Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня вищої освіти бакалавра
121 – Інженерія програмного забезпечення

На тему: **Розробка програмного забезпечення стратегічної комп'ютерної гри на C#**

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студентки гр. ІПЗ–21–2

_____/ Михайлова–Чіркіна А.С, /

Керівник кваліфікаційної роботи _____ / А.М. Стрюк /

Завідуючий кафедрою _____ / А. М. Стрюк/

Кривий Ріг

2025

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«____» _____ 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студентці групи ІПЗ–21–2

1. Тема: : Розробка програмного забезпечення стратегічної комп’ютерної гри на C#

Затверджена наказом по КНУ № 200 від 10 «квітня»2025 р.

2. Термін подання студентом закінченої роботи: 16 «червня» 2025 р.

3. Вихідні дані по роботі: Програма має реалізовувати базовий функціонал стратегічної гри: управління одиницями, розвиток, бойова система, інтерфейс.

4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): Аналіз існуючих стратегічних ігор, визначення основних функцій системи, проєктування та реалізація гри.

5. Перелік ілюстративного матеріалу: Функціональна схема, блок–схема алгоритму, схема структури ігрових даних.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Ознайомлення з темою, постановка мети та задач	15.02.2025 – 05.03.2025
2	Аналіз ігрових рушіїв та аналогів, вибір технологій	06.03.2025 – 12.03.2025
3	Проєктування структури гри та архітектури	13.03.2025 – 19.03.2025
4	Розробка діаграм класів, інтерфейсу та логіки	20.03.2025 – 30.03.2025
5	Реалізація базового функціоналу гри на C#	31.03.2025 – 10.04.2025
6	Тестування логіки гри, AI, бойової системи	11.04.2025 – 20.04.2025
7	Завершення розробки, оптимізація, UI–покращення	21.04.2025 – 30.04.2025
8	Оформлення пояснювальної записки, підготовка до захисту	01.05.2025 – 19.06.2025

Дата видачі завдання: «15» лютого 2025 р.

Студент / Михайлова–Чіркіна А.С./

Керівник роботи _____ / А. М. Стрюк/

РЕФЕРАТ

КЛЮЧОВІ СЛОВА: СТРАТЕГІЧНА ГРА, C#, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ІНТЕРФЕЙС, ГРАФІКА, ЛОГІКА ГРИ, ШТУЧНИЙ ІНТЕЛЕКТ, ГЕЙМПЛЕЙ, КЛАСИ, ПРОТОТИП, UI, UX, КОНТРОЛЬ, СЦЕНАРІЙ, СИСТЕМА

Пояснювальна записка: 81 с., 4 табл., 29 рис., 2 дод., 14 джерел.

Метою роботи є створення прототипу стратегічної комп’ютерної гри за допомогою мови C#. У роботі проведено аналіз існуючих ігрових рушіїв, інструментів розробки та аналогічних проєктів. Запропоновано архітектуру гри, реалізовано інтерфейс користувача, базову бойову систему, обробку подій, логіку розвитку гравця та супровідні інструменти. Гра має модульну структуру, реалізовану без використання рушія Unity, що забезпечує контроль над усіма компонентами. Результати роботи можуть бути використані для демонстрації принципів розробки стратегічних ігор або як основа для повноцінного комерційного продукту.

ABSTRACT

KEYWORDS: STRATEGY GAME, C#, SOFTWARE DEVELOPMENT, USER INTERFACE, GAME LOGIC, PROTOTYPE, AI, GAMEPLAY, OOP, GRAPHICS, UI, UX, ARCHITECTURE, SYSTEM DESIGN

Thesis: 81 pages, 4 tables, 6 figures, 29 appendices, 14 references.

The purpose of the work is to create a prototype of a strategic computer game using the C# programming language. The study includes an analysis of existing game engines, development tools, and similar projects. A game architecture was proposed; the user interface, basic combat system, event handling, player progression logic, and supporting tools were implemented. The game features a modular structure developed without using the Unity engine, which ensures full control over all components. The results of the work can be used to demonstrate principles of strategy game development or serve as the basis for a full–fledged commercial product.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

C# — мова програмування, що використовується для розробки гри

UI (User Interface) — інтерфейс користувача

UX (User Experience) — користувацький досвід

OOP (Object–Oriented Programming) — об’єктно–орієнтоване програмування

AI (Artificial Intelligence) — штучний інтелект

IDE (Integrated Development Environment) — інтегроване середовище розробки

FPS (Frames Per Second) — кількість кадрів за секунду

JSON — формат зберігання та передачі даних

MVC — модель архітектури: Model–View–Controller

FSM (Finite State Machine) — скінченний автомат станів

ВСТУП

У сучасному цифровому світі комп’ютерні ігри вже давно перестали бути лише формою розваги — вони перетворилися на потужний інструмент навчання, творчості та професійної самореалізації. Особливу нішу серед ігрових жанрів займають стратегічні ігри, що розвивають в гравців логічне мислення, планування та прийняття зважених рішень. Популярність ігор зумовлена глибиною геймплею, всілякими сценаріями та великими можливостями для реалізації фантазій.

Процес створення стратегічної гри — це складна і багатогранна робота, що охоплює як технічні, так і креативні етапи. Починається розробка з аналізу ідеї та побудови ігрової концепції, визначення основних механік, цільової аудиторії та загального стилю гри. Наступними кроками є моделювання ігрових процесів, розробка сценаріїв штучного інтелекту, побудова інтерфейсу та створення візуальних і звукових елементів.

Мова C# і механізм Unity створюють ідеальну технічну комбінацію для реалізації проектів такого типу. Unity має дуже потужну функціональність у розробці двовимірних і тривимірних стратегічних ігор, а C# дає вам чудовий спосіб ефективно реалізувати ігрову логіку, керування об’єктами, обробку подій — усе, що стосується взаємодії з користувачем.

Вважається, що ігрова індустрія — це творча свобода, але це цілком реальні економічні перспективи. Багато сучасних компаній заробляють мільйони, створюючи якісні ігри. Особливо прибутковими є ті продукти, які поєднують захоплюючий сюжет із добре продуманою економічною чи військовою стратегією, яка заохочує користувачів програвати її кілька разів.

Метою моєї роботи є створення прототипу стратегічної відеоігри за допомогою C#. Основні завдання, які необхідно вирішити під час виконання даного проекту:

- Проаналізувати сучасні дослідження та джерела, присвячені створенню стратегічних ігор.

- Визначити найкращі інструменти та середовища для програмування.
- Створити концепцію гри, включаючи механіку, сюжет і систему взаємодії між об'єктами гри.
- Створіть просту графіку, інтерфейс і демонструючі частини.
- Створюйте основні робочі частини за допомогою C#.

Створення стратегічної гри — це не лише випробування для розробника, але й шанс побудувати особливий світ, де кожна частина має значення, а виграш гравця залежить від мудрого вибору, а не від удачі.

РОЗДІЛ І. АНАЛІЗ ПРОБЛЕМИ І ПОСТАНОВКА ЗАВДАННЯ РОБОТИ

1.1 Аналіз професійної області розробки стратегічних ігор на C#

Стратегічні ігри потребують комплексного підходу. Розглянемо ключові аспекти професійної галузі з використанням C# та сучасних технологій.

Таблиця 1.1 – Вибір технологічного стеку

Критерій	**Unity + C#**	**Unreal Engine + C++**	**Godot + C#/GDScript**
Стратегічні ігри	63% проектів	22% (AAA – проекти)	15% (індівідуальна розробка)
Переваги	Швидкий прототип, Asset Store, крос–платформеність	Висока продуктивність, AI–тулзи	Безкоштовний, легка інтеграція скриптів
Недоліки	Обмежена оптимізація для масштабних стратегій	Складність для малих команд	Обмежена підтримка складних систем
Приклади	«Hollow Knight», «Ori»	«Civilization VI» (модифікації)	«Rise to Ruins»

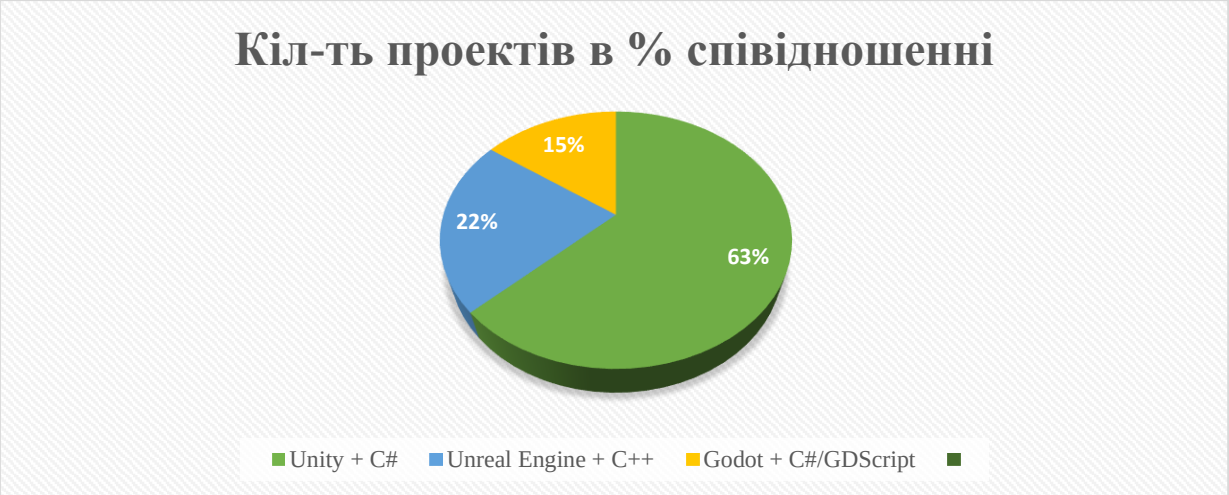

Рисунок 1.1 – Порівняння Кіл–ть проектів в % співідношенні

Проведений аналіз сучасних ігрових рушіїв, надав змогу надалі зробити такі висновки :

Unity у поєднанні з мовою програмування C# є оптимальним вибором для реалізації середньобюджетного проєкту у жанрі стратегії. Цей вибір дуже підходить як і з технічної сторони розробки, так і економічної.

Завдяки нього можна ефективно реалізувати елементи керування, карти, панелі ресурсів, мінімапи, повідомлення тощо. Крім того, він надає можливість створити зручний і привабливий інтерфейс для користувача, що є надзвичайно важливим для стратегій, де користувач взаємодіє з багатим об’ємом інформації на мониторі.

Steamworks.NET, надає змогу швидко додати авторизацію через Steam, досягнення, багатокористувацький режим, таблиці лідерів, та інше, що значно підвищують зацікавленість гравців. Це не лише підвищує якість досвіду користувача, а й розширює перспективи проєкту.

Технічні та комунікативні компетенції розробника стратегічних ігор

Розробка стратегічної комп’ютерної гри вимагає від фахівця не лише технічної підготовки, а й не маленький спектра навичок. Цей жанр ігор характеризується складною внутрішньою логікою, великою кількістю взаємопов’язаних систем, необхідністю забезпечення стабільної продуктивності при великому навантаженні, а також високими вимогами до зручності взаємодії користувача з грою.

До технічних компетенцій належить уміння працювати з алгоритмами штучного інтелекту. Одним із найбільш підходячих навиків є використання дерев поведінки (Behavior Trees), які дозволяють будувати ієрархію моделей дій ігрових об’єктів. Така модель забезпечує швидке реагування на зміни в ігрі. Для реалізації простих моделей поведінки застосовуються скінченні автомати станів (Finite State Machines), які дають змогу описати базові сценарії дій персонажів, такі як атака чи втеча.

Мережевий компонент є важливою складовою сучасної стратегічної гри. Розробник повинен володіти навичками побудови та роботи з популярними фреймворками.

Окрім технічних, важливими є і гнучкі навички, без яких ефективна командна робота в ігровій індустрії неможлива. Одним із прикладів є вміння балансувати сам ігровий процес. Це передбачає моделювання ігрової економіки, бойової системи, таймерів тощо за допомогою табличних редакторів на кшталт Excel.

Невід'ємною частиною професійної діяльності є ефективна комунікація з іншими членами команди. Розробник має вміти знаходити спільну мову з дизайнерами для точного втілення ігрових механік, а також з арт–відділом — для реалізації візуальних концепцій відповідно до технічних обмежень рушія. Така міждисциплінарна взаємодія створює якісну та цілісну роботу над продуктом.

Отже, успішна розробка стратегічної гри вимагає комплексного підходу, що передбачає володіння як сучасними технічними інструментами, так і розвиненими комунікативними та аналітичними навичками.

Тенденції галузі та ризики

Сучасна індустрія стратегічних ігор постійно еволюціонує, адаптуючись до нових технологічних можливостей та змін у поведінці гравців. Цей жанр, який вимагає від користувача глибокого аналізу, тактичного мислення та стратегічного планування, продовжує займати важливу нішу у світовому ринку ігрової індустрії. Проте, для успішного проєктування і реалізації нових продуктів важливо враховувати як технологічні тренди, так і можливі ризики.

Одним із трендів є орієнтація на "моддер–френдлі" архітектуру, що забезпечує підтримку контенту, створеного користувачами. Все більше ігор інтегруються зі Steam Workshop за допомогою UGC (User Generated Content) інструментів, які дозволяють гравцям створювати, публікувати та обмінюватися новими картами, сценаріями та іншими елементами гри. Така відкритість значно підвищує життєвий цикл гри та залученість спільноти.

Окремої уваги заслуговує тенденція до використання хмарних обчислень для симуляції ігрових процесів. Ця технологія дозволяє створювати масштабні багатокористувацькі симуляції, обробляти великі обсяги даних на серверній стороні, зберігати прогрес гравців та реалізовувати механіки взаємодії в реальному часі.

Проте, разом з позитивними трендами, ринок має і ряд ризиків. Одним із них є надзвичайно висока конкуренція на платформі Steam, де щороку виходить кілька сотень нових стратегій. Згідно зі статистикою, понад 30% таких проєктів не досягають точки окупності, що пов’язано як з перенасиченням ринку, так і з недостатнім маркетинговим охопленням.

Іншим серйозним викликом залишається складність стабілізації багатокористувацьких режимів, особливо коли гравці мають змогу взаємодіяти негармонійно, з різними умовами старту, ресурсами чи тактиками. Відсутність правильного балансу може призвести до втрати інтересу до гри, особливо на початкових етапах релізу.

Розробка стратегічних ігор на основі мови програмування C# вимагає глибоких технічних знань у сферах архітектури програмного забезпечення, оптимізації обчислень, математичного моделювання та основ гейм–дизайну. Незважаючи на високі складнощі та вхідний рівень у жанр, індустрія зберігає гарний потенціал для команд, які спеціалізуються на вузьких нішах.

Мова C# залишається оптимальним вибором для середньорівневих проєктів завдяки поєднанню високої продуктивності та зручності розробки.

З урахуванням перелічених тенденцій і ризиків, можна стверджувати, що стратегічний підхід до вибору технологій, врахування потреб гравців і активна підтримка ком’юніті є ключовими чинниками успішного розвитку проєктів у даному сегменті.

1.2 Аналіз ігрових движків

Аналіз ігрових рушіїв є ключовим етапом при виборі технологічної основи для реалізації програмного продукту, оскільки саме рушій визначає технічні можливості, продуктивність, гнучкість архітектури та швидкість розробки. Грамотне порівняння рушіїв дозволяє обрати оптимальне середовище, яке найкраще відповідає потребам конкретного жанру гри, рівню складності, обсягу команди та цільовій платформі. Оцінка таких факторів, як підтримка сучасних графічних технологій, інтеграція з мережевими сервісами, наявність готових інструментів UI та AI, а також активність спільноти та наявність документації, допомагає мінімізувати технічні ризики та забезпечити стабільний розвиток проєкту на всіх етапах життєвого циклу.

