Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка мобільної гри «Три в ряд» з використанням рушія Godot і GDScript»

Виконав: студент 4 курсу
групи 4ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Гайдей С.О.
(прізвище та ініціали)
Керівник: д.т.н., проф. каф. ПЗ Романюк О.Н.
(прізвище та ініціали)
Рецензент: к.т.н, доц. каф. ОТ Снігур А.В.
(прізвище та ініціали)

23.06.25

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н. _____
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гайдей Софії Олегівні

1. Тема роботи– «Розробка мобільної гри «Три в ряд» з використанням рушія Godot і GDScript»

Керівник роботи: Романюк Олександр Никифорович, д.т.н., проф. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: мова програмування GDScript, середовище розробки Godot.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз алгоритмів розробки мобільних ігор; розробка ключових алгоритмів гри; тестування програми.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Романюк О.Н., д.т.н., професор кафедри ПЗ	24.03.2025	02.06.202

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз аналогів розроблюваного програмного продукту	25.03.25 – 30.03.25	вик.
2	Розробка методів та алгоритмів програмного застосунку	31.03.25 – 16.04.25	вик.
3	Аналіз алгоритмів розробки мобільних ігор	17.04.25 – 07.05.25	вик.
4	Розробка ключових алгоритмів гри	08.05.25 – 24.05.25	вик.
5	Тестування програми	25.05.25 – 27.05.25	вик.
6	Оформлення матеріалів до захисту БКР	28.05.25 – 30.05.25	вик

Студент _____ (підпис) Гайдей С.О. (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) Романюк О.Н (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.932.2:004.93

Гайдей. С.О. Розробка мобільної гри «Три в ряд» використанням рушія Godot і GDScript: бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 73 с.

На укр. мові. Бібліогр. : 35 назв ; рис. : 58; табл. 3

У бакалаврській дипломній роботі проведено всебічний аналіз сучасного стану та тенденцій розвитку мобільних ігор на платформі Android. У рамках дослідження виконано порівняльний аналіз чинних аналогічних ігор, виявлено їхні основні функціональні можливості, переваги та недоліки.

У роботі обґрунтовано доцільність використання ігрового рушія Godot та мови програмування GDScript для реалізації мобільного застосунку, зважаючи на їхню відкритість, кросплатформність, ефективність у створенні 2D ігор і зручність інтеграції з Android. Розроблено мобільну казуальну гру «Три в ряд», яка містить в собі базові ігрові механіки характерні для обраного жанру, адаптивний інтерфейс користувача, а також систему підрахунку балів, збереження прогресу і переходу між рівнями.

Результати дипломної роботи демонструють можливість створення повноцінного, функціонального ігрового продукту на основі відкритих технологій. Напрацьовані підходи можуть бути використані як основа для подальшої розробки подібних мобільних ігор або розширення функціональності наявного проєкту.

Ключові слова: мобільна гра, три в ряд, Godot, GDScript, Android, ігровий рушій, інтерфейс користувача, розробка мобільних ігор, казуальні ігри.

ABSTRACT

UDC 004.932.2:004.93

Haidei S.O.Development of a mobile game «Match 3» using the Godot engine and GDScript: bachelor's qualification work in specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2025. 73 p.

In Ukrainian. Bibliography: 35 titles; fig.: 58; tab. 3

The bachelor's thesis provides a comprehensive analysis of the current state and trends in the development of mobile games on the Android platform. As part of the study, a comparative analysis of existing similar games was performed, their main functional capabilities, advantages and disadvantages were identified.

The paper substantiates the feasibility of using the Godot game engine and the GDScript programming language to implement a mobile application, taking into account their openness, cross-platform, efficiency in creating 2D games and ease of integration with Android. A mobile casual game "Three in a Row" was developed, which includes basic game mechanics characteristic of the chosen genre, an adaptive user interface, as well as a system for calculating points, saving progress and transitioning between levels.

The results of the thesis demonstrate the possibility of creating a full-fledged, functional game product based on open technologies. The developed approaches can be used as a basis for further development of similar mobile games or expanding the functionality of an existing project.

Keywords: mobile game, match-three, Godot, GDScript, Android, game engine, user interface, mobile game development, casual games.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. На даному етапі розвитку інформаційних технологій характеризується стрімким розвитком індустрії мобільних застосунків, зокрема ігор. Ігри для мобільних пристроїв стали не лише способом дозвілля, а й потужним інструментом впливу на масову культуру, освіту та навіть комерцію. В умовах наростальної популярності мобільних платформ, особливо операційної системи Android, спостерігається підвищений інтерес до розробки доступних, захопливих та технічно досконалих мобільних ігор.

Одним із найпопулярніших жанрів у сфері мобільного геймінгу є казуальні ігри, зокрема «Три в ряд». Їх популярність пояснюється простотою механіки, яскравою графікою, інтуїтивним інтерфейсом та здатністю утримувати увагу користувача протягом тривалого часу. Проте багато аналогів мають низку недоліків, серед яких надмірна складність для новачків, високе навантаження на ресурси пристрою, а також одноманітність ігрових механік. З огляду на це, створення власної гри цього жанру є актуальним і виправданим як з технічного, так і з науково-практичного погляду, оскільки дає змогу усунути зазначені недоліки та запропонувати більш збалансований і доступний ігровий продукт.

У рамках бакалаврської кваліфікаційної роботи було прийнято рішення реалізувати проєкт мобільної гри «три в ряд» з використанням відкритого ігрового рушія Godot [1] і мови програмування GDScript [2]. Такий вибір зумовлений низкою факторів, відкритістю платформи, її гнучкістю, підтримкою двовимірної графіки, активною спільнотою розробників та можливістю ефективного розгортання проєктів під Android.

Крім того, розробка гри саме за допомогою Godot та GDScript дозволяє глибше ознайомитися з сучасними підходами до створення інтерактивних додатків, зокрема з об'єктноорієнтованим програмуванням та застосуванням візуального редактора. Ці інструменти є не лише ефективними, а й доступними

для студентів та початківців у галузі дизайну ігор. Таким чином, реалізація проєкту дозволяє не лише створити завершений програмний продукт, а й розвинути професійні компетентності у сфері розробки ігор.

Особливу увагу доцільно приділити користувацькому інтерфейсу, адаптації гри до різних розмірів екранів та забезпеченню стабільної роботи програми на пристроях із різною потужністю. Це відповідає сучасним вимогам до якості мобільного програмного забезпечення та свідчить про комплексний підхід до реалізації теми дослідження.

Таким чином, тем бакалаврської кваліфікаційної роботи у якій розроблені методи та засоби підвищення інтерактивності є актуальним.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою даної роботи підвищення інтерактивності та функціональних можливостей мобільної гри шляхом використання відкритого ігрового рушія Godot та мови програмування GDScript.

Основними задачами дослідження є:

1. здійснити аналіз сучасного ринку мобільних ігор та визначити їх особливості;

2. дослідити методи створення алгоритмів для мобільних ігор;

3. реалізувати базову функціональність гри, включаючи ігрові механіки, систему рівнів та збереження даних гравця;

4. провести комплексне тестування гри та оцінити ефективність використаних засобів.

Технічне завдання на розробку наведено в додатку А.

Об’єкт дослідження — процес створення програмних модулів мобільної гри.

Предмет дослідження — методи та засоби створення модулів мобільної гри.

Методи дослідження. У процесі дослідження використовувались: загальні методи програмної інженерії для побудови архітектури застосунку, методи комбінаторики для формування ігрових комбінацій, теорія алгоритмів для реалізації логіки гри, а також методи обробки псевдовипадкових послідовностей для генерації ігрових рівнів.

Наукова новизна роботи.

1. Подальшого розвитку отримав метод побудови ігрового рівня, заснований на поетапній псевдовипадковій генерації з перевіркою на відсутність готових комбінацій, що дозволяє забезпечити збалансовану складність та підвищити якість гри без додаткових обчислювальних витрат.

2. Подальшого розвитку набув метод створення ігрової логіки з використанням динамічної варіативності на основі випадкових подій, що забезпечує різноманітність ігрових ситуацій, прогнозованість реакцій системи та стійкий інтерес користувача до процесу гри.

Практичне значення отриманих результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень розроблено алгоритми та створено програмні засоби для реалізації базової логіки, генерації рівнів та обробки ігрових подій у мобільній грі.

Особистий внесок здобувача. Усі наукові результати викладені у БДР, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: математичне моделювання комбінаторних систем у розробці казуальних ігор [3], особливості сучасних комп’ютерних ігор [4].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні положення БКР доповідалися та обговорювалися на Міжнародних та Всеукраїнських конференціях: XXVIII Міжнародна науково-технічна конференція «Технологія-2025» (Київ, 2025), XII Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології – 2025» (Київ, 2025).

Структура та обсяг БКР. БКР складається з вступу, чотирьох розділів, висновків, списку літератури, що містить 35 найменувань, 3 додатки. Робота містить 58 ілюстрацій, 3 таблиці. Загальний обсяг роботи складає 103 сторінки, основний зміст викладено на 73 сторінках друкованого тексту.

1 АНАЛІЗ АЛГОРИТМІВ РОЗРОБКИ МОБІЛЬНИХ ІГОР

1.1 Аналіз основних методів побудови мобільних ігор

Розробка мобільних ігор є складним процесом, що поєднує в собі елементи програмування, дизайну, геймдизайну [5], тестування та оптимізації під різні платформи. В умовах швидко наростальної конкуренції на ринку мобільних ігор вибір ефективного методу побудови гри має вирішальне значення. Основні підходи до створення мобільних ігор можна умовно поділити на кілька категорій. До них можна віднести нативну розробку, кросплатформну розробку з використанням ігрових рушіїв, а також розробка за допомогою спеціалізованих конструкторів без необхідності написання великої кількості складного коду програмування.

Нативна розробка мобільних ігор передбачає створення додатків із використанням мов програмування та інструментів, характерних для конкретної платформи Java [6] або Kotlin [7] для Android, Swift [8] або Objective-C [9] для iOS. Цей підхід забезпечує максимальний контроль над апаратними можливостями пристрою, високу продуктивність та доступ до всіх системних API. Проте розробка гри окремо для кожної платформи потребує більше ресурсів, часу і зусиль, а також команди розробників з різною технічною спеціалізацією.

Кросплатформна розробка [10] базується на використанні рушіїв, що дозволяють створювати гру один раз і експортувати її одразу на кілька платформ. Серед найпопулярніших таких рушіїв є Unity [11], Unreal Engine [12], Godot, Defold [13], Cocos2d-x [14] тощо. Застосування цих інструментів значно скорочує час розробки, зменшує витрати та полегшує підтримку гри. При цьому рушії надають широкий спектр інструментів для роботи з графікою, фізикою, анімацією, звуком та інтерфейсом користувача.

Особливої уваги заслуговує рушій Godot, який активно набирає популярності через відкритий вихідний код, безплатну ліцензію, активну спільноту та простий у використанні скриптовий інтерфейс GDScript, спеціально створений для потреб ігрової розробки. Godot дозволяє створювати як 2D, так і 3D ігри, забезпечуючи при цьому зручне середовище розробки, підтримку сценової структури та можливість використання візуального редактора. Завдяки своїй архітектурі рушій особливо ефективний для створення 2D проєктів, включно з іграми типу «три в ряд» (див. рисунок 1.1).

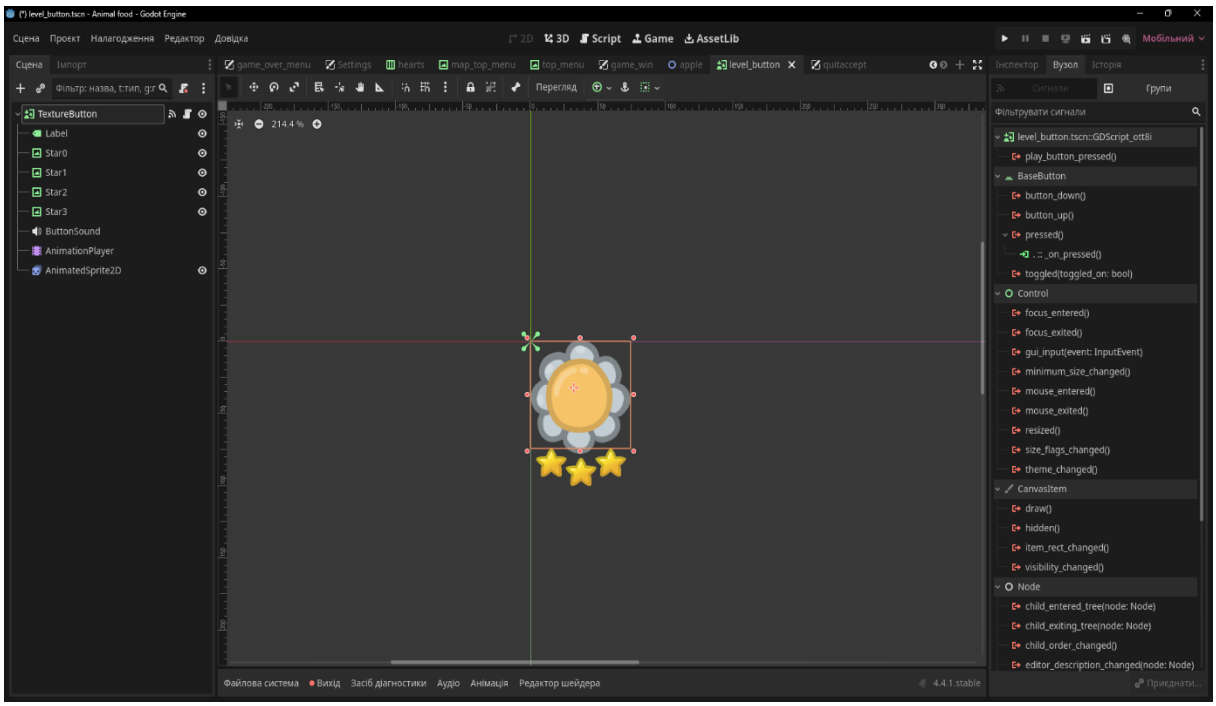

Рисунок 1.1 – інтерфейс Godot

Окрему категорію становлять так звані no-code або low-code платформи, такі як Buildbox [15], Gdevelop [16] або GameSalad [17]. Вони орієнтовані на користувачів без досвіду програмування і дозволяють створювати прості мобільні ігри шляхом компонування елементів за допомогою візуального інтерфейсу. Хоча ці інструменти значно полегшують вхід у розробку ігор, вони мають суттєві обмеження у функціональності та гнучкості проєктів. У контексті розробки повноцінної гри з нестандартною логікою та оптимізацією ці платформи часто поступаються традиційним рушіям.

Методи побудови мобільних ігор також відрізняються за рівнем абстракції. Наприклад, деякі рушії надають високорівневі інструменти, які дозволяють швидко створювати прототипи без необхідності писати багато коду, у той час, як інші орієнтовані на низькорівневу оптимізацію і точний контроль за усіма можливими процесами в грі. Вибір відповідного підходу залежить від посталених цілей проєкту, досвіду розробника, цільової платформи та функціональних вимог до гри (див. рисунок 1.2).

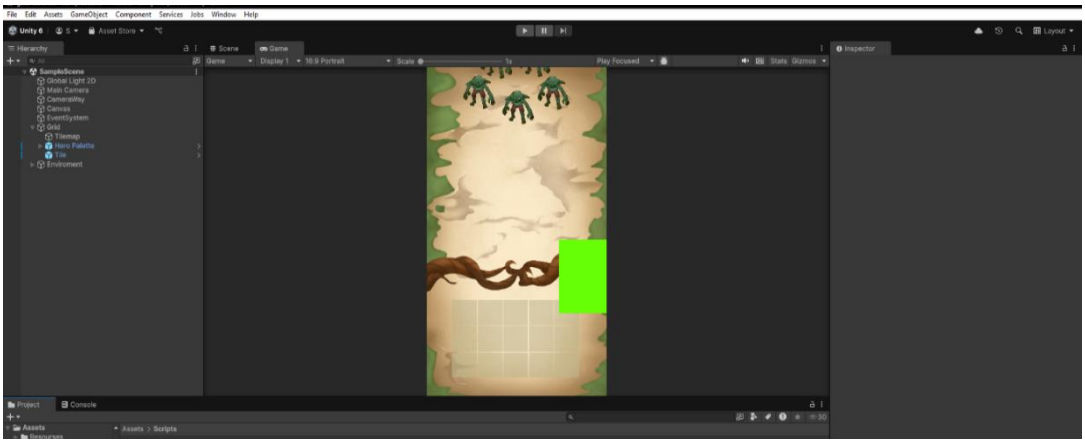

Рисунок 1.2 – Створення прототипу гри в Unity

Крім інструментів та підходів, важливе значення має і структура самого ігрового проєкту. Більшість сучасних рушіїв підтримують модульну або сценову архітектуру, яка дозволяє робити гру з окремих компонентів (див. рисунок 1.3).

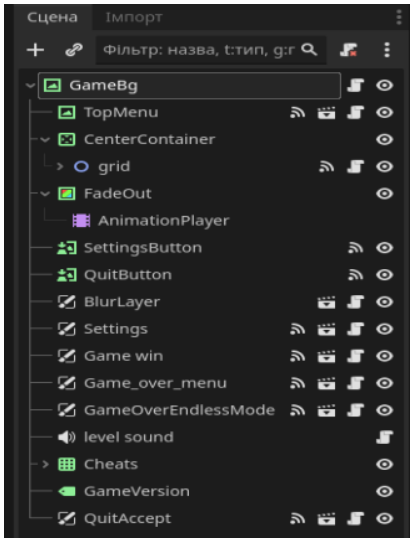

Рисунок 1.3 – Використання сцен в Godot

Це сприяє кращій масштабованості, тестуванню та повторному використанню коду. У випадку ігрового рушія Godot така структура реалізується через вузли та сцени, які можна вкладати один в одного для створення складних ієрархій (див. рисунок 1.4).

