнього, він працює у фоновому режимі на кінцевому пристрої, інтерпретуючи код, написаний на JavaScript.

IONIC – це багатоплатформовий SDK, або software development kit, з повністю відкритим кодом, який використовує фреймворк Cordova і плагіни Capacitor для розробки мобільних додатків. Користувачі можуть створювати додатки і налаштовувати їх для роботи з операційними системами Windows, iOS, Android, а також з сучасними браузерами. Ionic надає призначені для користувача компоненти і засоби для взаємодії з цими компонентами – наприклад, такі, як віртуальна прокрутка, вкладки, навігація, типографіка і т.д.

Також Ionic пропонує інтерфейс командного рядка і сервіси для вирішення інших завдань, наприклад, розгортання коду і автоматичного складання. Включає також і власну інтегровану середу розробки (IDE) – Ionic Studio. Розробник може підключати додаткові модулі фреймворка Cordova,

включати push-повідомлення, створювати значки додатків і навіть заставки до додатків.

ELECTRON, або ATOM SHELL – це фреймворк, розроблений GitHub. Він дозволяє вести розробку графічних додатків для операційних систем настільних комп'ютерів за допомогою веб-технологій. Фреймворк також включає Node.js, який дозволяє працювати з backend, і бібліотеку рендеринга з Chromium. Electron дозволяє створювати графічні додатки, використовуючи браузерні технології і розширюючи їх функціональність за допомогою системи доповнень. На відміну від веб-додатків, програми, розроблені на платформі Electron, являють собою виконувані файли без прив'язки до браузеру. При цьому переносити розроблений додаток для різних платформ не треба.

FRAMEWORK 7 – це безкоштовний фреймворк з відкритим кодом для розробки мобільних і веб-додатків. Також може бути використаний як інструмент для прототипування мобільних додатків. Багатоплатформовий фреймворк дозволяє створювати веб додатки для операційних систем iOS і Android.

Працюючи з Framework7, ви можете використовувати будь-які інструменти – створювати додатки на ньому так само просто, як і веб-сайти. Фреймворк поставляється з такими компонентами, як React, Vue.js і Svelte, забезпечуючи синтаксис, і структуровану базу даних. Також в нього вбудований набір готових до використання віджетів і елементів. Движок шаблонів Template7, підтримка стилів, в тому числі material design, великий список компонентів і різні свайпи – все це робить Framework7 відмінним кросплатформним рішенням на всі випадки життя.

Більшість IT компаній починають свій бізнес в інтернеті зі створення сайту. Надалі ресурс адаптується для роботи з мобільними пристроями, і при позитивній динаміці відвідуваності приймається рішення про створення програми для Android і iOS.

Додаток для мобільних пристроїв більш функціонально, в порівнянні з сайтом, має зручний для користувача інтерфейс і можливість роботи навіть без підключення до інтернету (з обмеженим функціоналом). На мобільному ринку сформувалося абсолютне домінування двох операційних систем. Це iOS і Android. Створюючи мобільні додатки для цих та інших ОС, основна увага приділяється таким питанням:

- ✓ Висока швидкість розробки;
- ✓ Надійність, стабільність у роботі;
- ✓ Простота в підтримці і випуск оновлень;
- ✓ Максимально ефективне використання можливостей платформи.

Залежно від цільової аудиторії і поставлених цілей розробка мобільних додатків для iOS і Android здійснюється з використанням нативних або кросплатформних технологій. У сучасних IT компаніях ви можете замовити створення, як нативного, так і кросплатформного додатків.

НАТИВНІ ДОДАТКИ

Кожна платформа має свої нативні мови, «рідні» для цієї операційної системи. Для iOS це SWIFT або Objective-C. Для системи Android нативними будуть мови Kotlin або Java.

Для звичайного користувача різниці практично немає – нативний або кросплатформний додаток буде встановлено на його пристрої. Але при більш ретельному вивченні роботи різних додатків можна помітити, що нативні рішення, написані для конкретної операційної системи, будуть більш зручними, мати інтуїтивно зрозумілий інтерфейс, і працювати будуть швидше. Відбувається це тому, що при створенні нативного додатка зв'язка UI / UX дизайнерів і програмістів більш ефективна.

Дизайнеру точно відомо, з якими UI рішеннями звикли працювати користувачі даної операційної системи (наприклад – кнопки «Back» і нижній Tab Bar для пристроїв під iOS). Точно також програміст буде розуміти, як краще реалізувати ту чи іншу UI особливість для мобільного пристрою під керуванням конкретної операційної системи.

В результаті користувач, запускаючи нативний додаток, інтуїтивно розуміє, як з ним взаємодіяти, навіть не вивчаючи новий для нього інтерфейс.

КРОСПЛАТФОРМЕНІ ДОДАТКИ

Розробка додатків для Android і iOS, якщо створювати окреме рішення для кожної операційної системи, збільшує час і вартість проєкту. З урахуванням великої кількості версій і різновидів операційних систем (в тому числі і малопоширених), писати окремий код під кожну платформу складно і недоцільно. Подібна ситуація і стала причиною появи кросплатформних додатків для мобільних пристроїв. Якщо нативні додатки створюються під конкретну операційну систему, то при написанні коду для кросплатформного рішення є можливість адаптувати ПЗ під будь-яку ОС.

Багатоплатформовий підхід використовує той факт, що розробка мобільних додатків під Android iOS ведеться на мовах розмітки і стилів. Це JavaScript, CSS і HTML, які використовуються при створенні сайтів. Такий підхід виправданий, оскільки в результаті більшість контенту представлено у вигляді HTML сторінок. Додатки такого типу пишуться і підходять практично для всіх існуючих мобільних гаджетів, оскільки в їх основі покладено принцип роботи браузера.

Незважаючи на те, що додатки називаються кросплатформними, один і той же виконуючий файл не можна запустити на мобільних пристроях під управлінням різних ОС. Припустимо, створений кросплатформний додаток скомпільовано для роботи в середовищі Android. Його виконуючий файл «* .apk» не вийде запустити на пристрої, що працює під управлінням iOS.

Кросплатформна розробка додатків для iOS і Android дозволяє значно оптимізувати процеси розробки. Будь-який кросплатформний додаток може бути з мінімальними зусиллями скомпільовано для різних платформ. І в результаті будуть отримані різні виконувані файли. Так, в рішенні для iOS виконуваний файл отримає розширення «* .ipa», а додаток для пристроїв під управлінням Android буде запускати додаток з файлу «* .apk».

ПЛЮСИ І МІНУСИ НАТИВНИХ РІШЕНЬ

До переваг нативних додатків відноситься:

- ✓ Продуктивність (використовувані технології мають прямі зв'язки з платформою, що підвищує швидкість і стабільність роботи програми).
- ✓ Ефективність (нативні додатки створюються для вирішення конкретних завдань в певному середовищі, що забезпечує кращу відповідність можливостей ПЗ з апаратними можливостями пристроїв).
- ✓ Більш зручний для користувача інтерфейс (досягається завдяки прямій інтеграції додатків з операційною платформою).
- ✓ Краще ранжування в магазинах додатків.

До недоліків нативних рішень відноситься:

- ✓ Збільшення часу на розробку
- ✓ Необхідний великий бюджет
- ✓ Несумісність з іншими мобільними платформами

В зв'язку з вибором однієї ОС цільова аудиторія скорочується до користувачів однієї мобільної платформи.

ПЕРЕВАГИ ТА НЕДОЛІКИ КРОСПЛАТФОРМНИХ РІШЕНЬ

Плюси кросплатформних додатків:

- ✓ Економія бюджету на створення мобільного застосування (можливість використання одного технологічного стека одночасно на всіх платформах).
- ✓ Швидкість і простота розгортання (розробникам кросплатформних додатків не доводиться витрачати час на вивчення кількох технологічних стеків для кожної платформи, вони працюють з одним універсальним стеком).
- ✓ Використання однакового інтерфейсу і UX (для просування мобільного додатка важливий як дизайн (UI), так і відчуття користувачів (UX), і розроблене однією командою рішення для всіх платформ забезпечує однаковий зовнішній вигляд і інтерфейс для кожного пристрою).

Мінуси кросплатформних рішень:

- ✓ Зниження гнучкості (зі своїми завданнями додаток впорається на будь-якій платформі, проте його адаптація для максимально ефективного використання можливостей кожної операційної системи буде проблемною).
- ✓ Зниження продуктивності.
- ✓ Є можливість невідповідності UI на різних платформах.
- ✓ Можливі проблеми з відправкою кросплатформних рішень в магазини додатків.

Вибір між нативним і кросплатформним додатком залежить від стратегії його просування та покладених функцій. Якщо потрібно відразу охопити максимальну аудиторію, а складні завдання перед мобільним додатком не стоять, то логічніше буде скористатися кросплатформним підходом. Це буде набагато швидше і дешевше ніж окрема розробка додатків для iOS і Android, ціна яких буде вдвічі більше. Якщо ви сумніваєтеся у виборі, ми допоможемо його зробити, оцінивши ситуацію на ринку, сам продукт, цільову аудиторію і т.д.

Створення нативного додатка виправдано, якщо його реалізація передбачає максимально використовувати можливості кожної платформи, і на першому етапі немає необхідності присутності відразу і в Google Play Store, і в Apple App Store.

Можна стартувати на одній платформі, і при досягненні успіху приступати до розширення ринку. На першому етапі буде логічніше створення додатків для Android, ціна яких в порівнянні з рішеннями для iOS нижче, а охоплення аудиторії – більше.

1.2. Огляд ринка мобільних ігор

На кінець 2020 року світовий ринок всіх ігрових додатків оцінюється у 159,3 мільярда доларів, причому 48% від цього 77,2 мільярда доларів находиться в мобільних іграх. Для порівняння: в минулих роках обсяги ігрового ринку були менше ніж 152 млрд. доларів, в музичній індустрії заробили 58 млрд. доларів, а касові збори в кінотеатрах склали 42,5 млрд.

доларів. Ігровий ринок, перевищує ринок звукозапису і кіноринок більше ніж на 50 млрд. доларів.

Існує декілька чинників, що призвели до успіху. Бізнес-модель значно змінилася за останні декілька років. Через цього споживачі купують менше різних додатків, але проводять в них більше часу ніж раніше. Щоб використовувати всі можливості, розробники ігор перейшли від старої моделі разової покупки товару до нової моделі мікротранзакції. Деякі з надпопулярних можливостей для монетизації в різноманітних іграх містять продаж різноманітних персонажів, зброї, пакетів розширення контенту, ящиків з різноманітними випадковими товарами та безлічі інших предметів для більш комфортної гри.

Ігрова індустрія буда довгий час головним рушієм сегмента мобільних додатків на планеті, але пандемія корона вірусу допомогла вийти на небувалий рівень: у порівнянні з 2019 роком кількість установок додатків у 2020 збільшилася на 47%. Це на 41% вище, ніж показники зростання минулого року. На цю мить часу користувачі мобільних платформ які грають в мобільні ігри більше ніж 50%, що зрівняло ігрову категорію додатків по популярності з музичною.

В США час, витрачений людьми на роботу з мобільними пристроями офіційно перевищив час витрачений на перегляд телебачення.

Ігри на мобільній платформі мають головну перевагу перед іншими платформами оскільки ви маєте можливість грати будь-де. Від простих повсякденних ігор до приголомшливих драйвових новинок, які швидко набирають популярність легко граються та підходять для коротких сесій в вільний час протягом цілого дня, в ігри грає майже кожен демографічний прошарок людства. Середній вік мобільних геймерів становить 36,6(у порівнянні з 28,1 у 2014), гендерний розкол становить 53,6% жінок, 46,4% чоловіків, а третина всіх гравців у віці 32-50 років – це далеко від традиційних стереотипів.

В США з 2011 ігри визнані я окремий вид мистецтв.

У 2021 році мобільний ринок збільшиться у зв’язку з розвитку нових технологій і тенденцій.

1. З розповсюдженням 5G почався бурхливий розвиток хмарних ігор

2020 рік став роком 5G, що призвело до початку глобального впровадження в всіх країнах світу даної технології не дивлячись на ситуацію з пандемією COVID 19. Комерційне впровадження 5G почалося у 2020, а широкомасштабні рухи продовжаться до 2025 року.

2. Хмарові сервіси дадуть більший можливостей для користувачів мобільних платформ для гри в мобільні ігри

У 2020 році компанії Google і Microsoft випустили свої нові хмарні ігрові рішення. Віцепрезидент з маркетингу в компанії IronSource, описує це: «В цілому малоймовірно, що за короткий час хмарні гри замінять консолі й ПК або будь-яким чином вплинуть на аудиторію казуальних гравців що може перетворити їх на фанатів хмарних рішень. З мого досвіду, всі нові формати та технології які з’являються на ринку мобільних ігор не перевернуть весь ринок кардинально, але створять декілька нових прошарків. Розширюючи діапазон можливого ігрового досвіду нових та старих геймерів і створюючи більший доступ людям до традиційних ігор, хмарні рішення, ймовірно внесуть значний розвиток всього ринку ігор».