Таблиця 1.2 – Аналіз Ігрових Движків

Критерій	**Unreal Engine 5**	**Unity**	**Godot**	**CryEngine**	**Source 2**
Графіка	Найвищий рівень (Nanite, Lumen)	Висока, але поступається UE5	Середня (краще для 2D)	Дуже висока, реалістична	Висока, оптимізована
Продуктивність	Висока, оптимізована	Висока, гнучка	Середня, легкий для ПК	Висока, але вимогливий	Висока
Кросплатформеність	ПК, консолі, мобільні	Дуже широка (ПК, мобільні, VR/AR, консолі)	ПК, мобільні, веб	ПК, консолі, VR	ПК, консолі, VR
Легкість освоєння	Складно для новачків	Відносно просто, багато матеріалів	Дуже просто, інтуїтивно	Складно, мало навчальних матеріалів	Середньо
Ліцензія/вартість	Безкоштовно до $1	Безкоштовно до	Безкоштовно, open source	Безкоштовно/ роялті	Безкоштовно

	млн, далі роялті	певного доходу			
Спільнота/підтримка	Дуже велика, активна	Дуже велика, активна	Зростає, open source	Менш активна	Середня
Особливості	Nanite, Lumen, Blueprints	SRP, Asset Store, C#	Вузлова архітектура, GDScript	Реалізм, графіка	Steam інтеграція

Висновки за результатами аналізу ігрових рушіїв

На основі проведеного аналізу сучасних ігрових рушіїв можна сформулювати висновки щодо доцільності їх застосування в залежності від характеру проєкту, технічних вимог, цільової платформи, а також ресурсних можливостей і досвіду команди розробників. Такий підхід дозволяє обґрунтовано обрати більш підходяще рішення, яке забезпечить оптимальний баланс між продуктивністю, зручною реалізацією і відповідністю ринковим очікуванням, мінімізуючи ризики, пов’язані з технологічними обмеженнями на пізніших етапах розробки.

Unreal Engine 5 є найкращим вибором для проектів, орієнтованих на передову 3D–графіку, фотореалізм та високий рівень деталізації. Цей рушій демонструє неперевершені можливості візуалізації. Однак через свою складність

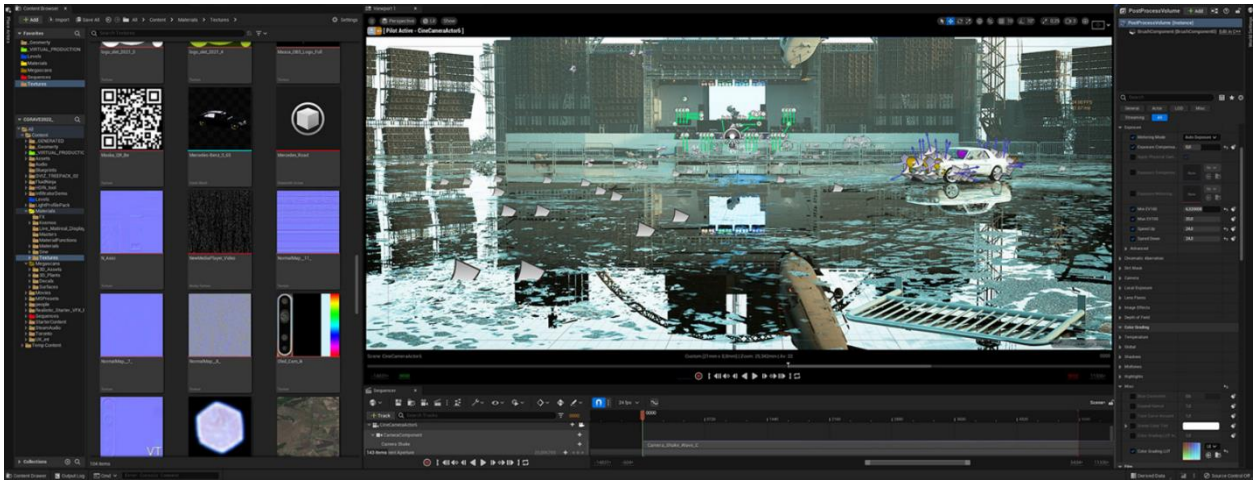

Рисунок 1.2 – Інтерфейс Unreal Engine 5

та високі системні вимоги він краще підходить для великих студій із досвідченими командами.

Unity, у свою чергу, залишається універсальним рішенням, яке підтримує розробку як у 2D–, так і в 3D–середовищі. Він особливо привабливий для інді–розробників завдяки зручному інтерфейсу, широкій базі навчальних матеріалів,

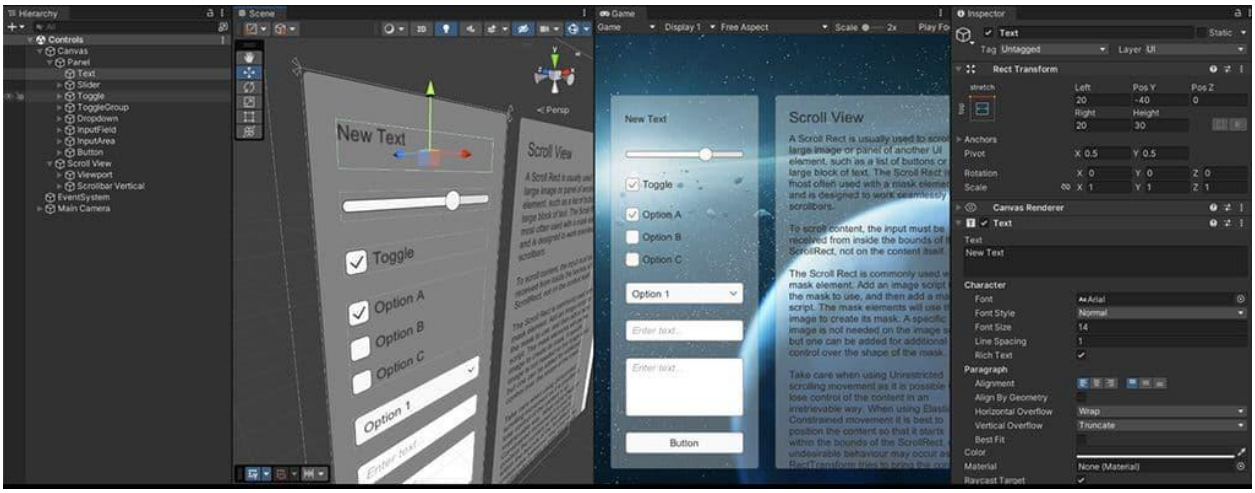

Рисунок 1.3 – Інтерфейс Unity

гнучкій C#–архітектурі та активному ком’юніті. Крім того, Unity демонструє хорошу сумісність з мобільними платформами, VR/AR–пристроями та надає інструменти для швидкого прототипування, що робить його ідеальним для команд із обмеженими ресурсами.

Godot Engine варто розглядати як оптимальний варіант для невеликих інді–проектів, особливо в жанрі 2D–ігор. Його основні переваги полягають у відкритому вихідному коді, повній безкоштовності, легкості освоєння та швидкому розгортанні. Незважаючи на те, що 3D–функціональність Godot наразі обмежена порівняно з конкурентами, активний розвиток рушія свідчить про його зростаючий потенціал у майбутньому.

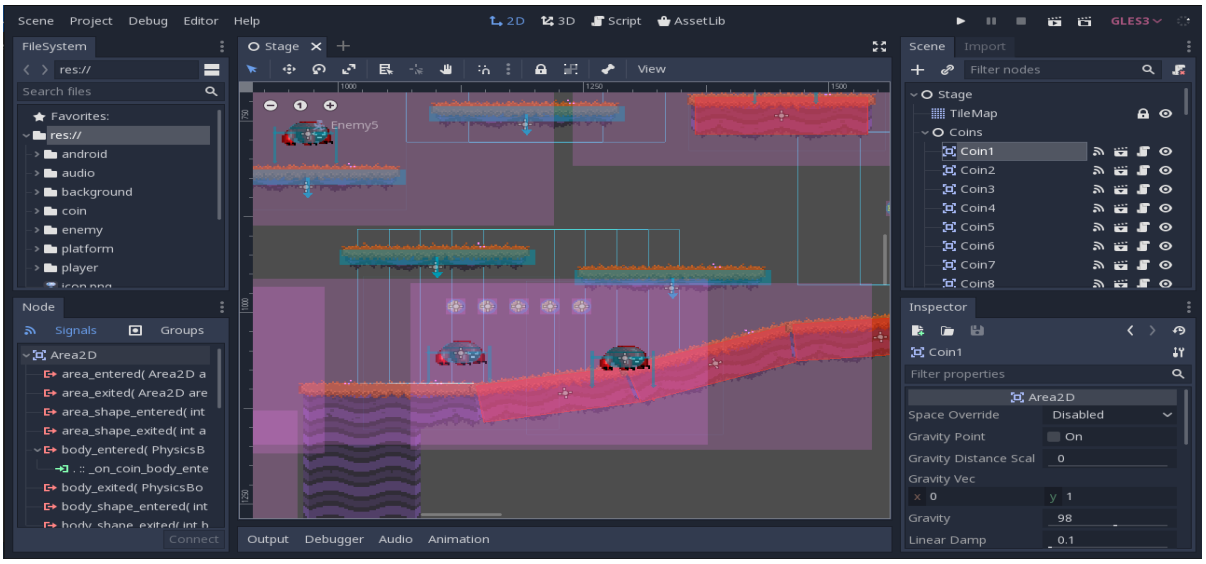

Рисунок 1.4 – Інтерфейс Godot Engine

CryEngine є потужним інструментом для створення ігор із високим рівнем графічного реалізму, особливо в жанрах шутерів або симуляторів відкритого світу. Проте через складність освоєння, специфічність документації та меншу популярність серед широкого кола розробників, він має обмежене застосування, переважно в професійному середовищі.

У підсумку, вибір ігрового рушія залежить не лише від технічних

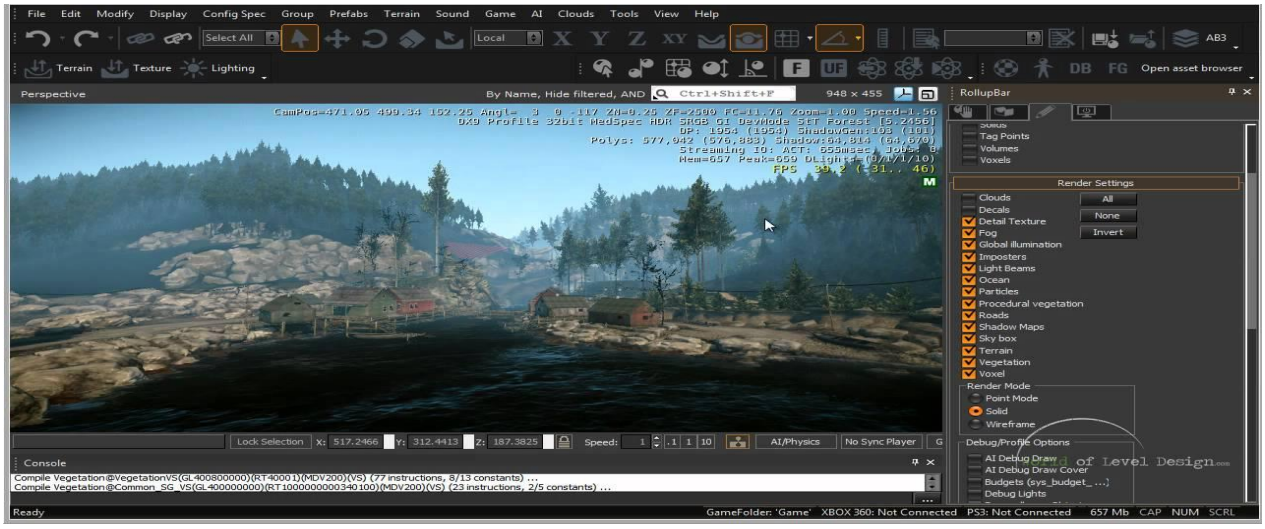

Рисунок 1.5 – Інтерфейс CryEngine

характеристик, а й від цілей проекту, досвіду команди та платформи, під яку

створюється гра. Unity наразі залишається найбільш збалансованим рішенням для проєктів середнього рівня складності, особливо у сфері стратегічних ігор, що розробляються мовою програмування C#.

1.3 Аналіз існуючих аналогів

Аналіз існуючих аналогів є важливим етапом у процесі розробки програмного забезпечення, оскільки він дозволяє не лише глибше зрозуміти ринкові очікування та потреби цільової аудиторії, а й виявити сильні та слабкі сторони конкурентних продуктів.

Нижче представленная таблиця аналізу доволі популярних стратегічних ігор які допоможуть визначитись з головними критерями гри які очикують побачити на ринку гравці.

Таблиця 1.3 Аналіз існуючих ігор

Назва гри	**Рік**	**Движок**	**Ключові особливості**	**Графіка**	**Механіки**
Celeste	2018	XNA/MonoGame	Складний платформінг, сюжет	Піксель–арт	Dash, climbing
Hollow Knight	2017	Unity	Метроїдванія, атмосфера, дослідження	2D–анімація	Бої, апгрейди
Ori and the Blind Forest	2015	Unity	Візуальна казка, емоційний сюжет	2.5D, мальована	Платформінг, головоломки
Rayman Legends	2013	UbiArt	Кооператив, гумор, анімація	2D–анімація	Платформінг, секрети

Celeste (2018)

- Рушій: Microsoft XNA, MonoGame
- Опис: Гравець керує Медлін, яка намагається піднятися на гору Celeste, долаючи складні перешкоди. Гра поєднує точний платформінг з емоційним сюжетом про боротьбу з тривогою і депресією. У грі є механіка dash — ривок

у повітрі в 8 напрямках, а також лазання по стінах. Рівні дуже складні, з багатьма секретами і альтернативними шляхами (B–Sides, C–Sides).

- Візуальний стиль: Піксель–арт у стилі 8–біт, мінімалістичний, але виразний.
- Фото: Скриншоти показують яскраві піксельні рівні з платформами.

Рисунок 1.5 – Інтерфейс Celeste 2018

Hollow Knight (2017)

- Рушій: Unity
- Опис: Метроїдванія з похмурою атмосферою, глибоким сюжетом і великим досліджуваним світом. Гравець бореться з ворогами, покращує здібності і відкриває нові області.
- Візуальний стиль: Ручна 2D–анімація з деталізованими персонажами і фонами, стилізована під похмуре фентезі.
- Фото: Скриншоти показують темні, атмосферні локації з плавною анімацією персонажів і ефектами бою.

Рисунок 1.6 – Інтерфейс Hollow Knight (2017)

Ori and the Blind Forest (2015)

- Рушій: Unity
- Опис: Емоційна історія з елементами платформінгу і головоломок. Гравець досліджує яскравий і мальовничий світ, розв’язуючи завдання і долаючи перешкоди.
- Візуальний стиль: Мальована 2.5D графіка з яскравими кольорами і плавною анімацією.
- Фото: Скриншоти демонструють яскраві, мальовничі пейзажі.

Рисунок 1.7 – Інтерфейс Ori and the Blind Forest (2015)

Rayman Legends (2013)

- Рушій: UbiArt Framework
- Опис: Кооперативний платформер з гумором, багатьма секретами і чудовою анімацією. Гра орієнтована на веселу гру з друзями.
- Візуальний стиль: Яскрава 2D–анімація в мультяшному стилі з деталізованим промальовуванням персонажів і оточення.
- Фото: Скриншоти показують яскраві рівні з багатьма деталями і динамічними анімаціями персонажів.

Рисунок 1.8 – Інтерфейс Rayman Legends (2013)

ВИСНОВОК

Ці ігри демонструють різні підходи до створення 2D платформерів, що поєднують якісний геймплей, продуманий сюжет, а також унікальний візуальний стиль.

Геймплей цих ігор базується на точному контролі персонажа, цікавих механіках руху (наприклад, dash у Celeste), дослідженні світу і розвитку героя (Hollow Knight), поєднанні платформінгу з головоломками (Ori) та веселому кооперативному проходженні (Rayman Legends). Важливо ретельно

продумувати механіки, баланс складності і взаємодію гравця з ігровим світом, щоб забезпечити захоплюючий ігровий досвід.

Планування розробки має включати аналіз цільової аудиторії, створення діаграм варіантів використання, роботу з tilemap, анімацією та камерою. Візуальний стиль повинен бути цілісним і відповідати тематиці гри — від піксель–арту до мальованої 2.5D графіки або мультяшної анімації.

Вивчення успішних прикладів допомагає розробникам запозичувати кращі практики, уникати помилок і створювати якісну продукцію, яка буде цікава широкій аудиторії. Загалом, ці ігри демонструють, що навіть невеликі студії можуть створювати глибокі, емоційні та технічно досконалі проекти.

Загалом висновок такий: для розробника важливо поєднувати правильниі технологічні вибори, продуманий дизайн, унікальний візуальний стиль і сюжет, що зацікавить гравця, щоб створити гру, яка буде затребувана на і так вже перегруженному ринку. Аналіз таких ігор, як Celeste, Hollow Knight, Ori and the Blind Forest та Rayman Legends, є відмінною основою для розуміння сучасних трендів і стандартів у розробці 2D платформерів.