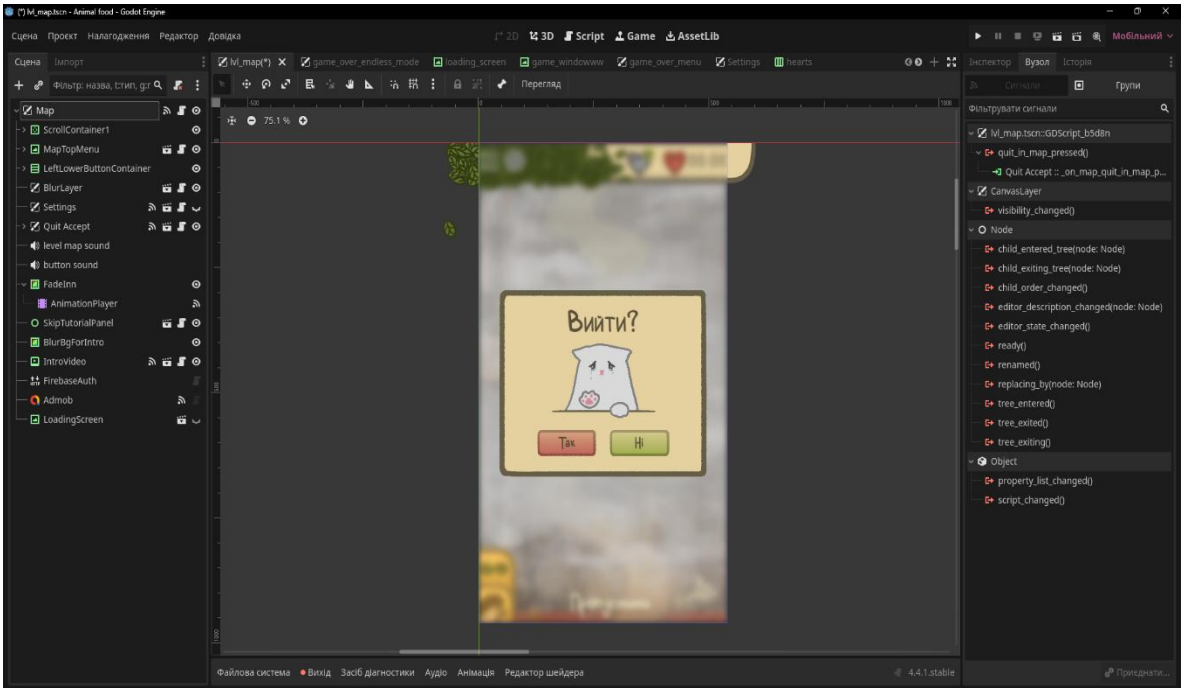

Рисунок 1.4 – Створення головної сцени гри яка містить в собі багато вузлів та дочірніх сцен

Слід також враховувати питання продуктивності, особливо у випадку розробки для мобільних пристроїв, які мають обмежені ресурси в порівнянні з настільними комп’ютерами, ноутбуками або консолями. Покращення ігрового процесу, графіки, обробки подій, а також мінімізація споживання енергії є критично важливими аспектами. Саме тому під час вибору рушія або підходу до розробки варто враховувати не лише зручність реалізації, а й чисельні технічні обмеження цільової платформи. Godot у цьому плані демонструє хороші результати завдяки легкій архітектурі та можливості тонкого налаштування сцен і моніторингу ресурсів.

Ще одним важливим аспектом у побудові мобільних ігор є підтримка інструментів для швидкого тестування і розгортання проєктів. У багатьох

сучасних рушіях, передбачено можливість миттєвого запуску гри без необхідності повної збірки проєкту. Це значно пришвидшує цикл розробки, дозволяє оперативно вносити зміни та спрощує процес налагодження. Крім того, сучасні ігрові рушії підтримують експортування проєкту на різні мобільні платформи, зокрема Android та iOS, із мінімальними налаштуваннями, що особливо актуально для інді розробників та невеликих команд.

У процесі побудови мобільних ігор також важливо враховувати монетизацію, підтримку локалізації, збору аналітики та інтеграцію з платформними сервісами, наприклад, Google Play Services, App Store, рекламою, push-повідомленнями тощо. Хоча не всі рушії забезпечують ці функції за замовчуванням, багато з них підтримують відповідні плагіни або дозволяють додавати SDK від сторонніх розробників.

Таким чином, методи побудови мобільних ігор мають широкий спектр реалізацій, від нативного програмування до повноцінних ігрових рушіїв та конструкторів. Кожен підхід має свої переваги й недоліки, які потрібно враховувати при виборі технології для створення мобільної гри. Для невеликих проєктів або індивідуальних розробників особливо привабливими є безплатні ігрові рушії з відкритим вихідним кодом, які не обмежують доступ до всієї функціональності рушія та дозволяють повністю контролювати весь процес розробки.

Отже, проведений аналіз основних методів розробки мобільних ігор показує, що сучасні методи побудови мобільних ігор охоплюють широкий спектр підходів від нативної розробки до використання кросплатформних ігрових рушіїв і візуальних конструкторів. Кожен із них має свої сильні сторони та обмеження, які впливають на вибір технології залежно від цілей проєкту, технічних вимог, бюджету та рівня підготовки команди розробників. Нативна розробка забезпечує максимальний контроль і продуктивність, але вимагає значних ресурсів. Кросплатформні рушії дозволяють значно скоротити час і зусилля, необхідні для створення гри на декількох платформах одночасно. Конструктори без коду зручні для швидкого створення простих проєктів, проте

мають обмежену функціональність. Таким чином, вибір оптимального методу розробки мобільної гри має ґрунтуватися на комплексному врахуванні технічних, організаційних та творчих чинників.

1.2 Основні вимоги до розробки мобільних ігор

Основні вимоги до розробки мобільних ігор формуються з урахуванням комплексного поєднання геймплею графічного оформлення технічної стабільності та зручності використання на широкому спектрі пристроїв. Першим завданням є створення цікавого геймплею що залучає гравця з перших хвилин і спонукає до довготривалої взаємодії. В ігровій механіці слід передбачити плавну криву складності збалансований прогрес та систему мотивації через досягнення або віртуальні винагороди. При цьому підхід до дизайну рівнів і сценаріїв гри має враховувати особливості сенсорного управління [18] що дозволить уникнути незручностей під час торкання екрана (див. рисунок 1.5).

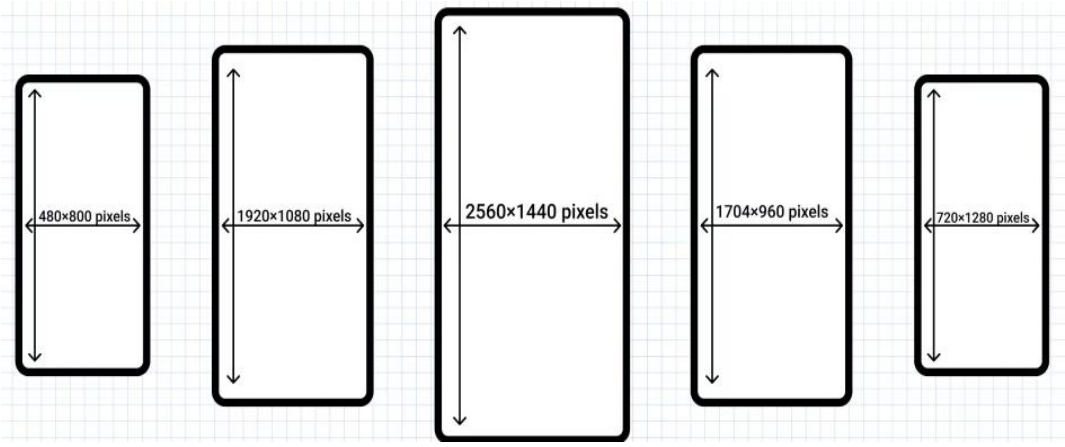

Рисунок 1.5 – Порівняння різних екранів телефонів

Візуальна частина мобільної гри повинна поєднувати естетику та продуктивність. Для двовимірних елементів використовуються 2д спрайти які оптимізуються під різні роздільні здатності екранів, а для динамічних ефектів застосовують 3д моделі або шейдерні рішення. Важливо налаштувати рівні деталізації ресурсів, щоб зменшити навантаження на графічний адаптер і

забезпечити рівномірну частоту кадрів. Незалежно від типу графіки художній стиль і колірна гама мають створювати впізнаваний образ гри (див. рисунок 1.6).

Рисунок 1.6 – Підготовка 2д спрайтів

У технічному вимірі продуктивність та оптимізація відіграють ключову роль. Розробник контролює кількість активних об’єктів у кадрі використовує механізми кешування текстур [19] і звукових файлів, а також активно застосовує інструменти профілювання, щоб виявити ділянки коду що сповільнюють роботу. Додаткові випробування на пристроях із різними характеристиками дозволяють вчасно коригувати налаштування якості візуального відтворення та цикл обробки логіки гри. Збалансований підхід запобігає перегріву смартфонів і передчасному розрядженню акумулятора.

Користувацький інтерфейс і система управління є невіднятною частиною якісного досвіду користування. Усі елементи інтерфейсу розміщуються таким чином, щоб не перекривати важливу частину ігрового поля, а натомість

забезпечувати швидкий доступ до основних функцій. Розміри кнопок відповідають рекомендаціям виробників операційних систем що зменшує ризик ненавмисних натискань. Взаємодія з меню налаштувань та інформаційними вікнами відбувається без значних затримок що підвищує комфорт під час ігрової сесії (див. рисунок 1.7).

Рисунок 1.7 – Приклад вдалого дизайну користувацького інтерфейсу

Кросплатформність та адаптивний дизайн [20] передбачають можливість перенесення гри на різні операційні системи мобільні та настільні без значного перероблення коду. Для експорту під Android і iOS застосовуються відповідні інструменти що забезпечують коректне масштабування графіки та перенастроювання управління з урахуванням сенсора або миші й клавіатури при портуванні на комп’ютери. Конфігурація проєкту дозволяє гнучко контролювати параметри рендерингу і вводити різні зміни в інтерфейс незалежно від платформи.

Звукове оформлення відіграє важливу роль у створенні атмосфери. Кожна дія гравця супроводжується відповідним аудіоефектом в той час, як фоновий музична доріжка підкреслює загальний настрій, а не відриває від процесу гри. В налаштуваннях пропонується регулювати рівень музики та звукових сигналів окремо, щоб кожен користувач мав можливість адаптувати звучання під власні вподобання (див. рисунок 1.8).

Рисунок 1.8 – Регулювання гучності різних звукових доріжок

Нефункціональні вимоги [21] включають безпеку обробки даних локалізацію інтерфейсу та збір аналітики. Забезпечення захищеного збереження прогресу гравця або авторизації через зовнішні сервіси здійснюється за допомогою протоколів що відповідають міжнародним стандартам. Локалізація текстів сприяє залученню глобальної аудиторії. Інтеграція аналітичних модулів

дозволяє відстежувати показники залученості виявляти зони відтоку гравців та приймати обґрунтовані рішення щодо оновлення контенту (див. рисунок 1.9).

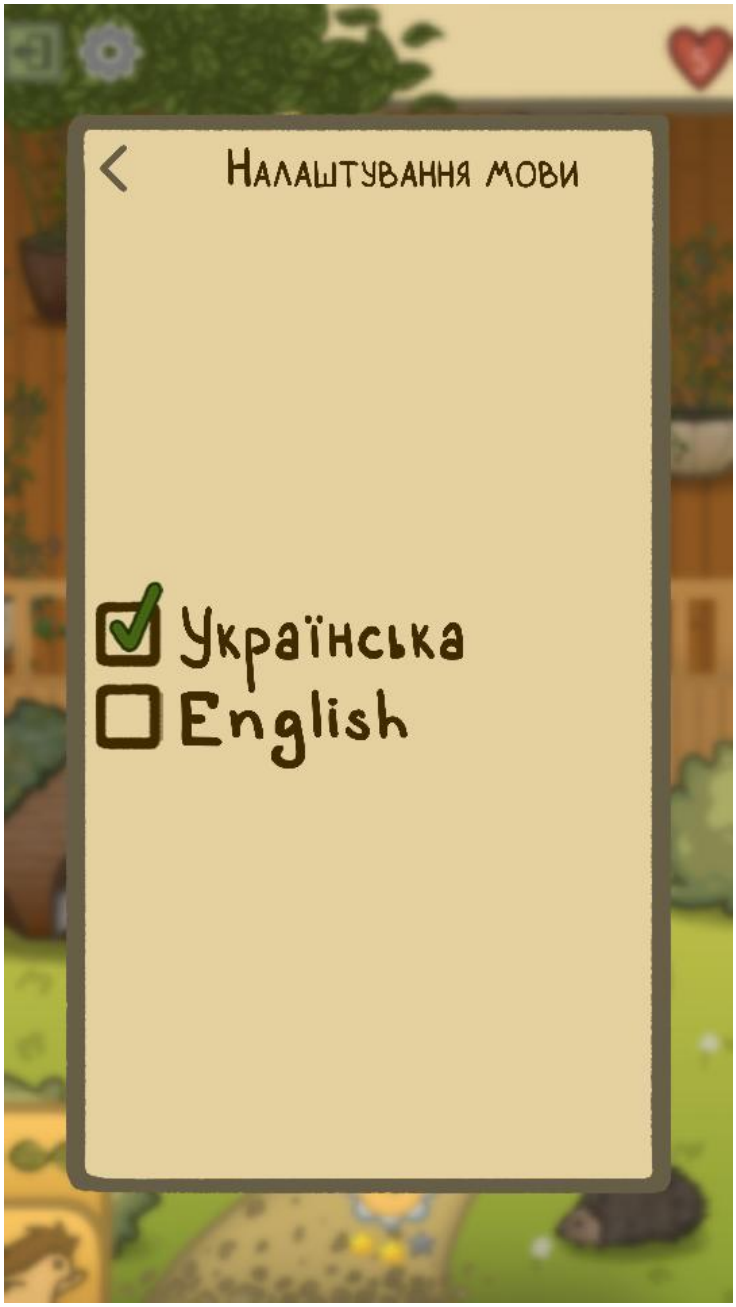

Рисунок 1.9 – Вибір мови

Підтримка коду та масштабованість проєкту досягаються через модульну архітектуру. При розділенні відповідальності між блоками обробки ігрового процесу відображення інтерфейсу та взаємодії з сервером команді простіше

вносити зміни та виправляти помилки. Документування ключових компонентів системи скорочує час адаптації нових розробників до проєкту.

У сучасному середовищі мобільних додатків успішність гри визначається не лише її ідеєю та сюжетом, а насамперед відповідністю низки ключових вимог, які формують загальну якість продукту та задоволення користувача. Першим завданням розробника є створення гармонійного поєднання захопливого ігрового процесу та інтуїтивно зрозумілого інтерфейсу що враховує особливості сенсорного керування. Збалансована крива складності підтримує мотивацію гравця і водночас дозволяє уникнути надмірних фрустрацій у перші хвилини знайомства з грою.

Візуальна частина та звуковий дизайн формують перше враження і задають тональність всього досвіду. Використання 2д спрайтів і 3д ефектів повинно бути обґрунтованим і оптимізованим для різних пристроїв аби забезпечити плавну анімацію та стабільну частоту кадрів. Паралельно з цим підбір звукових ефектів і музичних тем має вдало підкреслювати атмосферу гри не відволікаючи від головного завдання гравця.

Технічна частина розробки охоплює не лише оптимізацію ресурсів і профілювання продуктивності, а й забезпечення безпеки даних локалізацію інтерфейсу та збір аналітики для подальшого вдосконалення продукту. Кросплатформності та масштабованості системи надають можливість адаптувати гру для різних операційних систем і формфакторів без значних змін у кодовій базі. Лише комплексний підхід до кожного з перерахованих аспектів дає змогу створити конкурентоспроможний і довготривалий в експлуатації проєкт.

1.3 Аналіз сучасних ігрових рушіїв

Аналіз сучасних ігрових рушіїв передбачає розгляд трьох найпопулярніших рішень на ринку Unity, Unreal Engine та Godot.

Unity здобув широке визнання завдяки своїй простоті початкового освоєння та масивній екосистемі плагінів. Інструменти для розробки двовимірних проєктів у Unity створюють враження швидкого старту, а вбудовані засоби візуального редагування сцен спрощують роботу дизайнера. Проте з ростом складності проєкту виявляються слабкі місця в системі обробки великих тривимірних сцен і обмеження в ліцензійних умовах для комерційного використання на високих рівнях доходу (див. рисунок 1.10).

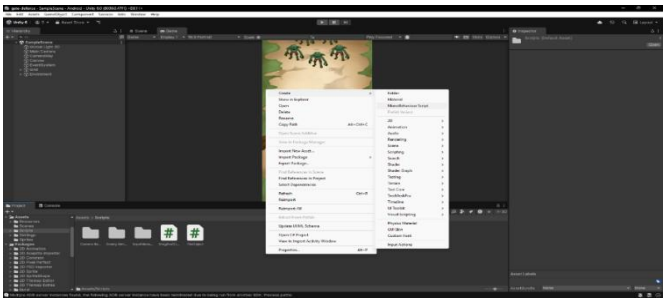

Рисунок 1.10 – Інтерфейс Unity

Unreal Engine вирізняється потужним рушієм рендерингу і дивовижним готовими рішеннями для створення фотореалістичного оточення. Розробники, які мають досвід у C++ [22] або прагнуть використати візуальне програмування через Blueprints [23], знайдуть Unreal Engine привабливим завдяки гнучкому контролю над кожним аспектом гри (див. рисунок 1.11).

Рисунок 1.11 – Інтерфейс Unreal Engine

Водночас новачкам доведеться стикнутися з високим порогом входження та значними апаратними вимогами вже на стадії прототипування, а обсяг вихідного коду рушія може ускладнити розуміння внутрішніх механізмів.

Godot постає альтернативою з відкритим кодом і вільною ліцензією, що не накладає жодних обмежень на рівні доходу чи розповсюдження. Цей рушій однаково добре підходить як для двовимірних, так і для тривимірних ігор, пропонуючи легку вагу та швидке завантаження редактора. Мова програмування GDScript створена з урахуванням потреб ігрових розробників вона синтаксично проста та зрозуміла навіть тим, хто раніше не працював з мовами сценаріїв [24]. Ця відкритість рушія сприяє формуванню активної спільноти користувачів що регулярно долучають нові модулі й удосконалюють наявні (див. рисунок 1.12).