3. Казуальні геймери не будуть купувати щомісячну підпуску на хмарові сервіси, що не скажеш про Хард-корних геймерів. Компанії Apple і Google на весь світ оголосили про свої нові сервіси, які працюють по новій системі платних підписок. Компанії Apple і Google пропонують гравцям підписку на нові ігрові сервіси – змінюючи свій дохід від ігрових покупок і реклами в додатках на щомісячну плату з гравця. Проте, не дивлячись на сформовану тенденцію монетизації компаніями. Ми все ще маємо безліч невідомих факторів для повноцінного аналізу та прогнозів на майбутнє: ні компанія Apple, ні компанія Google, ні сторонні розробники, що працюють з

компаніями, не пояснили повноцінну як буде працювати бізнес-модель даного напрямку в ігровій сфері.

1.3. Огляд ігор та ігрових платформ

У наш час кожен гравець може найти собі ігову платформу яка задовільнить його гаманець та смак. У цьому розділя я представлю багато ігрових класифакацій та платформ заради кращого розуміння їх ознак та параметрів.

1.3.1. Категорії

Ігрові категорії на різних ігрових платформах має різні функції для пошуку. Загалом на усіх платформах категорії ігор мають спільну назву та однакове призначення для гравців. Більшість ігор в наш час мають декілька категорій, але це не значить що ігри лише з однією категорію не можуть бути популярними.

Казуальні.[1] Ігри орієнтовані на більш широку та просту аудиторію гравців, на відмінно від дуже специфічної категорії «хард-корних» ігор. Можуть набувати властивості та принципи будь-якої з категорій ігор, казуальні ігри є «загальною» категорією відеоігор. Для казуальних ігор характерна наявність декількох простих правил для гри, значно коротший термін ігрових сесій, та вони не потребують від гравця наявності геймерських навичок чи знань.

Action[2] (з англ. - «Дія»). Ігри які наповнені великою кількістю ігрових подій та сцен, що потребують більш активної участі гравця в прийнятті ігрових рішень. Представники цієї категорії є однією з найкраших категорії ігор на мобільних платформах.

Стратегії.[3] Ця категорія ігор, у яких невизначеність ігового процесу і

часто автономні навички прийняття ігрових подій, але гравці завжди приймоють головні рішень, що мають високе значення при визначенні кінцевого результату. Практично всі стратегічні ігри вимагають від гравця внутрішнього мислення і навичок для високої ситуативної обізнаності.

Спорт.[4] Категорія спортивних ігор які на мобільних платформах зв’язані з такими жанрами як «симулятори» та «стратегії». Це ігри у яких симулюється гра у різні види спору. Більшість спортивних напрямків мають декілька різних варіантів ігор на їх базі. Спортивні ігри поділяються на декілька: деякі категорії частіше фокусуються на самому виді ігрововo спорту та грі (наприклад «NBA 2K21») у той час, як інші (наприклад «Championship Manager 2020») яка більше звертає увагу на стратегічні сторони спортивних реалій (тренування гравців, менеджмент клубу, сфера загалом). Особливістю всіх спортивних ігор є те, що користувач у них може зустріти реальні назви та імена професіональних спортивних ліг, гравців, ігрових клубів та інших. Також спортивні ігри мають тенденцію що року оновлювати свою гру, щоб зображати сучасний стан команд, та заробити на цьому велику суму грошей.

Це одна з найстаріших категорій ігор на всіх платформах світу.

Слова. Категорія ігор в основі яких закладена постійна гра зі словами.

Кросворди та ребуси, загадки, амбіграма - все це словесні ігри.

Симулятори. Категорія ігор яка є одна з найбільш популярно оскільки створення таких ігор є найбільш простою та кількість можливих варіантів є нескінченною. Категорія активно використовують розробниками в інших жанрах.

Сімейні. Для категорії сімейних ігор більш характерна наявність різних режимів для гри на одному пристрої для декількох користувачів.

Рольові.[5] Рольова гра («role-playing game» або скорочений варіант «RPG») – це категорія ігор, в якій користувачі беруть на себе роль ігрових персонажів у вигаданому ігровому всесвіті. Користувачі беруть на себе відповідальність за персонажа гри виконання ролей в рамках розповіді, або через буквальну дію персонажа, або через процес постійного прийняття рішень щодо розвитку свого ігрового персонажа. Дії, здійснені в рамках багатьох сесій ігор, приводять різних варіантів кінця гри за формальною

системою правил та наказів. Чудовим прикладом категорій є гра про Карла «Beholder».

Настільні. Настільна категорія ігор приваблює гравця в основному класичні версії наявний старих настільних ігор (нарди, аліас, монополія та інші) та оригінальні настільні ігри. Для цієї категорії ігор характерна повільність розвитку гри як і для стратегічних ігор.

Музика. Є однією з найскладніших категорій ігор та водночас креативним видом категорій на даний час. Музика є ядро всього ігрового процесу таких ігор. Світ ігор цієї категорії реагує лише на музику та спонукає гравця до дій. Гарний представник категорії є гра на основі музики та ритму «Beat Blade».

Карткові. Категорія завжди були пов'язана з категорією азартних ігор різнив варіантів Казино. Загалом - це порятовані версії різних класичних карткових ігор (наприклад «Покер»), або ігри з інших категорій однією з ключових механік яких є маніпуляції з карти. Представники категорії - «Горміти», «Pokemon», «Hearthstone».

1.3.2. Операційні системи

Ігри можна також систематизувати за видами та поколіннями підтримуваних операційних системам. З точки зору корпорацій для збільшення своїх постійних доходів та клієнтської бази гравців - вигідно підтримувати декілька різних видів та версій операційних систем. Питання розробника завжди полягає чи вигідно це робити? Чи принесе постійна або тимчасова підтримка, наприклад декількох не дуже популярних систем для ігор, чи призведе це до постійного збільшення нових гравців чи ні? Тому більшість компаній розробників підтримують обмежену кількість ігрових операційних систем, що дозволяє зменшити основних перечень ОС, що є цільовими на ринку або навіть обов'язковими платформами для максимального розповсюдження гри фанатів та гравців. Підтримувані Операційні системи існують трьох різних формфакторів: комп'ютерні системи, консольні системи та мобільні системи. До мобільних ОС належать

основні три вида – iOS, Android, та Windows Phone Також існує менш популярна Blackberry яка набула популярності у США, але загалом всі мобільні ОС є пригідними для експлуатації користувачем і використанням для проводженні часу у іграх. До комп'ютерних належать декілька гігантів світового масштабу Windows, MacOS та Linux з відкритим ОС. До консольних операційних систем призначенні Xbox OS, Orbis OS (Playstation OS), NintendoOS (Nintendo Switch OS).

IOS[6] (раніше була назва iPhone OS) компанія Apple розробила виключно для своїх мобільних гаджетів операційну систему. На даний момент всі мобільні пристрої та гаджети компанії працювали виключно на IOS, але у

2020 році всі пристрої iPad перейшли на нову операційну систему iPadOs. Лише операційна система Android більш популярна у світі мобільних пристроїв. IOS став основою для розробки всі інших операційних систем компанією Apple.

Android[7] - це мобільна операційна система, створена на основі модифікації доступної версії ядра Linux та різного програмного забезпечення доступного в Інтернеті з відкритим кодом, створеного основним чином для

всіх видів мобільних пристроїв на ринку із сенсорним екраном, таких як смартфони. Android розробляється об'єднанням різних розробників, відомим як Альянс відкритих телефонів і на постійній основі спонсорується компанією Google. Операційну систему оприлюднили у 2007 році, а перший комерційний пристрій на навій системі Android був запущений весною 2008 року.

Microsoft Windows[8] яку користувачі зазвичай називають Windows, - це група декількох своїх сімейств графічних операційних систем реального часу, усі вони розроблені компанією, або їх права належать Microsoft. Кожна сім'я в цій системі обслуговує певний сектор обчислювальної галузі. До активних сімей Microsoft Windows належать сім'я Windows NT та сім'я Windows IoT; вони можуть вливати на різні під сімейства, наприклад

Windows Server або Windows Embedded Compact (Windows CE). До уявних сімей Microsoft Windows належать такі системи як: Windows 9x, Windows Mobile та Windows Phone.

Лінукс[9] (англ. Linux, повна назва — GNU/Linux) — основна назва UNIX подібних операційних систем на базі однойменного ядра. Це один із перших прикладів створення вільного (free) та відкритого (з відкритим кодом, open source) програмного забезпечення (software). На відміну від власницьких операційних систем (на кшталт Microsoft Windows та MacOS X), їхні вихідні коди доступні всім для використання, зміни та поширення абсолютно вільно (в тому числі безкоштовно).

MacOS - POSIX сумісна операційна система створена корпорацією Apple. Є наслідувач Mac OS 9 — кінцевого релізу компанії «класичної» Mac OS — основною операційною системою створеною корпорацією Apple з 1984 року. OS X входить в нове сімейство операційних систем Apple OS X, до якого також належить інша ОС для мобільних пристроїв — Apple iOS. У macOS використовується основне ядро Darwin, засноване на базі мікроядра Mach, що містить код, написаний розробниками Apple та код, отриманий з операційних систем NeXTSTEP та FreeBSD. Apple macOS використовується для комп'ютерів Macintosh (Макінтош) на базі процесорів PowerPC та Intel (починаючи з версії 10.6,) macOS підтримує тільки комп'ютерні пристрої Mac на базі процесора Intel. Mac OS — стала другою за популярністю у світі операційна система.

1.3.3. Режими

Іноді розробники ігор класифікують свою гру за наявними режимами або ж за підключенням гри до мережі Інтернет. Понад 5 років тому нові ігрові проєкти завжди мали офлайн режими, тобто не були залежні від підключення до інтернету - то наданий момент ринок ігор збільшив свої обсяги так все частіше можна побачити серйозні та дорогі ігрові продукти, що постійно потребують підключення до інтернету для гри, або повністю стали онлайн проєктами.

Що відноситися до режимів в ігрових проєктах - то можна акцентувати увагу на три основні режими гри: одиничне проходження, кооперативне проходження з декількома гравцями та безумовно мультіплєєрний режим з можливістю грати на різних онлайн серверах. Для одиничний режим гри характерна повна відсутність інтернет зв'язку – кооперативне продження та мультиплеєрний режими в нових іграх є лише онлайн режимами гри. Не дивлячись на те, що є комбіновані рішення для ігор, іноді гравці можуть навіть зустріти одиночні ігри які потребують наявність мережі інтернет.

1.3.4. Ігрові платформи

Ігрові платформи, завдяки яким гравець може грати у відеоігри, розвиваються кожен день. Від звичайних ігрових автоматів в торгових центрах до продвинутих аркадних машин та великих комп'ютерних клубів. При цій кількості різних ігрових платформ існують 3 основних лідери. Перша ігрова платформа це ПК, друга ігрова платформа це мобільні телефони, а третя ігрова платформа це портативні консолі.

Персональні комп'ютери (**ПК**) - це одна із найбільш функціональна платформа для гравця який бажає зіграти в відео ігри, доступних на даний момент часу. Вони ймовірно будуть двох типів: настільні ПК або зручні ноутбуки. Ринок може запропонувати користувачеві будь який комп'ютер який буде вкладатися у бюджет людини. Обладнання комп'ютера повинно містити в собі потужну оперативну пам'ять на диску, а також новеньку відеокарту для обробки графічних даних, яка має назву GPU. Потужні GPU допомагають комп'ютеру отримання чудові результату. Разом з усіма можливими поширеними апаратними засобами для роботи ПК, такими як працюючий монітор, наявність клавіатури та миші, можна легко отримувати задоволення від ігор. Персональні комп'ютери іноді налаштовують на можливість використати додаткові пристрої, такі як ручний джойстик, щоб покращити управління в грі. Свій ігровий досвід можна збільшити завдяки підключенню комп'ютера до широкоформатного екрану телевізора завдяки наявності кабелю типу HDMI або VGA, які можуть бути підключенні до

телевізора. Завдяки усі можливостям перерахованих персональні комп'ютери можуть бути неймовірною ігровою платформою для кожного гравця.

Консолі - це гаджети, створені спеціально для отримання максимальних емоцій від гри у відеоігри. В більшості випадків вони продаються гравцю з пристроями для гри, такими як джойстик та головний блок, який проводить всі процеси обробки даних. Як правило їх підєднують до екранів домашніх телевізорів чи комп'ютерних моніторів, щоб побачити візуальну картинку від ігрової консолі. На даний момент часу ринку присутні декілька основних ігрових типів консолей. Популярними є, наприклад, домашня ігрова приставка **Xbox**[10] (до платформи xbox можна віднести всі ігрові консольні продукти компанії Microsoft, а саме: Xbox, Xbox Series, та інші. Ігрові консолі платформи Xbox є більш популярними у США та Канаді у порівнянні з Європейськими країнами), **Wii** та **PlayStation**[11] (до ігрової платформи playstation можна віднести всі консольні продукти компанії Sony, а саме: Playstation 1, Playstation 4, та інші.). Також є портативні консолі, такі як **Nintendo Switch**[12] (до платформи Nintendo відносяться всі консольні продукти компанії, а саме: Game&Watch, Nintendo 3DS, та інші). Ці портативні приставки мають невеликі розміри що дозволяє носити його користувачу будь де. Така різноманітність та простота використання призвела до того, що консолі одна з найпопулярніших платформ у світі.