РОЗДІЛ ІІ. ПРОЕКТУВАННЯ СТРАТЕГІЧНОЇ КОМП'ЮТЕРНОЇ ГРИ НА C#

2.1 Розробка технічної та проектної документації

На етапі підготовки до проекту розробляється базовий набір документації, який слугує основою для всієї подальшої роботи над стратегічною грою. Основними документами на цьому етапі є концептуальний опис проєкту (концепт–документ) та дизайн–документ. Вони визначають напрям розробки, геймплейну структуру, цільову аудиторію та технічні особливості гри.

Концепт–документ

Концепт–документ є коротким викладом ідеї гри, який дозволяє зрозуміти, чим вона приваблюватиме потенційного гравця, як виглядатиме після реалізації та чим буде унікальною. Це своєрідна «візитівка» проєкту, призначена як для внутрішнього користування в команді, так і для зовнішніх презентацій. У типовому концепт–документі відображаються такі ключові компоненти як:

- Тип гри та цільовий користувач — визначається жанр (наприклад, реальна стратегія, покрокова стратегія), платформа, на яку орієнтується проєкт, та основна демографія гравців (вік, ігровий досвід, інтереси).
- Ключові відмінності продукту (унікальні геймплейні особливості) — короткий перелік тих елементів, які роблять гру привабливою і вирізняють її серед інших. Це можуть бути нові ідеї в ігровому процесі, незвичайна механіка, стиль графіки, атмосфера або інноваційні підходи до розвитку сюжету. Завдяки цим елементам гра запам'ятовується і має більше шансів зацікавити користувачів.
- Опис ігрового процесу — узагальнений огляд основної механіки гри: які дії виконує гравець, у якому порядку, з якою метою. Також важливо зазначити, чому ці дії є цікавими і чому вони повторюються.

- Оцінка технічних вимог — попереднє визначення мінімальних і рекомендованих параметрів комп’ютера для коректної роботи гри. Це допоможе зорієнтувати розробку під цільову платформу.

Дизайн–документ

Дизайн–документ є розширеною технічною специфікацією гри. Він містить повне організоване і уявлення про внутрішню логіку системи, способи реалізації ігрових механік, інтерфейс, правила взаємодії об'єктів тощо. У практиці розробки прийнято виокремлювати такі частини дизайн–документа:

- Загальна концепція — включає в себе елементи концепт–документа, доповнені інформацією про технічну реалізацію ігрової ідеї.
- Функціональна структура гри — докладний опис процесів, з якими взаємодіє користувач. Вказуються можливості гравця, інтерфейсні елементи, сценарії бою, економічного розвитку, дипломатії, перемоги тощо.
- Технічна модель реалізації — визначає засоби, алгоритми, зразки проектування і технологічні рішення, що будуть застосовані. Наприклад: вибір двигуна, використання системи ECS, підхід до розробки ШІ або збереження поступу.
- План реалізації гри — попереднє розбиття проєкту на етапи (pre–production, prototype, alpha, beta, реліз), з орієнтовним графіком розробки, тестування та ітерацій покращення. За потреби також додається розподіл завдань у команді.

2.2 Побудова діаграми варіантів використання (Use Case Diagram)

Під час аналізу функціональних вимог до гри була створена UML–діаграму варіантів використання, яка показує основні сценарії взаємодії користувача з ігрою.

У грі одночасно може брати участь лише один гравець, який керує об’єктом за допомогою клавіатури. Основне керування транспортним засобом здійснюється клавішами A (рух ліворуч) та D (рух праворуч). Крім того,

натискання клавіші ESC дозволяє відкрити головне меню гри, де користувач може обирати різні параметри та функції.

Основні функціональні можливості головного меню:

- Налаштування гри – зміна графічних параметрів, конфігурація дисплею, регулювання гучності або повне вимкнення звукових ефектів.
- Перегляд досягнень – відображення найкращих результатів гравця, зафіксованих під час заїздів. Також є можливість перегляду світових рекордів та порівняння їх зі своїми.
- Початок гри – запуск першого рівня з попередньо збереженими налаштуваннями.

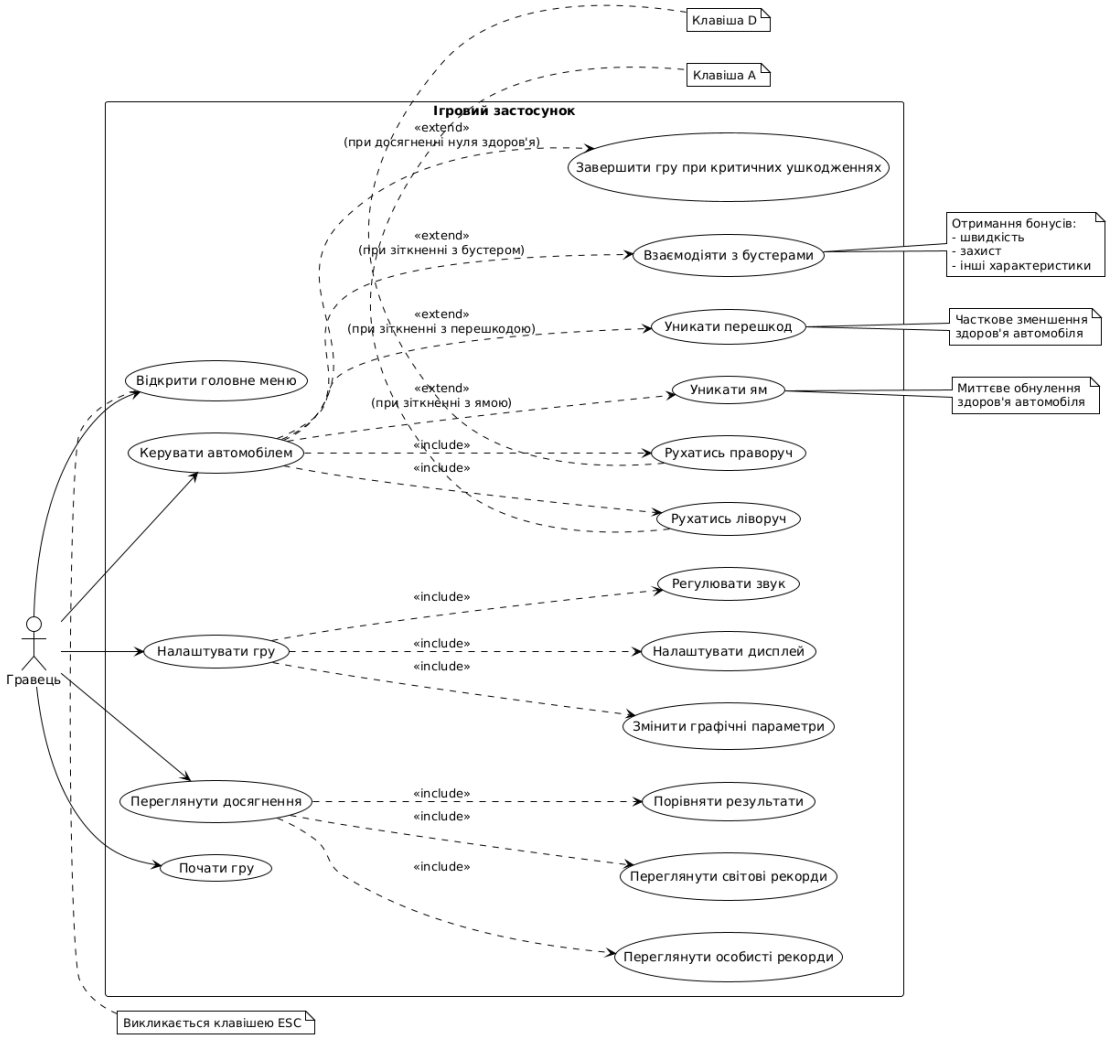

Рисунок 2.1 – Діаграма варіантів використання

Під час ігрового процесу гравець переміщує автомобіль вліво або вправо, перебуваючи на старті в центрі екрана. Завдяки руху бокових елементів карти створюється ефект поступального руху вперед. На шляху гравця трапляються різноманітні об’єкти, з якими він може взаємодіяти:

- Бустери – при зіткненні зі спеціальними спрайтами гравець отримує бонуси, які можуть впливати на швидкість, захист або інші характеристики.
- Перешкоди – при зіткненні з ними запас здоров’я автомобіля зменшується частково.
- Ями – контакт з ямою миттєво призводить до обнулення здоров’я транспортного засобу.

Гра завершується, якщо рівень здоров’я автомобіля досягає нуля, тобто у випадку критичних ушкоджень. Така механіка стимулює гравця до уважності, маневрування та використання бонусів у стратегічний спосіб.

2.3 Проєктування діаграми класів ігрового застосунку (Game Logic Class Diagram)

Діаграма класів ігрової логіки є центральним елементом архітектурного проектування стратегічної гри, оскільки вона визначає структуру взаємодії між основними компонентами ігрового процесу. При розробці цієї діаграми було враховано принципи об'єктно–орієнтованого програмування.

Основні класи ігрової логіки

Клас Player Клас Player є основним класом, що представляє гравця в ігровій системі. Він містить такі ключові атрибути:

- string Name – ім'я гравця для ідентифікації
- int Score – поточний рахунок гравця
- int Health – рівень здоров'я персонажа
- Position CurrentPosition – поточна позиція на ігровому полі

Методи класу включають:

- void Move(Direction direction) – переміщення гравця в заданому напрямку

- void TakeDamage(int damage) – зменшення здоров'я при отриманні пошкоджень
- bool IsAlive() – перевірка стану гравця

Клас GameObject Це абстрактний базовий клас для всіх ігрових об'єктів, що забезпечує нормалізований підхід до роботи з різними елементами гри:

- Position position – координати об'єкта в ігровому просторі
- Sprite sprite – графічне представлення об'єкта
- bool isActive – статус активності об'єкта

Абстрактні методи:

- abstract void Update() – оновлення стану об'єкта
- abstract void Render() – відображення об'єкта на екрані
- abstract void OnCollision(GameObject other) – обробка зіткнень

Клас Enemy Наслідує від GameObject та представляє ворожі об'єкти:

- int damage – шкода, яку завдає ворог
- MovementPattern pattern – шаблон руху ворога
- int hitPoints – кількість очок здоров'я

Специфічні методи:

- void AttackPlayer() – атака гравця
- void FollowMovementPattern() – слідування шаблону руху
- void DestroyOnCollision() – знищення при зіткненні

Клас Bonus Також наслідує від GameObject, представляє бонусні об'єкти:

- BonusType type – тип бонусу (здоров'я, очки, швидкість)
- int value – значення бонусу
- float duration – тривалість дії (для тимчасових бонусів)

Взаємодія між класами

Архітектура побудована таким чином, що клас Player взаємодіє з іншими об'єктами через систему зіткнень. GameManager координує всі взаємодії та керує станом гри. Використання абстрактного класу GameObject дозволяє легко додавати нові типи ігрових об'єктів без зміни основної логіки.

Клас CollisionManager відповідає за виявлення та обробку зіткнень між об'єктами, використовуючи алгоритми прямокутних або круглих зон зіткнення в залежності від типу об'єкта.

Переваги архітектури

Запропонована структура класів забезпечує:

- Модульність – кожен клас має чітко визначену відповідальність
- Розширюваність – легке додавання нових типів об'єктів
- Повторне використання коду – базові функціональності успадковуються
- Тестованість – окремі компоненти можна тестувати незалежно

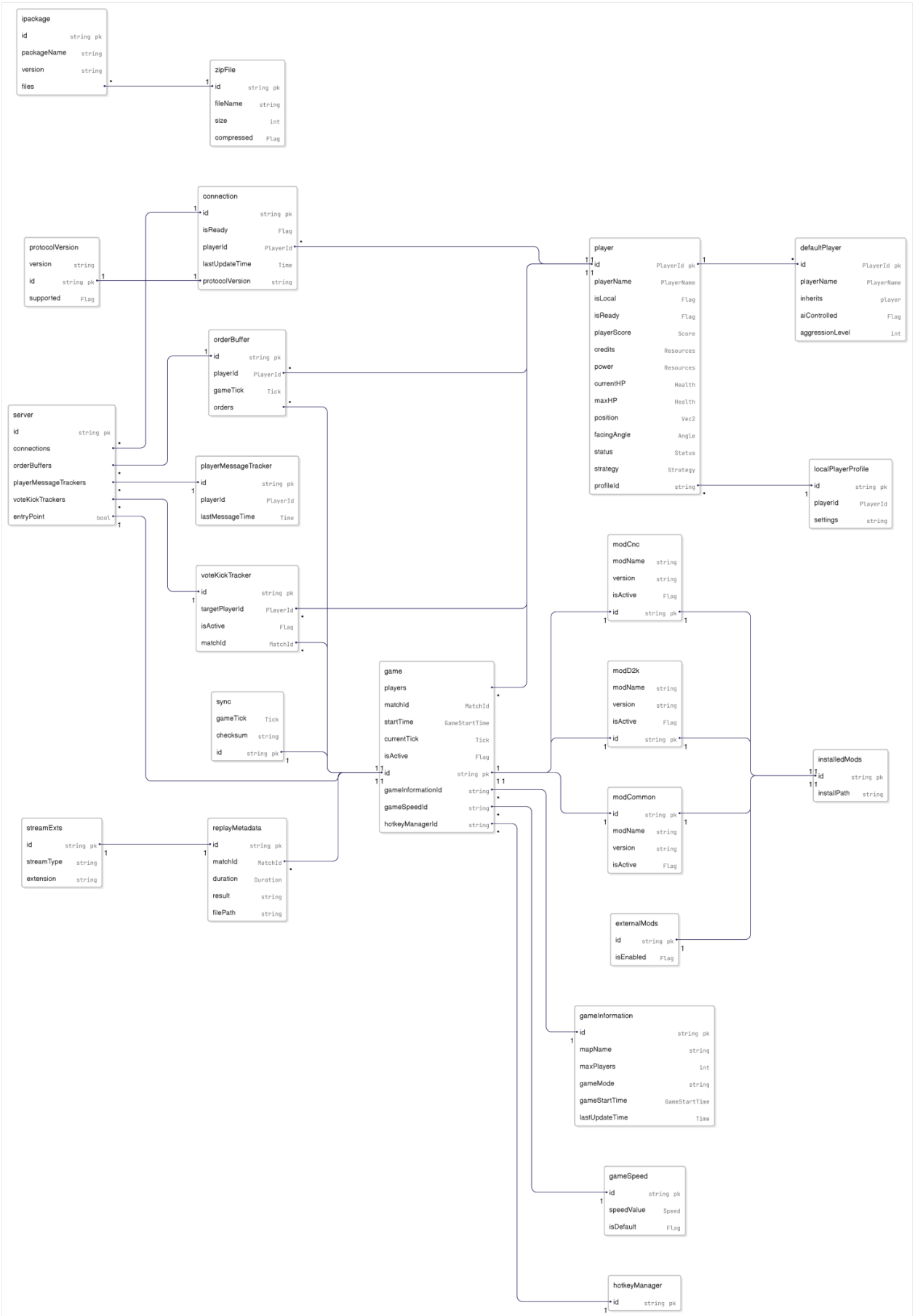

Рисунок 2.2 – Діаграма класів ігрового застосунку

2.4 Побудова загальної діаграми класів (System–Level Class Diagram)

Загальна діаграма класів представляє високорівневу архітектуру всієї системи, включаючи не лише ігрову логіку, а й компоненти користувацького інтерфейсу, управління ресурсами, звуковою системою та збереженням даних. Ця діаграма є критично важливою для розуміння взаємодії між різними підсистемами застосунку.

Архітектурні шари системи

Шар презентації (Presentation Layer) Верхній шар архітектури відповідає за взаємодію з користувачем:

- Клас MainWindow – головне вікно застосунку, що координує роботу всіх UI–компонентів
    - void InitializeComponents() – ініціалізація елементів інтерфейсу
    - void ShowMainMenu() – відображення головного меню
    - void StartGame() – запуск ігрового процесу
- Клас GameScreen – екран ігрового процесу
    - void RenderGame() – відображення ігрового стану
    - void HandleInput() – обробка користувацького вводу
    - void UpdateUI() – оновлення елементів інтерфейсу
- Клас MenuSystem – система меню
    - void ShowSettingsMenu() – налаштування гри
    - void ShowAchievements() – відображення досягнень
    - void HandleMenuNavigation() – навігація по меню

Шар бізнес–логіки (Business Logic Layer) Центральний шар, що містить основну логіку гри:

- Клас GameEngine – основний ігровий движок
    - void InitializeGame() – ініціалізація ігрової сесії
    - void UpdateGameState() – оновлення стану гри
    - void ProcessGameLogic() – обробка ігрової логіки
    - bool CheckWinCondition() – перевірка умов перемоги

- Клас ScoreManager – управління рахунком та досягненнями
  - void AddScore(int points) – додавання очок
  - void SaveHighScore() – збереження рекорду
  - List<Achievement> GetAchievements() – отримання досягнень
- Клас InputHandler – обробка користувацького вводу
  - void RegisterKeyBindings() – реєстрація клавіатурних комбінацій
  - void ProcessInput(InputEvent event) – обробка подій вводу
  - bool IsKeyPressed(Key key) – перевірка натискання клавіші

Шар даних (Data Layer) Нижній шар, що відповідає за роботу з даними:

- Клас FileManager – управління файлами
  - void SaveGameState(GameState state) – збереження стану гри
  - GameState LoadGameState() – завантаження стану гри
  - void SaveSettings(Settings settings) – збереження налаштувань
- Клас ConfigurationManager – управління конфігурацією
  - Settings LoadSettings() – завантаження налаштувань
  - void ApplySettings(Settings settings) – застосування налаштувань
  - void ResetToDefaults() – скидання до значень за замовчуванням

Паттерни проектування в архітектурі

Singleton Pattern Використовується для класів GameEngine та AudioManager, щоб гарантувати існування лише одного екземпляра цих критично важливих компонентів.