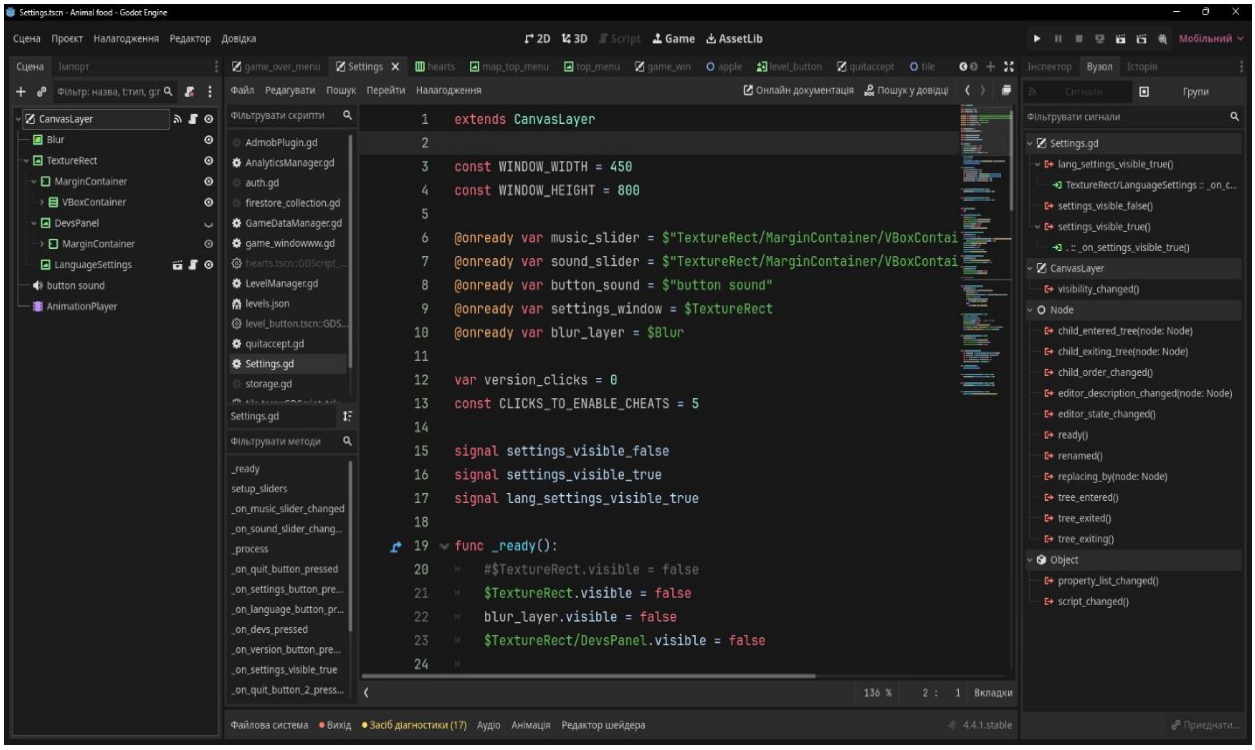

Рисунок 1.12 – Інтерфейс Godot

У Godot відсутній централізований магазин плагінів, однак користувачі швидко об’єднують власні напрацювання на GitHub та інструментах для керування версіями. Це створює необхідність ретельнішого відбору зовнішніх модулів, але з іншого боку підвищує загальний рівень контролю над проєктом і дозволяє уникнути тягаря надмірних залежностей. Крім того, Godot забезпечує гнучкий механізм експортному пакету для різних платформ включаючи Windows

macOS Linux Android iOS веб та інші цільові середовища без додаткових ліцензійних відрахувань (див. рисунок 1.13).

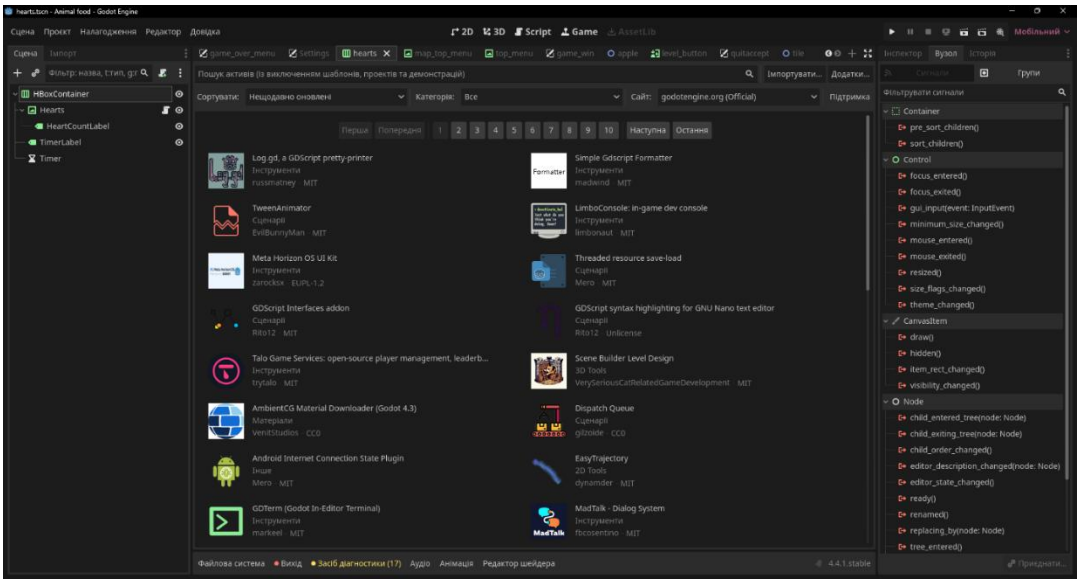

Рисунок 1.13 – Плагіни в Godot

Щодо продуктивності обробки двовимірної графіки Godot демонструє дуже добрі результати навіть на бюджетних пристроях, а в галузі тривимірної графіки постійно вдосконалює рендерингові алгоритми. Unity і Unreal Engine можуть запропонувати вищу продуктивність шляхом пропрієтарних оптимізацій, однак у Godot завдяки відкритому коду можливе швидке виявлення та виправлення вузьких місць у самому рушії. Важливим аспектом є також мобільна оптимізація де вага готового пакету вирішує багато питань щодо часу завантаження та пам’яті.

Вибір ігрового рушія часто визначається критеріями довгострокової підтримки, можливістю швидкої адаптації та потенціалом масштабування проєкту. При оцінці слід враховувати не лише поточні потреби, але й перспективу розвитку, чи буде рушій отримувати активні оновлення, як швидко реагують розробники двигуна на запити спільноти, і наскільки легко інтегрувати власні рішення до базової архітектури.

Порівняння функціональних характеристик сучасних ігрових рушіїв Unity, UnrealEngine та Godot наведено в таблиці 1.1.

Таблиця 1.1 – Таблиця порівняння функціональних характеристик сучасних ігрових рушіїв

Характеристика	Unity	UnrealEngine	Godot
Ліцензійні умови	Безкоштовна версія з обмеженнями доходу	Безкоштовна з роялті понад певний рівень доходів	Повністю безкоштовний
Початкова крива навчання	Низька	Висока	Низька
Підтримка двовимірної графіки	Висока	Середня	Висока
Оптимізація та продуктивність	Добра з пакетом Pro	Відмінна	Дуже добра на бюджетних пристроях
Спільнота та ресурси	Велика екосистема Asset Store	Активна спільнота і Marketplace	Швидкозростаюча спільнота
Кросплатформений експорт	Багатоплатформений	Багатоплатформений	Надзвичайно гнучкий
Підтримка мобільних платформ	Повна	Повна	Повна
Можливість модифікації рушія	Обмежена кодом Unity	Обмежена	Повна завдяки відкритому коду

У цій таблиці видно що Godot поєднує в собі переваги відкритої ліцензії простоти освоєння широкої підтримки платформ та активної спільноти що робить його найвдалішим вибором для більшості мобільних проєктів.

1.4 Висновки

У цьому розділі було розглянуто основні підходи до розробки мобільних ігор, виділені їхні переваги та обмеження в контексті нативної, кросплатформної та схеми no-code розробки. Аналіз показав, що нативна розробка забезпечує максимальний контроль над апаратними ресурсами та найвищу продуктивність, але водночас потребує більшої кількості ресурсів і залучення спеціалізованих команд для кожної платформи. Кросплатформні рушії надають швидкий старт і можливість єдиного коду для Android і iOS, що значно скорочує терміни виробництва й полегшує підтримку, проте може вимагати компромісів у контролі над низькорівневими оптимізаціями. Платформи з візуальними конструкторами демонструють переваги в простоті використання для початківців чи для створення прототипів, але їхня функціональність і гнучкість часто виявляються недостатніми для реалізації складних ігор.

Детальний огляд вимог до мобільних ігор висвітлив ключові аспекти, які визначають якість кінцевого продукту, продуманий геймплей із плавною кривою складності та системою мотивації, оптимізоване графічне оформлення у поєднанні зі стабільною продуктивністю, зручний інтерфейс, адаптований під сенсорне й периферійне управління, а також кросплатформність і масштабованість. Було підкреслено, що успіх мобільної гри залежить від уміння балансувати між естетикою, технічними обмеженнями пристроїв і потребами користувачів, що вимагає комплексного підходу на всіх етапах розробки, від прототипування до розгортання та підтримки.

Порівняльний аналіз рушіїв Unity, Unreal Engine та Godot показав, що кожен із них має свої сильні сторони. Unity вирізняється розвиненою екосистемою й доступністю для 2D проєктів, Unreal Engine потужним рендерингом і візуальними інструментами, тоді як Godot поєднує відкриту ліцензію, простоту освоєння через GDScript і широкі можливості кросплатформного експорту. Саме Godot забезпечує найбільшу гнучкість, від модульності коду та можливості модифікації рушія до ефективної підтримки та

2D, і 3D ігор, що робить його оптимальним вибором для реалізації мобільного казуального проєкту жанру «три в ряд».

Необхідно також врахувати роль інтеграції з користувачем і зворотного зв’язку у процесі підтримки та розвитку гри. Активне залучення гравців через оновлення контенту, сезонні події або невеликі тематичні доповнення забезпечує довготривалу зацікавленість і формує відчуття спільноти навколо проєкту. Важливо передбачити механізми швидкого розгортання змін, як у вигляді свіжих патчів, так і масштабних оновлень, що дозволить оперативно реагувати на відгуки та випускати нові можливості без значних простоїв сервісу.

Також слід звернути увагу на бізнес-модель і монетизацію, адже без чіткого розуміння способів отримання доходу проєкт ризикує залишитися нерентабельним. Моделі «free-to-play» з механіками внутрішньоігрових покупок чи валютою та різноманітними пропозиціями користувачам мають бути ретельно збалансовані, щоб не зруйнувати ігровий досвід. Одночасно коректна аналітика фінансових показників, від кількості транзакцій до середнього чека створює міцну основу для планування майбутніх оновлень та розширення аудиторії.

Отже, узагальнивши всі розглянуті аспекти, можна констатувати, що для розробки конкурентоспроможної мобільної гри необхідно обирати той рушій та методологію, які найкраще відповідають конкретним завданням проєкту, ресурсам команди та бізнес-цілям. Водночас для дипломної роботи, мета якої створити адаптивну, легку в освоєнні та технічно досконалу казуальну гру, рушій Godot із GDScript є найбільш обґрунтованим вибором.

2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ ГРИ

2.1 Розробка алгоритмів побудови рівнів ігор жанру три в ряд

Розробка ігрового поля для мобільної казуальної гри жанру три в ряд вимагає ретельного балансу між випадковістю й контрольованою складністю. На початковому етапі створення рівня необхідно сформувати матрицю фішок таким чином, щоб гравець не отримав автоматичних комбінацій ігрових елементів одразу після завантаження рівня. Інакше гра втрачає сенс, гравець одразу одержує бонуси без жодних зусиль. Тому більшість комерційних рішень спираються на каскадний метод генерації. Кожну клітинку заповнюють випадковим чином, а потім здійснюють перевірку розташування навколо неї. Якщо утворюється комбінація з трьох або більше однакових фішок, алгоритм відкидає поточне значення та вибирає нове. Цей підхід гарантує відсутність готових ланцюжків і дозволяє миттєво контролювати базову конфігурацію поля.

Удосконалити якість початкової генерації допомагає застосування псевдовипадкових числових послідовностей, які дозволяють створювати контрольовану стохастику. В цьому випадку для цілей генерації використовують функції випадкового числа з фіксованим початковим зерном. Завдяки цьому при однаковому початковому значенні зерна гра відтворює ідентичні конфігурації поля, що спрощує налагодження та тестування. Водночас зміна параметрів зерна дає змогу отримати різні варіанти рівня з близькою до бажаної частотою появи бонусних комбінацій. Серед переваг цього підходу є можливість зберігати й відтворювати успішні комбінації або ж навпаки аналізувати особливо вдалий розподіл фішок для подальшого вдосконалення балансу.

Попри достатню ефективність поетапної перевірки комбінацій, її використання ускладнюється при високій кількості різновидів фішок. Зі збільшенням різноманіття ігрових елементів складність росте експоненційно. Щоб запобігти затримкам при завантаженні рівня, в ігрових рушіях

застосовують пакетну перевірку [25]. Пакетна перевірка полягає в тому, що весь масив згенерованих значень проходить через один прохід алгоритму визначення ланцюжків. За допомогою сканування рядків і стовпців одночасно в одному циклі фільтрують усі трійки. Коли на етапі первинної генерації знаходяться поодинокі порушення умови відсутності готових ланцюжків, вони усуваються під час другого короткого прохідного етапу. Цей механізм значно економить ресурси мобільного пристрою та мінімізує паузи на етапі початку рівня.

Процедурна генерація [26] з використанням шумових алгоритмів забезпечує плавність переходів між різними зонами складності та додає рівню характер. Функція шуму Перліна [27] дає змогу створити карту інтенсивності появи бонусних фішок або перешкод. Наприклад у нижній частині матриці значення Перліна може бути задане таким чином, що інтенсивність бонусних елементів мінімальна, тоді як у центрі або біля верхнього краю матриці вона зростає. Завдяки такому підходу можна домогтися природного візуального розподілу зон з високою ймовірністю появи спеціальних комбінацій на фоні загального випадкового заповнення. Цей принцип використовують як у класичних міні іграх, так і у великих казуальних проєктах, де дизайн рівня диктує необхідність поступового нарощування складності (див. рисунок 2.1).

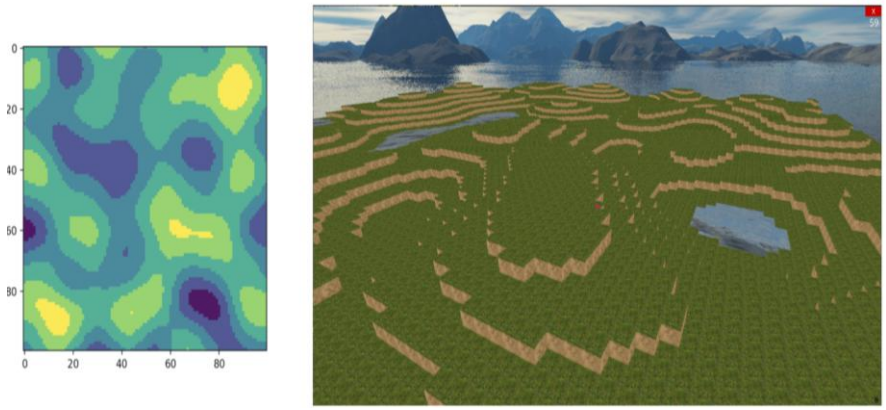

Рисунок 2.1 – Застосування шуму Перліна у відомій грі «Minecraft»

Кластерний аналіз [28] може стати корисним доповненням до процедурних та комбінаційних методів при побудові особливо динамічних рівнів. Спочатку

ігрове поле розбивають на кластери розміром від трьох на три до шести на шість клітинок. У межах кожного кластеру аналізують частоту появи однойменних фішок і при виявленні надмірної однорідності застосовують локальну генерацію. Завдяки цьому методу зберігається випадковість у межах окремих секторів, але загальна матриця виглядає більш структуровано та естетично. Кластери із різними характеристиками можна домінувати окремими видами бонусів або перешкод, забезпечуючи додатковий рівень геймплейного челенджу.

Більш складні механізми оптимізації використовують моделі регресійного аналізу або стохастичні методи оптимізації. Наприклад, для автоматичного налаштування складності рівня застосовують лінійну регресію [29], яка аналізує залежність середньої кількості ходів до завершення від кількості фішок на полі. На основі цих даних система може коригувати параметри генератора так, щоб досягалася середня метрика швидкості проходження на рівні на рівні 70 відсотків успішності. Це дозволяє автоматично адаптувати ігрове поле під різний рівень досвіду гравця.

Ще одним перспективним напрямом є використання генетичних алгоритмів для еволюційного вдосконалення конфігурацій. Генетичний алгоритм створює спочатку великий пул кандидатів на стартові поля, оцінює їх за критерієм складності або часу проходження, а потім відбирає кращі конфігурації. На наступному кроці відбувається мутація та кросинговер матриць фішок, формуючи нові покоління. З часом система здатна оптимізувати такі параметри як рівень інтенсивності бонусів і частота появи перешкод. Цей підхід особливо корисний для турнірних режимів або сезонних подій, де дуже важливо підтримувати високий рівень взаємодії гравця.

Нарешті, інтеграція комбінованих підходів дає змогу синтезувати найкраще з кожного алгоритму. Наприклад, спочатку застосовувати каскадний метод із перевіркою на готові ланцюжки для первинної генерації, потім використовувати карту шуму Перліна для налаштування зон бонусних елементів, а на фінальному етапі здійснити кластерний аналіз та оптимізацію за допомогою регресійного методу. Така багаторівнева система дає можливість

створити максимально збалансований і водночас різноманітний рівень, здатний зацікавити гравця протягом усього часу проходження.

Таким чином, аналіз алгоритмів побудови ігрового поля в жанрі ігор три в ряд вимагає врахування низки факторів від контролю початкового розподілу фішок до процедурних методів формування зон і автоматичної оптимізації складності. Застосування комбінованих рішень дозволяє реалізувати безперервний цикл генерації рівнів, що адаптується під потреби гравця і зберігає баланс між викликом і винагородою. Це створює міцний фундамент для подальшої реалізації мобільної казуальної гри на рушії Godot із використанням GDScript.

2.2 Розробка методу динамічної варіативної логіки гри

У центрі внутрішньої логіки розроблюваної гри «три в ряд» лежить ітеративний цикл взаємодії користувача з ігровим полем, в якому кожен рух фішок ретельно обробляється з урахуванням наявності комбінацій та можливості повернення ходу назад. На початковому етапі алгоритму відбувається ініціалізація ігрового поля й очікування дії гравця, він обирає дві сусідні фішки для переставляння місцями. Після вибору система виконує функцію перестановки, яка змінює положення обраних елементів і готує поле до перевірки.

Якщо перестановка не призвела до утворення хоча б однієї групи з трьох і більше однакових фішок у суцільному ряду або стовпці, запускається процедура повернення ходу назад. У цьому випадку початкові позиції обраних фішок відновлюються, а користувачеві повертається право зробити нову спробу. Такий підхід не дозволяє виконати хід, який не утворює комбінацій, й гарантує, що кожен успішний рух приносить ігрову винагороду у вигляді видалення фішок та нарахування додаткових балів.

Якщо ж після перестановки виявлено одну або більше комбінацій з трьох і більше фішок одного типу, алгоритм переходить до етапу пошуку відповідних

комбінацій. Ця функція здійснює сканування всіх рядків і стовпців і формує перелік груп, які необхідно знищити. Після складання списку відповідних фрагментів виконується їх видалення з поля та активується механізм падіння верхніх елементів у звільнені клітини. Водночас у верхньому ряду з’являються нові фішки, згенеровані згідно з обраним методом заповнення, що забезпечує безперервність циклу гри (див. рисунок 2.2).