Мобільна (всі мобільні девайси та гаджети незалежно від виробника рахуються як мобільні платформи. Це телефони, планшети, смартгодинник основних сімейств iOS, Android, Windows Phone, Blackberry)

1.4. Основні завдання

Моїм основним завданням в даній дипломної роботи є проектування та створення мобільного додатку гри під платформу android (зважуючи усі аспекти розробки додатку гри, від програмування, до дизайну гри та інш.).

Висновок до розділу 1

В першому розділі моєї дипломної роботи ми пройшли ринкові аспекти ігрової індустрії, класифікацію ігор та порівняння найпопулярніших видів ігрових платформ Мобільні ігри значно розвинулися завдяки розвитку нових технологій таких які інтернет та постійному покрашенню розробки мобільних платформ. Телефон більше не являється тільки засобом зв’язку між людьми на даний момент часу це така ж основна річ що людина не уявляє можливість проводження свого вільного часу. На даний момент мобільні технології та різні 'види мобільних ігор є найбільш перспективною областю нових технологій. Через це розробка мобільної гри є актуальним завданням.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ ДЛЯ РОЗРОБКИ МОБІЛЬНОЇ ГРИ

2.1. Загальні відомості про ігрові рушії

Зараз мобільні ігри створюються на базі вибраним розробником GE. Нам для початку потрібно зрозуміти що собою уявляють GE перед тим я переходити до порівняння різних видів.

Ігровий рушій - це програма, яка виконує керування технічною стороною гри яка буда створена на базі цього рушія, що дозволяє розробнику зменшити кількість кроків для розробки гри шляхом стандартизації та систематизації її базової структури. На даний момент часу, головним значенням для даних ігрових рушіїв являється можливість створення кросплатформних ігор (наприклад розробка гри може відбуватися одночасно для ПК, Android, PS4 та Xbox One. Увесь можливий контент, що який використовується розробником під час створення своїх ігор (музика, графічний дизайн, анімації й т.д.) розробники ігор називають ассетами (англ. Assets). Головну функціональність будь якої гри як правило забезпечує ігровий рушій, який складається з декількох головних рушіїв:

1. Графічний рушій рендерингу – головною задачею якого є представлення двовимірного простору або тривимірного простору за допомогою комп’ютерної графіки;
2. Рушій фізичних законів – підсистема рушія, яка відповідає за достовірну симуляцію фізичних законів нашого світу в віртуальний вимір;
3. Звук гри, система закодованих скрипів, анімація персонажів;
4. ігровий штучний інтелект – головний набір методик, які дозволяють розробнику створити ілюзію інтелекту в поведінці ігорових персонажів, що контролюються комп’ютером;
5. мережевий код – частина рушія, за допомогою яких розробник може створювати онлайн гру. Цей код відповідає за взаємодію гравця та сервер між собою;
6. багато потоковість даних;

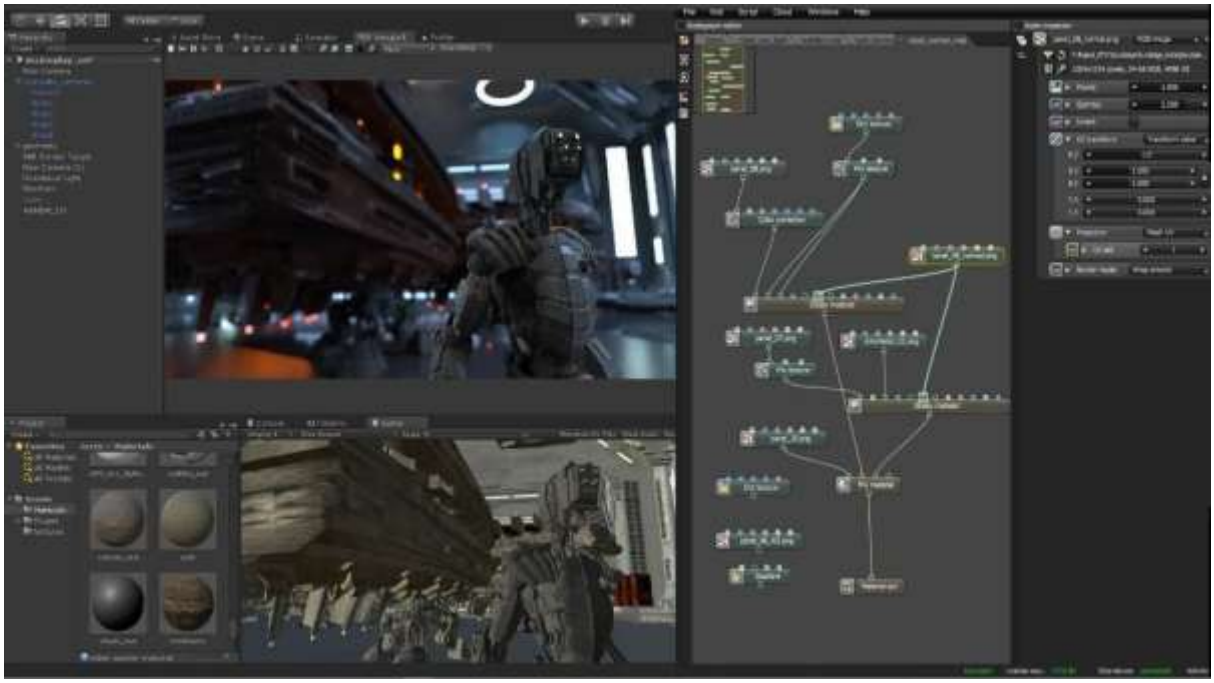
7. граф сцени – це структура даних, що використовується розробником для роботи з векторною графікою;

2.2. Приклади сучасних GE, їх плюси та мінуси.

На даний момент існує безліч різних GE. В пункті 2.2 я розберу найбільш відомі світові рушії.

2.2.1. Unity

Unity – це не лише простий GE, а й кросплатформне багато функціональне середовище розробки ігор, що було створенна американською компанією Unity Technologies. Релізу ігрового рушія прийшов на 2005 рік, і з тих пір цей рушій постійно розвивається. Технологія рушія Unity дозволяє створювати різні ігри та програми, що будуть підтримуватися приблизно на 25 різних ігрових платформах, найбільш явними лідерами є комп'ютери, ігрові консолі та мобільні телефони. GE виступає в ролі повноцінного середовища для створення мобільних ігор, тому що ігровий рушій об'єднав різні програмні засоби, що застосовується при розробці ПО – текстовий редактор, отладчік, дебагер і т.д. Редактор рушія Unity має простий та



доступний інтерфейс для користувача, що підтримує технологію Drag & Drop.[13]

Рисунок 2.1 – Інтерфейс редактора Unity

Для написання будь яких скриптів в Unity можна використати мову програмування C#, при цьому до версії 2017.1 ігровий рушій підтримував модифікацію язика програмування UnityScript, а також рушій підтримував мову програмування Boo, але лише до 5 версії ігрового рушія. На даний момент на Unity написали та розробили безліч додатків, що можуть охопити різних гравців, платформ та жанрів. Unity користується популярністю не тільки у інді розробників, але й в професійних компаніях.

Unity має декілька головних плюсів:

- В редакторі рушія застосовується компонентно-орієнтований підхід завдяки цьому розробник може створювати об’єкти і додати в них різні компоненти рушія.
- Легкий для використання Drag and Drop інтерфейс графічного редактора дає право малювати різні види карт та ставити об’єкти на них в режимі реального часу з можливість тестування результату.
- Наявність великої кількості бібліотек асетів та плагінів, що допомагає інді розробника спростити процес розробки гри.
- Можливість розробника гри в один клік змінити платформу.

Що ж до мінусів, то вони наступні:

- Складність при створенні багатокомпонентних схемам і ускладнення процесу розробки при підключенні до проєкту сторонні бібліотеки.
- При створенні складних сцен з великою кількістю компонентів рушій негативно вливає на оптимізацію гри, що може спричини проблеми розробнику гри.
- Ігри, створені на базі рушія Unity завжди займають багато місця на різних платформах. Навіть найпростіша гра створена на Unity може займати велику кількість місця на диску. І якщо для комп’ютера це не критичний об’єм пам’яті, то для мобільних носіїв може не вистачити вільного місця.

В кінці хотілось би розповісти про ліцензію. Unity доступний безкоштовний рушій, що дає право інді розробникам створювати ігри на готовому GE. Але є декілька важливих нюансів в безкоштовній версії рушія існують деякі невеликих обмежень, скажімо, що перед запуском вашої гри створеною на цьому рушії буде демонструватись логотип компанії Unity та дохід від вашої гри не повинен бути більше ніж 100 тис. доларів в рік. В випадку коли сума перевищує ліміт розробник гри мусить придбати професійну версію GE, що буде коштувати розробнику 125 доларів за місяць або 1800 доларів за одноразовий платіж.

Нижче ми бачимо приклади ігор розроблених на рушії Unity.

Рисунок 2.2 – Приклад гри на Unity. Rust

Рисунок 2.3 – Приклад гри на Unity. Ori and the Blind Forest

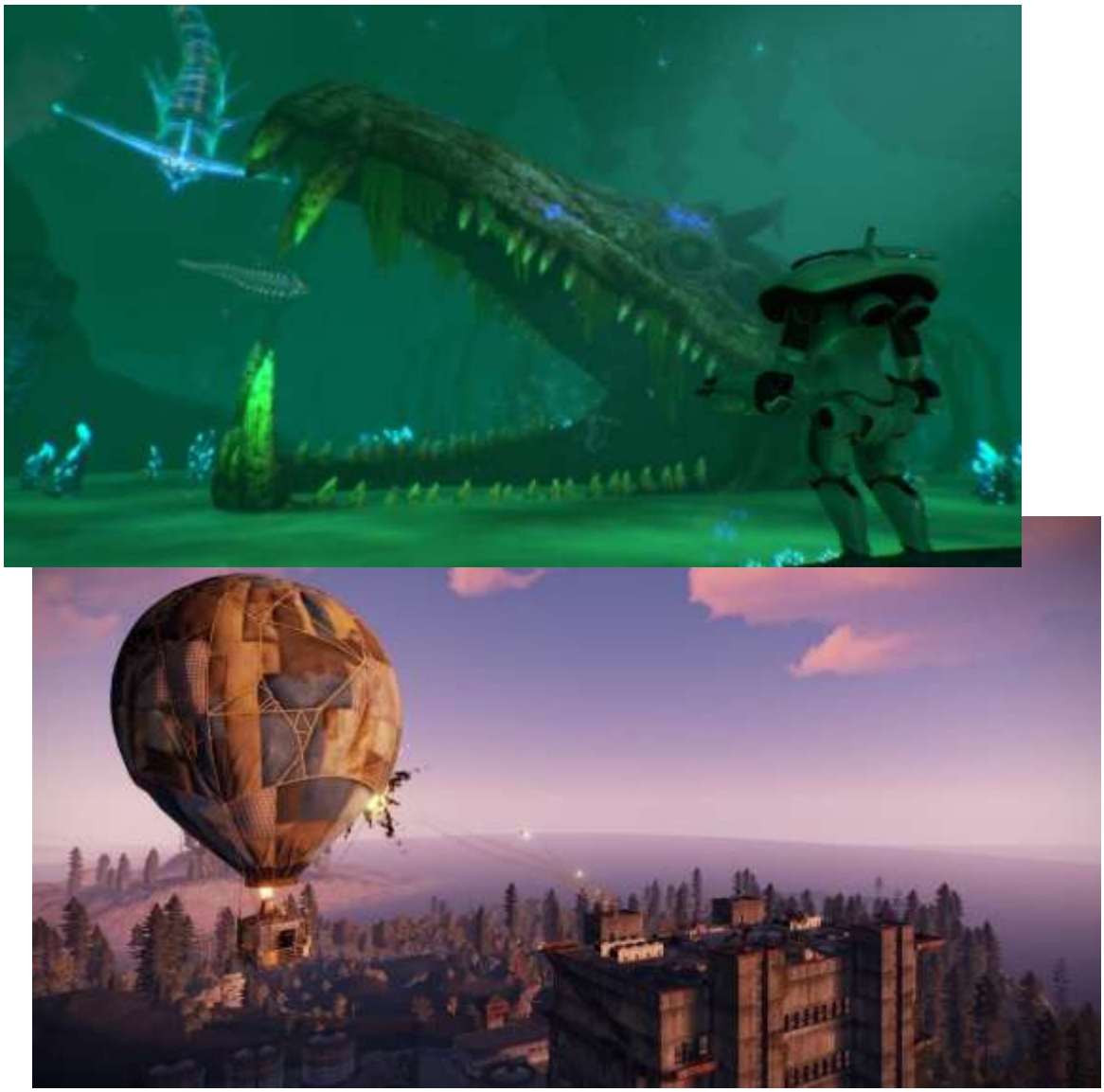


Рисунок 2.4 – Приклад гри на Unity. Subnautica

2.2.2. Unreal Engine 4

На черзі в нас буде розгдянутий Unreal Engine 4 (UE4) – ігровий рушій створений та оновлюваний компанією Epic Games. Знак 4 означає теперішню версію цього ігрового рушія вихід якої відбувся 2014 році, перша версія рушія Unreal Engine 1 була опублікована в 1998 році. Як і рушій Unity, ви можете розробляти гру або додаток під будь яку ігрову платформу.