Observer Pattern Реалізується між GameEngine та UI–компонентами для автоматичного оновлення інтерфейсу при зміні ігрового стану.

Factory Pattern Клас ObjectFactory використовується для створення ігрових об'єктів різних типів, забезпечуючи централізоване управління їх створенням.

MVC Pattern Загальна архітектура слідує принципам Model–View–Controller, де ігрова логіка (Model) відокремлена від представлення (View) через контролери.

Міжшарова взаємодія

Взаємодія між шарами здійснюється через чітко визначені інтерфейси, що забезпечує слабке зв'язування компонентів. Шар презентації взаємодіє з бізнес–логікою через інтерфейси IGameController та IMenuController, а бізнес–логіка з шаром даних через IDataProvider та IConfigurationProvider.

Така архітектура забезпечує високу масштабованість системи та можливість незалежного розвитку окремих компонентів без впливу на інші частини системи.

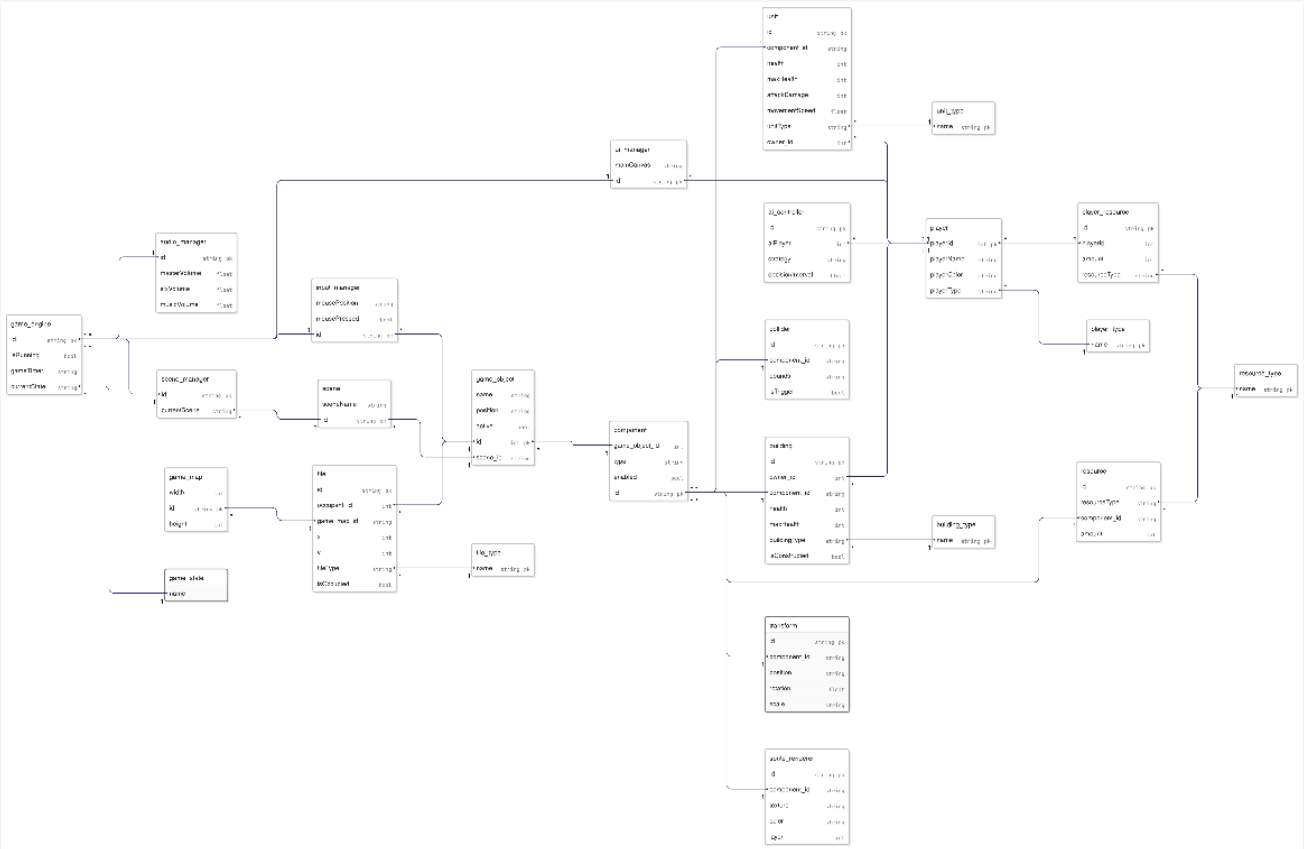

Рисунок 2.3 – Діаграма класів

2.5 Розробка прототипу інтерфейсу користувача (UI/UX–моделювання)

Проектування користувацького інтерфейсу для стратегічної гри вимагає особливої уваги до ергономіки, інтуїтивності та ефективності взаємодії. Прототип UI/UX був розроблений з урахуванням специфіки жанру та потреб

цільової аудиторії, з акцентом на зручність управління та швидкість доступу до ключових функцій.

Принципи дизайну інтерфейсу

Мінімалізм та функціональність Інтерфейс побудований за принципом “менше – це більше”, де кожен елемент має чітке призначення та логічне розташування. Відсутність зайвих декоративних елементів дозволяє гравцю сфокусуватися на ігровому процесі.

Консистентність Всі елементи інтерфейсу використовують єдину стилістику, типографіку та колірну палітру. Кнопки, іконки та текстові поля мають однаковий стиль оформлення в усіх частинах застосунку.

Доступність Інтерфейс розрахований на користувачів з різним рівнем досвіду. Важливі елементи мають достатній розмір для комфортного використання, а контрастність кольорів забезпечує читабельність тексту.

Структура головного меню

Компоновка елементів Головне меню розташоване в центрі екрану та містить чотири основні розділи:

- «Нова гра» – запуск нової ігрової сесії
- «Налаштування» – доступ до параметрів гри
- «Досягнення» – перегляд прогресу та рекордів
- «Вихід» – завершення роботи застосунку

Візуальна ієрархія Кнопки меню розташовані вертикально з рівномірними інтервалами. Найважливіша кнопка “Нова гра” виділена більшим розміром та яскравішим кольором. Використовується м'який градієнт фону для створення глибини без відволікання уваги.

Інтерактивність Всі інтерактивні елементи мають візуальний відгук на наведення курсора миші – плавна зміна кольору та легкий ефект тіні. При натисканні кнопки демонструють анімацію “натискання” для підтвердження дії.

Дизайн ігрового інтерфейсу

Розподіл екранного простору Ігровий екран організований за принципом максимізації ігрової області:

- 80% екрану відведено безпосередньо під ігрове поле
- 15% займає панель інформації (рахунок, здоров'я, час)
- 5% резервується для службових елементів (пауза, налаштування)

Панель стану гравця Розташована в верхній частині екрану та містить:

- Індикатор здоров'я у вигляді горизонтальної смужки з плавним кольоровим переходом (зелений–жовтий–червоний)
- Лічильник очок з великим, чітко читабельним шрифтом
- Таймер гри у форматі MM:SS
- Іконка паузи в правому верхньому куті

Система повідомлень Для інформування гравця про важливі події (збирання бонусів, отримання пошкоджень, досягнення рекордів) використовується система спливаючих повідомлень. Вони з'являються в центрі екрану з напівпрозорим фоном та автоматично зникають через 2–3 секунди.

UX–сценарії взаємодії

Сценарій запуску гри

1. Користувач бачить головне меню одразу після запуску
2. Натискання на “Нова гра” призводить до плавного переходу на ігровий екран
3. Коротка анімація завантаження (1–2 секунди) готує гравця до початку
4. Гра починається з центральної позиції автомобіля

Сценарій паузи та налаштувань

1. Натискання ESC відкриває меню паузи поверх ігрового екрану
2. Ігровий процес зупиняється, фон затемнюється
3. Доступні опції: продовжити, налаштування, головне меню
4. Повернення в гру відбувається миттєво без додаткових завантажень

Сценарій завершення гри

1. При досягненні нульового здоров'я з'являється екран результатів

2. Відображається отриманий рахунок та порівняння з рекордом
3. Анімація святкування при встановленні нового рекорду
4. Швидкий доступ до повторного запуску або повернення в меню

Адаптивність та масштабованість

Підтримка різних роздільностей Інтерфейс адаптується до роздільностей від 1024x768 до 4K, використовуючи відносні розміри та позиціонування елементів. Текст масштабується пропорційно для збереження читабельності.

Оптимізація для швидкої взаємодії Всі критичні дії (пауза, перезапуск, вихід) доступні через гарячі клавіші. Час відгуку інтерфейсу не перевищує 100 мілісекунд для забезпечення комфортного ігрового досвіду.

Можливості розширення Архітектура UI передбачає легке додавання нових елементів – додаткових панелей інформації, нових типів повідомлень або розширених налаштувань без перепроектування основної структури.

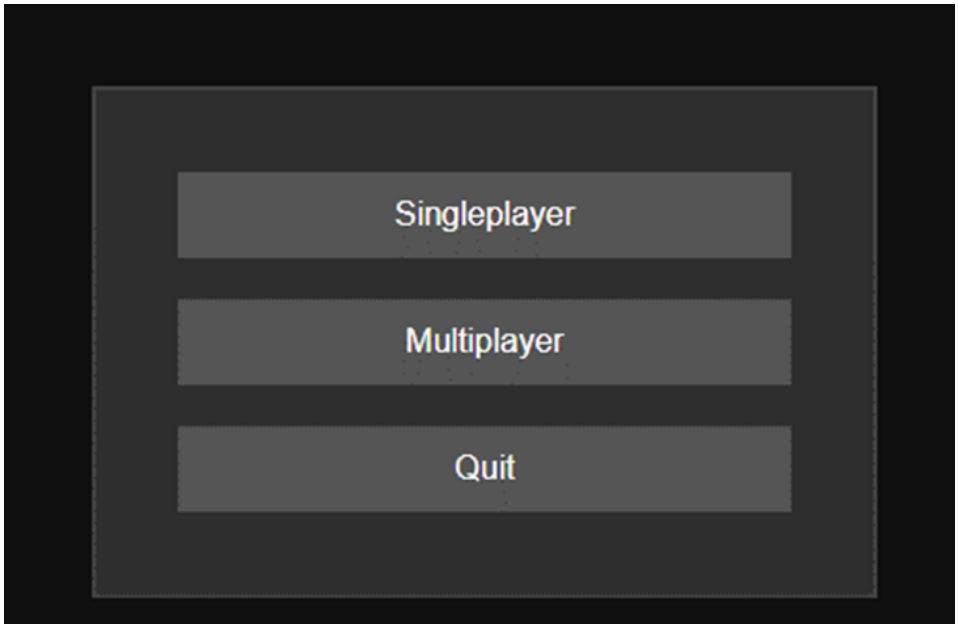

Рисунок 2.1 - Головне меню гри

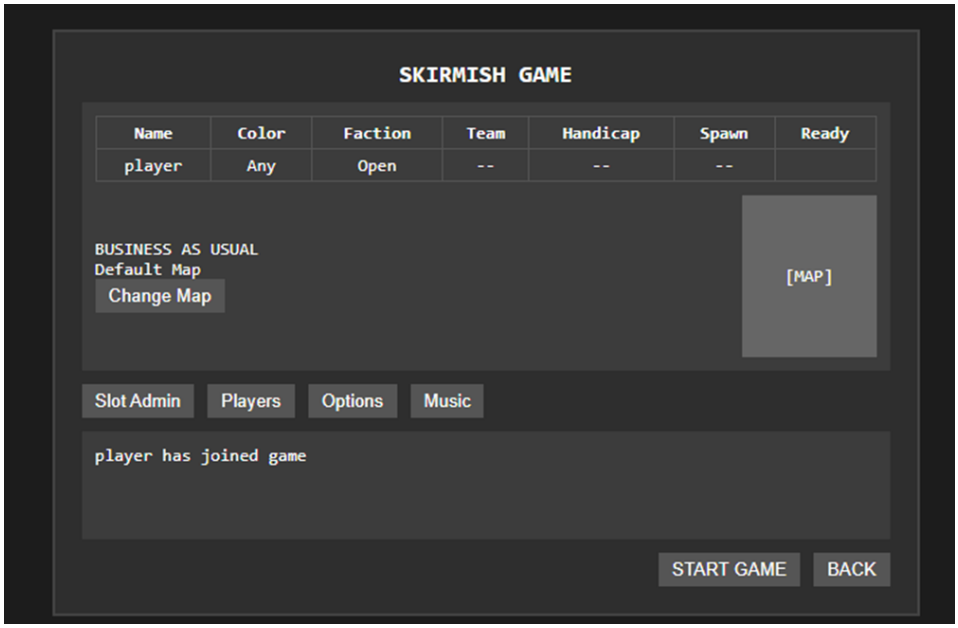

Рисунок 2.2 – Меню одиночної сесії гри

РОЗДІЛ ІІІ. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СТРАТЕГІЧНОЇ ГРИ

3.1 Обґрунтування вибору технології: розробка гри на C# без ігрового рушія Unity

При проєктуванні стратегічної гри особливу увагу було приділено вибору відповідного інструментарію. Не дивлячись на те, що Unity є одним із найпопулярніших рушіїв для розробки 2D та 3D–ігор, у контексті поточного проєкту було прийнято рішення відмовитися від використання рушія та реалізувати гру безпосередньо на C#, з використанням базових засобів графічної обробки (наприклад, Windows Forms, WPF або MonoGame).

Таке рішення обгрунтовується кількома ключевими аспектами:

По–перше це простота механіки гри.

Розробка цієї стратегісної гри не потребує складної фізики, 3D–графіки, розгалужених анімацій чи інтеграції сторонніх інструментів. Основний ігровий процес базується на логіці керування об’єктами, обробці подій, таймерів, та простому UI — усе це ефективно реалізується у межах середовища C#.

По–друге, контроль над архітектурою гри

При розробці без рушія програміст має повний контроль над усіма аспектами проєкту: від процесу візуалізації до керування ресурсами. Це дозволяє уникнути надлишкової абстракції, яку нав'язують ігрові рушії, а також забезпечує гнучкість в оптимізації.

3. Швидкість навчання і впровадження

Розробка на «чистому» C# є більш прозорою для студентського або навчального проєкту. Немає потреби витрачати час на освоєння специфіки Unity Editor, його організації, налаштування шаблонів об’єктів, керування сценами тощо. Це дозволяє сфокусуватися саме на логіці гри, а не на інструментах.

4. Мінімальні системні вимоги та розмір проєкту

Використання C# без рушія дозволяє значно зменшити фінальний розмір гри, знизити системні вимоги, а також уникнути встановлення додаткових бібліотек. Проєкт буде зручнішим у розповсюдженні та тестування, особливо якщо він призначений для desktop–платформи.

5. Навчальний інтерес

З точки зору розробника–початківця, ручна реалізація базових механік — це чудова можливість засвоїти принципи побудови ігрових циклів, роботи з графікою, таймерами, подіями та шаблонами проєктування (наприклад, патерн «Observer» або «Game Loop») без надмірної залежності від готових рішень.