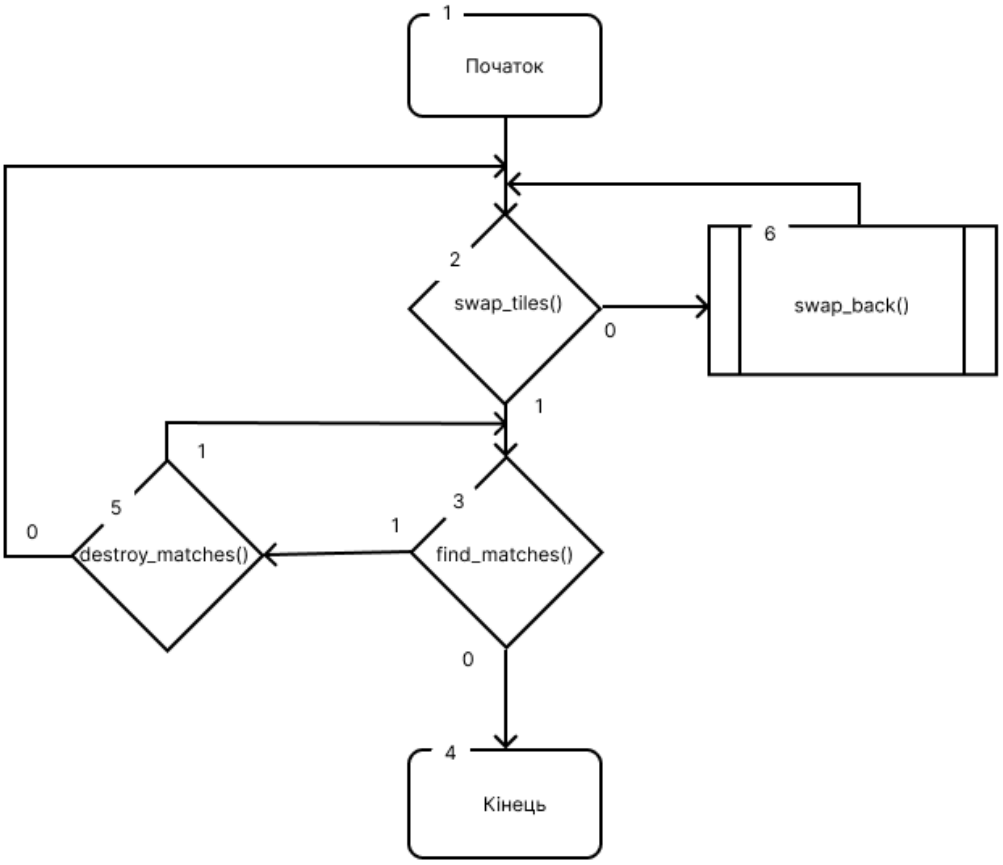

Рисунок 2.2 – Блок-схема головного алгоритму гри

Після модифікації розміщення фішок на полі з урахуванням гравітації запускається повторний цикл пошуку готових комбінацій. Така каскадна обробка гарантує видалення ланцюгових реакцій, коли одна падіння іноді запускає появу нових груп у добудованому полі. Кожний такий прохід вкладається у внутрішній цикл до моменту, коли перевірка не виявить жодної нової трійки фішок. У разі відсутності додаткових груп ітерація завершується, а управління передається назад до очікування дії гравця.

Наприкінці кожного завершеного циклу проводиться перевірка стану гри на предмет задоволення умов перемоги або вичерпання доступних ходів. Якщо користувачу вдалось виконати завдання рівня, наприклад зібрати певну кількість фішок заданого типу, генерується повідомлення про переможне завершення рівня і відбувається перехід до екрана підсумків. У випадку відсутності успішних комбінацій та одночасного закінчення ходів спрацьовує екран поразки, після чого гравцеві пропонується спробувати пройти рівень ще раз.

Таким чином, основний алгоритм гри складається з циклічних етапів перестановки, перевірки наявності комбінацій, їх видалення разом із подальшим падінням та генерацією нових фішок, перевірки завершення каскаду й оцінки результату. Така структура забезпечує безперервний динамічний процес гри, коли кожен хід гравця впливає на зміни на полі й створює нові можливості для формування виграшних груп, підтримуючи цікавість та залученість користувачів.

2.3 Розробка методу ігрової логіки на основі станів

Загальний алгоритм роботи мобільної казуальної гри жанру «три в ряд» починається з ініціалізації ігрового середовища і формування початкового розміщення фішок на ігровому полі. Після завантаження рівня гравець отримує можливість вибору двох сусідніх елементів для обміну їх місцями. Якщо обмін не призводить до утворення комбінацій з трьох або більше фішок одного типу в рядку чи стовпці, система автоматично повертає обрані елементи у вихідні позиції та дозволяє гравцеві виконати новий хід. У разі ж виникнення комбінацій відбувається пошук усіх груп, призначених до видалення, їх знищення та подальше падіння елементів згори з одночасною генерацією нових фішок. Цей процес повторюється доти, поки на полі не припиняться каскадні утворення готових ланцюжків. По завершенню каскаду здійснюється перевірка стану рівня за умовами успішного виконання цілі або вичерпання ходів.

Для візуалізації логіки алгоритму та ключових переходів між станами було розроблено декілька діаграм станів. Перша діаграма ілюструє основний цикл взаємодії користувача з ігровим полем і включає початковий етап очікування ходу, перевірку правильності ходу та виконання процедур swap tiles find matches і destroy matches. Спочатку перевіряється результат функції swap tiles якщо повертається значення що комбінацію не знайдено відбувається виклик процедури swap back після чого управління повертається на початок циклу. Якщо ж комбінація знайдена запускається модуль пошуку трійок та видалення груп фішок який у разі успіху знову звертається до перевірки готових комбінацій і так до моменту припинення каскаду (див. рисунок 2.3).

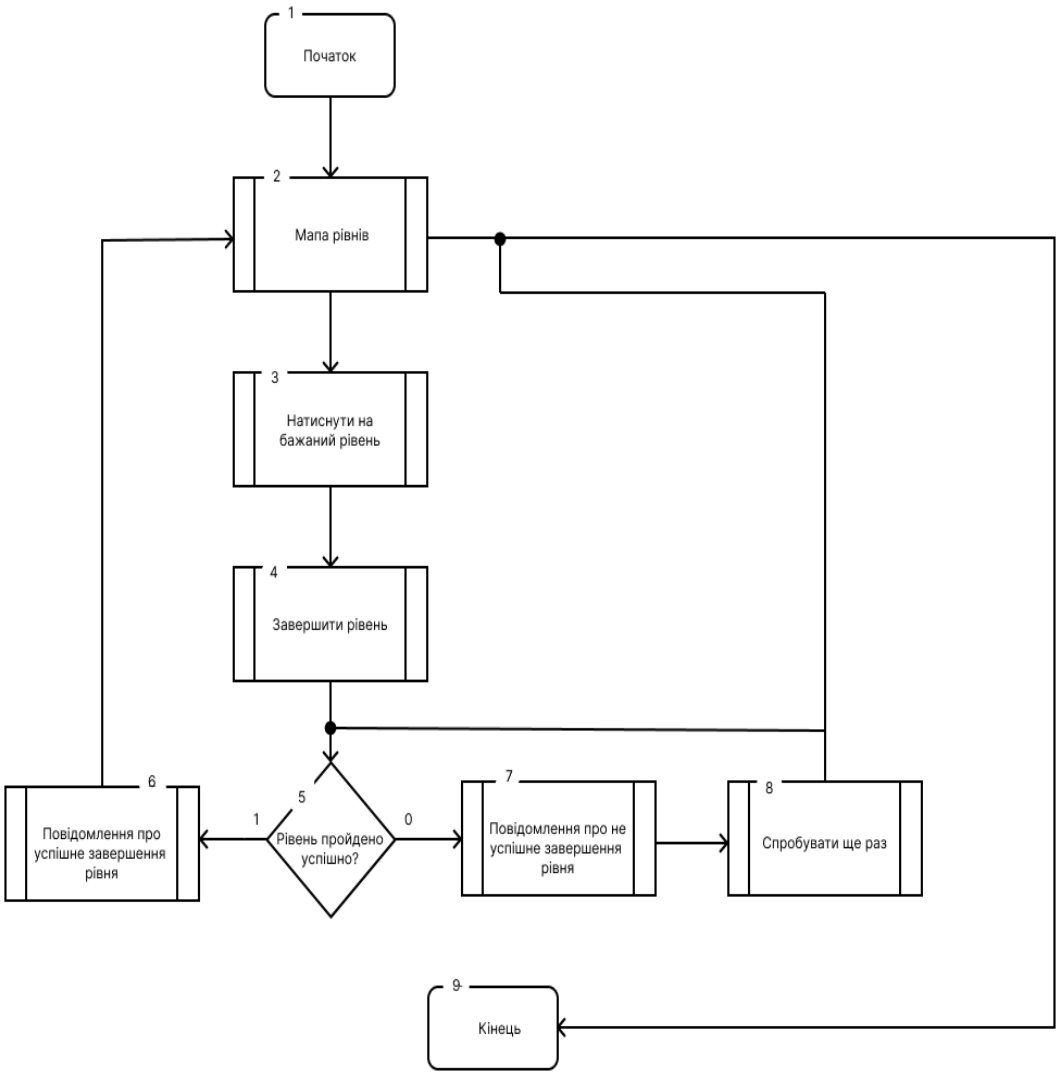

Рисунок 2.3 – Основний цикл взаємодії користувача з ігровим полем

Друга діаграма станів відображає роботу інтерфейсу на рівні навігації між екраном карти рівнів і власне ігровим екраном. У ній показано послідовність відкриття гри вибір рівня перехід до вікна гри виконання проходження рівня та перевірку умов успішності або поразки. У разі виграшу гравець повертається до карти рівнів для нового вибору завдання, а в разі поразки йому пропонується повторити спробу той самий рівень (див. рисунок 2.4).

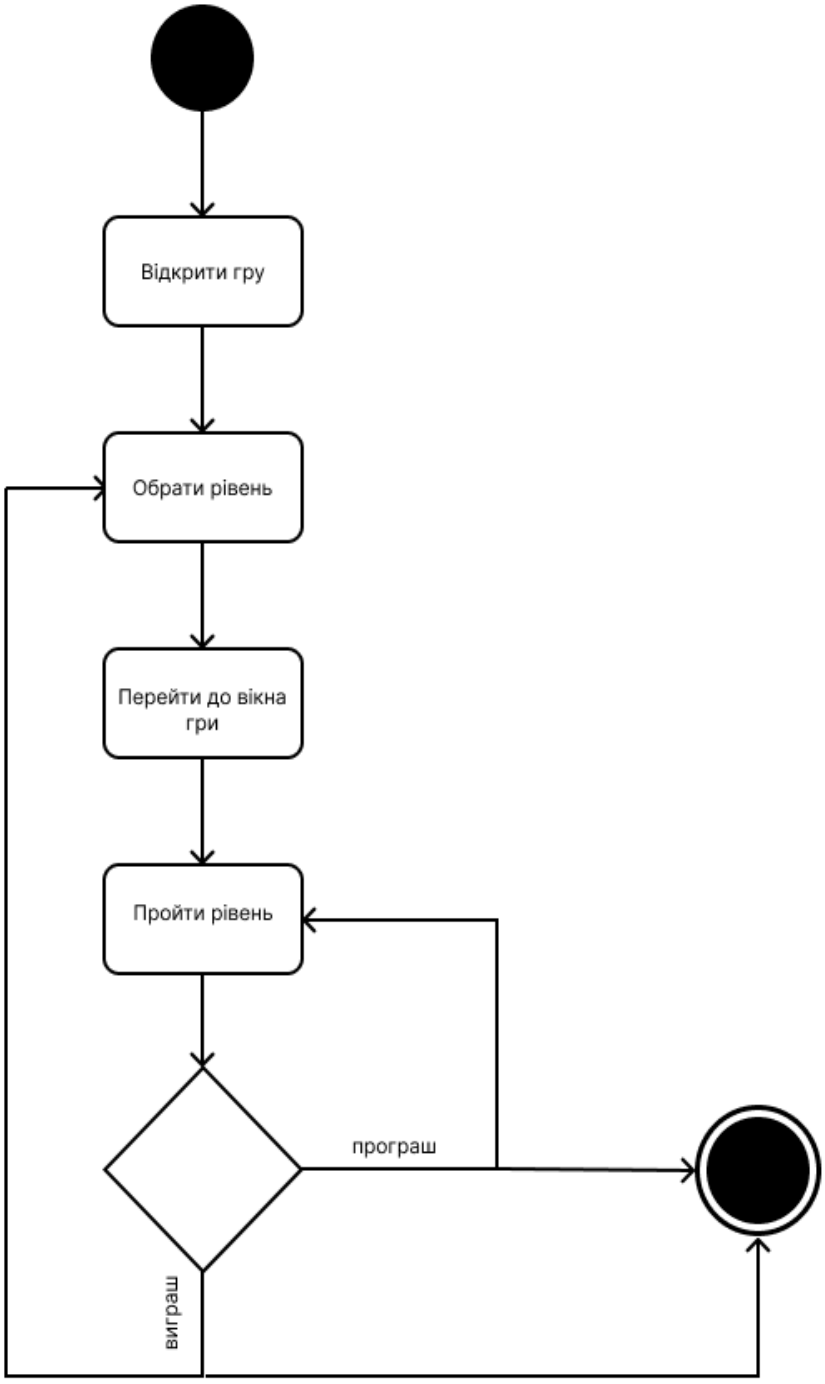

Рисунок 2.4 – Діаграма станів

Усі діаграми станів були узгоджені з логікою реалізації на рушії Godot та мовою GDScript. Вони слугують наочним інструментом даючи чітке бачення можливих переходів і допомагаючи запобігати дублюванню станів або пропуску обробки критичних умов. Такий підхід гарантує високу стабільність роботи гри та передбачуваність її поведінки за будь-яких умов взаємодії користувача з інтерфейсом.

2.4 Висновки

У другому розділі було здійснено всебічний аналіз та проєктування алгоритмічної частини мобільної казуальної гри жанру три в ряд. Спершу було ретельно вивчено методи генерації початкового розподілу фішок на ігровому полі з урахуванням необхідності уникнення випадкових готових комбінацій. Обґрунтовано застосування каскадного підходу до заповнення матриці гри коли кожна випадкова фішка проходить перевірку на предмет утворення трійок і за потреби генерується заново, що гарантує відсутність простих виграшних ходів на початку рівня. Додатково було досліджено переваги використання псевдовипадкових зерен для відтворюваності польових конфігурацій що спрощує процес налагодження і тестування. Завдяки цьому рішення стає можливим зберігати успішні набори фішок для подальшого аналізу і порівняння продуктивності різних версій алгоритму.

Наступним етапом розробки було впровадження процедурних методів з використанням шуму Перліна який дозволяє створювати гладкі карти інтенсивності появи бонусних елементів на полі. Цей підхід формує природні переходи між зонами з меншим і більшим рівнем складності що знижує одноманітність і підвищує зацікавленість гравця. Крім того застосування кластерного аналізу дало змогу розбити ігрове поле на окремі сектори де масштаби локальної випадковості контролюються окремо що запобігає надмірному скупченню однакових фішок в одній ділянці. Завдяки поєднанню

цих методів вдалося досягти гармонійного балансу між передбачуваністю і випадковістю ігрового процесу.

У цьому розділі також було розглянуто методи математичної оптимізації які дозволяють автоматично налаштовувати параметри генерації залежно від даних про середню кількість ходів до завершення рівня. Використання лінійної регресії та стохастичних методів оптимізації створює основу для адаптивної зміни складності що підвищує рівень залученості та знижує ймовірність відмови гравця в результаті занадто різкого стрибка навантаження. Окреслено перспективи застосування генетичних алгоритмів для еволюційного покращення початкових карт рівнів що може бути корисно при розробці турнірних режимів або сезонних подій.

Ключовим складником другого розділу стала розробка детального ітеративного алгоритму основного ігрового циклу який складається з етапів перестановки фішок пошуку комбінацій видалення знайдених груп каскадного падіння нових елементів і повторної перевірки на готові ланцюги до повного зникнення комбінацій. Було описано механізм повернення ходу назад якщо перестановка не утворює жодного виграшного поєднання що забезпечує безпрограшний сценарій кожного руху гравця. Така послідовність дій гарантує динамічний ігровий процес в якому кожний хід приносить очікувану винагороду і водночас не допускає марнотратного пересування фішок.

Окрему увагу в розділі було приділено моделюванню станів програми та побудові UML діаграм [30] які наочно демонструють переходи між ключовими модулями й екранними станами. Створено декілька схем станів що відображають логіку функцій swap tiles find matches destroy matches, а також навігацію між екраном мапи рівнів і результатами рівня. Графічні моделі стали ефективним інструментом для верифікації правильності переходів і дозволяють вчасно виявити потенційні ситуації пропуску обробки виняткових умов.

Отже, результати цього розділу безпосередньо переходять у наступні фази розробки та гарантують узгоджений і контрольований процес створення мобільної казуальної гри жанру «три в ряд».

3 РОЗРОБКА КЛЮЧОВИХ ПРОГРАМНИХ МОДУЛІВ МОБІЛЬНОЇ ГРИ

3.1 Аналіз аналогів розроблюваного програмного застосунку

Зі зростанням популярності мобільних ігор, особливо в жанрі казуальних головоломок, ринок наповнився великою кількістю ігор формату три в ряд. Цей жанр вирізняється простотою своїх механік, короткою тривалістю ігрових сесій та широкою привабливістю для різних вікових груп, що робить його ідеальним для мобільних платформ. Завдяки цьому, проєкти такого типу активно розвиваються й урізноманітнюються, залучаючи гравців новими візуальними рішеннями, сюжетними елементами, системами нагород тощо.

Однак, попри велику кількість подібних ігор, не всі вони однаково якісно реалізовані або доступні для користувачів пристроїв із невисокою продуктивністю. У багатьох аналогах спостерігається надмірна кількість графічних ефектів, агресивна монетизація, високі вимоги до ресурсів пристрою або одноманітний ігровий процес. Це відкриває можливості для створення нових продуктів, які краще відповідатимуть потребам сучасного користувача.

У цьому розділі буде проведено порівняльний аналіз наявних ігор аналогів у жанрі три в ряд за різними критеріями, зокрема за графічним складником, ігровою механікою, системою мотивації, продуктивністю та зручністю для кінцевого користувача. Метою такого аналізу є виявлення сильних і слабких сторін конкурентів та визначення напрямів удосконалення при створенні нової гри, яка буде краще адаптована до пристроїв з обмеженими ресурсами та забезпечить позитивний досвід для ширшого кола користувачів.

Серед численних аналогів розроблюваної мобільної гри варто виділити декілька ключових застосунків. Candy Crush [31], Bejeweled Classic [32] та Harry Potter: Puzzles & Spells [33], кожен з яких має свої унікальні особливості та переваги. Candy Crush (див. рисунок 3.1), є одним із найвідоміших представників

жанру три в ряд. Гра випущена компанією King. Вона вирізняється яскравим візуальним оформленням, великою кількістю рівнів, щоденними завданнями та інтеграцією соціальних функцій, які дозволяють гравцям змагатися з друзями.

Рисунок 3.1 – Приклад роботи додатку Candy Crush

Гра побудована на безперервному циклі нових викликів, з кожним рівнем додаються нові механіки такі як бомби, обмеження за ходами, желе тощо. Цей підхід дозволяє зберігати інтерес користувача (див. рисунки 3.2 - 3.3).

Рисунок 3.2 – Приклад механік гри Candy Crush

Рисунок 3.3 – Приклад різноманіття рівнів Candy Crush

Bejeweled Classic (див. рисунок 3.4) дна з перших комерційно успішних ігор формату три в ряд, яка задала стандарти жанру. Розроблена компанією PopCap Games, ця гра має більш класичний та стриманий стиль, без надмірної візуальної перевантаженості. (див. рисунок 3.5).