UE4 підтримує створену Epic Games систему візуального програмування на рушії, що має назву Blueprint. Елементи ігрового процесу, що використовується блоками вузлів які базується на використанні ігрового інтерфейсу в редакторі UE. Blueprinys є скриптованою мовою для визначення якої можна використувати об'єктно-орієнтований клас чи обєкт [14]

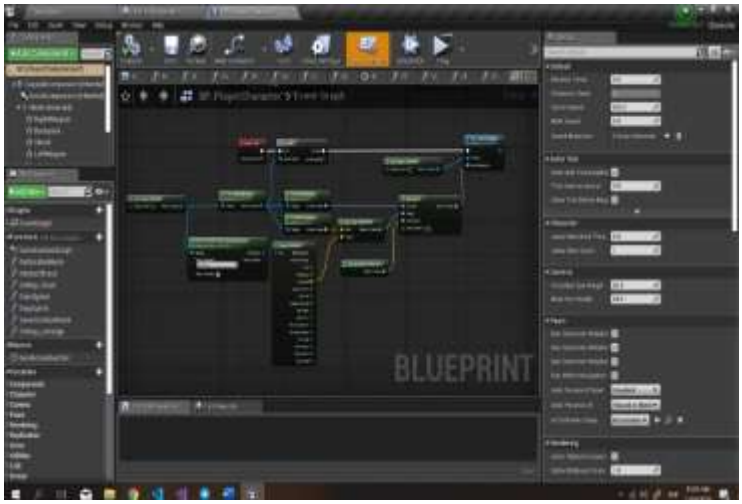


Рисунок 2.5 – Редактор Blueprint класів та загальний вигляд Blueprints

Редактор Blueprint потужна і гнучка система, тому що дає права користувачам використовувати більшість можливостей для створення концепцій та інструментів, які як правило мають лише програмісти. Кріс того, в UE4 є спеціальний блок для Blueprint, що доступна в модифікованій версії C++, яка дає право програмістам створювати любі базові системи, які будуть збільшені дизайнерами. Також програміст може використовуючи знання C++ створювати нові блоки для власного використання в Blueprints. Деякі потужні компанії використовують різні технології «Unreal» як головну основу для створення власного ігрового рушія. Так на основі рушія Unreal Engine 2 був створений новий рушій «Flesh Engine».

З плюсів даного ігрового рушія можна відмітити декілька:

• Система Blueprint дозволяє людям без знання язика програмування C++ створювати свої ігри на рушії UE4 не набираючи жодного рядку кода

• Система візуального програмування Blueprint пришвидшує створення прототипу гри та перевірки його в рушії.

Майже безкінечно кількість доступних асетів та плагінів для створення своєї гри через платформу Unreal Marketplace. Для стимулювання нових розробників компанія Epic Games що місяця роздають платні асети та плагіни безкоштовно новим розробникам ігор.

• Розробники UE4 постійно збільшують навчальні матеріали гайди та проводять бесіди.

Серед мінусів рушія можна вказати наступні недоліки:

• Великі проблеми з налаштуванням штучного інтелекту: при збільшення на одному рівні гри занадто багато ігрових істот з штучним інтелектом, спроби ігрового рушія обробити дії всіх істот одночасно призводить до постійного падіння fps тому більшості розробникам прийдеться знаходити способи обмеження або зменшення діяльності всіх істот з AI.

• Високий поріг входження для розробників ігор. Студент в сфері розробки та створення комп'ютерних та мобільних ігор. Без огляду на те, що всю гру можна розробити не написавши жодного рядку коду, все ж для простої людини у якої немає навичок в даній сфері цей ігровий рушій буде являтися занадто складним через дуже широкий функціонал, що рекомендує Unreal Engine 4 для розробників.

Потрібно також згадати про ліцензію даного ігрового рушія. З 3 березня 2015 року Unreal Engine 4 став повністю безкоштовним для всі розробників. Але розробники зобов'язані передавати 5% від продажу любої

гри компанії Epic Games, якщо квартальний дохід розробника перевищує 3000 доларів.

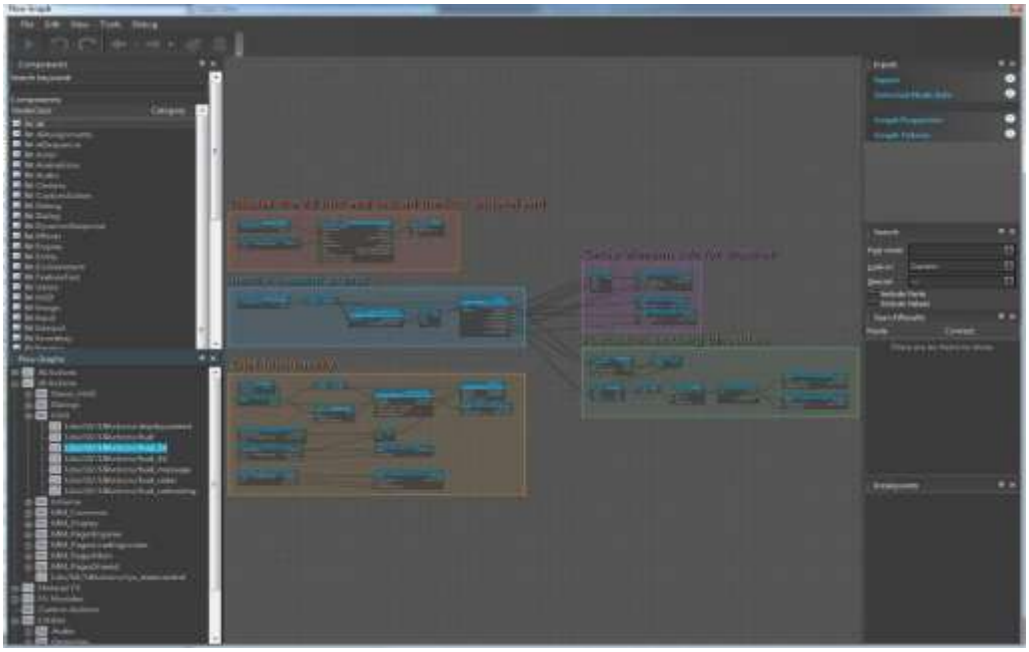
Рисунок 2.6 Приклад гри на Unreal Engine. Mortal Kombat 11

Рисунок 2.7 Приклад гри на Unreal Engine. Fortnite

Рисунок 2.8 Приклад гри на Unreal Engine. Man of Medan

2.2.3. CryEngine V

Останній ігровий рушій в моєму списку Crytek - CryEngine V, що був анонсований весною 2016 році. Він представляє собою оновлену версію CryEngine 4, в якій було встановлена підтримка шоломів віртуальної реальності, та написання скриптів на мові програмування C#. Створення проекту в даному GE відбувається завдяки мові програмування C++, а також існує нова система підтримки Flowgraph, що схожа по моделі використання на Blueprint від Epic Games. Та на відміну від Blueprint, Flowgraph набагато повільніше виконує команди тому, що він не може самостійно генерувати читаємий код. Із за цього не можливо утворити складну систему, оскільки



ігровий рушій буде не оптимізован. 15]

Рисунок 2.9 – Технологія Flowgraph в CryEngine

Ігровий рушій постачається компанією безкоштовно разом з SDK, та з запуском версії програми 5.5 всі ігрові розробники повинні виплачувати 5% від прибутку проекту, отриманих ними від ігор, що користуються останньою версією даного GE, що перевищує 5000 доларів. Розробники, що використовують старіші версії рушія можуть подати заявку на звільнення від виплат, якщо в майбутньому вони не будуть обновлятися до версії 5.5 або вище.

Серед плюсів ігрового рушія CryEngine V можна виділити наступні:

- Постійна підтримка найновіших технологій, таких як DirectX12, Vulkan API, VR, та постійне написання скриптів на мові програмування C#.
- Фотореалістична графіка в ігровому рушії, що при правильній розробці проекту може перевершити будь-які інші ігри створені за допомогою рушія Unity або Unreal Engine 4. Також, ігровий рушій підтримує realtime render, що дозволяє в декілька кліків протестувати щойно створений рівень гри.
- GameSDK – інструмент, що йде при встановленні разом з ігровим рушієм, та на базі якого можливо в декілька кліків створювати власні ігри.

Що ж до мінусів, то вони наступні:

- Дуже малий вибір початкових асетів та плагінів в магазині CryEngine Marketplace в порівнянні з магазинами Unreal Marketplace або магазинами Unity Asset Store.
- Присутні деякі перешкоди при створенні фінальної версії продукта.
- Присутні деякі обмеження при розробці онлайн проектів, стандартна версія рушія без змін може підтримувати не більше 32 гравців онлайн

Рисунок 2.10. Приклад гри на рушії CryEngine.

Рисунок 2.11 Приклад гри на рушії CryEngine. Hunt: Showdown

Рисунок 2.12 – Приклад гри на рушії CryEngine. Kingdom Come

2.3 Шаблони та демо-проекти

Після запуску ігрового рушія для розробника буде запропоновано декілька різних можливостей. Перше це створити проект, друга можливість вибрати готові Progects або пройти різні курси в Tutorials. Для розробника є декілька шаблонів такі як 2D, 3D, 3D, With Extras та інші.

Також розробник може протестувати декілька демо-проектві

- Roll a Ball
- Create Kit: FPS
- Platfermer Microgame
- Create Kir: Puzzle
- Beginner Scripting

2.4. Збірка

Рушії дозволяє зібрати будь який проект під всі популярні платформи, від ПК і консолей до мобільних пристроїв. Для ПК і мобільних пристроїв вистачає одного кліка, а ось для інших платформ може знадобитися додатковий SDK.

2.5. Процес реалізації

```
using UnityEngine;

public class Player
{
    public int HP, Mana, Manapool;
    const int MAX_MANAPOOL = 10;

    public Player()
    {
        HP = 30;
        Mana = Manapool = 2;
    }

    public void RestoreRoundMana()
    {
        Mana = Manapool;
    }

    public void IncreaseManapool()
    {
        Manapool = Mathf.Clamp(Manapool + 1, 0, MAX_MANAPOOL);
    }

    public void GetDamage(int damage)
    {
        HP = Mathf.Clamp(HP - damage, 0, int.MaxValue);
    }
}
```

Рисунок 2.13 - Приклад реалізації класу Player

На рисунку зображений процес реалізації класу Player який виконує функцію HP яка віднімається після удару по гравцю, та Mana яка

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

public class SpellTarget : MonoBehaviour, IDropHandler
{
    public void OnDrop(PointerEventData eventData)
    {
        if (!GameManagerScr.Instance.IsPlayerTurn)
            return;

        CardController spell = eventData.pointerDrag.GetComponent<CardController>(),
                       target = GetComponent<CardController>();

        if (spell &&
            spell.Card.IsSpell &&
            spell.IsPlayerCard &&
            target.Card.IsPlaced &&
            GameManagerScr.Instance.CurrentGame.Player.Mana >= spell.Card.Manacost)
        {
            var spellCard = (SpellCard)spell.Card;

            if ((spellCard.SpellTarget == SpellCard.TargetType.ALLY_CARD_TARGET &&
                 target.IsPlayerCard) ||
                (spellCard.SpellTarget == SpellCard.TargetType.ENEMY_CARD_TARGET &&
                 !target.IsPlayerCard))
            {
                GameManagerScr.Instance.ReduceMana(true, spell.Card.Manacost);
                spell.UseSpell(target);
                GameManagerScr.Instance.CheckCardsForManaAvaliability();
            }
        }
    }

}
```

збільшується з кожним кроком.

Рисунок 2.14 - Приклад реалізації класу SpellTarget

На рисунку зображений процес реалізації класу SpellTarget який відповідає за роботу карт Spell як для союзник карт та і для ворожих. Дія цих

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

public class AttackedCard : MonoBehaviour, IDropHandler {

    public void OnDrop(PointerEventData eventData)
    {
        if (!GameManagerScr.Instance.IsPlayerTurn)
            return;

        CardController attacker = eventData.pointerDrag.GetComponent<CardController>(),
                       defender = GetComponent<CardController>();

        if (attacker &&
            attacker.Card.CanAttack &&
            defender.Card.IsPlaced)
        {
            if (GameManagerScr.Instance.EnemyFieldCards.Exists(x => x.Card.IsProvocation) &&
                !defender.Card.IsProvocation)
                return;

            GameManagerScr.Instance.CardsFight(attacker, defender);
        }
    }
}
```

карт може призвести до збільшення або зменшення Hp інших карт або героїв.