Таблиця 3.1 – Порівняння рушія і «чистого» C#

Критерій	Unity + C#	C# без рушія (WinForms/WPF/MonoGame)
Порог входу	Вищий (необхідне знання Editor, систем Unity)	Нижчий (достатньо базових знань C#)
Керування ресурсами	Частково абстраговане, автоматизоване	Повністю в ручному режимі (гнучкіше, але складніше)
Розмір фінальної збірки	~100–200 МБ навіть для простої гри	5–30 МБ залежно від графічної бібліотеки
Продуктивність у 2D	Добра, але з невеликим оверхедом	Вища (немає фонових систем Unity)
Наявність фізики / 3D	Вбудовані компоненти	Потребує реалізації вручну

Налагодження логіки	Через інспектори та скрипти	Повністю через код і дебаг–інструменти C#
Гнучкість налаштувань UI	Висока, але з вивченням UI Toolkit/uGUI	Простий UI (Forms/WPF) або кастомний
Придатність для навчального проєкту	Іноді надмірно складний	Ідеальний для засвоєння фундаментальних концепцій

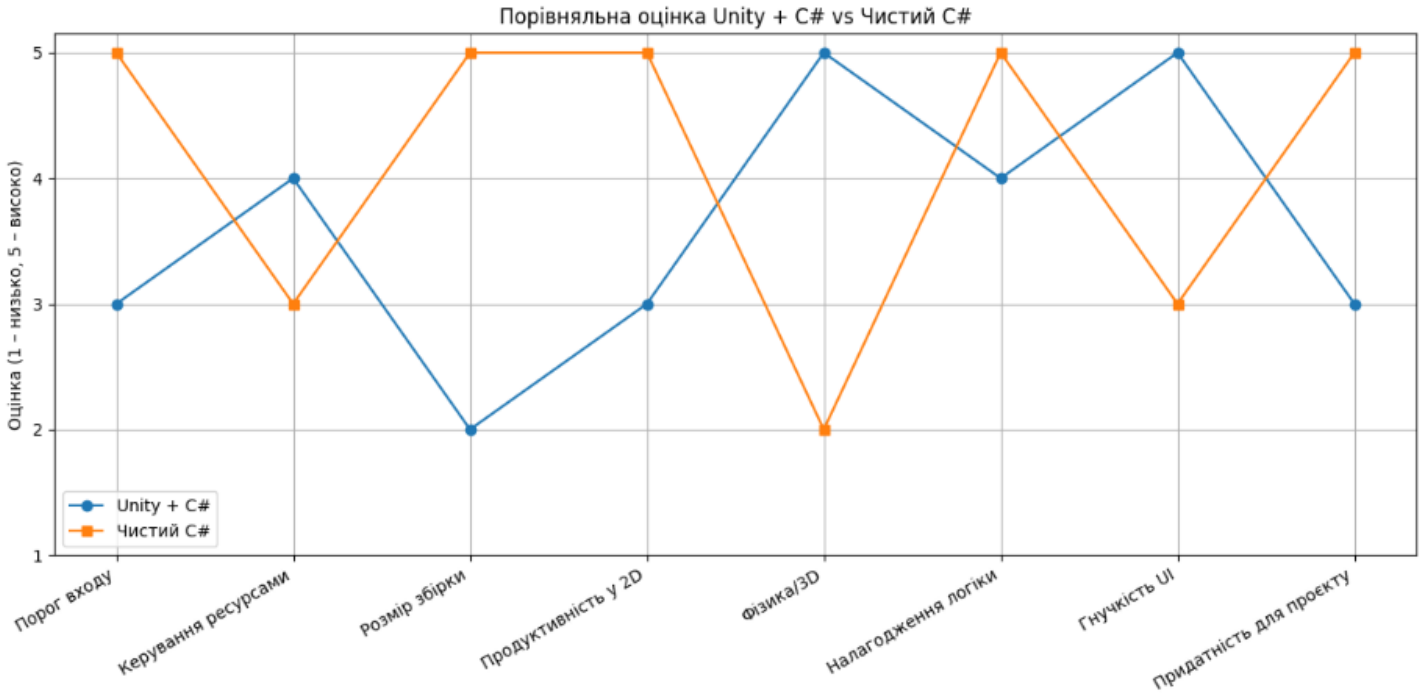

Рисунок 2.1 – Графік Unity + C# vs Чистий C# (без рушія)

Висновок

Для простої стратегічної гри з базовим UI, відсутністю складної 3D–графіки та необхідності прямого контролю над логікою, використання C# без рушія є не лише ефективним, а й продуктивно вигідним вибором. Це забезпечує

простішу архітектуру, глибше занурення в програмування, а також менші системні вимоги гри.

3.2 Опис структури програмного забезпечення

Програмне забезпечення побудоване за принципами об'єктно–орієнтованого програмування з використанням архітектури компонентно–сутнісної системи (Entity–Component–System). Структура проекту організована в декілька ключових модулів, що забезпечують різні аспекти функціонування гри.

Архітектурний огляд

Система організована навколо центральних концепцій:

Світ (World) – контейнер для всіх ігрових об'єктів та логіки

Гравці (Players) – представлення учасників гри

Актори (Actors) – ігрові сутності з набором компонентів

Інтерфейс користувача – система віджетів для взаємодії

Клас Player – основа системи гравців

Центральним елементом архітектури є клас Player, який інкапсулює всю інформацію про гравця в грі:

```
public class Player : IScriptBindable, IScriptNotifyBind, ILuaTableBinding,
ILuaEqualityBinding, ILuaToStringBinding
{
    public readonly Actor PlayerActor;
    public readonly string PlayerName;
    public readonly string InternalName;
    public readonly FactionInfo Faction;
    public readonly bool NonCombatant = false;
    public readonly bool Playable = true;
    public readonly int ClientIndex;
    public readonly CPos HomeLocation;
    public readonly int Handicap;
    public readonly PlayerReference PlayerReference;
    public readonly bool IsBot;
    public readonly string BotType;
    public readonly Shroud Shroud;
    public readonly FrozenActorLayer FrozenActorLayer;

    public Color Color { get; private set; }
    public WinState WinState = WinState.Undefined;
    public World World { get; }
```

```
 }
```

Ключові характеристики класу Player:

Ідентифікація гравця:

PlayerName – відображуване ім'я гравця

InternalName – внутрішній ідентифікатор

ClientIndex – індекс клієнта в мережевій грі

PlayerMask – унікальна бітова маска для швидкого визначення союзників/ворогів

Ігрові властивості:

Faction – фракція гравця з розв'язанням випадкових фракцій

HomeLocation – стартова позиція на карті

Handicap – модифікатор складності

WinState – поточний стан перемоги/поразки

Системи видимості:

Shroud – туман війни для гравця

FrozenActorLayer – шар “заморожених” акторів для оптимізації

Конструктор та ініціалізація

Конструктор класу Player демонструє складний процес ініціалізації:

Реалізація графічного інтерфейсу користувача

```
 public Player(World world, Session.Client client, PlayerReference pr,
MersenneTwister playerRandom)
 {
    World = world;
    InternalName = pr.Name;
    PlayerReference = pr;

    // Визначення типу карти
    inMissionMap = world.Map.Visibility.HasFlag(MapVisibility.MissionSelector);
    botInfos =
World.Map.Rules.Actors[SystemActors.Player].TraitInfos<IBotInfo>();

    if (client != null)
    {
       // Ініціалізація для реального гравця або бота
       ClientIndex = client.Index;
       color = client.Color;
       PlayerName = client.Name;
```

```
            BotType = client.Bot;

            // Розв'язання фракції з урахуванням випадковості
            Faction = ResolveFaction(world, client.Faction, playerRandom,
!pr.LockFaction);
            DisplayFaction = ResolveDisplayFaction(world, client.Faction);

            // Призначення точки появи
            var assignSpawnPoints =
world.WorldActor.TraitOrDefault<IAssignSpawnPoints>();
            HomeLocation = assignSpawnPoints?.AssignHomeLocation(world, client,
playerRandom) ?? pr.HomeLocation;
        }
        else
        {
            // Ініціалізація для карточного гравця
            ClientIndex = world.LobbyInfo.Clients.FirstOrDefault(c => c.IsAdmin)?.Index
?? 0;
            // ... решта ініціалізації
        }
    }
```

Система відносин між гравцями

Клас реалізує складну систему визначення відносин:

```
    {
        if (this == other)
            return PlayerRelationship.Ally;

        if (other == null || other.Spectating)
            return NonCombatant ? PlayerRelationship.Neutral : PlayerRelationship.Ally;

        if (AlliedPlayersMask.Overlaps(other.PlayerMask))
            return PlayerRelationship.Ally;

        if (EnemyPlayersMask.Overlaps(other.PlayerMask))
            return PlayerRelationship.Enemy;

        return PlayerRelationship.Neutral;
    }
```

Ця система використовує бітові маски для швидкого визначення союзників та ворогів, що критично важливо для продуктивності в реальному часі.

Система інтерфейсу користувача

Інтерфейс користувача організований через систему YAML–файлів, що описують структуру віджетів. Приклад головного меню:

```
Container@MAINMENU:
    Logic: CustomMainMenuLogic
    Children:
```

```
Container@MENUS:
   X: (WINDOW_RIGHT - WIDTH) / 2
   Y: WINDOW_BOTTOM / 2 - HEIGHT / 2
   Width: 200
   Height: 160
   Children:
      Background@MAIN_MENU:
         Children:
            Button@SINGLEPLAYER_BUTTON:
               X: PARENT_RIGHT / 2 - WIDTH / 2
               Y: 20
               Width: 140
               Height: 30
               Text: label-singleplayer-title
               Font: Bold
            Button@MULTIPLAYER_BUTTON:
               X: PARENT_RIGHT / 2 - WIDTH / 2
               Y: 60
               Width: 140
               Height: 30
               Text: label-multiplayer-title
               Font: Bold
```

Особливості UI архітектури:

Контейнерна структура:

Кожен елемент UI є контейнером або віджетом

Підтримка вкладеності через систему Children

Автоматичне позиціонування відносно батьківських елементів

Система координат:

Використання відносних координат (PARENT_RIGHT, WINDOW_BOTTOM)

Підтримка математичних виразів для позиціонування

Адаптивність до різних роздільностей екрану

Логіка обробки:

Прив'язка логіки через атрибут Logic

Система обробки клавіатурних комбінацій

Підтримка локалізації через текстові ключі

Інтеграція зі скриптовою системою

Клас Player інтегрований з Lua–скриптовою системою через реалізацію відповідних інтерфейсів:

```
#region Scripting interface
Lazy<ScriptPlayerInterface> luaInterface;

public void OnScriptBind(ScriptContext context)
{
   luaInterface ??= Exts.Lazy(() => new ScriptPlayerInterface(context, this));
}

public LuaValue this[LuaRuntime runtime, LuaValue keyValue]
{
   get => luaInterface.Value[runtime, keyValue];
   set => luaInterface.Value[runtime, keyValue] = value;
}
```

Це дозволяє сценаріям безпосередньо взаємодіяти з об'єктами гравців, що критично важливо для створення місій та модифікацій.

Оптимізації та продуктивність

Архітектура включає декілька оптимізацій:

Бітові маски для швидкого визначення відносин між гравцями

Ледача ініціалізація (Lazy<T>) для скриптових інтерфейсів

Кешування розв'язаних імен гравців

Заморожені актори для економії пам'яті в туману війни

Модуль штучного інтелекту

Система штучного інтелекту організована через набір утилітарних класів та трейтів, що забезпечують різні аспекти поведінки ботів.

Клас AIUtils – утилітарні функції ШІ

```
public static class AIUtils
{
   public static bool IsAreaAvailable<T>(World world, Player player, Map map, int
radius, HashSet<string> terrainTypes)
   {
      var cells = world.ActorsHavingTrait<T>().Where(a => a.Owner == player);

      return cells.Select(a => map.FindTilesInCircle(a.Location, radius)
         .Count(c => map.Contains(c) &&
terrainTypes.Contains(map.GetTerrainInfo(c).Type) &&
            Util.AdjacentCells(world, Target.FromCell(world, c))
               .All(ac => map.Contains(ac) &&
terrainTypes.Contains(map.GetTerrainInfo(ac).Type))))
                  .Any(availableCells => availableCells > 0);
   }
```

```
        public static ILookup<string, ProductionQueue> FindQueuesByCategory(Player
player)
        {
            return player.World.ActorsWithTrait<ProductionQueue>()
                .Where(a => a.Actor.Owner == player && a.Trait.Enabled)
                .Select(a => a.Trait)
                .ToLookup(pq => pq.Info.Type);
        }
    }
```

Основні функції AIUtils:

Аналіз доступності території – перевірка можливості розміщення будівель

Пошук черг виробництва – категоризація доступних засобів виробництва

Підрахунок акторів – статистичний аналіз ігрових об'єктів

Очищення блокувань – автоматичне переміщення юнітів для звільнення шляхів

Система спостереження за ворогами

Трейт EnemyWatcher реалізує механізм відслідковування та сповіщення про ворожі юніти:

```
    [TraitLocation(SystemActors.Player)]
    sealed class EnemyWatcherInfo : TraitInfo
    {
        [Desc("Interval in ticks between scanning for enemies.")]
        public readonly int ScanInterval = 25;

        [Desc("Minimal ticks in-between notifications.")]
        public readonly int NotificationInterval = 750;
    }

    sealed class EnemyWatcher : ITick
    {
        readonly HashSet<Player> discoveredPlayers;
        HashSet<uint> lastKnownActorIds;
        HashSet<uint> visibleActorIds;

        void ITick.Tick(Actor self)
        {
            if (self.Owner.Shroud.Disabled || self.Owner.IsBot ||
                !self.Owner.Playable || self.Owner.PlayerReference.Spectating)
                return;

            foreach (var actor in self.World.ActorsWithTrait<AnnounceOnSeen>())
```

```
        {
            if (actor.Actor.AppearsFriendlyTo(self))
                continue;

            if (!actor.Actor.CanBeViewedByPlayer(self.Owner))
                continue;

            // Логіка сповіщення про виявлення
        }
    }
}
```

Ключові особливості EnemyWatcher:

Періодичне сканування – перевірка видимих ворогів з заданим інтервалом

Система сповіщень – уникнення спаму через мінімальні інтервали між повідомленнями

Відслідковування стану – збереження інформації про раніше виявлених акторів

Інтеграція з системою видимості – врахування туману війни

Архітектурні принципи ШІ системи

- Модульність: Кожен аспект ШІ реалізований як окремий трейт, що дозволяє гнучко налаштовувати поведінку різних типів ботів.
- Ефективність: Використання кешування та періодичних перевірок замість постійного моніторингу зменшує навантаження на процесор.
- Розширюваність: Статичні утилітарні методи можуть використовуватися різними реалізаціями ШІ без дублювання коду.

Ця архітектура забезпечує баланс між гнучкістю, продуктивністю та простотою підтримки, що є критично важливим для ігрового движка реального часу.

Структура інтерфейсу користувача

Інтерфейс користувача реалізований через систему віджетів, де кожен елемент інтерфейсу представлений окремим класом. Головний клас управління

меню CustomMainMenuLogic наслідується від базового класу ChromeLogic та відповідає за координацію роботи всіх елементів головного меню.