Рисунок 3.4 – Приклад рівня Bejeweled Classic

Рисунок 3.5 – Приклад графіки Bejeweled Classic

Головною особливістю Bejeweled Classic є простота і відсутність складних механік чи сюжетних вставок, що робить її привабливою для гравців, які шукають базовий формат гри без відвабливих факторів. У порівнянні з іншими іграми, BejeweledClassic менш вимоглива до ресурсів пристрою, що робить її хорошим вибором для слабших телефонів. Водночас відсутність розгалуженого прогресу, сюжетної лінії чи системи рівнів може знизити інтерес користувачів після кількох годин гри. Також у грі майже відсутні соціальні функції або нагороди, що обмежує рівень залучення гравців у довгостроковій перспективі.

Harry Potter: Puzzles & Spells, розроблена Zynga, поєднує у собі традиційну механіку три в ряд з ліцензованим візуальним стилем всесвіту Гаррі Поттера. Гра вирізняється своєю високоякісною графікою, численними анімаціями та сюжетними елементами, що дозволяють гравцям зануритися у магічну атмосферу Хогвартсу (див. рисунок 3.6).

Рисунок 3.6 – Приклад рівня в Harry Potter: Puzzles & Spells

Сюжет гри розвивається впродовж проходження рівнів, де гравець зустрічає вже знайомих персонажів з книг та фільмів, колекціонує магічних істот та вивчає закляття. (див. рисунок 3.7). Однак, через свою графічну насиченість та об’ємність контенту, додаток має високі системні вимоги та може споживати значні ресурси пристрою. Це робить його менш зручним для великої частки користувачів зі слабкими телефонами (див рисунок 3.8).

Рисунок 3.7 – Приклад головного меню в Harry Potter: Puzzles & Spells

Рисунок 3.8 – Приклад рівня головоломки в Harry Potter: Puzzles & Spells

В порівнянні можна побачити що хоча всі три ігри належать до одного жанру три в ряд, кожна з них має свої особливості, сильні та слабкі сторони. Candy Crush приваблює яскравою графікою, великою кількістю рівнів та постійними оновленнями, але потребує потужного пристрою й активно використовує внутрішні покупки. Bejeweled Classic вирізняється простотою, стабільною роботою та низькими вимогами до ресурсів, однак має обмежений функціонал та швидко втрачає інтерес користувача. Harry Potter: Puzzles & Spells пропонує глибше занурення в ігровий світ шляхом сюжетної частини та привабливого візуального стилю, проте є надто вимогливою до системних ресурсів і схильною до агресивної монетизації.

В результаті аналізу було створено таблицю (таблиця 3.1) для порівняння ігор аналогів з власною розробкою.

Таблиця 3.1 – Порівняльні характеристики аналогів

Характеристики	Candy Crush	Bejeweled Classic	Harry Potter: Puzzles & Spells	Власна розробка
Графічний інтерфейс	1	0	1	1
Різномаїття ігрових механік	1	0	1	1
Оптимізація	0	1	0	1
Кількість ігрових рівнів	1	1	1	1
Сумарний коефіцієнт	3	2	3	4

Представлена таблиця наочно демонструє порівняльний аналіз трьох популярних ігор жанру три в ряд, CandyCrush, BejeweledClassic та HarryPotter: Puzzles&Spells, а також оцінку власної розробки за чотирма ключовими критеріями, графічний інтерфейс, різноманіття ігрових механік, оптимізація та

кількість ігрових рівнів. Кожен із параметрів оцінювався за бінарною шкалою 1 відповідність, 0 недостатній рівень, що дозволяє об'єктивно оцінити загальні переваги та недоліки кожного з продуктів.

Як видно з таблиці, CandyCrush та HarryPotter: Puzzles&Spells мають сильні сторони в частині графіки, процесу ігрового різноманіття та багатого контенту, але поступаються у сфері оптимізації. Це підтверджує, що ці ігри можуть бути менш придатними для пристроїв з обмеженими ресурсами. Bejeweled Classic, навпаки, демонструє хорошу оптимізацію, однак суттєво поступається за сучасним візуальним оформленням та функціональністю.

Власна розробка отримала найвищу сумарну оцінку 4 бали, що свідчить про її збалансованість. Вона поєднує в собі яскравий графічний інтерфейс, різноманіття механік, достатню кількість рівнів та належну оптимізацію, що робить її конкурентоспроможною навіть у порівнянні з визнаними представниками жанру. Це також підтверджує доцільність створення нового ігрового продукту, який зможе зайняти нішу на ринку мобільних ігор для менш продуктивних пристроїв.

Доцільність розробки нової казуальної гри жанру три в ряд зумовлена стійкою популярністю цього типу ігор серед широкої аудиторії користувачів. Ігри такого формату вирізняються простою механікою, що не потребує складного навчання, швидким зануренням в ігровий процес, а також здатністю забезпечити короткі, але насичені сесії гри, що ідеально підходить для мобільних користувачів. За останні роки ринок ігор три в ряд показує стабільно високі показники завантажень та прибутків, що свідчить про сталий попит на такі продукти. Проте більшість доступних ігор орієнтовані на пристрої середнього або високого класу, що залишає поза увагою значну частину користувачів із менш потужними телефонами або обмеженим доступом до ресурсів.

У цьому контексті розробка оптимізованої гри три в ряд, орієнтованої на мобільні пристрої з невеликим обсягом оперативної пам’яті та низькою продуктивністю процесора, є актуальним та обґрунтованим кроком. Така гра дозволить заповнити прогалину на ринку, пропонуючи гравцям збалансовану

комбінацію графіки, ігрового процесу, різноманітності та продуктивності. Крім того, продумана архітектура та використання рушія Godot забезпечують кращу масштабованість, легкість у підтримці та розширенні функціоналу. Таким чином, розробка гри не лише відповідає потребам ринку, а й створює можливості для подальшого розвитку проєкту, включно з додаванням нових режимів, візуальних тем та соціальних елементів.

3.2 Аналіз даних

Перш ніж розпочати безпосередню реалізацію програмних модулів, необхідно визначити підходи до організації даних та вибрати методи їхньої обробки. Для власної розробки була обрана модульна структура, коли кожен ключовий компонент гри, такий як меню рівнів, ігрове вікно та вікно завершення матчу, працює із власними наборами даних, водночас взаємодіючи через чітко визначені інтерфейси та сигнали ігрового рушія. Такий підхід забезпечує ізоляцію логіки відповідальності та полегшує тестування окремих модулів. Зокрема, для збереження прогресу та налаштувань користувача застосовується вбудований API файлової системи Godot, яке дозволяє серіалізувати структури JSON і зберігати їх у локальних файлах.

У меню рівнів ключовим є завдання відображення доступності кожного рівня й обробка даних про пройдені етапи. Для цього передбачено дві функціональні частини, такі як оптимізація завантаження ресурсів та збереження стану. Оптимізація досягається динамічним підвантаженням мініатюр для кожного рівня тільки при запуску меню, що дозволяє знизити обсяг завантажуваної пам’яті. Дані про пройдені рівні, найвищі бали та відкриті бонуси зберігаються у єдиному JSON [34] файлі, доступ до якого здійснюється через єдиний клас менеджер рівнів.

У модулі ігрового вікна реалізуються основні ігрові механіки, побудова поля, алгоритми пересування фішок та виявлення ланцюжків із трьох і більше елементів. Для кожного об’єкта поля створюється окремий ресурс спрайт із властивістю позиція, а управління рухом здійснюється за допомогою корутин на

базі сигналів Godot. Логіку бомби як бонусного об’єкта реалізовано через патерн «Strategy», коли для кожного типу детонації, навколишні клітинки або рядок, викликається відповідний метод який відповідає за вибух. По завершенню матчу модуль обробки результатів рівня збирає статистику ходів, визначає перемогу чи поразку та передає дані в менеджер збережень для фіксації досягнень.

Розробка усіх перелічених компонентів проходитиме в середовищі Godot із використанням GDScript. Тестування функціоналу відбуватиметься як вручну на різних емуляторах Android, так і автоматизовано з використанням вбудованого Unit Test Framework Godot.

Завдяки такій організації даних та інструментарію розробки стає можливим отримання гнучкої, розширюваної архітектури, яка готова до подальшої інтеграції додаткових сервісів та аналітики.

3.3 Розробка програмного модуля меню рівнів

Модуль меню рівнів відповідає за відображення списку доступних ігрових локацій, динамічне підвантаження мініатюр і збереження позиції прокручування, щоб користувач міг повернутися до попереднього перегляду без втрати контексту. Для реалізації безперервного та плавного скролінгу застосовано власні функції керування значенням scroll_vertical контейнера ScrollContainer, зокрема методи smooth_scroll та start_deceleration, які використовують Tween-служби рушія для м’якого зниження швидкості прокручування. Такий підхід забезпечує відчуття природної інерції та покращує взаємодію з меню на сенсорному екрані пристрою (див. рисунок 3.9).

Щоб зберегти місце, на якому користувач зупинився, було створено механізм нормалізації поточної позиції скролу на відрізку від 0 до 1 через обчислення співвідношення current_scroll до max_scroll та інверсію цього значення. Під час виходу з дерева сцени функція _exit_tree викликає save_scroll_position, яка передає нормалізовану позицію в клас-менеджер GameDataManager. При повторному відкритті меню за допомогою

restore_scroll_position() відбувається асинхронне очікування кадру await get_tree().process_frame, а потім прокручування до збереженої позиції відповідно до максимального значення max_scroll контейнера (див. рисунок 3.10).

Рисунок 3.9 – Прокручування мапи рівнів

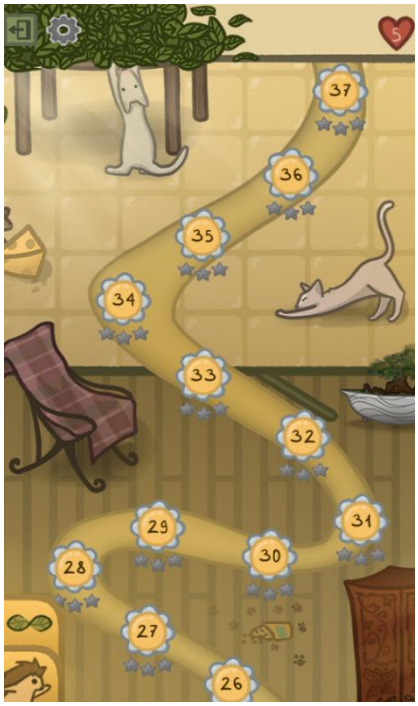

Рисунок 3.10 – Відновлення збереженої позиції прокручування мапи рівнів

Обробка взаємодії користувача відбувається у функції _input, яка ігнорує події, якщо відкриті інші накладені панелі, такі як діалог підтвердження виходу, відеоінструкція тощо, а також враховує поточний режим налаштувань. При

дотику до екрана встановлюється змінна is_touching, фіксується початкова позиція дотику, а під час руху пальцем обчислюється відстань перетягування, так досягається відчуття ручного прокручування. По відпусканню дотику викликається start_deceleration, що плавно зупиняє прокручування до моменту, коли воно не потребує втручання користувача (див. рисунок 3.11).

Рисунок 3.11 – Розпізнавання доторку лише на верхньому відкритому вікні

Окрім механіки прокручування, модуль містить обробники кнопок виходу і підтвердження закриття меню (див. рисунок 3.11), при натисканні на кнопку

виходу відтворюється звук, активується вікно підтвердження, а у випадку повторного натискання виконується емісія сигналу quit_in_map_pressed для переходу до головного меню гри. Така реалізація дозволяє уніфікувати звуковий супровід і логіку переходів між станами інтерфейсу, забезпечуючи користувачу зрозуміле та послідовне повернення до геймплею або завершення гри.

Підсумовуючи розробку модуля меню рівнів, слід відзначити, що інтегрована система плавного прокручування за допомогою Tween-служб рушія значно покращує користувацький досвід і створює природне відчуття інерції при навігації списком рівнів. Механізм нормалізації та збереження позиції скролу гарантує, що гравець завжди повернеться до того місця, де зупинився, що особливо важливо для комфортного ознайомлення з доступними локаціями у разі перерваної сесії гри.

Одночасно реалізований підхід до обробки подій введення враховує накладені інтерфейсні шари та дозволяє уникнути конфліктів між різними режимами взаємодії, що зберігає цілісність UX навіть за умов роботи декількох вікон чи діалогів. Завдяки чіткому розмежуванню відповідальностей, від фіксації початкової точки дотику до плавного гальмування прокручування, досягається висока точність та відчуття безперервності руху, що відповідає сучасним стандартам сенсорних інтерфейсів.

Кнопкова логіка виходу з гри та підтвердження закриття меню забезпечує єдиний підхід до обробки звукового супроводу та сигналів рушія, що спрощує підтримку й подальший розвиток модуля меню рівнів. Взаємодія між GameDataManager і самим меню рівнів дозволяє централізовано управляти збереженням даних, тим самим значуще знижуючи ризик дублювання коду та можливих помилок при роботі зі збереженням прогресу гравця. В результаті було отримано цілком надійний, гнучкий та зручний у використанні графічний інтерфейс користувача, готовий до масштабування та інтеграції з іншими модулями розроблюваної мобільної гри.

3.4 Розробка програмного модуля ігрового вікна

Розробка модуля ігрового вікна охоплює реалізацію базової ігрової механіки, що забезпечує формування поля, двовимірної сітки розміром N×M, обмін позиціями фішок, пошук ліній із трьох і більше елементів, детонацію бонусних об’єктів і каскадне заповнення вивільнених клітин. Для представлення поля у пам’яті використовується матриця tile_grid[i][j], де кожен елемент містить посилання на об’єкт Tile із властивістю type, колір або спеціальна роль та вектором grid_position для відтворення у вікні. Ініціалізація поля відбувається випадковим чином із гарантією відсутності готових збігів на старті, що реалізовано через багаторазове генерування сусідніх трійок із контролем комбінацій. Комбінаторний підхід із перевіркою всіх рядків і стовпців підряд по N+M операцій (див. рисунок 3.12).

Рисунок 3.12 – Ігрова сітка

Пересування фішок в межах поля реалізовано через метод swap_tiles, який виконує взаємозамінність позицій у матриці та одночасно анімує їхнє

переміщення з плавною інтерполяцією за допомогою Tween-служб рушія. Алгоритм перевіряє умову суміжності, дві фішки можуть обмінюватися тільки якщо їхні координати відрізняються на одиницю за однією віссю й рівні за другою. У разі якщо після обміну не утворюється жодного збігу, викликається метод swap_back, що повертає фішки на попередні місця, зберігаючи цілісність матриці. Обчислювальна складність перевірки збігів після кожного обміну становить, оскільки для кожного осередку перевіряються максимальні ланцюжки в горизонтальному та вертикальному напрямах (див. рисунок 3.13).

Рисунок 3.13 – Момент перетягування ігрових фішок

Пошук збігів здійснюється функцією find_matches, яка аналізує кожен рядок та стовпець через «ковзне вікно» довжини $k \geq 3$. Використовуючи двовимірну матрицю суміжності, алгоритм визначає зони послідовних елементів однакового type і позначає їх для знищення. З наукового погляду це завдання можна трактувати як пошук підрядкових патернів у рядках та стовпцях, де кожен патерн, це підмасив довжини ≥ 3 з константною ознакою. Теоретично така

операція належить до класу задач сканування послідовностей із лінійною складністю, що забезпечує масштабованість механіки на полях будь-яких розмірів.

Бонусні об’єкти бомбочки додають стратегічної глибини й новизни в ігровий процес. У розроблюваному проєкті передбачено три види бомб. Лінійна бомба активується при створенні хрестоподібного збігу й очищає всю відповідну горизонтальну або вертикальну лінію. Обчислення ураженої зони відбувається через просте ітеративне просування вздовж осі до країв матриці. Об’ємна бомба має радіус дії R, задаваний параметром, і знищує всі фішки на відстані ≤R. Алгоритм формування множини уражених клітин реалізовано через генерацію квадратної області із фільтром за наведеною формулою, що є прикладом застосування дискретної геометрії. Колірна бомба активується при комбінації п’яти однакових фішок і очищує всі елементи на полі заданого кольору, що реалізовано через один прохід по матриці з перевірками рівності властивості type (див. рисунок 3.14).

Рисунок 3.14 – Фішки бомбочки

Генерація кожного виду бомб та їхня інтеграція в каскадні реакції вимагає застосування алгоритмів злиття множин та поетапного оброблення видалених осередків. Після детонації виконується перерахунок сусідніх елементів, падіння гравітації. Наукова модель, взаємодія із вільними клітинами згідно з алгоритмом fill_down, що перебирає стовпці й по черзі спускає всі не null значення донизу. Цей процес повторюється доти, доки не припиняться нові збіги, формуючи каскад, проте на практиці обмежений кількістю рівнів зернистості та глибиною каскаду (див. рисунок 3.15).

Рисунок 3.15 – Каскадна реакція

Усі математичні й алгебраїчні операції реалізовані з акцентом на наукову новизну, поєднання дискретних алгоритмів пошуку патернів, геометричних обчислень у площині й побудови каскадних функцій із використанням операторів множин. Це дозволяє не лише створювати плавний і захопливий геймплей, а й забезпечує гнучкість налаштувань механік, адаптацію до різної складності й науково обґрунтовану оцінку продуктивності.

Підсумовуючи розробку модуля ігрового вікна, слід відзначити, що реалізована система формування поля, механіка обміну фішок і пошук ланцюжків із трьох однакових елементів дозволяють гравцю відчути економічну цілісність ігрового процесу. Впровадження бомб із різними алгоритмами дії, лінійної, об’ємної та колірної, що додає стратегії та варіативності, а використання корутин забезпечує плавність анімацій та послідовне оброблення каскадних реакцій. Завдяки науково обґрунтованим підходам до обчислення уражених зон, перевірки сусідніх елементів і заповнення вивільнених клітин, модуль поєднує високу продуктивність із гнучкими налаштуваннями механік та готовий до майбутнього розширення функціоналу.

3.5 Розробка програмного модуля вікна завершення ігрового матчу

Розробка вікна завершення матчу охоплює два взаємопов’язані підмодулі, вікно поразки та вікно перемоги. Обидва компоненти побудовані на основі єдиної сцени з розташованими шарами BlurLayer, який розмиває фон, та головним контейнером вікна, що містить елементи інтерфейсу, кнопки та анімаційні елементи. Плавність появи й зникнення вікон забезпечується через AnimationPlayer, де для кожного стану передбачено окрему анімацію «show_in» для поступової появи вікна і «hide_out» для його приховання.

У разі поразки основний виклик ініціалізується сигналом _on_grid_game_over, що запускає відтворення звукового ефекту невдачі через loss_sound.play, встановлює вікно по центру екрана викликом _center_over_window, а також робить видимими шари BlurLayer та сам контейнер

з повідомленням про поразку. Після цього запускається анімація «show_in», що за допомогою Tween-служб ігрового рушія забезпечує плавне збільшення прозорості вікна від нуля до повної видимості. Одночасно викликається update_stars_display, яка, хоча в режимі поразки найчастіше приховує значки зірок, перевіряє поточний режим гри, наприклад, нескінченний режим не відображає жодних зірок і відображає лише ті елементи, що відповідають збереженим досягненням гравця. Після завершення цієї процедури у компоненті GameDataManager автоматично оновлюється лічильник життів через decrease_heart, що реалізує загальну логіку обмеження кількості спроб у класичному «free-to-play» стилі (див. рисунок 3.16).