Рисунок 2.15 Приклад реалізації класу AttackedCard

На рисунку зображений процес реалізації класу AttackedCard котрий відповідає за перевірку можливості атакувати інші карти. Якщо на одній або декількох картах знаходяться властивість defender інші карти зобов’язані атакувати їх як єдині можливі цілі.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class AttackedHero : MonoBehaviour, IDropHandler {

    public enum HeroType
    {
        ENEMY,
        PLAYER
    }

    public HeroType Type;
    public Color NormalCol, TargetCol;

    public void OnDrop(PointerEventData eventData)
    {
        if (!GameManagerScr.Instance.IsPlayerTurn)
            return;

        CardController card = eventData.pointerDrag.GetComponent<CardController>();

        if (card &&
            card.Card.CanAttack &&
            Type == HeroType.ENEMY &&
            !GameManagerScr.Instance.EnemyFieldCards.Exists(x => x.Card.IsProvocation))
        {
            GameManagerScr.Instance.DamageHero(card, true);
        }
    }

    public void HighlightAsTarget(bool highlight)
    {
        GetComponent<Image>().color = highlight ?
                                      TargetCol :
                                      NormalCol;
    }
}
```

Рисунок 2.15 Приклад реалізації класу AttackedHero

На рисунку зображений процес реалізації класу AttackedHero який відповідає за можливість або не можливість інших карт атакувати героя. А також розпізнавання картами ворожого та свого героя.

Висновок до розділу 2

Стрімке зростання мобільних ігор у сфери розваг призвело до створення спеціальних програм для їх створення – ігрових рушіїв. Дані технології призначені для того, щоб полегшити життя розробникам ігор, оскільки більшість проблем, що стосуються базових потреб для створення ігор вирішені всередині самого рушія і дають можливість програмісту зосередитись на розробці ігрової механіки, а не на взаємодії з апаратним забезпеченням.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ КРОСПЛАТФОРМНОГО ІГРОВОГО ЗАСТОСУНКУ

3.1. Ідея

Перед узгодженням теми роботи я уже мав деякі ідеї для ігри та механізму його геймплею. Натхненням для мене була невеличка гра під назвою WORDLE, автором якої є Джош Вордл. Після виходу у світ ця, здавалося б, до нестями проста гра з часом набула небачаного розголосу у мережі. Одна із найбільших газет у США — The New York Time — помітила її та, побачивши перспективи, викупила у Джоша за семизначну суму. Та це й не дивно, адже уже в грудні 2021 року кількість гравців уже сягала 1 мільйон.. Простий, захоплюючий геймплей — ось це мета, яку я маю досягнути. Трішки подумав та розробив перший варіант геймплею. Поле з 9 клітинами (як клавіатура для набору номера) та 2 гравці — це усе що включав у себе перший прототип застосунку. Гравці ходять по черзі. Один гравець ховається від іншого поки той намагається його знайти, ставши на його клітинку. Кілька раундів і переможець нагороджується балами. Трохи погравши у цей прототип я дійшов до висновку, що на такому великому полі вкрай складно спіймати свого опонента, тому той, хто ховається у більшості випадків отримував перемогу. Отже, доцільним було скоротити кількість клітин збалансувавши шанси на перемогу обох гравців. Спочатку до 6 клітин, а потім до 4. На цьому і зупинився.

3.2 Вибір платформи

Мобільні застосунки — це ідеальна платформа для такої простої гри. Хоча, у майбутньому було б також не погано мати можливість з легкістю інтегрувати гру у веб-сайт, або запускати на desktop. Так як основна платформа — це мобільна, то я розглядав рішення для розробки мобільних кросплатформних застосунків. Провівши невеличку пошукову роботу з'ясував, що на даний момент є досить багато таких рішень, але найпопулярніші із них —React Native та Flutter. Натикався також на Xamarin,

Cordova та інші. Єдиним мінусом цих рішень було невелике комюніті, відповідно мала кількість бібліотек та більша ймовірність застрягнути на якісь проблемі, не знайшовши спосіб вирішення. Також насторожила невелика кількість не закритих issue у офіційних репозиторіях фреймворків. Також були ігрові движки — Unity, Game Maker тощо. Але їхні інструменти дещо відрізнялися від моїх потреб, так як це все таки рушії для розробки 3D ігор, з можливістю 2D. А я вибирав рішення, які безпосередньо націлені на мобільні платформи, відповідно саме на цих платформах розробники фреймворків роблять акцент і намагаються досягнути максимальної швидкодії, ресурсозатратності і ефективності розробки. Вибираючи між RN в Flutter, вирішальним фактором була швидкодія. У Flutter є свій власний довідок для малювання графіки на екрані, а React Native використовує мости щоб зв'язати код Java Script з нативними елементами інтерфейсу, що і в результаті приводить до посадки FPS.

3.3. Дизайн

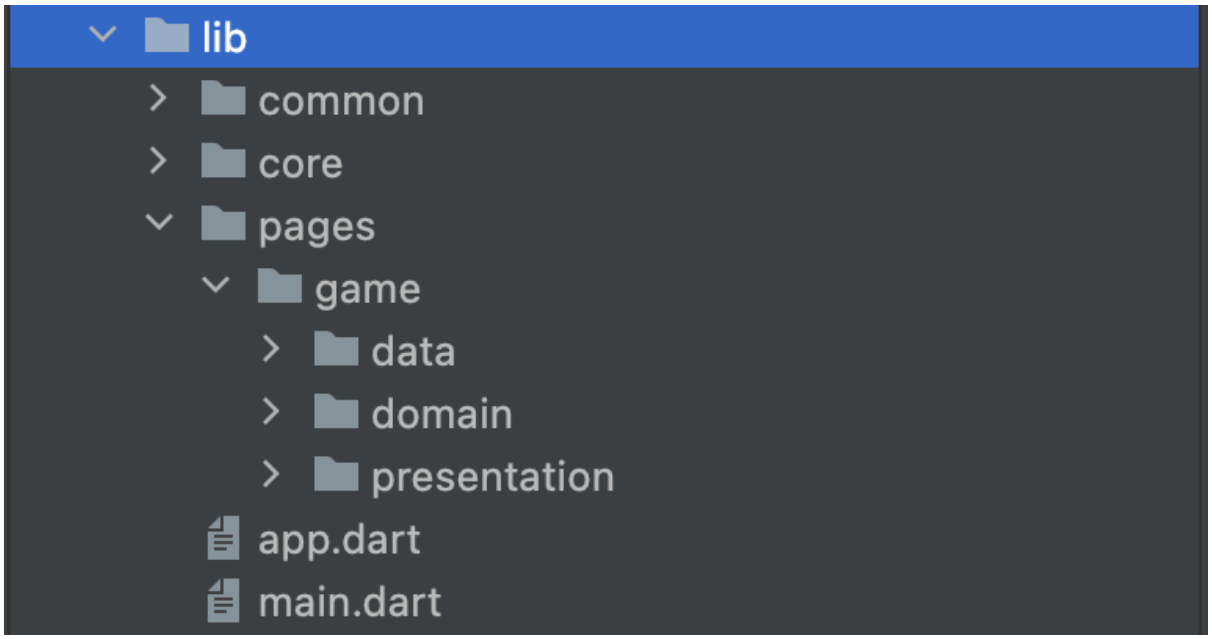
Дизайн був розроблений у програмі Figma.

3.4. Написання коду

Створення Flutter-проекту в Android Stu.dio (скріни)

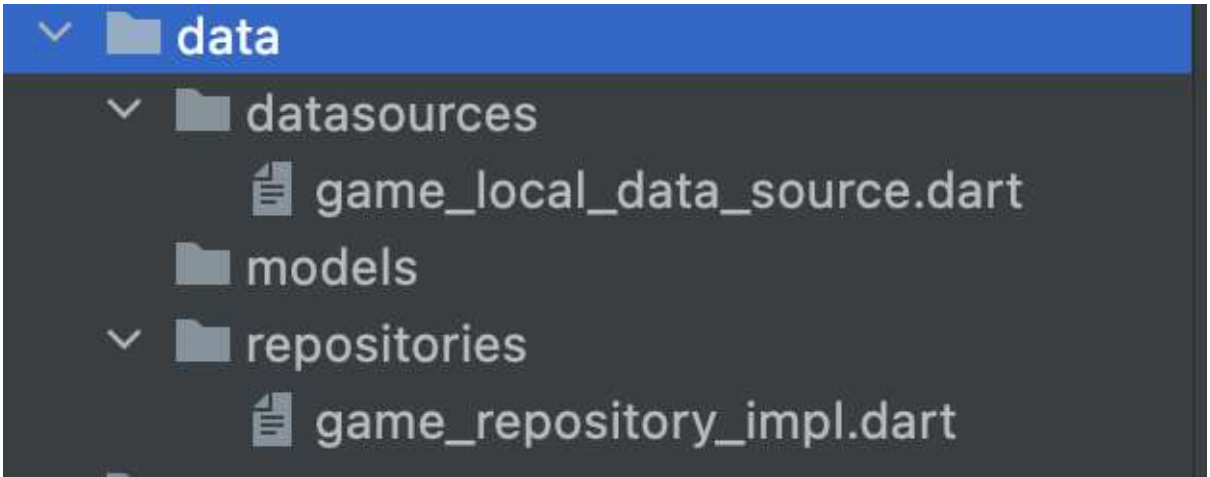
Підготовка проекту (завантаження ресурсів (іконок, шрифтів, текстів), підключення бібліотек)

Проектування архітектури застосунку. Clean Architecture (https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html). Розбивання на шари.



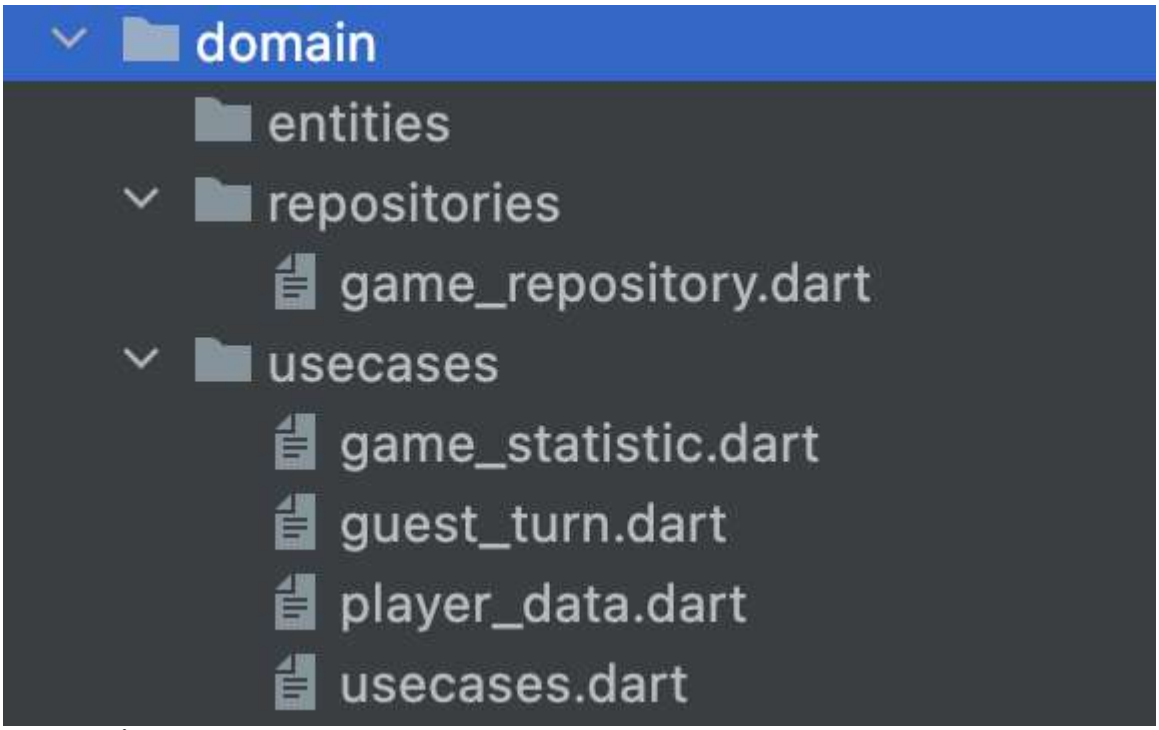
Структура пакетів коду:

data — відповідає за збереження даних, низькорівневий код роботи з



внутрішнім та віддаленим сховищем. datasources — джерела, з яких надходять дані. repositories — реалізація патерну проектування repository, який є точкою перетину двох шарів архітектури, а саме data і domain.