Класифікація меню системи

Система меню організована ієрархічно та включає наступні типи:

```
{
   Main,          // Головне меню
   Singleplayer,  // Меню одиночної гри
   Extras,        // Додаткові можливості
   MapEditor,     // Редактор карт
   StartupPrompts, // Початкові підказки
   None           // Відсутність активного меню
}
```

Панелі управління грою

Кожна панель управління відповідає за конкретний аспект геймплею:

```
protected enum MenuPanel
{
   None,      // Відсутність активної панелі
   Missions,  // Панель місій
   Skirmish,  // Панель швидкої гри
   Multiplayer,// Панель мультиплеєра
   MapEditor,  // Панель редактора карт
   Replays,   // Панель перегляду реплеїв
   GameSaves  // Панель збережених ігор
}
```

Ініціалізація системи

Процес ініціалізації головного меню відбувається через конструктор класу CustomMainMenuLogic, який використовує патерн Dependency Injection для отримання необхідних залежностей:

```
[ObjectCreator.UseCtor]
public CustomMainMenuLogic(Widget widget, World world, ModData modData)
{
   rootMenu = widget;
   rootMenu.Get<LabelWidget>("VERSION_LABEL").Text =
modData.Manifest.Metadata.Version;

   // Ініціалізація елементів інтерфейсу
   InitializeMainMenu(widget);
   InitializeSingleplayerMenu(widget, modData);
   InitializeExtrasMenu(widget, world);
```

```
    InitializeMapEditorMenu(widget, modData);

    // Налаштування системи новин
    InitializeNewsSystem(widget, modData);

    // Налаштування профілю гравця
    InitializePlayerProfile(widget, world);

    // Запуск системи Discord інтеграції
    DiscordService.UpdateStatus(DiscordState.InMenu);
}
```

Управління станом меню

Система використовує централізоване управління станом через метод SwitchMenu(), який забезпечує правильні переходи між різними розділами інтерфейсу:

```
void SwitchMenu(MenuType type)
{
    menuType = type;
    DiscordService.UpdateStatus(DiscordState.InMenu);

    // Оновлення підказок при наведенні миші
    Game.RunAfterTick(Ui.ResetTooltips);
}
```

Інтеграція з зовнішніми сервісами

Система новин

Система новин інтегрована з GitHub API для отримання актуальної інформації про оновлення та події:

```
void LoadAndDisplayNews(GitHubWebServices webServices, Widget newsBG)
{
    if (!Game.Settings.Game.FetchNews)
        return;

    var newsButton =
newsBG.GetOrNull<DropDownButtonWidget>("NEWS_BUTTON");
    if (newsButton != null)
    {
        DisplayNews(webServices);
        newsButton.OnClick = () => OpenNewsPanel(newsButton);

        if (webServices.NewsAlert)
            OpenNewsPanel(newsButton);
    }
}
```

Discord інтеграція

Система підтримує інтеграцію з Discord для відображення поточного статусу гравця:

DiscordService.UpdateStatus(DiscordState.InMenu);

Обробка подій життєвого циклу

Клас реалізує правильну обробку подій створення та знищення об'єктів:

```
protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        Game.OnRemoteDirectConnect -= OnRemoteDirectConnect;
        Game.BeforeGameStart -= RemoveShellmapUI;
    }

    Game.OnShellmapLoaded -= OpenMenuBasedOnLastGame;
    base.Dispose(disposing);
}
```

Система локалізації

Усі текстові рядки в інтерфейсі підтримують локалізацію через систему атрибутів:

[TranslationReference]

const string LoadingNews = “label–loading–news”;

[TranslationReference(“author”, “datetime”)]

const string AuthorDateTime = “label–author–datetime”;

Патерни проектування

У коді використовуються наступні патерни проектування:

Observer Pattern – для обробки подій користувацького інтерфейсу

Strategy Pattern – для різних типів меню та панелей

Factory Pattern – для створення віджетів інтерфейсу

Singleton Pattern – для глобальних сервісів (Discord, Settings)

Така архітектура забезпечує високу гнучкість системи, легкість тестування та можливість швидкого додавання нових функцій без порушення існуючої логіки роботи програми.

Основні складові сервера:

Тип сервера (ServerType) — визначає режим роботи: локальний, мережевий, мультиплеєр або виділений сервер.

Стан сервера (ServerState) — дозволяє відслідковувати стадію ігрової сесії: очікування гравців, гра запущена, завершення роботи.

Список з'єднань (List<Connection> Conns) — активні клієнти, підключені до сервера.

Подієва система (BlockingCollection<IServerEvent> events) — обробка асинхронних подій, таких як підключення/відключення клієнтів, прийом пакетів, пінг–запити.

Ігрова логіка та обробка команд (InterpretCommand, StartGame, ReceiveOrders) — реалізовано через інтерфейси та делегування у відповідні компоненти гри.

Приклад ініціалізації сервера

Нижче наведено фрагмент конструктора класу Server, який демонструє ініціалізацію TCP–сокетів, запуск потоку прослуховування клієнтів і налаштування базових параметрів:

```
public Server(List<IPEndPoint> endpoints, ServerSettings settings, ModData
modData, ServerType type)
{
    Log.AddChannel(“server”, “server.log”, true);
    foreach (var endpoint in endpoints)
    {
        var listener = new TcpListener(endpoint);
        try
        {
            listener.Start();
            listeners.Add(listener);

            new Thread(() =>
            {
```

```
                while (true)
                {
                    if (State != ServerState.WaitingPlayers)
                    {
                        listener.Stop();
                        return;
                    }

                    if (listener.Server.Poll(1000000, SelectMode.SelectRead))
                    {
                        try
                        {
                            events.Add(new
 ConnectionConnectEvent(listener.AcceptSocket()));
                        }
                        catch (Exception) { }
                    }
                }
            })
            { IsBackground = true }.Start();
        }
        catch (SocketException ex)
        {
            Log.Write("server", $"Failed to listen on {endpoint}: {ex.Message}");
        }
    }

    Type = type;
    Settings = settings;
    ModData = modData;
    ...
 }
```

Цей код показує, як сервер відкриває вказані порти для підключення клієнтів і асинхронно приймає нові з'єднання.

Система подій

Серверна логіка реалізована через систему подій, кожна з яких має відповідний клас, що реалізує інтерфейс IServerEvent. Наприклад:

```
 sealed class ConnectionConnectEvent : IServerEvent
 {
    readonly Socket socket;
    public ConnectionConnectEvent(Socket socket) => this.socket = socket;

    void IServerEvent.Invoke(Server server)
    {
        server.AcceptConnection(socket);
```

```
    }
}
```

Це дозволяє обробляти події асинхронно в одному потоці виконання, що оптимізує роботу сервера і запобігає блокуванню при високих навантаженнях.

Синхронізація клієнтів

При прийомі даних від клієнтів сервер виконує перевірку коректності, автентифікацію, а також забезпечує синхронізацію стану гри через метод ReceiveOrders. У разі розсинхронізації або помилки сервер може завершити ігрову сесію.

Логування та обробка збоїв

Для аналізу подій використовується окремий канал логів server.log. Крім того, вбудовано механізм запису повторів (ReplayRecorder), що дозволяє зберігати сесії гри.

Основні складові сервера:

Тип сервера (ServerType) — визначає режим роботи: локальний, мережевий, мультиплеєр або виділений сервер.

Стан сервера (ServerState) — дозволяє відслідковувати стадію ігрової сесії: очікування гравців, гра запущена, завершення роботи.

Список з'єднань (List<Connection> Conns) — активні клієнти, підключені до сервера.

Подієва система (BlockingCollection<IServerEvent> events) — обробка асинхронних подій, таких як підключення/відключення клієнтів, прийом пакетів, пінг–запити.

Ігрова логіка та обробка команд (InterpretCommand, StartGame, ReceiveOrders) — реалізовано через інтерфейси та делегування у відповідні компоненти гри.

Приклад ініціалізації сервера

Нижче наведено фрагмент конструктора класу Server, який демонструє ініціалізацію TCP–сокетів, запуск потоку прослуховування клієнтів і налаштування базових параметрів:

Лобі гри

Клас CustomLobbyLogic є частиною клієнтської логіки користувацького інтерфейсу лобі в модифікації гри. Він реалізує логіку взаємодії користувача в лобі: управління слотами, чатами, картами, налаштуваннями та взаємодію з Discord.

Основні функції CustomLobbyLogic:

Управління слотами гравців:

Додавання ботів.

Призначення команд (Free–for–all, Humans vs Bots).

Зміна кольору, фракції, команди, спавн–пойнтів, статусу готовності.

Відображення гравців та спостерігачів.

Чат–система:

Підтримка загального чату та командного.

Блокування чату при санкціях.

Система сповіщень про нові повідомлення.

Інтеграція з Discord через IRC.

Управління мапами:

Вибір карти з попереднім переглядом.

Перевірка доступності мапи.

Автоматичне оновлення та завантаження недоступних карт.

Панель адміністрування:

Примусове починання гри.

Кік гравців.

Зміна опцій лобі.

Режими: Options, Players, Music, Servers.

Обробка стану підключення:

У разі невдалого підключення, метод ConnectionStateChanged відкриває відповідну панель інтерфейсу для повторного введення паролю чи вибору модифікації:

```
 void ConnectionStateChanged(OrderManager om, string password,
NetworkConnection connection)
{
   if (connection.ConnectionState == ConnectionState.NotConnected)
   {
      Ui.CloseWindow();
      // Відкриває вікно повідомлення
      var switchPanel = CurrentServerSettings.ServerExternalMod != null ?
"CONNECTION_SWITCHMOD_PANEL" : "CONNECTIONFAILED_PANEL";
      Ui.OpenWindow(switchPanel, new WidgetArgs()
      {
         { "orderManager", om },
         { "connection", connection },
         { "password", password },
       });
   }
}
```

Старт гри:

Кнопка «Почати гру» активується тільки за дотримання наступних умов:

Користувач є хостом.

Обрана мапа є доступною.

Усі обов’язкові слоти зайняті.

Вистачає доступних точок спавну.

У разі гри без ботів — не менше двох гравців.

```
 void StartGame()
{
   if (modData.MapCache[map.Uid].Status == MapStatus.Available)
   {
      gameStarting = true;
      orderManager.IssueOrder(Order.Command("startgame"));
   }
}
```

Зв’язок із Discord

При вході в лобі, CustomLobbyLogic оновлює статус Discord Rich Presence — відображається карта, кількість гравців, сервер. Це реалізовано через DiscordService.UpdateStatus(...).

```
public Server(List<IPEndPoint> endpoints, ServerSettings settings, ModData
modData, ServerType type)
{
    Log.AddChannel(“server”, “server.log”, true);
    foreach (var endpoint in endpoints)
    {
        var listener = new TcpListener(endpoint);
        try
        {
            listener.Start();
            listeners.Add(listener);

            new Thread(() =>
            {
                while (true)
                {
                    if (State != ServerState.WaitingPlayers)
                    {
                        listener.Stop();
                        return;
                    }
                    if (listener.Server.Poll(1000000, SelectMode.SelectRead))
                    {
                        try
                        {
                            events.Add(new
ConnectionConnectEvent(listener.AcceptSocket()));
                        }
                        catch (Exception) { }
                    }
                }
            })
            { IsBackground = true }.Start();
        }
        catch (SocketException ex)
        {
            Log.Write(“server”, $”Failed to listen on {endpoint}: {ex.Message}”);
        }
    }

    Type = type;
    Settings = settings;
    ModData = modData;
    ...
}
```

Цей код показує, як сервер відкриває вказані порти для підключення клієнтів і асинхронно приймає нові з'єднання.

Система подій

Серверна логіка реалізована через систему подій, кожна з яких має відповідний клас, що реалізує інтерфейс IServerEvent. Наприклад:

```
sealed class ConnectionConnectEvent : IServerEvent
{
   readonly Socket socket;
   public ConnectionConnectEvent(Socket socket) => this.socket = socket;

   void IServerEvent.Invoke(Server server)
   {
      server.AcceptConnection(socket);
   }
}
```

Це дозволяє обробляти події асинхронно в одному потоці виконання, що оптимізує роботу сервера і запобігає блокуванню при високих навантаженнях.

Синхронізація клієнтів

При прийомі даних від клієнтів сервер виконує перевірку коректності, автентифікацію, а також забезпечує синхронізацію стану гри через метод ReceiveOrders. У разі розсинхронізації або помилки сервер може завершити ігрову сесію.

Логування та обробка збоїв

Для аналізу подій використовується окремий канал логів server.log. Крім того, вбудовано механізм запису повторів (ReplayRecorder), що дозволяє зберігати сесії гри.

Цей розділ можна доповнити у наступних главах описом клієнтської частини, інтерфейсу адміністратора та обробки специфічних команд гравця. Якщо потрібен розширений варіант із конкретними структурами даних, логікою обробки повідомлень або із прикладами типових сценаріїв, — можу підготувати продовження.

Клієнтська частина та запуск гри

Клас Game є центральним статичним класом, що координує запуск, ініціалізацію, підключення до сервера, створення локального сервера, візуалізацію та обробку ігрового циклу.

Ключові обов’язки Game:

Завантаження модів та налаштувань.

Ініціалізація рендерингу, аудіо та інших підсистем.

Управління локальним або мережевим з’єднанням.

Запуск локального сервера для одиночної гри або тестування.

Організація головного ігрового циклу (рендерінг та логіка).

Ведення лог–файлу активності через метод LogTEXT.WriteTEXT(...).

Підключення до сервера

Підключення клієнта до сервера реалізується через метод JoinServer:

```
public static OrderManager JoinServer(ConnectionTarget endpoint, string password, bool recordReplay = true)
{
    var newConnection = new NetworkConnection(endpoint);
    if (recordReplay)
        newConnection.StartRecording(() => TimestampedFilename());

    var om = new OrderManager(newConnection);
    JoinInner(om);
    CurrentServerSettings.Password = password;
    CurrentServerSettings.Target = endpoint;

    return om;
}
```

Цей метод створює мережеве з’єднання, ініціалізує менеджер команд (OrderManager) і готує клієнт до роботи в багатокористувацькому середовищі.

Створення локального сервера

У грі підтримується створення локального або скріміш–сервера (гра проти ботів) через метод CreateLocalServer:

```
public static ConnectionTarget CreateLocalServer(string map, bool isSkirmish = false)
{
    LogTEXT.WriteTEXT(“Skirmish Game”, $”{Settings.Player.Name}”);
    var settings = new ServerSettings()
    {
        Name = “Skirmish Game”,
```

```
            Map = map,
            AdvertiseOnline = false
        };

        var endpoints = new List<IPEndPoint>
        {
            new(IPAddress.Loopback, 0)
        };

        server = new Server.Server(endpoints, settings, ModData, isSkirmish ?
ServerType.Skirmish : ServerType.Local);

        return server.GetEndpointForLocalConnection();
    }
```

Це дозволяє швидко запускати гру без необхідності зовнішнього з'єднання, що корисно для тестування чи одиночних сценаріїв.

Ігровий цикл

Основний цикл гри виконується в методі Loop() і поділений на:

LogicTick() — обробка логіки гри, команд гравців, порядків.

RenderTick() — оновлення графіки, рендерінг об'єктів і інтерфейсу.

Sleep — динамічна затримка для оптимізації використання ресурсів CPU.

Такий підхід дозволяє підтримувати стабільну частоту кадрів та чуйність інтерфейсу навіть при високому навантаженні.

Логування та аналітика

У коді реалізована функція LogTEXT.WriteTEXT(...), яка дозволяє вести власні логи гри в окремий файл:

```
    public static void WriteTEXT(string category, string message)
    {
        var logMessage = $"{DateTime.Now:yyyy-MM-dd HH:mm:ss} [{category}]
{message}";
        lock (_lock)
        {
            File.AppendAllText(LogFilePath, logMessage + Environment.NewLine);
        }
    }
```

Це може бути корисним для подальшого аналізу сесій, статистики, відлагодження та контролю доступу.

3.3 Розробка інструкції з експлуатації програмного забезпечення

Запуск та налаштування

У цьому розділі ми роздивимося як запускати та налаштовувати окрему сесію гри «Назва гри».

Крок 2.1 Запуск гри «Назва гри»

Розділ буде пов’язаний з меню запуском гри.

Рисунок 3.1 - Інтерфейс головного меню

На даний момент в гри реалізовано лише «Singleplayer», тобто гра на одну людину, у майбутньому буде реалізований режим «multiplayer» – гра на декілька людей. Тому натискаємо на кнопку «Singleplayer» та переходимо у наступне меню.

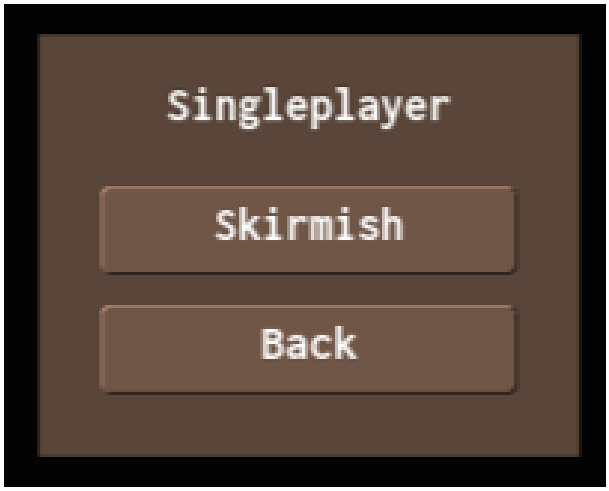

Рисунок 3.2 - Інтерфейс меню

У даному меню реалізовано створення локальної мережі для одиночної гри. У майбутньому буде реалізовано функціонал збереження та перед перегляд гри.

натиснувши на кнопку «Skirmish» ми переходимо у меню налаштування ігрової сесії.

Крок 2.2 Налаштування ігрової сесії гри «Назва гри»

У цьому розділі ми роздивимося налаштування ігрової сесії гри «Назва гри».

Перше на що можна звернути увагу – це реалізацію чату.

Рисунок 3.3 Інтерфейс налаштування сессії

Далі на скріні ми роздивимося налаштування панелі «Faction» розділі «Players».

Дана панель дає можливість обрати фракцію кожна з яких має свою реалізацію Штучного Інтелекту. Більш детально розписано у 3 розділі.

Наступна панель яку ми розглянемо називається «Team».