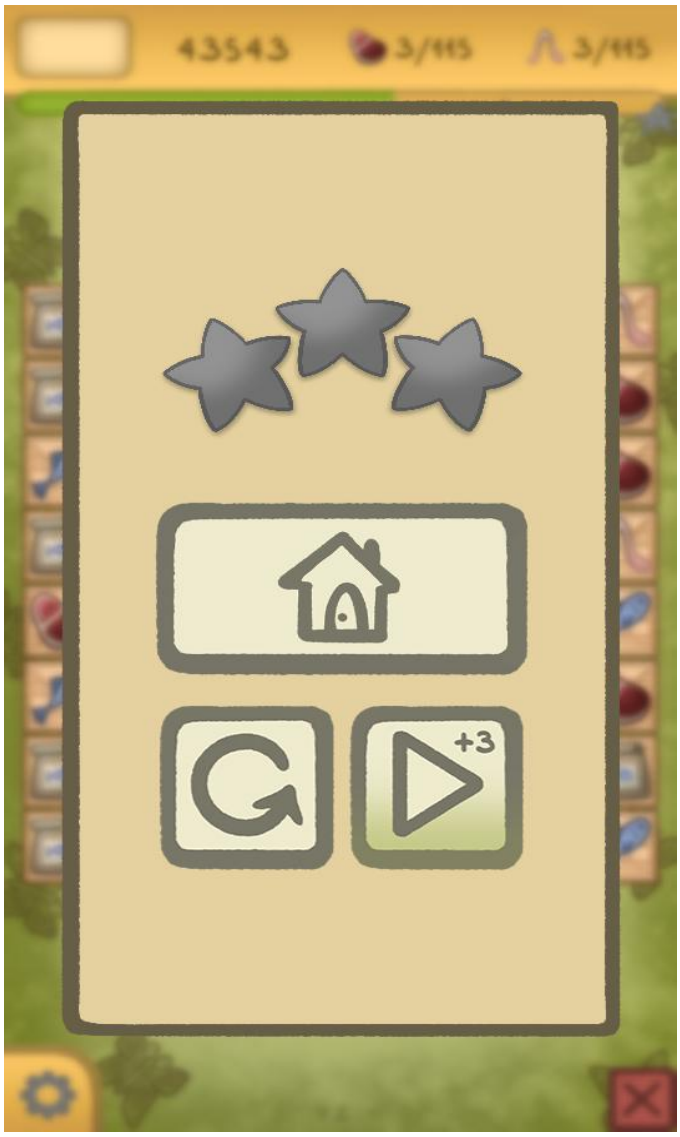

Рисунок 3.16 – Завершення гри

Кнопки вікна поразки «Restart» і «Retry» обробляються окремими методами _on_restart_pressed та _on_retry_pressed. Перший одразу відтворює звуковий ефект натискання button_sound.play та емітує сигнал reload_button_pressed, який ловиться на вищому рівні керування сценою, що дозволяє повернути гравця в до вибору рівня. Другий перевіряє доступну кількість життів через get_hearts якщо вони є, то за допомогою get_tree.reload_current_scene відбувається перезавантаження поточної сцени, для повторної спроби проходження рівня. У випадку якщо життів не лишилося, гравець повинен зачекати кілька хвилин для того, щоб життя оновились, до цього моменту подальша взаємодія грою блокується (див. рисунок 3.17).

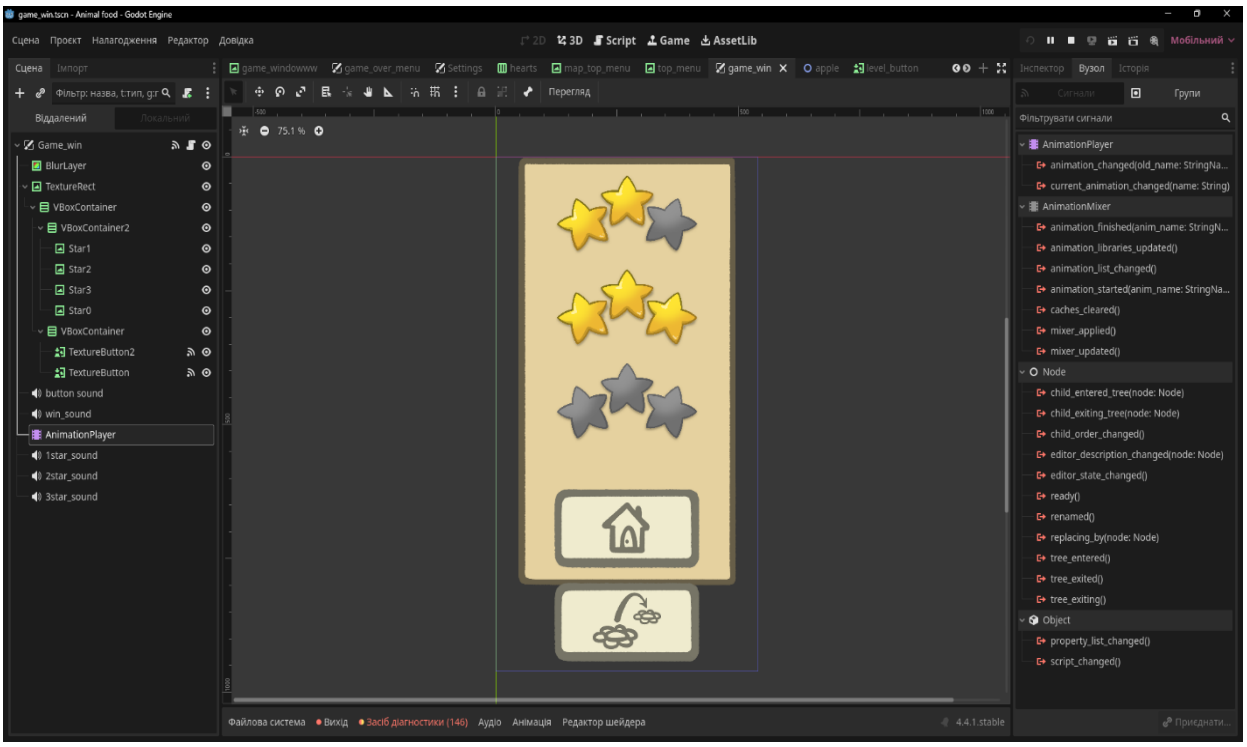

Рисунок 3.17 – Розробка сцени віна поразки

Відображення результатів поразки та активація кнопок завершуються автоматичним закриттям вікна через анімацію «hide_out» за запитом іншого модуля керування станами гри або після вибору гравцем подальшої дії. Завдяки цьому вікно поразки не лише інформує про невдалу спробу, а й коректно

інтегрується в загальний потік гри, зберігаючи цілісність UX та уникаючи раптових переривань взаємодії з грою.

Вікно перемоги запускається при надходженні сигналу _on_grid_declare_game_win_true, який спершу приховує початковий фон екрана перемоги та виконує метод _center_win_window, що встановлює вікно в центр екрана з урахуванням динамічного масштабування. Одразу ж програється звуковий ефект успіху win_sound.play, після чого "show_in" поступово показує вікно. Паралельно відбувається обчислення кількості зірок calculate_stars, де рівень досягнень гравця порівнюється з порогами 0.5·max_score, 0.75·max_score і 1·max_score, що дає можливість науково обґрунтовано ранжувати успішність проходження рівня (див. рисунок 3.18).

Рисунок 3.18 – Вікно перемоги

Після завершення анімації викликається show_stars, яка по черзі відображає відповідні спрайти зірок з ефектом поступового прояву і

відтворенням індивідуальних звукових ефектів starN_sound.play. Це поєднання візуальних та аудіопідтримок створює емоційний піковий момент, підсилюючи відчуття досягнення. Наступний рівень за допомогою _on_texture_button_pressed відкривається лише в тому разі, якщо він розблокований і не перевищує ліміт max_unlocked_level; в іншому випадку програма повертає гравця до карти рівнів, використовуючи _transition_to_map (див. рисунок 3.19).

Рисунок 3.19 – Анімація переходу до меню вибору рівнів

У випадку успіху також виконується оновлення прогресу в LevelManager.update_level_progress, що зберігає нові дані про найбільший досягнутий рівень, кількість зірок та розблоковує наступний рівень. Після цього змінюється сцена на вигляд карти або на сценарій гри з новим рівнем, залежно від вибору гравця, що забезпечує безшовний перехід між етапами проходження.

Таким чином, розроблений модуль вікна перемоги та поразки об’єднує в собі елементи інтерфейсу, анімаційні переходи, звукові ефекти та алгоритмічну логіку обробки результатів матчу, створюючи завершений цикл взаємодії із гравцем. Інтеграція з менеджерами даних GameDataManager та LevelManager гарантує коректне збереження прогресу, а використання GDScript корутин і Tween-служб рушія Godot забезпечує плавні й чітко синхронізовані анімації, що відповідають високим стандартам якості мобільних ігор.

3.6 Розробка глобального модуля збереження та управління грою

Глобальний модуль збереження та управління гри GameDataManager виступає серцем системи збереження внутрішніх станів та налаштувань користувача, забезпечуючи узгоджений доступ до даних із будь-якої частини програми. Він завантажується рушієм як єдиний синглтон при старті через автозавантаження, що гарантує наявність єдиного екземпляра цього класу протягом усього часу роботи. Після ініціалізації відразу перевіряється наявність файлу save_data.json; якщо такий файл існує й містить коректно сформований JSON-рядок, модуль виконує розбір його вмісту та відновлює внутрішні поля, зокрема ідентифікатор поточного рівня, параметри звуку, кількість життів та інші збережені стани. У разі відсутності чи пошкодження файлу створюється нова структура за замовчуванням, що містить належні початкові значення, зокрема стартовий рівень, стандартну гучність звуку тощо.

Процес збереження прогресу реалізований у вигляді серіалізації єдиного об’єкта стану гри у формат JSON і запису цього рядка у локальний файл. Внутрішній механізм спочатку формує об’єкт із вкладеними підрозділами, куди

автоматично збираються всі необхідні дані, а потім викликає методи рушія для відкриття файлу у режимі запису та збереження отриманого рядка. Якщо під час запису виникає помилка, вона ловиться всередині блоку обробки помилок, значення помилки зберігається у внутрішній змінній і фіксується для подальшого аналізу, а сам рядок відкладається у буфер очікування, аби повторити спробу запису пізніше. Таке рішення забезпечує надійність і стійкість до тимчасових збоїв файлової системи.

Зворотний процес відновлення даних при повторному запуску гри полягає у зчитуванні текстового вмісту файлу, його передачі в парсер JSON і валідації отриманого об’єкта. Модуль перевіряє наявність усіх очікуваних ключів перед тим, як розгорнути їх у відповідні поля, що дозволяє уникнути збоїв під час зміни структури збереження між версіями програми. Якщо в процесі валідації виявляються нові або відсутні параметри, їм автоматично присвоюються значення за замовчуванням, після чого оновлений об’єкт негайно записується назад у файл, що гарантує сумісність із поточною версією гри без необхідності ручних міграцій даних.

Усі підмодулі гри, меню рівнів, ігрове вікно, вікна перемоги та поразки, взаємодіють із GameDataManager через чітко визначені виклики методів, не звертаючись безпосередньо до файлової системи. Наприклад, при завершенні рівня модуль оновлює внутрішній стан гри, порівнюючи поточний результат із попереднім рекордом і за потреби замінюючи його на кращий. Після цього змінені дані негайно серіалізуються й зберігаються. Таким чином, прогрес гравця оновлюється централізовано й послідовно, без дублювання коду в різних частинах проєкту.

Кожна зміна налаштувань звуку або інтерфейсу виконується через відповідні методи керування, які одночасно коригують внутрішні поля і викликають збереження. Це дає змогу забезпечити миттєве відновлення цих налаштувань при подальших запусках гри. За потреби реалізується механізм «гарячих» оновлень, відкладені у буфері зміни автоматично записуються під час

наступного виклику основного методу збереження, що дозволяє працювати без переривання ігрового сеансу.

Архітектурно модуль побудований так, що його легко розширити для надсилання даних на віддалений сервер або інтеграції з хмарним сховищем. Достатньо реалізувати додатковий інтерфейс, який у разі успішного локального запису буде дублювати JSON-рядок у мережевий запит. Це не потребуватиме змін у решті коду, оскільки клієнтська частина гри взаємодіє лише з універсальними методами збереження та завантаження.

Важливою властивістю є також централізована обробка версійності даних. GameDataManager зберігає номер поточної структури збереження та при запуску порівнює його з номером, що вказаний у файлі. Якщо вони не збігаються, модуль ініціює процес оновлення даних, підставляючи нові поля та коригуючи застарілі, після чого автоматично перезаписує файл у відповідному форматі. Це дозволяє змінювати внутрішні формати без ризику втрати прогресу користувача.

У результаті застосування описаних рішень глобальний модуль збереження та управління іграми забезпечує прозору й надійну роботу з усіма даними, необхідними для функціонування клієнтської частини. Завдяки використанню єдиної точки входу, послідовній валідації, обробці помилок і відкладеному буферу запису досягається висока стійкість до відмов і простота супроводу архітектури, що відповідає сучасним вимогам до мобільних додатків.

3.7 Висновки

В цьому розділі було створено чітко структуровану архітектуру, яка умовно розбивається на чотири взаємопов’язані компоненти, меню рівнів, ігрове вікно, вікно завершення матчу та глобальний менеджер даних. Кожен із цих модулів виконує свою роль у загальному циклі взаємодії користувача з грою, при цьому всі вони тісно інтегровані через єдиний інтерфейс і механізми сигналів рушія Godot. Така модульна організація не лише полегшує розробку та

тестування окремих частин програми, а й гарантує гнучкість при подальшому розширенні або зміні функціоналу.

Меню рівнів було реалізовано із застосуванням власного алгоритму плавного скролінгу, який забезпечує відчуття інерції та природну навігацію по списку доступних локацій. Механізм нормалізації й збереження позиції скролу дозволяє зберігати контекст перегляду навіть після виходу з меню, що покращує комфорт користувача під час тривалих ігрових сесій.

Основний ігровий модуль поєднує традиційну для жанру «три в ряд» сітку елементів із механіками пересування фішок, пошуку ліній та детонації бонусів. Алгоритмічна частина включає лінійний пошук збігів через «ковзне вікно», комбінаторну перевірку початкової конфігурації поля та математично обґрунтовані методи формування зон ураження для різних типів бомб. Використання корутин дозволяє організувати послідовні кроки анімації й обробки каскадних реакцій без блокування головного потоку, що позитивно позначається на плавності відображення ігрового процесу.

Вікна перемоги й поразки побудовані на спільній сцені з узгодженими анімаціями появи й зникнення, а також звуковим супроводом, що створює емоційні пікові моменти завершення рівня. Логіка відображення результатів ґрунтується на розрахунку зіркового рейтингу через порогові значення відносно максимального балу, а механізми реагування на кнопки «повторити» або «далі» зберігають узгодженість із внутрішнім станом прогресу. Це дозволило досягти єдиного стандарту взаємодії в усіх випадках завершення матчу.

Глобальний модуль збереження та управління даними є ключовим елементом архітектури, що централізовано обслуговує всі запити на читання й запис ігрового стану. Завдяки автозавантаженню як єдиного синглтона весь цикл серіалізації й десеріалізації JSON здійснюється без розпорошення коду по різних компонентах. Продумана валідація структури й механізм відкладеного повторного запису у випадку помилок гарантують збереження прогресу навіть за нестабільних умов роботи файлової системи.

Таким чином, цей розділ демонструє закономірний перехід від теоретичних вимог до практичної реалізації ключових блоків гри. Погодженість інтерфейсних шаблонів, алгоритмічних підходів та механізмів обміну даними забезпечує цілісний, стійкий і зручний для користувача продукт. Отримана архітектура готова до подальшої еволюції: від додавання нових типів бонусів та режимів гри до розширення аналітичних можливостей і масштабування на інші платформи. Усі напрацювання закладають надійну основу для створення якісної мобільної гри, що відповідає сучасним стандартам і очікуванням аудиторії.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування мобільної гри на Android [35], це процес оцінювання роботи додатка на мобільних пристроях з цією операційною системою з метою перевірки його функціональності, безпеки та правильності виконання.

На першому етапі здійснюється інсталяція тестової версії застосунку на Android-пристрій. Далі проводяться функціональні тести, що дозволяють перевірити правильність введення й виведення даних, а також відповідність поведінки додатка заданій бізнес-логіці. Завершальним етапом є тестування безпеки, яке включає перевірку на збереження конфіденційних даних, виявлення потенційних вразливостей і проведення аналізу на проникнення.

Здійснимо перевірку мапи рівнів. У разі спроби затримати палець на мапі та перетягнути її вверх, повинна відбутись обробка доторку всередині алгоритму, тоді мапа перетягнеться вверх. Результат зображено на рисунку 4.1.

Рисунок 4.1 – Результат перетягування екрана

Після цього можна провести тест графічних анімацій. С ігровій сцені з мапою рівнів представлено кілька анімацій. Результат зображено на рисунках 4.2 та 4.3.

Рисунок 4.2 – Результат виконання анімації 1

Рисунок 4.3 – Результат виконання анімації 2

Після перевірки мапи рівнів, наступний крок – це спроба перевірити вікно гри, а саме алгоритм пошуку збігів. Перемістивши однакові фішки в ряд можна побачити чи дійсно працюють всі створені алгоритми. Підтвердженням позитивного результату є те що фішки переміщаються, видаляються та зараховуються. Візуально це можна побачити на верхній панелі де будуть відображатись набрані ігрові очки та цілі (див. рисунок 4.4).

Рисунок 4.4 – Зараховані ігрові бонуси

Ці етапи тестування спрямовані на забезпечення високої якості та надійності мобільної гри для користувачів під управлінням операційної системи Android.

Рисунок 4.5 – Тестування рівнів

Також було протестовано випадковий набір рівнів щоб виявити можливі помилки у написаннях умов рівня, завантаження ігрових фішок та завантаження сітки гри (див. рисунок 4.5).

Рисунок 4.6 – Тестування певних фішок

На рисунку 4.6 можна побачити тест випадкового рінвя на те чи правильно завантажився набір певних фішок, в цьому випадку рибних.

Рисунок 4.7 – Тестування умов для перемоги

Наступним етапом стало тестування умов завершення рівня, перемоги та поразки. Імітуючи різні стани ігрового поля, штучно встановлюючи необхідні параметри було протестовано, чи коректно спрацьовують сигнали _on_grid_declare_game_win_true та _on_grid_game_over. Особливо ретельно досліджувалося відображення екрану перемоги, правильність розрахунку зірок за алгоритмом із порогами 0,5·max_score, 0,75·max_score, 1·max_score та синхронність програвання звукових ефектів у пік переможної анімації. Також пріоритет було надано перевірці таймінгів анімацій у режимі реального пристрою, адже на емуляторі вони могли йти швидше чи повільніше залежно від обчислювальної потужності пристрою.