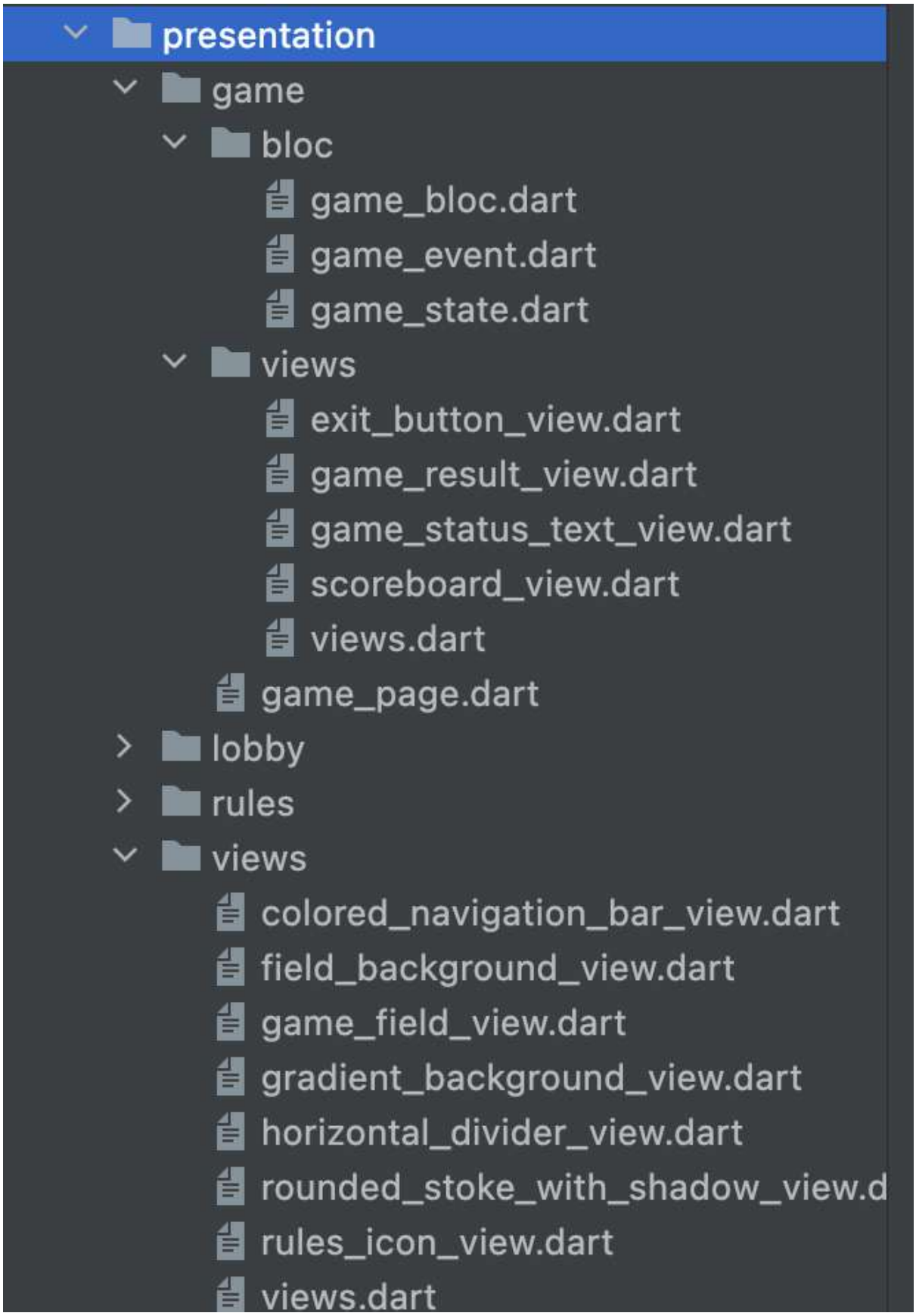
Domain — шар бізнес-логіки. repositories — інтерфейс патерну проектування repository. usecases — опис всього однієї дії, яка є частину



бізнес-логіки.

Presentation — шар, який відповідає за відображення на екрані.

game, lobby, rules — пакети, що стосуються за відповідної сторінку гри. bloc — реалізація патерну проектування BloC, про який буде іти мова далі. views — класи, які описують віджет — елемент користувацького інтрефейсу.


presentation
game
bloc
game_bloc.dart
game_event.dart
game_state.dart
views
exit_button_view.dart
game_result_view.dart
game_status_text_view.dart
scoreboard_view.dart
views.dart
game_page.dart
lobby
rules
views
colored_navigation_bar_view.dart
field_background_view.dart
game_field_view.dart
gradient_background_view.dart
horizontal_divider_view.dart
rounded_stoke_with_shadow_view.d
rules_icon_view.dart
views.dart

3.5. Керування станом

Flutter використовує декларативний підхід у своїй реалізації. Це означає, що Futter будує свій користувацький інтерфейс в залежності від стану застосунку. На відмінну від інтерактивного підходу, під час розробки з декларативним підходом нам потрібно думати ЩО саме має буде відображено на екрані, а не ЯК саме це має бути відображено. Декларативний підхід має безліч переваг та застосовується у багатьох фреймворках, навіть нативно мобільна розробка активно переходить на цей підхід. За цим майбутнє. https://docs.flutter.dev/development/data-and-backend/state-mgmt/intro
https://docs.flutter.dev/development/data-and-backend/state-mgmt/declarative
https://docs.flutter.dev/development/data-and-backend/state-mgmt/ephemeral-vs-app
https://docs.flutter.dev/development/data-and-backend/state-mgmt/simple
https://docs.flutter.dev/development/data-and-backend/state-mgmt/options

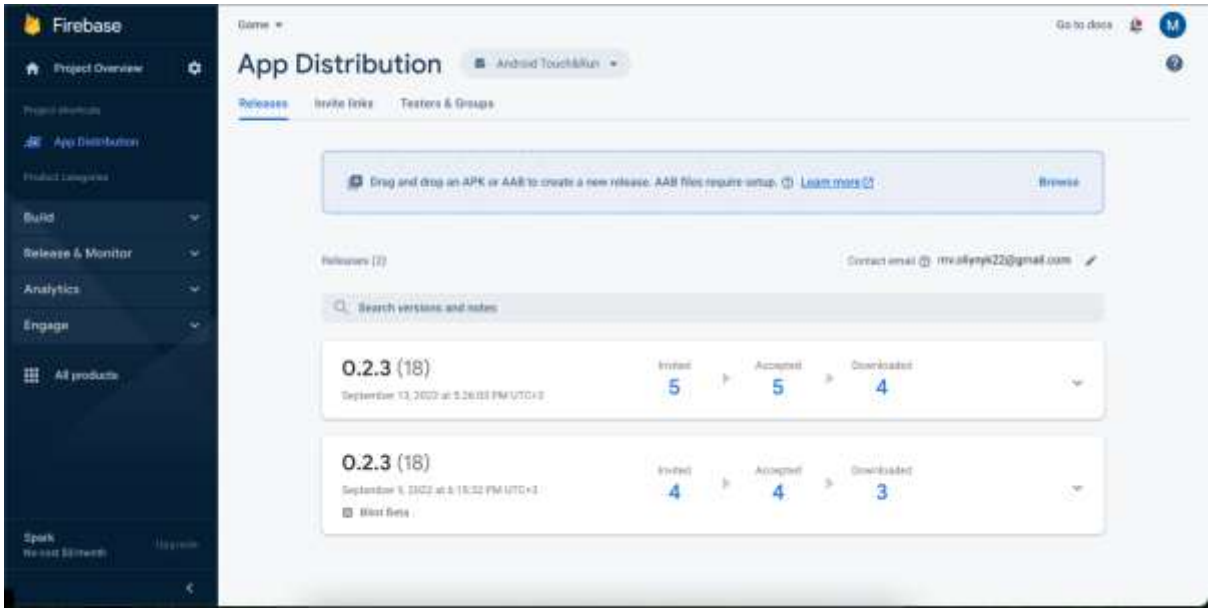
Виходячи з вищесказаного варто було б розважливо вибрати необхідну реалізацію керування станом програми. Так як стандартний метод керування станом потребує великої кількості написання рутинного та шаблонного коду, то розробники потурбувалися про це, щоб цей процес став більш приємний та швидким. Тому існує кілька варіантів керування станом у Flutter, один із найпопулярніших та найстабільнішим є BLoC, про який ми уже згадували раніше. Детальна документація значно допомогла опанувати цей паттерн. https://bloclibrary.dev/#/ru/whybloc
https://bloclibrary.dev/#/ru/
https://bloclibrary.dev/#/ru/flutterbloccoreconcepts

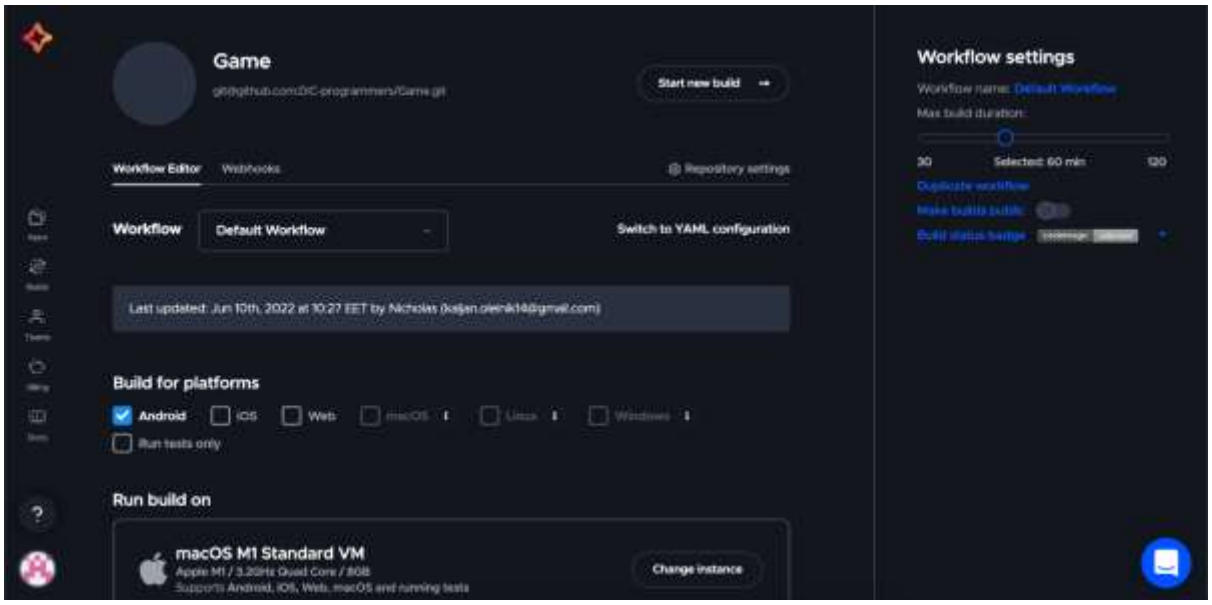
3.6. Тестування та відладка

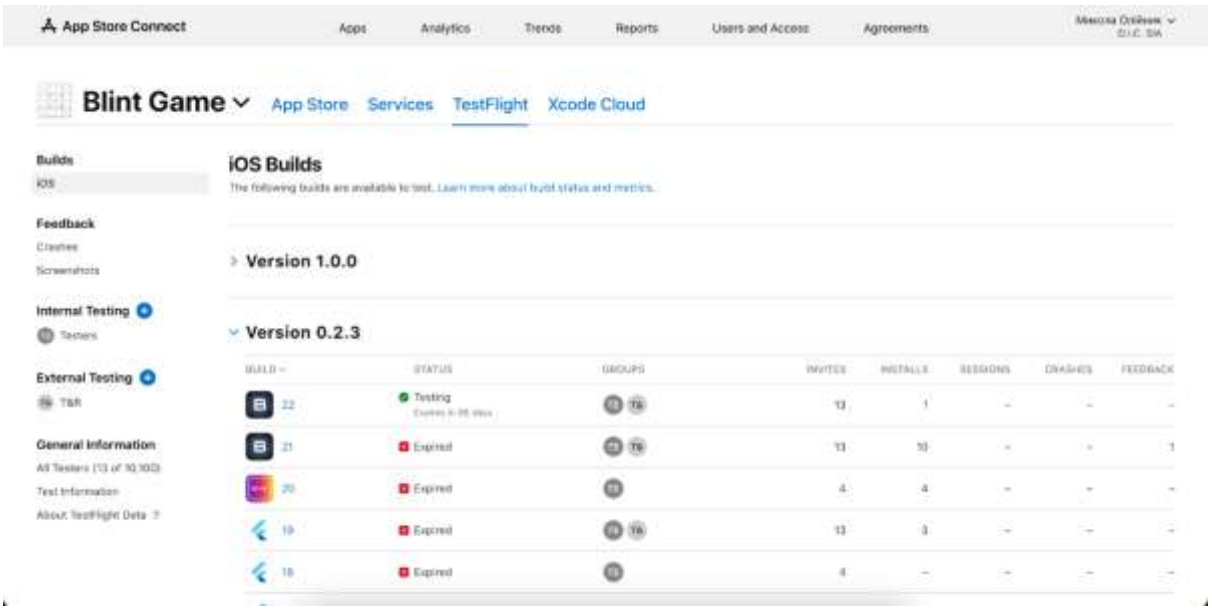
Наступний етап — це тестування додатку та його відладка, виправлення помилок, вдосконалення, оптимізація та рефакторинг коду.

Деплой (CI/CD)

Для того, аби додаток був доступний для користувачів у тестовому режимі було використано сервіс дистрибуції — Firebase App Distribution для Android та TestFlight для iOS. Так як процес вивантаження аж двох додатків займає певну кількість часу і є досить рутинною справою, то я вирішив, що буде чудовою ідеєю реалізувати автоматичний деплой і дистрибуцію, так звання — Continuous Integration/Continuous Delivery. Для цього я підключив та налаштував цей процес з допомогою сервісу Codemagic.io, що дозволило мені робити деплой всього кліком миші на віддаленому пристрої!

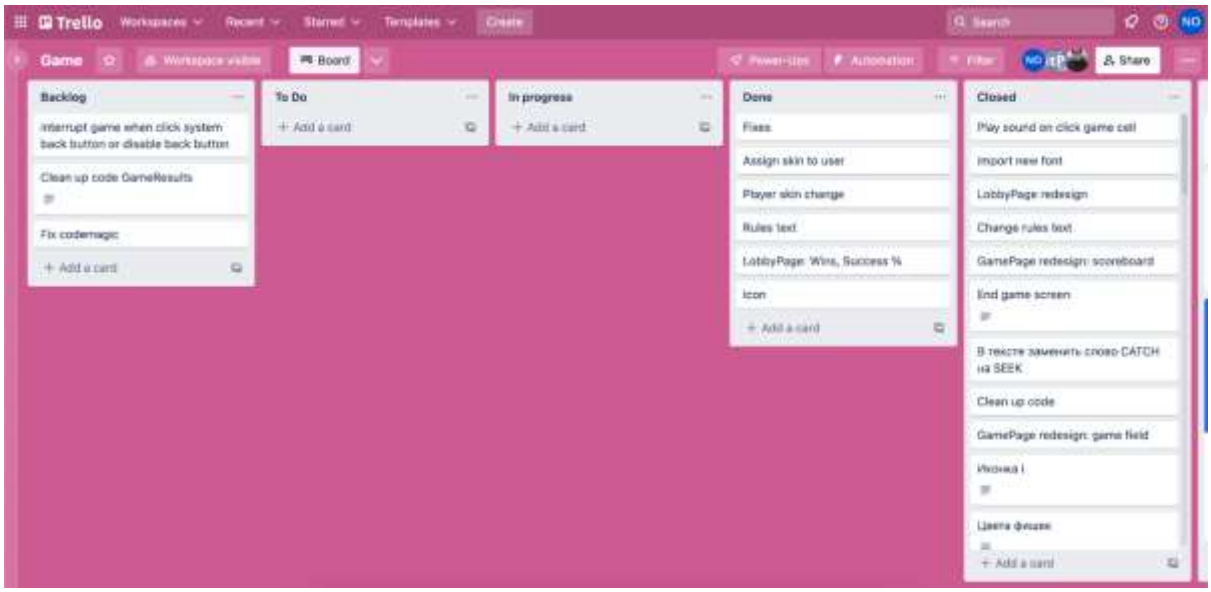






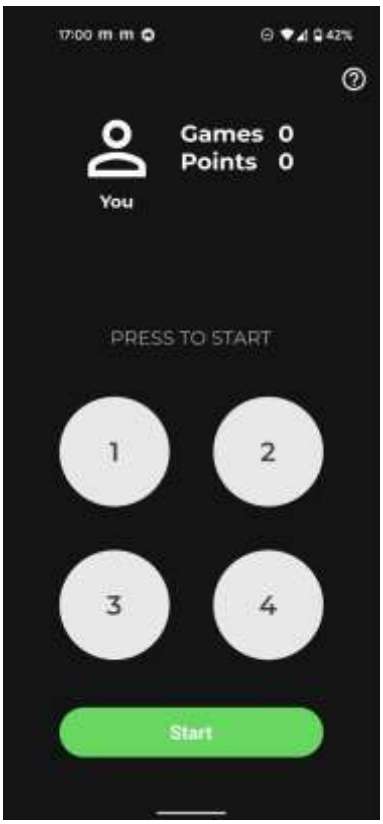
3.7. Організація роботи

Для оптимізації процесу розробки я вирішив доцільним на практиці застосувати набуті мною знання та організувати свою роботу у Trello, де записував усі свої завдання у вигляді карток. Для організації дошки мені підійшов принцип роботи KANBAN. Моя дошка виглядає наступним чином:



3.8. Мобільний ігровий кросплатформерний застосунок Touch&Run

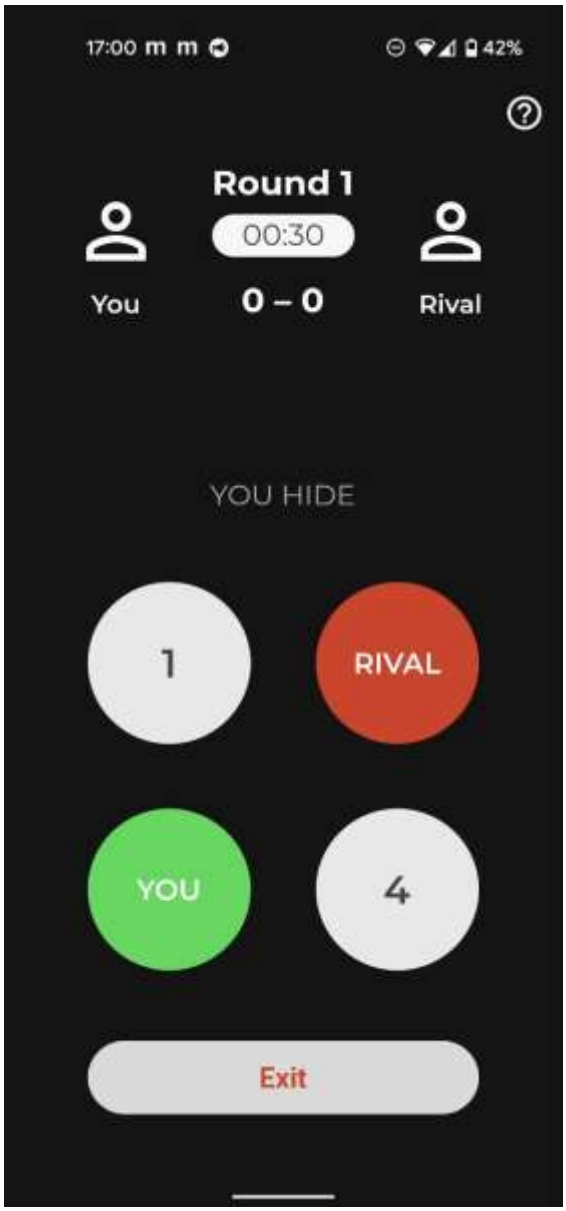
Розроблено з допомогою кросплатформерного фреймфорка Flutter, розробленого компанією Google. Використовувана мова програмування Dart.



Опис:

Застосунок являє собою ігровий додаток.

При запуску бачимо домашню сторінку зі статистикою ігор (Games — кількість зіграних ігор, Points — кількість зароблених очок):



При натисканні кнопки Start попадаємо на ігрову сторінку:

Правила гри наступні:

1. Один проти одного грають два гравці.

2. Гра складається із чотирьох раундів тривалістю 30 секунд кожен раунд.

3. Ігрове поле складається із 4 кругів, на яких стоять фішки гравців.

4. Протягом одного раунду один гравець повинен пересунути свою фішку на той круг, під яким ховається фішка другого гравця. А другий гравець, відповідно, повинен не дати виявити свою фішку, в тому числі змінюючи її розташування на ігровому полі.

5. Гравці по черзі роблять хід, натискаючи на один із кругів ігрового поля.

5.1. Перший хід завжди робить гравець, який ховає свою фішку.

5.2. На виконання одного ходу дається 5 секунд, після закінчення яких право ходу переходить до іншого гравця.

5.3. Гравець, який ховається, бачить на своєму екрані всі переміщення гравця, який шукає.

5.4. Гравець, який шукає, не бачить переміщення гравця, який ховається, та діє навмання.

5.5. Будь-який гравець в наступному ході може не переміщати свою фішку на інший круг та залишитись на тому самому ж місці.

6. Раунд достроково закінчується, якщо гравець, який шукає, перемістив свою фішку на той круг, під яким ховається фішка другого гравця. У цьому випадку перемога дістається першому гравцю.

7. Якщо до закінчення часу раунду гравець, який шукає фішку другого гравця, не зміг її виявити, то перемога дістається другому гравцю.

8. У наступному раунді гравці змінюються ролями.