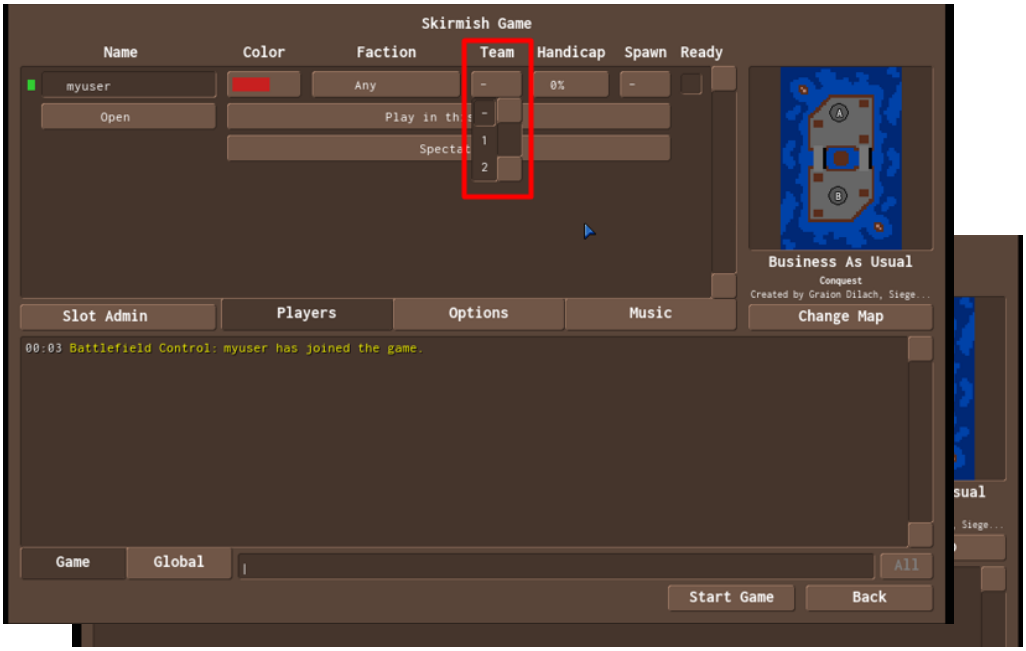

Рисунок 3.5 - Інтерфейс панелі «Team»

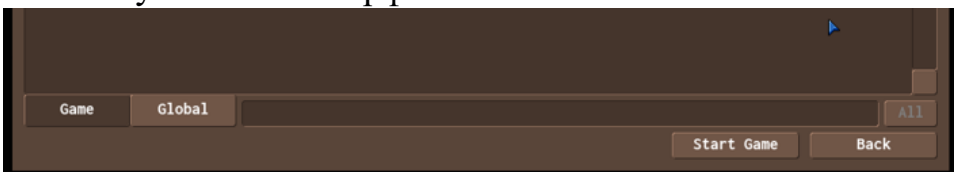

Рисунок 3.4 - Інтерфейс панелі «Players»

У панелі реалізована функція вибору команди за яку ми будемо грати.

Паналь яка йде далі це панель яка називається «Spawn».

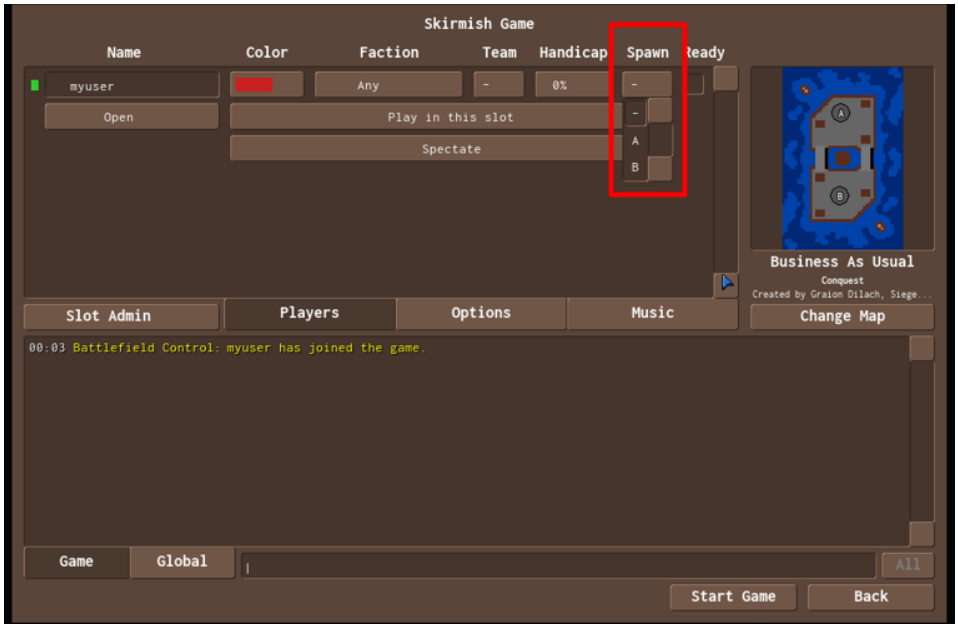

Рисунок 3.6 - Інтерфейс панелі «Spawn»

У цій панелі ми обираємо місце на карті з якого будемо починати гру та місце розташування нашої бази (найголовніша споруда). У майбутньому коли буде реалізована можливість додавати інші карти ми будемо мати можливість обирати різні карти із списку по кнопці «Change map».

І останнє поле у цьому розділі це «Color».

У полі можна обирати колір ваших юнітів та споруд.

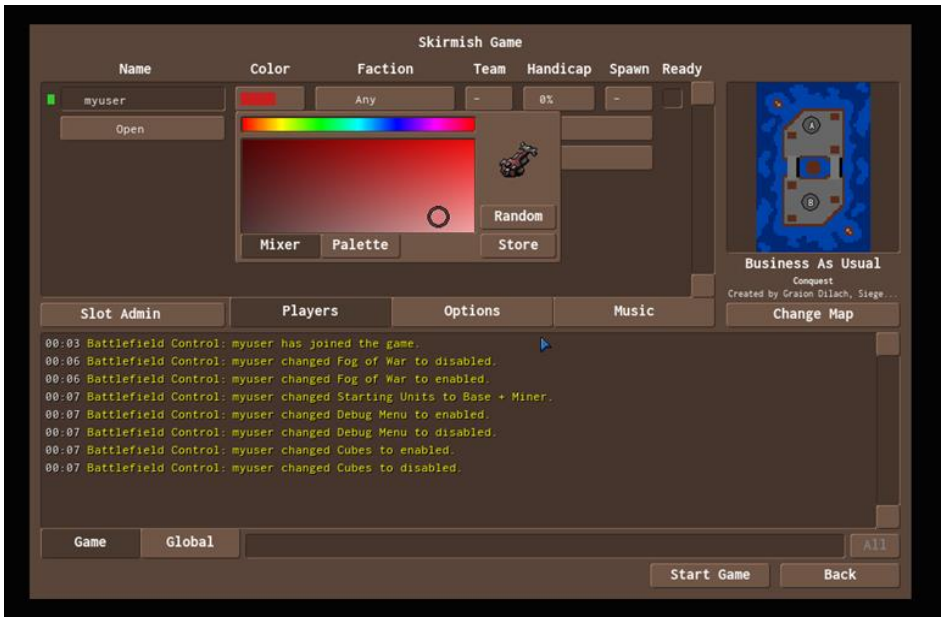

Рисунок 3.7 - Інтерфейс панелі «Color»

Далі ми розглянемо наступний розділ який називається «Options»

У розділі маємо 9 полів чекерів та 4 поля випадайки.

Рисунок 3.8 - Інтерфейс розділу «Options»

Поля чекери:

- Explored map – якщо чекер включений то уся карта буде видна одразу і її не треба буде досліджувати.
- Short Game – якщо чекер включений то гра закінчується коли база суперника буде знищена.
- Separate team spawns – якщо чекер включений то гравці без призначених точок появи почнуть гру якомога далі від ворожих гравців.
- Fog of war – якщо чекер включений то він зменшує огляд суперників по карті.
- Build off Allies – якщо чекер включений то у гравців є можливість розміщати свої бази на території суперника та грати з ним у команді.
- Scrap – якщо чекер включений то на місці знищених юнітів або споруд буде сміття яке не дозволить використовувати дану точку.
- Cubes – якщо чекер включений то за кожного вбитого юніта або споруди ми отримуємо грошову винагороду.

- Limit build area – якщо чекер включений то в грі у вас буде обмежена область для будівництва.
- Debug menu – якщо чекер включена то у вас буде можливість використати меню для адміністратора.

Наступне поле яке ми має це поле випадайка з назвою «Starting cash».

Рисунок 3.9 - Інтерфейс розділу «Starting cash»

В даному полі ми обираємо стартовий капітал для гри.

Наступне поле називається «Time limit».

Рисунок 3.10 - Інтерфейс розділу «Time limit»

В даному полі ми обираємо часові границі гри

Далі йде поле «Startinig Units».

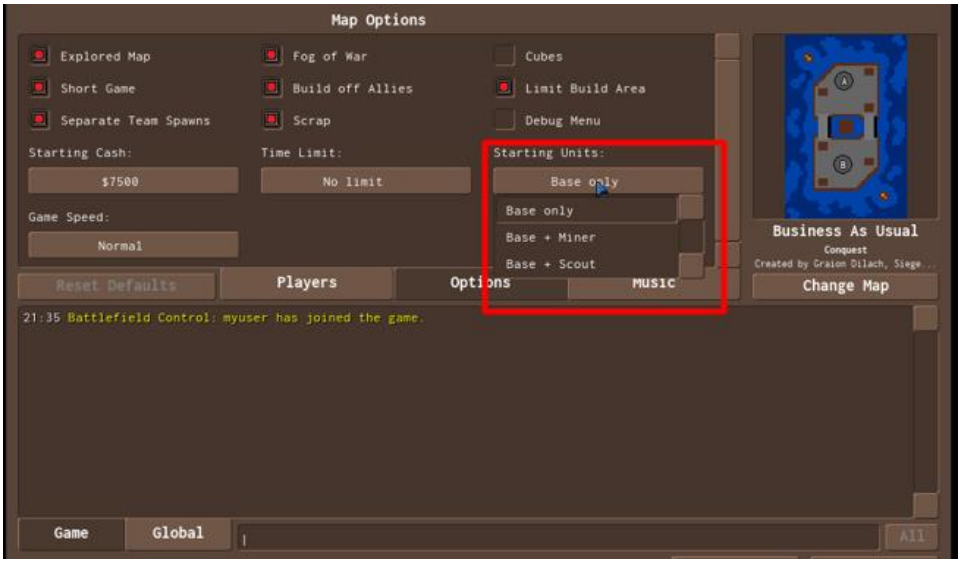

Рисунок 3.11 - Інтерфейс поля «Startinig Units»

Дане поле використовується для того щоб налаштувати те з яким юнітом ми будемо починати гру.

Останнє поле у даному розділі має назву «Game speed»

Рисунок 3.12 - Інтерфейс поля «Game speed»

Поле потрібно для того щоб обрати швидкість з якою буде розвиватися гра.

Ігровий процес

У цьому розділі ми роздивимося ігровий процес у створеній сесії гри «Назва гри».

Крок 3.1 «HEALTH BAR»

У цьому розділі ми роздивимося що таке «HEALTH BAR»

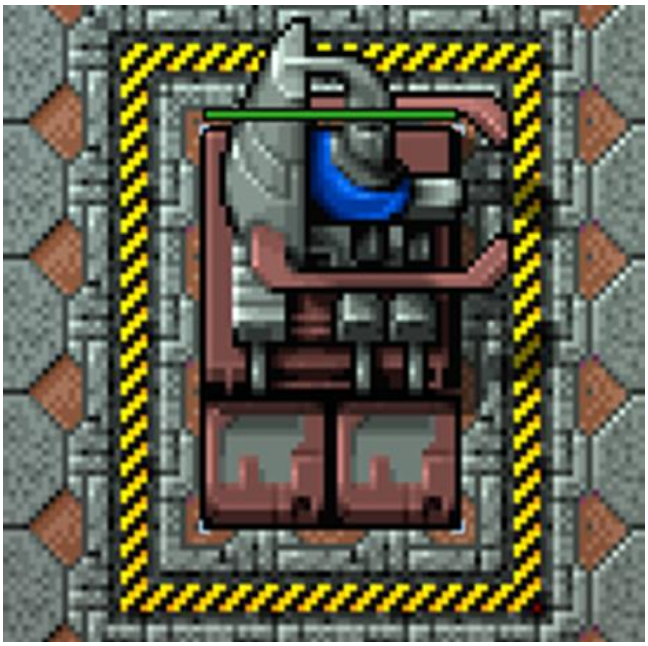

Рисунок 3.13 - Інтерфейс «HEALTH BAR»

«HEALTH BAR» – це візуальне подання здоров'я персонажа або сутності в комп'ютерних іграх, яке зазвичай відображається у вигляді шкали або смуги. На даному скріні показаний HEALTH BAR споруди. А на скріні нижче

Рисунок 3.14 - Інтерфейс юніта

показаний HEALTH BAR юніта.

Крок 3.2 «Опис об’єкта»

На скріні нижче є опис споруди на яку ми навели курсор миші.

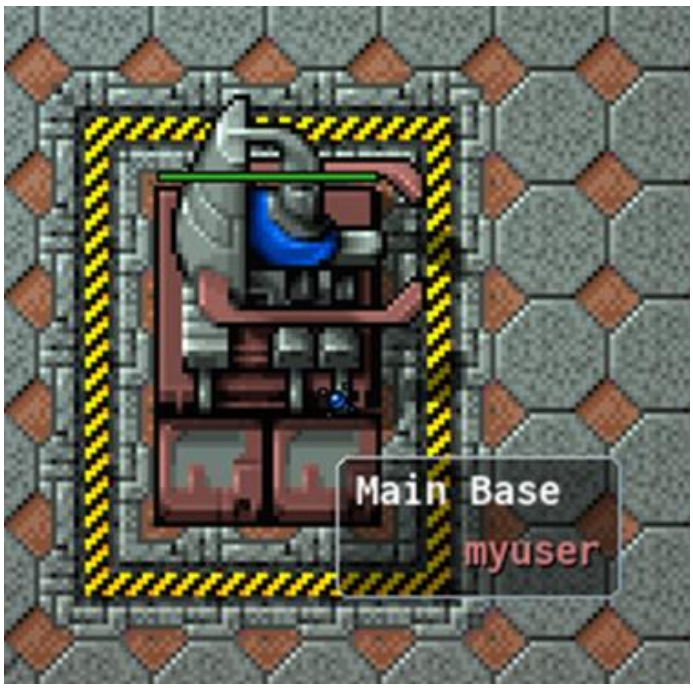

Рисунок 3.15 - Інтерфейс споруди

На наступному скріні в нас опис юніта на якого ми навели курсор миші.

Крок 3.3 «Опис інтерфейсу в ігровій сесії»

У розділі ми роздивимося сам інтерфейс вже в ігровій сесії який будемо використовувати для певних дій або для відстеження економіки, ігрового часу та ліміт кількості юнітів на карті.

Рисунок 3.17 - Інтерфейс ігрової карти

Рисунок 3.16 - Інтерфейс описа юніта

Ще в даному інтерфейсі ми спостерігаємо за картою та тим що на ній розміщено.