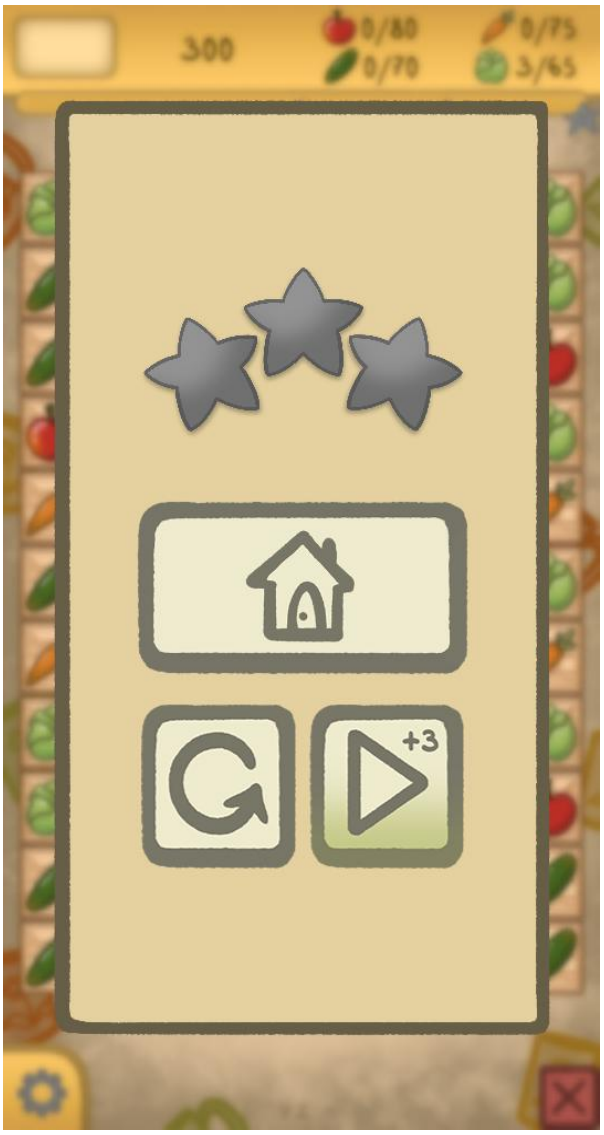

Рисунок 4.8 – Тестування умов для позарки

На рисунку 4.8 можна побачити тест на дотримання умов поразки. Чи вірно працює підрахунок ходів та обробка сигналів яка пов’язана з оголошенням поразки

Рисунок 4.9 – Тестування нарахування зірок

Відповідно до кількості набраних балів за проходження рівня гравець може отримати від 0 до 3 зірок. На рисунку 4.9 зображено успішне тестування нарахування зірок та проходження рівня.

4.2 Розробка інструкції користувача

Тестування мобільної гри на Android розпочинається з упорядкування середовища та налаштування підключення емуляторів і реальних пристроїв із різними характеристиками, щоб упевнитися в однаковій коректності поведінки програми незалежно від пристрою. На кожному етапі перевірялися не лише функціональні сценарії, але й продуктивність, рівень споживання оперативної пам’яті, навантаження на центральний процесор та час відгуку після торкання екрану в найнапруженіших сценах із численними анімаціями каскадів. Це дозволило виявити «вузькі місця» в алгоритмах обробки збігів та візуалізації, своєчасно оптимізувати таймери оновлення екрану та зменшити складність внутрішніх циклів без шкоди для плавності анімації.

Деталі щодо мінімальної та рекомендованої конфігурації пристрою для запуску застосунку можна знайти в таблицях 4.1 та 4.2.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	Qualcomm Snapdragon 835
Об’єм оперативної пам’яті	2 ГБ для 64-разрядної системи
Місце на жорсткому диску	1 ГБ
Операційна система	Android 11

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	2.8GHz Octa-core Google Tensor
Об’єм оперативної пам’яті	4 ГБ для 64-разрядної системи
Місце на жорсткому диску	4 ГБ
Операційна система	Android 14

Після встановлення гри достатньо торкнутися її значка, щоб розпочати сеанс. Перший екран завантаження демонструє логотип гри та коротку анімацію, після якої автоматично відкривається меню рівнів. Якщо під час завантаження були виявлені оновлення даних або нова версія збереження, спочатку відбувається їх синхронізація; це гарантує, що ваш прогрес буде відновлено незалежно від пристрою, з якого ви заходите у гру.

У меню рівнів представлено багато рівнів та локацій, кожен з яких маркований номером та відзначений зірковим рейтингом за попередні проходження. Для прокручування списку досить провести пальцем вгору або вниз, після чого система плавно зупиняється завдяки вбудованому алгоритму інерційного скролінгу. Позиція, на яку ви зупинилися, зберігається автоматично, тому навіть після повторного запуску гри гравець повернетеся до того ж місця. Доступні ті рівні, які вже розблоковано відповідно до прогресу гравця, заблоковані рівні позначені сірим кольором і відкриваються, щойно буде виконано умови попереднього рівня.

Після торкання кнопки рівня і підтвердження вибору починається завантаження ігрового поля. Воно складається з двовимірної сітки, заповненої елементами різних кольорів. Завдання гравця утворювати лінії з трьох або більше однакових фішок, обмінюючи їх місцями одним легким рухом пальця. Якщо після обміну не виникає жодного збігу, елементи автоматично повернуться на попередні позиції, але анімація відображатиме цю спробу, дозволяючи миттєво перейти до наступного ходу. Кожен успішний збіг супроводжується анімацією вибуху та акустичним ефектом, що підсилює відчуття задоволення від виконаної дії.

У процесі гри гравець зможе створювати бонусні об’єкти бомбочки різного типу. Лінійні бомби очищують цілий рядок або стовпець залежно від напрямку збігу, кольорові бомби усувають усі фішки одного відтінку, а об’ємні бомби з радіусом дії охоплюють навколишні клітини в межах чітко визначеного радіусу. Для їх активації необхідно утворити комбінацію з чотирьох або більше фішок і першочергово вийти за рамки звичайної трійки. Вибухи бомб супроводжуються

спеціальними візуальними ефектами та динамічними звуковими сигналами, а подальше падіння залишених елементів ґрунтується на моделі гравітації, що створює каскадні реакції й дарує додаткові бали.

Під час кожного рівня на верхній панелі відображається поточний рахунок та кількість ходів або інший ліміт, встановлений для досягнення мети. Як тільки гравець виконав усі умови рівня набравши потрібну кількість очок або прибравши всі спеціальні блоки за відведену кількість ходів, гра автоматично переходитиме до екрану перемоги. Тут спрацює система нарахування зірок за заданими порогами, а анімація та звуковий супровід створять святковий настрій. Якщо ж ходи закінчаться, а ціль не буде досягнута, гравець опиниться на екрані поразки з пропозицією спробувати знову або повернутися до меню рівнів. У разі поразки кількість життів автоматично зменшується, і якщо вони вичерпуються, може знадобитися кілька хвилин очікування відновлення, щоб отримати додаткову спробу пройти бажаний рівень.

У налаштуваннях гри, доступних через іконку шестерні у верхньому куті головного меню, можна відрегулювати гучність музики та звукових ефектів, змінити мову інтерфейсу або скинути прогрес до початкового рівня. Будь-які зміни в параметрах негайно зберігаються та відновлюються при наступному запуску гри.

Якщо під час гри виникають складнощі, наприклад, графічні артефакти або затримки анімації, рекомендується перевірити наявність вільного місця у пам’яті пристрою та оновити його операційну систему до останньої версії. У разі критичних багів можна звернутися до підтримки через відповідну форму у налаштуваннях, опис проблеми автоматично прикріпить лог-файл з останніми подіями, що допоможе оперативно усунути неполадки.

4.3 Висновки

Проведене тестування продемонструвало, що реалізовані механізми обробки взаємодії користувача, алгоритми пошуку та знищення збігів, а також

підсистеми відображення результатів проходять перевірку не лише в умовах контрольованого середовища емуляторів, але й на реальних пристроях із різним апаратним забезпеченням. Завдяки послідовному підходу до оптимізації продуктивності вдалося досягти стабільної частоти кадрів у всіх критичних сценах, що безпосередньо впливає на ігрове відчуття. Ретельне тестування сценаріїв повернення до меню рівнів засвідчило високу надійність механізму збереження та відновлення стану скролу, що покращує UX навіть у разі частих перезапусків гри.

Юзабіліті-аналіз підтвердив, що сенсорне керування й інтерфейс модуля ігрового вікна зрозумілі новачкам і не вимагають додаткових інструкцій для початку гри. Невеликі коригування у визначенні граничних зон дотику та оптимізація затримки між подіями touch_input і swap_tiles знизили кількість помилкових викликів назад до початкової позиції елемента, що покращило плавність взаємодії та зменшило фрустрацію користувачів.

Тестування вікон завершення матчу, як перемоги, так і поразки підтвердило коректність відображення зірок, звукових ефектів і переходів до подальших етапів гри без будь-яких «мертвих» станів чи ситуацій, коли кнопки не реагують на натискання. Синхронне програвання AnimationPlayer та виклики сигналів забезпечують чіткість і передбачуваність ігрового циклу, що є критично важливим для підтримки інтересу гравця та зменшення часу очікування між рівнями.

Загалом, результати випробувань свідчать про високу стабільність і надійність гри. У підсумку отримана версія мобільної гри відповідає сучасним вимогам якості та залишається ефективною й адаптивною в умовах швидкої еволюції мобільних пристроїв і операційних систем.

ВИСНОВКИ

У результаті виконання бакалаврської дипломної роботи було розроблено казуальну гру жанру «три в ряд» із використанням рушія Godot та мови GDScript. Спершу визначено функціональні та нефункціональні вимоги, проведено аналіз існуючих рішень на ринку та обґрунтовано вибір інструментарію для реалізації проєкту.

У другому розділі детально досліджено алгоритми побудови ігрового поля, застосування каскадної генерації з перевіркою на готові комбінації, процедурні методи з використанням шуму Перліна, кластерний аналіз та оптимізаційні підходи на основі регресійного моделювання і генетичних алгоритмів. Побудовані UML діаграми чітко відобразили послідовність основних етапів, від ініціалізації поля до видалення комбінацій і падіння нових фішок.

Третій розділ присвячено розробці ключових модулів у середовищі Godot та GDScript. Реалізовано ітераційний цикл гри, механізм перевірки ходів і функцію повернення назад у разі відсутності комбінацій, інтегровано кешування текстур і асинхронну обробку ресурсів для оптимальної продуктивності на мобільних пристроях. Окрему увагу приділено побудові інтерфейсу з урахуванням адаптивного дизайну, сенсорного управління та кросплатформних особливостей.

Четвертий розділ описує тестування розробленої гри, яке підтвердило її працездатність і відповідність очікуванням. Крім того, визначено мінімальні та рекомендовані вимоги до пристроїв для використання застосунку, а також розроблено інструкцію зі встановлення програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GodotEngine [Електронний ресурс] – Режим доступу до ресурсу: https://godotengine.org/ (дата звернення: 11.06.2025)

2. GDScriptDocumentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.godotengine.org/en/stable/getting_started/scripting/gdscript/gdscript_basics.html (дата звернення: 11.06.2025)

3. Гайдей С.О. Математичне моделювання комбінаторних систем у розробці казуальних ігор // Технологія-2025. Київ, 2025.

4. Гайдей С.О. Особливості сучасних комп'ютерних ігор // Інформаційні технології – 2025. Київ, 2025.

5. Schell J. The Art of Game Design: A Book of Lenses. CRCPress, 2019.

6. Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com/ua/ (дата звернення: 11.06.2025)

7. Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/ (дата звернення: 11.06.2025)

8. Swift [Електронний ресурс] – Режим доступу до ресурсу: https://swift.org/ (дата звернення: 11.06.2025)

9. Objective-C [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/objectivec (дата звернення: 11.06.2025)

10. OpenGL ES 3.2 Programming Guide. Addison-Wesley, 2016.

11. Unity [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com/ (дата звернення: 11.06.2025)

12. UnrealEngine [Електронний ресурс] – Режим доступу до ресурсу: https://www.unrealengine.com/ (дата звернення: 11.06.2025)

13. Defold [Електронний ресурс] – Режим доступу до ресурсу: https://defold.com/ (дата звернення: 11.06.2025)

14. Cocos2d-x [Електронний ресурс] – Режим доступу до ресурсу: https://www.cocos.com/en/ (дата звернення: 11.06.2025)

15. Buildbox [Електронний ресурс] – Режим доступу до ресурсу: https://www.buildbox.com/ (дата звернення: 11.06.2025)

16. GDevelop [Електронний ресурс] – Режим доступу до ресурсу: https://gdevelop.io/ (дата звернення: 11.06.2025)

17. GameSalad [Електронний ресурс] – Режим доступу до ресурсу: https://gamesalad.com/ (дата звернення: 11.06.2025)

18. Krug S. Don't Make Me Think. New Riders, 2005.

19. Lake A. Game Development with Godot and GDScript. Apress, 2023.

20. Freeman E., Robson E. Head First Design Patterns. O'Reilly, 2021.

21. ISO/IEC 25010:2011 – Software product quality requirements and evaluation (SQuaRE).

22. Stroustrup B. The C++ Programming Language. 4th ed. Addison-Wesley, 2013.

23. BlueprintVisualScripting [Електронний ресурс] – Режим доступу до ресурсу:https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/ (дата звернення: 11.06.2025)

24. Fowler M. Domain-Specific Languages. Addison-Wesley, 2010.

25. Knuth D. The Art of Computer Programming. Vol. 2. Addison-Wesley, 1997.

26. Smed T., Hakonen H. Algorithms and Networking for Computer Games. Wiley, 2006.

27. Perlin K. An Image Synthesizer // ACM SIGGRAPH, 1985.

28. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. Springer, 2009.

29. James G., Witten D., Hastie T., Tibshirani R. An Introduction to Statistical Learning. Springer, 2013.

30. UMLSpecification [Електронний ресурс] – Режим доступу до ресурсу: https://www.omg.org/spec/UML/ (дата звернення: 11.06.2025)

31. CandyCrushSaga [Електронний ресурс] – Режим доступу до ресурсу: https://king.com/game/candycrush (дата звернення: 11.06.2025)

32. BejeweledClassic [Електронний ресурс] – Режим доступу до ресурсу: https://www.ea.com/games/bejeweled (дата звернення: 11.06.2025)

33. HarryPotter: Puzzles&Spells [Електронний ресурс] – Режим доступу до ресурсу: https://harrypotterpuzzlesandspells.com/ (дата звернення: 11.06.2025)

34. JSONSpecification [Електронний ресурс] – Режим доступу до ресурсу: https://www.json.org/ (дата звернення: 11.06.2025)

35. AndroidDevelopers – Testingapps [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio/test (дата звернення: 11.06.2025)

ДОДАТКИ

Додаток А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» 09 2025 р.

Технічне завдання

на бакалаврську кваліфікаційну роботу «Розробка мобільної гри «Три в ряд» з використанням рушія Godot і GDScript» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи
д.т.н., проф. Романюк О.Н.
24 09 2025 року.

Виконала:
студентка гр.4ПІ-21б Гайдей С .О.
24 березня 2025 року

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота«Розробка мобільної гри «Три в ряд» з використанням рушія Godot та мови програмування GDScript».

Галузь застосування – мобільні ігри.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №18 від 18 лютого 2025 р.

3. Мета та призначення розробки.

Метою роботи є підвищення інтерактивності та функціональних можливостей мобільної 2D гри жанру «три в ряд» шляхом використання відкритого ігрового рушія Godot та мови програмування GDScript.

4. Вихідні дані для проведення НДР

1. Schell J. The Art of Game Design: A Book of Lenses. CRC Press, 2019.

2. Lake A. Game Development with Godot and GDScript. Apress, 2023.

3. Freeman E., Robson E. Head First Design Patterns. O'Reilly, 2021.

4. GodotEngine [Електронний ресурс] – Режим доступу до ресурсу: https://godotengine.org

5. Технічні вимоги.

Вхідні дані —Дії користувача в грі.

Вихідні дані — Результати пройденого рівня.

6. Конструктивні вимоги.