9. За перемогу в кожному раунді присуджується одне очко.

10. У грі перемагає той гравець, який виграє більше раундів та набере більше очок.

10.1. Гра достроково закінчується у випадку, якщо один із гравців виграє три раунди поспіль.

11. Очки за виграні раунди також нараховуються гравцеві, що програв.

12. У разі нічиї проводиться повторна гра, і так триває доти, доки один із гравців не переможе в черговій грі.

13. У разі дострокового переривання гри очки за виграні раунди гравцям не нараховуються.

Даний застосунок підключений до платформи Firebase та її можна завантажити за наступним посиланням:

https://drive.google.com/file/d/1wK3PF8Dx0SAfrtqPuvJZGBS0aEdoSGwH/view?usp=sharing
https://appdistribution.firebase.google.com/testerapps/1:1041118481099:android:9cfa59dcb9c4d25ccea94e/releases/6q7tkjthb0cp0

Висновки до 3 розділу

В цьому розділі були досліджені засоби розробки мобільних додатків React Native та Flutter та їх порівняння між собою.

Були досліджені різниці в підходах, процесі розробки та у підтримці існуючих додатків. Також Flutter був порівняний з нативною розробкою мобільних додатків та різницю в підходах між ними. Також була досліджена різниця в принципах побудови графічного інтерфейсу користувача, та розібран декларативний підхід в розробці інтерфейсів.

Крім того, був розроблений ігровий застосунок на Flutter, за допомогою якого можливо досліджувати швидкодію та ефективність роботи мобільних додатків, написаними за допомогою засоба Flutter.

Цей плагін здатен реєструвати швидкодію рендерингу мобільного додатку на кінцевому девайсі користувача в debug та release збірках, виводити дані на екран або у логи девайсу і також записувати швидкодію з плином часу. Цей плагін може використовуватися в будь-якому проєкті на Flutter іншими розробниками для вимірювання швидкодії та працює як на Android, так і на iOS.

ВИСНОВКИ

За проведеним аналізом можна зробити висновки, що технології та принципи, які використовує Flutter, є більш перспективними, надійними, зручними та швидкими до використання при розробці кросплатформенних мобільних додатків, ніж ті, що використовує його найближчий та найпопулярніший аналог – React Native.

Фреймворк Flutter випереджає React Native у розробці мобільний додатків за наступними параметрами:

• Продуктивність та швидкодія створеного програмного забезпечення.

• Простота проєктування та розробки користувацького інтерфейсу (UI).

• Час, потрібний для розробки продукту.

• Надійність та краща підтримка від розробників та товариством розробників.

• Поточний тренд зацікавленності у технології стрімко зростає та вже випередив React Native.

• Фреймворк має можливість компіляції під десктопні операційні системи (Windows, macOS, Linux) та збірку під вебдодаток, який відтворюється у браузері.

В свою чергу, React Native продовжує випереджати Flutter за наступними параметрами:

• Поточна кількість розробників, які здатні розробляти програму цією технологією більша, ніж на Flutter.

• Використання більш поширеної мови програмування (JavaScript замість Dart).

• Деякі частини логіки можна перевикористовувати у вебдодатках, які написані за допомогою бібліотеки React.js.

• Все ще більше вакансій на ринку, ніж у Flutter.

Також Flutter є доброю альтернативною більш дорожчої та довшої нативної розробки під кожну платформу окремо, якщо розроблюваний мобільний додаток не використовує 3D графіку, та не взаємодіє на низькому

рівні з hardware-частиною мобільних пристроїв. Тобто, якщо додаток більше зав’язаний на взаємодію з віддаленим сервером.

Крім того, був розроблений мобільний ігровий кросплатформерний застосунок Touch&Run.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Shaun Lewis, Mike Dunn. Native Mobile Development: A Cross-Reference for iOS and Android. Chapter 1. 2019. P. 1-3.

2. Shaun Lewis, Mike Dunn. Native Mobile Development: A Cross-Reference
for iOS and Android. Chapter 1. 2019. P. 4-5.

3. Niranjan Kumar. Learn Android. Chapter 1. 2020. P. 4-5.

4. Nate Ebel. Mastering Kotlin. Chapter 13. 2019. P. 267.

5. Steve Derico. Introducing IOS 8: Swift Programming from Idea to App Store. Vol. 3. 2014. P. 55-56.

6. John M. Wargo. Learning Progressive Web Apps. Chapter 8. 2020. P.150.

7. Dan Hermes. Xamarin Mobile Application Development. Vol. 1. 2015 P. 1-5.

8. Jan Rabe. Everything that is wrong with Xamarin and why it is bad for you.
Medium. 2018. P. 5-10

9. Prajyot Mainkar, Salvatore Giordano. Google Flutter Mobile Development
Quick Start Guide. Chapter 1. 2019. P. 5-10.

10. Barry Burd. Flutter For Dummies. Part 1. 2020. P. 22-23.

11. Google Trends "Flutter vs React Native" – [Електронний ресурс] – Режим доступу: https://trends.google.com/trends/explore?date=2017-01-01%202020-08-01&q=Flutter,React%20Native.

12. Alessandro Biessek. Flutter for Beginners: An introductory guide. Chapter 3. 2019. P. 99.

13. Flutter Документація – [Електронний ресурс] – Режим доступу: https://flutter.dev/docs/resources/architectural-overview.

14. Chris Sells. Canonical enables Linux desktop app support with Flutter. – Medium Flutter – 2020. Chapter 14. 2019. P. 376.

15. Flutter Showcase – [Електронний ресурс] – Режим доступу: https://flutter.dev/showcase.

16. Дмитрий Беспалов, Наталия Коробейникова. Операционные системы
реального времени и технологии разработки кроссплатформенного программного обеспечения. Харьков. Часть 2. 2019. С. 163.

17. Vladimir Novick. React Native – Building Mobile Apps with JavaScript. Chapter 4. 2017. P. 114.

Додаток А

ЛІСТИНГ ПРОГРАМ

Приклад додатку на Flutter мовою Dart, який виводить текст та кнопку:

```
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Welcome to Flutter',
home: Scaffold(
appBar: AppBar(
title: Text('Welcome to Flutter'),
),
body: Center(
child: Column(
children: [
Text('Hello World'),
SizedBox(height: 20),
ElevatedButton(
onPressed: () {
print('Click);
},
child: Text('A button),
),
],
),
),
),
);
}
}
```

Приклад JSX-коду, який використовує React Native:

```
<Button color="blue" shadowSize={2}>
Click Me
</Button>
```

Приклад в що транслюється JSX код у React Native:

```
React.createElement(
```

```
96
Button,
{color: 'blue', shadowSize: 2},
'Click Me'
);
```

Приклад додатку ≪Hello World≫ у React Native:

```
import React from 'react';
import { Text, View } from 'react-native';
import ToolbarAndroid from '@react-native-community/toolbar-android';
const HelloWorldApp = () => {
return (
<ToolbarAndroid
title="AwesomeApp">
<View
style={{
flex: 1,
justifyContent: "center",
alignItems: "center"
}}>
<Text>Hello, world!</Text>
</View>
</ToolbarAndroid>
)
}
export default HelloWorldApp;
```

Приклад додатку ≪Hello World≫ у Flutter:

```
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Welcome to Flutter',
home: Scaffold(
appBar: AppBar(
title: Text('Welcome to Flutter'),
),
body: Center(
```

```
child: Text('Hello World'),
),
),
);
}
}
```

Приклад роботи зі станом у Flutter за використанням StatefulWidget та State у віджетів:

```
MyHomePage({Key key, this.title}) : super(key: key);
final String title;
@override
_MyHomePageState createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
int _counter = 0;
void _incrementCounter() {
setState(() {
_counter++;
});
}
@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Text(widget.title),
),
body: Center(
child: Column(
children: <Widget>[
Text('You have pushed the button this many times:'),
Text('$_counter'),
],
),
),
floatingActionButton: FloatingActionButton(
onPressed: _incrementCounter,
child: Icon(Icons.add),
),
);
```

```
}
}
```

Приклад роботи зі станом у React Native за використанням State компонента:

```
class Clock extends React.Component {
constructor(props) {
super(props);
this.state = {date: new Date()};
}
render() {
return (
<div>
<h1>Привіт, світе!</h1>
<h2>Зараз {this.state.date.toLocaleTimeString()}.</h2>
</div>
);
}
}
```

Код розробленого додатку на Flutter для пошуку фільмів.

```
class MoviesApp extends StatelessWidget {
@override
Widget build(BuildContext  context) {
98
return MaterialApp(
title: 'Пошук фільмів',
home: MoviesAppHome(),
);
}
}
```

Код головного екрану зі списком:

```
class MoviesAppHome extends StatefulWidget  {
@override
MoviesAppHomeState createState() => MoviesAppHomeState();
}
class MoviesAppHomeState extends State<MoviesAppHome> {
final searchTextController = new TextEditingController();
String searchText = "";
@override
void dispose() {
```

```
searchTextController.dispose();
super.dispose();
}
void itemClick(Movie item) {
Navigator.push(
context,
MaterialPageRoute(
builder: (context) => MovieDetail(
movieName: item.title,
imdbId: item.imdbID,
)),
);
}
@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Center(child: Text('Пошук фільмів')),
),
body: Column(
children: <Widget>[
Container(
child: Row(children: <Widget>[
Flexible(
child: TextField(
controller: searchTextController,
decoration: InputDecoration(
hintText: "Фраза для пошуку",
fillColor: Colors.white,
border: new OutlineInputBorder(
borderRadius: new BorderRadius.circular(25.0),
borderSide: new BorderSide(
),
),
//fillColor: Colors.green
),
),
),
IconButton(
```

```
icon: Icon(Icons.search),
tooltip: 'Пошук фільмів',
color: Colors.lightBlue,
onPressed: () {
setState(() {
searchText = searchTextController.text;
SystemChannels.textInput.invokeMethod('TextInput.hide');
});
},
),
]),
padding: EdgeInsets.all(10),
color: Colors.grey[200],
),
if (searchText.length > 0)
99
FutureBuilder<List<Movie>>(
future: searchMovies(searchText),
builder: (context, snapshot) {
if (snapshot.hasData) {
return Expanded(
child: MovieList(
movies: snapshot.data,
itemClick: this.itemClick)
);
} else if (snapshot.hasError) {
return Text("${snapshot.error}");
}
return CircularProgressIndicator();
}),
],
));
}
}
```

Код екрану деталей фільму:

```
class MovieDetail extends StatelessWidget {
final String movieName;
final String imdbId;
MovieDetail({this.movieName, this.imdbId});
```

```
@override
Widget build(BuildContext  context) {
return Scaffold(
appBar: AppBar(
title:  Text(this.movieName),
),
body: FutureBuilder<MovieInfo>(
future: getMovie(this.imdbId),
builder:  (context, snapshot) {
if (snapshot.hasData) {
return Container(
padding: EdgeInsets.all(12),
child: Column(
crossAxisAlignment:  CrossAxisAlignment.start,
children:  <Widget>[
Container(
padding: EdgeInsets.fromLTRB(0, 10, 0, 10),
alignment:  Alignment.center,
child: Image.network(
snapshot.data.poster,
width: 200,
),
),
Card(
child: Container(
padding: EdgeInsets.all(12),
child: Column(children:  [
Text(snapshot.data.plot, textAlign:  TextAlign.justify),
SizedBox(height:  15),
PaddedText("Рік : " + snapshot.data.year),
PaddedText("Жанр : " + snapshot.data.genre),
PaddedText("Режесура : " + snapshot.data.director),
PaddedText("Час : " + snapshot.data.runtime),
PaddedText("Рейтинг : " + snapshot.data.rating),
PaddedText("IMDB рейтинг : " + snapshot.data.imdbRating),
PaddedText("Meta Score : " + snapshot.data.metaScore),
],
),
),
```

```
)
]));
} else if (snapshot.hasError) {
return Text("${snapshot.error}");
}
return Center(child: CircularProgressIndicator());
}),
);
100
}
}
class MovieItem extends StatelessWidget {
final Movie movie;
MovieItem({this.movie});
@override
Widget build(BuildContext context) {
return Container(
child: Row(
mainAxisAlignment: MainAxisAlignment.start,
children: <Widget>[
Column(children: <Widget>[
if (this.movie.poster != "N/A")
Image.network(this.movie.poster, height: 100, width: 100)
]),
Expanded(child: Column(
crossAxisAlignment: CrossAxisAlignment.start,
mainAxisAlignment: MainAxisAlignment.start,
children: <Widget>[
Text(
this.movie.title,
overflow: TextOverflow.ellipsis,
maxLines: 2,
),
Text(this.movie.year),
Text(this.movie.type)
]),)
],
),
padding: EdgeInsets.all(10),
```

```
margin: EdgeInsets.only(top: 12, left: 10, right: 10),
decoration:
new BoxDecoration(borderRadius: BorderRadius.all(Radius.circular(16)),color:
Colors.white),
);
}
}
```

Код моделей та сервісу для комунікації з сервером:

```
class Movie {
final String title;
final String year;
final String type;
final String poster;
final String imdbID;
Movie({this.title, this.year, this.type, this.poster, this.imdbID});
factory Movie.fromJson(Map<String, dynamic> json) {
return Movie(
title: json['Title'],
year: json['Year'],
type: json['Type'],
poster: json['Poster'],
imdbID: json['imdbID']);
}
}
class MovieInfo {
final String title;
final String year;
final String rating;
final String runtime;
final String genre;
final String director;
final String plot;
final String poster;
final String imdbRating;
final String metaScore;
MovieInfo(
{this.title,
this.year,
this.rating,
```

```
101
this.runtime,
this.genre,
this.director,
this.plot,
this.poster,
this.imdbRating,
this.metaScore});
factory MovieInfo.fromJSON(Map<String, dynamic> json) {
return MovieInfo(
title: json['Title'],
year: json['Year'],
rating: json['Rated'],
runtime: json['Runtime'],
genre: json['Genre'],
director: json['Director'],
plot: json['Plot'],
poster: json['Poster'],
imdbRating: json['imdbRating'],
metaScore: json['Metascore']);
}
}
Future<List<Movie>> searchMovies(keyword) async {
final response = await http.get('$API_URL$API_KEY&s=$keyword');
log(response.body);
if (response.statusCode == 200) {
Map data = json.decode(response.body);
if (data['Response'] == "True") {
var list = (data['Search'] as List)
.map((item) => new Movie.fromJson(item))
.toList();
return list;
} else {
throw Exception(data['Error']);
}
} else {
throw Exception('Something went wrong !');
}
}
```

```
Future<MovieInfo> getMovie(movieId) async {
final response = await http.get('$API_URL$API_KEY&i=$movieId');
log(response.body);
if (response.statusCode == 200) {
Map data = json.decode(response.body);
if (data['Response'] == "True") {
return MovieInfo.fromJSON(data);
} else {
throw Exception(data['Error']);
}
} else {
throw Exception('Something went wrong !');
}
}
```

Код плагіну для перевірки швидкодії у Flutter, код для Android:

```
@Override
public void onMethodCall(@NonNull MethodCall call, @NonNull Result result) {
if (call.method.equals("getRefreshRate")) {
try {
WindowManager windowManager = (WindowManager)
context.getSystemService(Context.WINDOW_SERVICE);
float refreshRate = windowManager.getDefaultDisplay().getRefreshRate();
result.success(refreshRate);
} catch (Exception e) {
result.success(null);
}
102
} else {
result.notImplemented();
}
}
```

Для iOS:

```
- (double)displayRefreshRate:(CADisplayLink *)link {
NSInteger preferredFPS = link.preferredFramesPerSecond;
// maximumFramesPerSecond property.
if (preferredFPS != 0) {
return @(preferredFPS).doubleValue;
}
return @([UIScreen mainScreen].maximumFramesPerSecond).doubleValue;
```

```
}
```

Flutter-код для отримання даних з нативних платформ:

```
static Future<double> get getRefreshRate async {
final double fpsHz = await _channel.invokeMethod('getRefreshRate');
return fpsHz;
}
```

Код плагіну:

```
class FpsPlugin {
static FpsPlugin get instance {
if (_instance == null) {
_instance = FpsPlugin.();
}
return _instance;
}
static Fps _instance;
static const _maxFrames = 120;
final lastTime = ListQueue<FrameTiming>(_maxFrames);
TimingsCallback _timingsCallback;
List<FpsCallback> _callBackList = [];
FpsPlugin._() {
_timingsCallback = (List<FrameTime> timings) {
_computeFps(timings);
};
SchedulerBinding.instance.addTimingsCallback(_timingsCallback);
}
registerCallBack(FpsCallback back) {
_callBackList?.add(back);
}
unregisterCallBack(FpsCallback back) {
_callBackList?.remove(back);
}
cancel() {
if (_timingsCallback == null) {
return;
}
SchedulerBinding.instance.removeTimingsCallback(_timingsCallback);
}
double _fpsHz;
Duration _frameInterval;
```

```
Future<void> _computeFps(List<FrameTiming> lastFrames) async {
for (FrameTime frames in lastFrames) {
lastTime.addFirst(frames);
}
103
while (lastTime.length > _maxFrames) {
lastTime.removeLast();
}
var lastFramesSet = <FrameTime>[];
if (_fpsHz == null) {
_fpsHz = await FpsPlugin.getRefreshRate;
}
if (_frameInterval == null) {
_frameInterval =
Duration(microseconds: Duration.microsecondsPerSecond ~/ _fpsHz);
}
var drawFramesCount = lastFramesSet.length;
// FPS / 60 = drawCount / (drawFramesCount + droppedCount)
// costCount = (drawFramesCount + droppedCount)
// FPS ≈ 60 * drawFramesCount / costCount
int droppedCount = 0;
var costCount = lastFramesSet.map((frame) {
// 15ms ~/ 16ms = 0
// 16ms ~/ 16ms = 0
// 17ms ~/ 16ms = 1
int droppedCount =
(frame.totalSpan.inMicroseconds ~/ _frameInterval.inMicroseconds);
return droppedCount +
1;
}).fold(0, /(a, b) => a + b);
droppedCount = costCount - drawFramesCount;
double fps = drawFramesCount * _fpsHz / costCount;
lastFrames.clear();
_callBackList?.forEach((callBack) {
callBack(fps, droppedCount.toDouble());
});
}
}
```