Інтерфейс програми поовинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні. Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

–пояснювальна записка до бакалаврської кваліфікаційної роботи;

–технічне завдання;

–лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз аналогів розроблюваного програмного продукту	25.03.25 – 30.03.25
2	Розробка методів та алгоритмів програмного застосунку	31.03.25 – 16.04.25
3	Аналіз алгоритмів розробки мобільних ігор	17.04.25 – 07.05.25
4	Розробка ключових алгоритмів гри	08.05.25 – 24.05.25
5	Тестування програми	25.05.25 – 27.05.25
6	Оформлення матеріалів до захисту БКР	28.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Усі етапи бакалаврської роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

81

Додаток Б

Протокол перевірки на наявність текстових запозичень

Назва роботи: «Розробка мобільної гри «Три в ряд» з використанням рушія Godot і GDScript»

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра ПЗ, факультет ФІТКІ, 4ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 6,4 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату,фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник_____ д.т.н., проф. каф. ПЗ Романюк О.Н.
(підпис) (прізвище, ініціали, посада)

Здобувач_____ Гайдей С.О.
(підпис) (прізвище, ініціали)

Додаток В

Лістинг програми

```
func save_scroll_position():
	var max_scroll = scroll_container.get_v_scroll_bar().max_value
	var current_scroll = scroll_container.scroll_vertical
	var normalized_position = 1.0 - (current_scroll / max_scroll)
	GameDataManager.save_scroll_position(normalized_position)
func restore_scroll_position():
	var saved_position = GameDataManager.load_scroll_position()
	#print("Loaded normalized scroll position: ", saved_position)
	if saved_position != null:
		await get_tree().process_frame
		var max_scroll = scroll_container.get_v_scroll_bar().max_value
		scroll_container.scroll_vertical = max_scroll * (1.0 - saved_position)
		#print("Restored scroll position: ", scroll_container.scroll_vertical)
	else:
		scroll_to_bottom()
func _on_scroll_changed(value):
	var max_scroll = scroll_container.get_v_scroll_bar().max_value
	var normalized_position = 1.0 - (value / max_scroll)
	GameDataManager.save_scroll_position(normalized_position)
	map_location.update_visible_buttons()
func _ready() -> void:
	load_bg_texture()
	$FadeOut/AnimationPlayer.play("fade_out")
	await $FadeOut/AnimationPlayer.animation_finished
	$FadeOut.visible = false
```

```
    func reposition_tiles():
        for i in width:
            for j in height:
                if i < all_tiles.size() and j < all_tiles[i].size() and all_tiles[i][j]
!= null:
                    all_tiles[i][j].position = grid_to_pixel(i, j)
                    all_tiles[i][j].scale = Vector2(scale_factor,
scale_factor)
    func is_in_array(array, item):
        for i in array.size():
            if array[i] == item:
                return true
        return false

    func make_2d_array():
        var array = []
        for i in width:
            array.append([])
            for j in height:
                array[i].append(null)
        return array
    func change_bomb(bomb_type, tile):
        if tile == null or not is_instance_valid(tile):
            return
        if bomb_type == BombType.ADJACENT:
            tile.make_adjacent_bomb()
        elif bomb_type == BombType.ROW:
            tile.make_row_bomb()
        elif bomb_type == BombType.COLUMN:
            tile.make_column_bomb()
```

```
func find_matches():
    if game_finished:
        return
    print("func find_matches start")
    var matches_found = false
    var all_matches = []
    var best_bomb = null
    var largest_single_match = 0
    for i in width:
        for j in height:
            if all_tiles[i][j] != null and not all_tiles[i][j].is_tile_bomb():
                var current_color = all_tiles[i][j].color
                var horizontal_matches = [Vector2(i, j)]
                var vertical_matches = [Vector2(i, j)]

                for x in range(i+1, width):
                    if all_tiles[x][j] != null and all_tiles[x][j].color
== current_color and not all_tiles[x][j].is_tile_bomb():
                        horizontal_matches.append(Vector2(x,
j))
                    else:
                        break

                for x in range(i-1, -1, -1):
                    if all_tiles[x][j] != null and all_tiles[x][j].color
== current_color and not all_tiles[x][j].is_tile_bomb():
                        horizontal_matches.append(Vector2(x,
j))
                    else:
                        break
```

```
                for y in range(j+1, height):
                    if all_tiles[i][y] != null and all_tiles[i][y].color
== current_color and not all_tiles[i][y].is_tile_bomb():
                        vertical_matches.append(Vector2(i, y))
                    else:
                        break

                for y in range(j-1, -1, -1):
                    if all_tiles[i][y] != null and all_tiles[i][y].color
== current_color and not all_tiles[i][y].is_tile_bomb():
                        vertical_matches.append(Vector2(i, y))
                    else:
                        break

                largest_single_match = max(largest_single_match,

    horizontal_matches.size(),

    vertical_matches.size())

                var bomb_type = BombType.NONE
                var bomb_position = Vector2(i, j)

                if not all_tiles[i][j].is_tile_bomb():
                    if horizontal_matches.size() >= 3 and
vertical_matches.size() >= 3:
                        bomb_type = BombType.ADJACENT
                    elif horizontal_matches.size() == 4:
                        bomb_type = BombType.ROW
```

```
                elif vertical_matches.size() == 4:
                    bomb_type = BombType.COLUMN
                elif horizontal_matches.size() >= 5 or vertical_matches.size() >= 5:
                    bomb_type = BombType.COLOR

            if horizontal_matches.size() >= 3:
                for match_pos in horizontal_matches:
                    if not all_matches.has(match_pos):
                        all_matches.append(match_pos)
                matches_found = true

            if vertical_matches.size() >= 3 and vertical_matches != horizontal_matches:
                for match_pos in vertical_matches:
                    if not all_matches.has(match_pos):
                        all_matches.append(match_pos)
                matches_found = true

            if bomb_type != BombType.NONE:
                if best_bomb == null or bomb_type > best_bomb.type:
                    best_bomb = {
                        "type": bomb_type,
                        "position": bomb_position,
                        "color": current_color
                    }

        var total_matches = 0
        for match in all_matches:
```

```
                var match_size = all_matches.size()

                if GameDataManager.is_endless_mode_enabled():
                    match match_size:
                        5:
                            current_counter_value += 1
                        4:
                            current_counter_value += 1
                        3:
                            current_counter_value += 1
                    emit_signal("update_counter", current_counter_value)

                if all_tiles[match.x][match.y] != null and not all_tiles[match.x][match.y].is_tile_bomb():
                    match_and_dim(all_tiles[match.x][match.y])
                    add_to_array(match)
                    total_matches += 1
            if matches_found:
                combo_count += 1
                play_combo_sound()
            else:
                combo_count = 0
            if largest_single_match > 0:
                emit_signal("match_made", current_matches.size(), largest_single_match)
            if best_bomb != null:
                print("Створення бомби типу ", best_bomb.type, " на позиції ", best_bomb.position)
                if all_tiles[best_bomb.position.x][best_bomb.position.y] != null and not all_tiles[best_bomb.position.x][best_bomb.position.y].is_tile_bomb():
```

```
                    emit_signal("check_goal", best_bomb.color)
                    make_bomb(best_bomb.type, best_bomb.position.x,
best_bomb.position.y)
            get_bombed_tiles()
            destroy_timer.start()

    func is_piece_null(column, row, array = all_tiles):
            if array[column][row] == null:
                    return true
            return false

    func match_and_dim(tile): #add sound
            if tile != null:
                    tile.matched = true
                    tile.dim()

    func add_to_array(value, array_to_add = current_matches):
            if !array_to_add.has(value):
                    array_to_add.append(value)

    func is_tile_null(column, row):
            if all_tiles[column][row] == null:
                    return true
            return false

    func find_bombs():
            var bomb_created = false
            for i in current_matches.size():
                    var current_column = current_matches[i].x
                    var current_row = current_matches[i].y
```

```
                if all_tiles[current_column][current_row] == null:
                        continue
                var current_color = all_tiles[current_column][current_row].color
                var col_matched = 0
                var row_matched = 0
                for j in current_matches.size():
                        var this_column = current_matches[j].x
                        var this_row = current_matches[j].y
                        if all_tiles[this_column][this_row] == null:
                                continue
                        var this_color = all_tiles[this_column][this_row].color
                        if this_column == current_column and current_color == this_color:
                                col_matched += 1
                        if this_row == current_row and current_color == this_color:
                                row_matched += 1
                if not bomb_created:
                        if col_matched == 5 or row_matched == 5:
                                make_bomb(BombType.COLOR, current_column, current_row)
                                bomb_created = true
                                return
                        if col_matched >= 3 and row_matched >= 3:
                                make_bomb(BombType.ADJACENT, current_column, current_row)
                                bomb_created = true
                                return
                        if col_matched == 4:
                                make_bomb(BombType.COLUMN, current_column, current_row)
```

```
                    bomb_created = true
                    return
                if row_matched == 4:
                    make_bomb(BombType.ROW, current_column, current_row)
                    bomb_created = true
                    return

    func destroy_matches():
        if game_finished:
            return
        var was_matched = false
        var tiles_destroyed = 0
        var match_size = 0
        bombs_to_process.clear()
        for i in width:
            for j in height:
                if all_tiles[i][j] != null and all_tiles[i][j].matched:
                    if all_tiles[i][j].is_tile_bomb():
                        bombs_to_process.append(all_tiles[i][j])
                        was_matched = true
                        match_size += 1

        for i in width:
            for j in height:
                if all_tiles[i][j] != null and all_tiles[i][j].matched:
                    if not all_tiles[i][j].is_tile_bomb():
                        was_matched = true
                        match_size += 1
                        emit_signal("check_goal", all_tiles[i][j].color)
```

```
                                emit_signal("update_score", tile_value * streak)
                                all_tiles[i][j].explode()
                                all_tiles[i][j] = null
                                tiles_destroyed += 1
        if tiles_destroyed > 0:
                emit_signal("match_made", match_size)
        move_checked = true
        if was_matched:
                process_next_bomb()
        else:
                swap_back()
                streak_timer.stop()
                match_streak = 0
        current_matches.clear()

func collapse_columns():
        if game_finished:
                return
        for i in width:
                for j in range(height - 1, -1, -1):
                        if all_tiles[i][j] == null and !restricted_fill(Vector2(i, j)):
                                for k in range(j - 1, -1, -1):
                                        if all_tiles[i][k] != null:
                                                all_tiles[i][k].move(grid_to_pixel(i, j))
                                                all_tiles[i][j] = all_tiles[i][k]
                                                all_tiles[i][k] = null
                                                break
        refill_timer.start()

func refill_columns():
```

```
    if game_finished:
        return
    streak += 1
    var tiles_moved = false
    for i in width:
        for j in height:
            if all_tiles[i][j] == null && !restricted_fill(Vector2(i, j)):
                spawn_single_tile(i, j)
                all_tiles[i][j].position = grid_to_pixel(i, -1)
                all_tiles[i][j].move(grid_to_pixel(i, j))
                tiles_moved = true
    if tiles_moved:
        moving_tiles_sound.play()
    after_refill()

func spawn_new_tile(i, j):
    var tile_type = randi() % possible_tiles.size()
    var new_tile = possible_tiles[tile_type].instantiate()
    new_tile.reset_bomb_status()
    new_tile.is_bomb = false
    add_child(new_tile)
    all_tiles[i][j] = new_tile
    return new_tile

func after_refill():
    #print("Перевірка після заповнення")
    var found_match = false
    for i in width:
        for j in height:
            if all_tiles[i][j] != null && !all_tiles[i][j].is_tile_bomb():
```

```
                        if match_at(i, j, all_tiles[i][j].color):
                            found_match = true
                            break

        if found_match:
            state = wait
            find_matches()
            get_bombed_tiles()
            destroy_timer.start()
        else:
            state = move
            streak = 1
            move_checked = false
            combo_count = 0
            if is_moves:
                if not (GameDataManager.is_tutorial_mode and
current_counter_value <= 2):
                    current_counter_value -= 1
                    emit_signal("update_counter", current_counter_value)
                    if current_counter_value == 0:
                        declare_game_over()
            check_and_shuffle()


    func match_all_in_column(column):
        if column in checked_columns:
            return
        checked_columns.append(column)
        for i in height:
            if all_tiles[column][i] != null:
```

```
                    if all_tiles[column][i].is_row_bomb:
                        match_all_in_row(i)
                    if all_tiles[column][i].is_adjacent_bomb:
                        find_adjacent_tiles(column, i)
                    all_tiles[column][i].matched = true

func match_all_in_row(row):
    if row in checked_rows:
        return
    checked_rows.append(row)
    for i in width:
        if all_tiles[i][row] != null:
            if all_tiles[i][row].is_column_bomb:
                match_all_in_column(i)
            if all_tiles[i][row].is_adjacent_bomb:
                find_adjacent_tiles(i, row)
            all_tiles[i][row].matched = true

func find_adjacent_tiles(column, row):
    for i in range(-1, 2):
        for j in range(-1, 2):
            if  is_in_grid(Vector2(column + i, row + j)):
                if all_tiles[column + i][row + j] != null:
                    all_tiles[column + i][row + j].matched = true

func get_bombed_tiles():
    checked_rows.clear()
    checked_columns.clear()
    activated_bombs.clear()
```

```
            var bombs_to_activate = []

            for i in width:
                for j in height:
                    var tile = all_tiles[i][j]
                    if tile != null and tile.matched and tile.is_tile_bomb():
                        bombs_to_activate.append({"tile": tile, "pos": Vector2(i, j)})

            for bomb_data in bombs_to_activate:
                var bomb = bomb_data.tile
                if is_instance_valid(bomb) and not activated_bombs.has(bomb):
                    activated_bombs.append(bomb)
                    call_deferred("activate_bomb", bomb)
        func setup_goals(level_data):
            for child in goal_container.get_children():
                child.queue_free()
            var goals = level_data.get("goals", [])
            for goal in goals:
                if "max_needed" in goal and "texture" in goal and "goal_string" in goal:
                    var texture = load(goal["texture"])
                    if texture:
                        make_goal(goal["max_needed"], texture, goal["goal_string"])
                        #print("Goal displayed: ", goal["goal_string"], " - ", goal["max_needed"])
                    else:
                        print("Error: Could not load texture ", goal["texture"])
                else:
```

```
                print("Error: Invalid goal data ", goal)
        #print("Total goals displayed: ", goal_container.get_child_count())

func _on_grid_update_score(amout_to_change):
        if $"../CenterContainer/grid".game_finished:
                return
        current_score += amout_to_change
        update_score_bar()
        score_label.text = str(current_score)
        update_stars_display()



func _on_grid_update_counter(new_value):
        if $"../CenterContainer/grid".game_finished:
                return
        current_count = int(round(new_value))

        #print(current_count)
        var is_moves = $"../CenterContainer/grid".is_moves
        if is_moves:
                update_moves_display(current_count)
        else:
                var minutes: int = current_count / 60
                var seconds: int = current_count % 60
                counter_label.text = "%01d:%02d" % [minutes, seconds]
```

Додаток Г

Презентація

««Розробка мобільної гри «Три в ряд» з використанням рушія Godot і GDScript»»

Студентка: ст. гр. 4ПІ-21б Гайдей С.О.
Керівник: д.т.н., професор Романюк О.Н.

Рисунок Г.1 – слайд 1

Актуальність

- Одним із найпопулярніших жанрів у сфері мобільного геймінгу є казуальні ігри, зокрема «Три в ряд». Їх популярність пояснюється простотою механіки, яскравою графікою, інтуїтивним інтерфейсом та здатністю утримувати увагу користувача протягом тривалого часу. Проте багато аналогів мають низку недоліків, серед яких надмірна складність для новачків, високе навантаження на ресурси пристрою, а також одноманітність ігрових механік. З огляду на це, створення власної гри цього жанру є актуальним і виправданим як з технічного, так і з науково-практичного погляду, оскільки дає змогу усунути зазначені недоліки та запропонувати більш збалансований і доступний ігровий продукт.

Рисунок Г.2 – слайд 2

Мета та завдання дослідження

- Метою даної роботи підвищення інтерактивності та функціональних можливостей мобільної 2D гри жанру «три в ряд» шляхом використання відкритого ігрового рушія Godot та мови програмування GDScript.

Наукова новизна

1. Подальшого розвитку отримав метод побудови ігрового рівня, заснований на поетапній псевдовипадковій генерації з перевіркою на відсутність готових комбінацій, що дозволяє забезпечити збалансовану складність та підвищити якість гри без додаткових обчислювальних витрат.

2. Подальшого розвитку набув метод створення ігрової логіки з використанням динамічної варіативності на основі випадкових подій, що забезпечує різноманітність ігрових ситуацій, прогнозованість реакцій системи та стійкий інтерес користувача до процесу гри.

Рисунок Г.3 – слайд 3

Основними задачами дослідження є:

- здійснити аналіз сучасного ринку мобільних ігор та визначити їх особливості;
- дослідити методи створення алгоритмів для мобільних ігор;
- реалізувати базову функціональність гри, включаючи ігрові механіки, систему рівнів та збереження даних гравця;
- провести комплексне тестування гри та оцінити ефективність використаних засобів.

Рисунок Г.4 – слайд 4

Практичне значення отриманих результатів

- Практичне значення отриманих результатів полягає в тому, що на основі проведеного аналізу створення ігор розроблено мобільну гру «Три в ряд», яка поєднує в собі зручність та швидкодію.

Рисунок Г.5 – слайд 5

Аналіз основних методів побудови мобільних ігор

- Розробка мобільних ігор є складним процесом, що поєднує в собі елементи програмування, дизайну, геймдизайну, тестування та оптимізації під різні платформи.
- Нативна розробка мобільних ігор передбачає створення додатків із використанням мов програмування та інструментів, характерних для конкретної платформи.
- Кросплатформна розробка базується на використанні рушіїв, що дозволяють створювати гру один раз і експортувати її одразу на кілька платформ.

Рисунок Г.6 – слайд 6

Вибір ігрового рушія

- Особливої уваги заслуговує рушій Godot, який активно набирає популярності через відкритий вихідний код, безплатну ліцензію, активну спільноту та простий у використанні скриптовий інтерфейс GDScript, спеціально створений для потреб ігрової розробки.

Рисунок Г.7 – слайд 7

Розробка методу динамічної варіативної логіки гри

У центрі внутрішньої логіки розроблюваної гри «три в ряд» лежить ітеративний цикл взаємодії користувача з ігровим полем, в якому кожен рух фішок ретельно обробляється з урахуванням наявності комбінацій та можливості повернення ходу назад.

Якщо перестановка не призвела до утворення хоча б однієї групи з трьох і більше однакових фішок у суцільному ряду або стовпці, запускається процедура повернення ходу назад.

Якщо ж після перестановки виявлено одну або більше комбінацій з трьох і більше фішок одного типу, алгоритм переходить до етапу пошуку відповідних комбінацій.

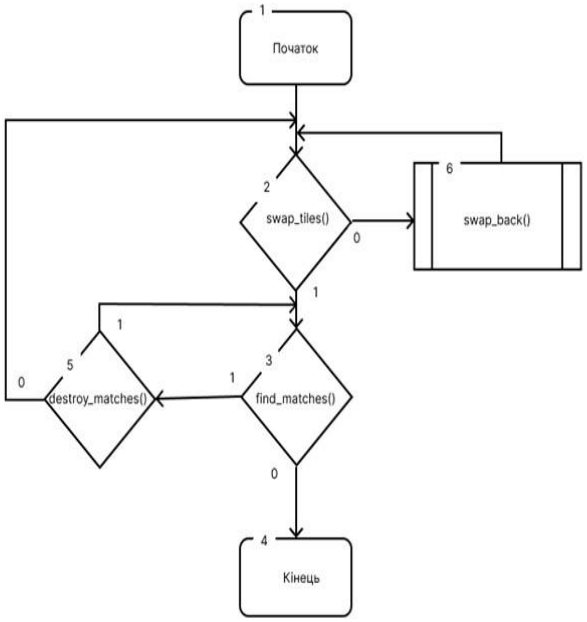

Рисунок Г.8 – слайд 8

Розробка методу ігрової логіки на основі станів

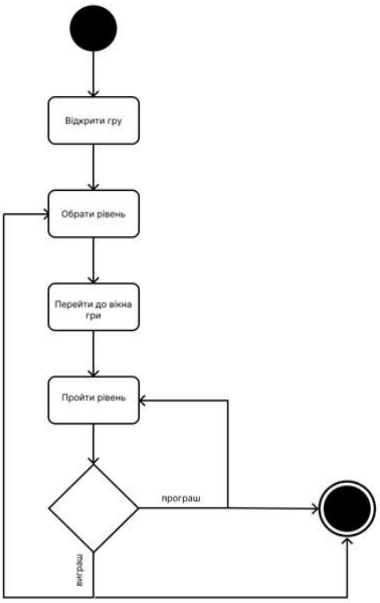

Рисунок Г.9 – слайд 9

Аналоги

Candy Crush

Bejeweled Classic

Рисунок Г.10 – слайд 10

Аналоги

Harry Potter: Puzzles & Spells

Власна розробка

Рисунок Г.11 – слайд 11

Тестування гри

Результат проходження рівня

Процес проходження рівня

Рисунок Г.12 – слайд 12

Тестування гри

Вибір рівня

Процес проходження рівня

Рисунок Г.13 – слайд 13

Висновки

- У результаті виконання бакалаврської дипломної роботи було розроблено казуальну гру жанру «Три в ряд» із використанням рушія Godot та мови GDScript. Спершу визначено функціональні та нефункціональні вимоги, проведено аналіз існуючих рішень на ринку та обґрунтовано вибір інструментарію для реалізації проєкту.

Рисунок Г.14 – слайд 14