Одна з можливостей яка присутня в цій грі це ставити траєкторію руху обраного юніта як показано на скріні нижче. Траєкторію руху визначена зеленою смужкою

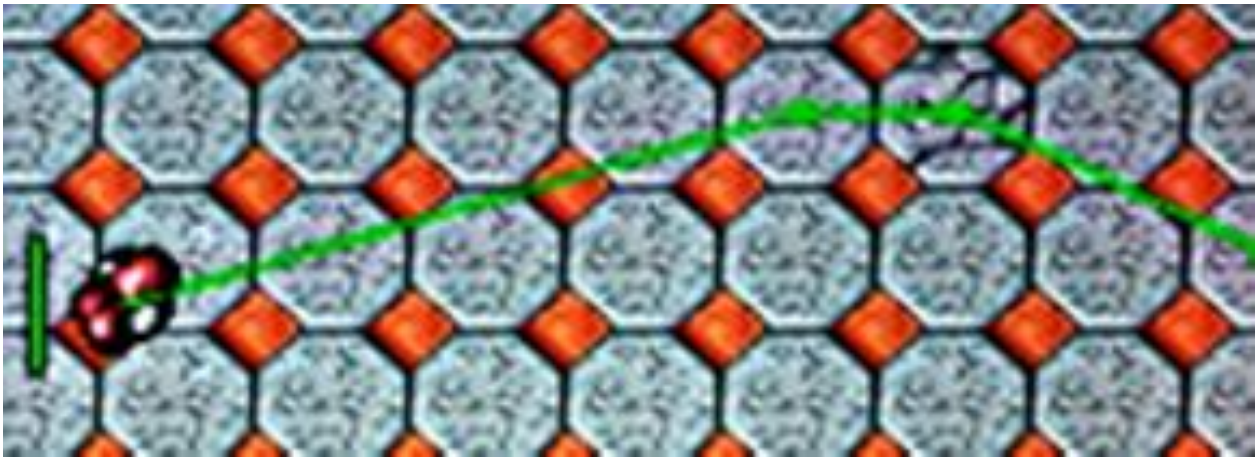

Рисунок 3.17 - Інтерфейс руху юніта

ВИКОРАСТАНІ ДЖЕРЕЛА

1. Шрейєр Дж. Кров, піт та пікселі. Зворотній бік індустрії відеоігор. — Харків: Фоліо, 2021.
2. Елетронна версія Walters, D. MonoGame Mastery: Build a Multi–Platform 2D Game and Reusable Game Engine (Прямо присвячена розробці ігор без рушія)
3. Харрісон Ф. *Вивчаємо C# через розробку ігор на Unity*. — Київ: Видавництво “Ранок”, 2022.
4. Steam. *Статистика Steam та ігор*. — https://store.steampowered.com/stats/stats/?l=ukrainian
5. FoxmindEd. *Книги для занурення в індустрію ігор*. — https://foxminded.ua/knygy–dlya–zanurennya–v–industriyu–igor/
6. KNEU. *Дослідження, аналіз та апробація серйозних ігор і симуляцій*. — https://ivo.kneu.edu.ua/ua/dosl_glot/proj_soit/s_games_simul/
7. Стаття про UML–діаграми, – Режим доступу: https://www.quality–assurance–group.com/type–uml–diagrams/;
8. Інформація про алгоритми, – Режим доступу: https://uk.wikipedia.org/wiki/Алгоритм;
9. Офіційний сайт Visual Studio, – Режим доступу: https://visualstudio.microsoft.com/ru/vs/;
10. Огляд на гру *Celeste*: https://www.ign.com/articles/2018/01/25/celeste–review
11. Офіційний сайт гри *Hollow Knight*: https://www.hollowknight.com/
12. Сторінка інформації про гру *Hollow Knight*: https://en.wikipedia.org/wiki/Hollow_Knight
13. Огляд на гру *Ori and the Blind Forest*: https://www.eurogamer.net/ori–and–the–blind–forest–review

14. Аналітика гри *Rayman Legends*:
https://www.ign.com/articles/2013/08/26/rayman–legends–review

ДОДАТОК А

```
using System.Collections.Generic;
using System.Linq;
using OpenRA.Mods.Common.Traits;
using OpenRA.Traits;
namespace OpenRA.Mods.Common
{
 public enum BuildingType { Building, Defense, Refinery }
 public enum WaterCheck { NotChecked, EnoughWater, NotEnoughWater,
DontCheck }
 public static class IntelliHelper
 {
 public static bool IsAreaAvailable<T>(EnvWorld world, Gamer player, Map
zoneMap, int radius, HashSet<string> terrainTypes)
 {
 var cells = world.ActorsHavingTrait<T>().Where(a => a.Owner == player);
 return cells.Select(a => zoneMap.FindTilesInCircle(a.Location, radius)
 .Count(c => zoneMap.Contains(c) &&
terrainTypes.Contains(zoneMap.GetTerrainInfo(c).Type) &&
 Util.AdjacentCells(world, Target.FromCell(world, c))
 .All(ac => zoneMap.Contains(ac) &&
terrainTypes.Contains(zoneMap.GetTerrainInfo(ac).Type))))
 .Any(availableCells => availableCells > 0);
 }
 public static ILookup<string, ProductionQueue> FindQueuesByCategory(Gamer
player)
 {
 return player.EnvWorld.ActorsWithTrait<ProductionQueue>()
 .Where(a => a.Actor.Owner == player && a.Trait.Enabled)
 .Select(a => a.Trait)
 .ToLookup(pq => pq.Info.Type);
 }
 public static int CountActorsWithNameAndTrait<T>(string actorName, Gamer
owner)
 {
 return owner.EnvWorld.ActorsHavingTrait<T>().Count(a => a.Owner == owner &&
a.Info.Name == actorName);
 }
 public static int
CountActorByCommonName<T>(ActorIndex.OwnerAndNamesAndTrait<T> actorIndex)
 {
 return actorIndex.Actors.Count(a => !a.IsDead);
 }
 public static void BotDebug(string format, params object[] args)
 {
 if (Game.Settings.Debug.BotDebug)
 TextNotificationsManager.Debug(format, args);
```

```
        }
        public static IEnumerable<Order> ClearBlockersOrders(List<CPos> tiles, Gamer
owner, Actor ignoreActor = null)
        {
        var world = owner.EnvWorld;
        var adjacentTiles = Util.ExpandFootprint(tiles, true).Except(tiles)
        .Where(world.Map.Contains).ToList();
        var blockers = tiles.SelectMany(world.ActorMap.GetActorsAt)
        .Where(a => a.Owner == owner && a.IsIdle && (ignoreActor == null || a !=
ignoreActor))
        .Select(a => new TraitPair<IMove>(a, a.TraitOrDefault<IMove>()))
        .Where(x => x.Trait != null);
        foreach (var blocker in blockers)
        {
        CPos moveCell;
        if (blocker.Trait is Mobile mobile)
        {
        var availableCells = adjacentTiles.Where(t => mobile.CanEnterCell(t)).ToList();
        if (availableCells.Count == 0)
        continue;
        moveCell = blocker.Actor.ClosestCell(availableCells);
        }
        else if (blocker.Trait is Aircraft)
        moveCell = blocker.Actor.Location;
        else
        continue;
        yield return new Order("Move", blocker.Actor, Target.FromCell(world, moveCell),
false)
        {
        SuppressVisualFeedback = true
        };
        }
        }
        }
       }
       using System;
       using System.Collections.Generic;
       using System.Linq;
       using OpenRA.Graphics;
       using OpenRA.Mods.Common;
       using OpenRA.Mods.Common.Traits;
       using OpenRA.Mods.Common.Widgets;
       using OpenRA.Mods.Common.Widgets.Logic;
       using OpenRA.Network;
       using OpenRA.Primitives;
       using OpenRA.Traits;
       using OpenRA.Widgets;
       namespace OpenRA.Mods.HV.Widgets.Logic
       {
        public class LobbyXSystem : ChromeLogic, INotificationHandler<TextNotification>
        {
```

```
[TranslationReference]
const string Add = “options-slot-admin.add-bots”;
[TranslationReference]
const string Remove = “options-slot-admin.remove-bots”;
[TranslationReference]
const string ConfigureBots = “options-slot-admin.configure-bots”;
[TranslationReference(“count”)]
const string NumberTeams = “options-slot-admin.teams-count”;
[TranslationReference]
const string HumanVsBots = “options-slot-admin.humans-vs-bots”;
[TranslationReference]
const string FreeForAll = “options-slot-admin.free-for-all”;
[TranslationReference]
const string ConfigureTeams = “options-slot-admin.configure-teams”;
[TranslationReference]
const string Back = “button-back”;
[TranslationReference]
const string TeamChat = “button-team-msgLine”;
[TranslationReference]
const string GeneralChat = “button-general-msgLine”;
[TranslationReference(“seconds”)]
const string ChatAvailability = “label-msgLine-availability”;
[TranslationReference]
const string ChatDisabled = “label-msgLine-disabled”;
static readonly Action DoNothing = () => { };
readonly ModMatrix modData;
readonly Action onStart;
readonly Action onExit;
readonly CmdMgr orderManager;
readonly WorldRenderer worldRenderer;
readonly bool skirmishMode;
readonly Ruleset modRules;
readonly WebServices services;
readonly InternetRelayChat internetRelayChat;
enum PanelType { Players, Options, Music, Servers, Kick, ForceStart }
PanelType panel = PanelType.Players;
enum ChatPanelType { Lobby, Global }
ChatPanelType msgView = ChatPanelType.Lobby;
readonly CtrlUnit lobby;
readonly CtrlUnit editablePlayerTemplate;
readonly CtrlUnit nonEditablePlayerTemplate;
readonly CtrlUnit emptySlotTemplate;
readonly CtrlUnit editableSpectatorTemplate;
readonly CtrlUnit nonEditableSpectatorTemplate;
readonly CtrlUnit newSpectatorTemplate;
readonly ScrollPanelWidget lobbyChatPanel;
readonly Dictionary<TextNotificationPool, CtrlUnit> chatTemplates = new();
readonly TextFieldWidget chatTextField;
readonly CachedTransform<int, string> chatAvailableIn;
readonly string chatDisabled;
readonly ScrollPanelWidget players;
```

```
        readonly Dictionary<string, LobbyFaction> factions = new();
        readonly IColorPickerManagerInfo colorManager;
        readonly TabCompletionLogic tabCompletion = new();
        MapPreview zoneMap;
        Session.MapStatus mapStatus;
        bool chatEnabled;
        bool disableTeamChat;
        bool insufficientPlayerSpawns;
        bool teamChat;
        int lobbyChatUnreadMessages;
        int globalChatLastReadMessages;
        int globalChatUnreadMessages;
        bool updateDiscordStatus = true;
        bool resetOptionsButtonEnabled;
        Dictionary<int, SpawnOccupant> spawnOccupants = new();
        readonly string chatLineSound;
        readonly string playerJoinedSound;
        readonly string playerLeftSound;
        readonly string lobbyOptionChangedSound;
        bool MapIsPlayable => (mapStatus & Session.MapStatus.Playable) ==
Session.MapStatus.Playable;
        void ConnectionStateChanged(CmdMgr om, string password, NetworkConnection
connection)
        {
        if (connection.ConnectionState == ConnectionState.NotConnected)
        {
        Ui.CloseWindow();
        void OnConnect()
        {
        Game.OpenWindow("SERVER_LOBBY", new WidgetArgs()
        {
        { "onExit", onExit },
        { "onStart", onStart },
        { "skirmishMode", false }
        });
        }
        Action<string> onRetry = pass => ConnectionLogic.Connect(connection.Target,
pass, OnConnect, onExit);
        var switchPanel = CurrentServerSettings.ServerExternalMod != null ?
"CONNECTION_SWITCHMOD_PANEL" : "CONNECTIONFAILED_PANEL";
        Ui.OpenWindow(switchPanel, new WidgetArgs()
        {
        { "orderManager", om },
        { "connection", connection },
        { "password", password },
        { "onAbort", onExit },
        { "onQuit", null },
        { "onRetry", onRetry }
        });
        }
        }
```

```
        [ObjectCreator.UseCtor]
        internal LobbyXSystem(CtrlUnit widget, ModMatrix modData, WorldRenderer
worldRenderer, CmdMgr orderManager,
        Action onExit, Action onStart, bool skirmishMode, Dictionary<string, MiniYaml>
logicArgs)
        {
        zoneMap = MapCache.UnknownMap;
        lobby = widget;
        this.modData = modData;
        this.orderManager = orderManager;
        this.worldRenderer = worldRenderer;
        this.onStart = onStart;
        this.onExit = onExit;
        this.skirmishMode = skirmishMode;
        modRules = modData.DefaultRules;
        services = modData.Manifest.Get<WebServices>();
        internetRelayChat = modData.Manifest.Get<InternetRelayChat>();
        Game.LobbyInfoChanged += UpdateCurrentMap;
        Game.LobbyInfoChanged += UpdatePlayerList;
        Game.LobbyInfoChanged += UpdateDiscordStatus;
        Game.LobbyInfoChanged += UpdateSpawnOccupants;
        Game.LobbyInfoChanged += UpdateOptions;
        Game.BeforeGameStart += OnGameStart;
        Game.ConnectionStateChanged += ConnectionStateChanged;
        ChromeMetrics.TryGet("ChatLineSound", out chatLineSound);
        ChromeMetrics.TryGet("PlayerJoinedSound", out playerJoinedSound);
        ChromeMetrics.TryGet("PlayerLeftSound", out playerLeftSound);
        ChromeMetrics.TryGet("LobbyOptionChangedSound", out
lobbyOptionChangedSound);
        var name = lobby.GetOrNull<LabelWidget>("SERVER_NAME");
        if (name != null)
        name.GetText = () => orderManager.LobbyInfo.GlobalSettings.ServerName;
        var mapContainer = Ui.LoadWidget("MAP_PREVIEW",
lobby.Get("MAP_PREVIEW_ROOT"), new WidgetArgs
        {
        { "orderManager", orderManager },
        { "getMap", (Func<(MapPreview, Session.MapStatus)>)(() => (zoneMap,
mapStatus)) },
        {
        "onMouseDown", (Action<MapPreviewWidget, MapPreview,
MouseInput>)((preview, mapPreview, mi) =>
        LobbyUtils.SelectSpawnPoint(orderManager, preview, mapPreview, mi))
        },
        { "getSpawnOccupants", (Func<Dictionary<int, SpawnOccupant>>)(() =>
spawnOccupants) },
        { "getDisabledSpawnPoints", (Func<HashSet<int>>)(() =>
orderManager.LobbyInfo.DisabledSpawnPoints) },
        { "showUnoccupiedSpawnpoints", true },
        { "mapUpdatesEnabled", true },
        {
        "onMapUpdate", (Action<string>)(uid =>
```

```
        {
        orderManager.IssueOrder(Order.Command("zoneMap " + uid));
        Game.Settings.Server.Map = uid;
        Game.Settings.Save();
        })
        },
        });
        mapContainer.ShowWhen = () => panel != PanelType.Servers;
        UpdateCurrentMap();
        var playerBin = Ui.LoadWidget("LOBBY_PLAYER_BIN",
lobby.Get("TOP_PANELS_ROOT"), new WidgetArgs());
        playerBin.ShowWhen = () => panel == PanelType.Players;
        players = playerBin.Get<ScrollPanelWidget>("LOBBY_PLAYERS");
        editablePlayerTemplate = players.Get("TEMPLATE_EDITABLE_PLAYER");
        nonEditablePlayerTemplate =
players.Get("TEMPLATE_NONEDITABLE_PLAYER");
        emptySlotTemplate = players.Get("TEMPLATE_EMPTY");
        editableSpectatorTemplate =
players.Get("TEMPLATE_EDITABLE_SPECTATOR");
        nonEditableSpectatorTemplate =
players.Get("TEMPLATE_NONEDITABLE_SPECTATOR");
        newSpectatorTemplate = players.Get("TEMPLATE_NEW_SPECTATOR");
        colorManager =
modRules.Actors[SystemActors.EnvWorld].TraitInfo<IColorPickerManagerInfo>();
        foreach (var f in
modRules.Actors[SystemActors.EnvWorld].TraitInfos<FactionInfo>())
        factions.Add(f.InternalName, new LobbyFaction { Selectable = f.Selectable, Name
= f.Name, Side = f.Side, Description = f.Description?.Replace(@"\n", "\n") });
        var gameStarting = false;
        Func<bool> configurationDisabled = () => !Game.IsHost || gameStarting ||
        panel == PanelType.Kick || panel == PanelType.ForceStart || !MapIsPlayable ||
        orderManager.LocalClient == null || orderManager.LocalClient.IsReady;
        var mapButton = lobby.GetOrNull<ButtonWidget>("CHANGEMAP_BUTTON");
        if (mapButton != null)
        {
        mapButton.ShowWhen = () => panel != PanelType.Servers;
        mapButton.IsDisabled = () => gameStarting || panel == PanelType.Kick || panel ==
PanelType.ForceStart ||
        orderManager.LocalClient == null || orderManager.LocalClient.IsReady;
        mapButton.OnClick = () =>
        {
        var onSelect = new Action<string>(uid =>
        {
        var status = modData.MapCache[uid].Status;
        if (uid == zoneMap.Uid || (status != MapStatus.Available && status !=
MapStatus.DownloadAvailable))
        return;
        orderManager.IssueOrder(Order.Command("zoneMap " + uid));
        Game.Settings.Server.Map = uid;
        Game.Settings.Save();
        });
```

```
modData.MapCache.UpdateMaps();
Ui.OpenWindow("MAPCHOOSER_PANEL", new WidgetArgs()
{
{ "initialMap", modData.MapCache.PickLastModifiedMap(MapVisibility.Lobby) ??
zoneMap.Uid },
{ "remoteMapPool", orderManager.ServerMapPool },
{ "initialTab", MapClassification.System },
{ "onExit", modData.MapCache.UpdateMaps },
{ "onSelect", Game.IsHost ? onSelect : null },
{ "filter", MapVisibility.Lobby },
});
};
}
var slotsButton =
lobby.GetOrNull<DropDownButtonWidget>("SLOTS_DROPDOWNBUTTON");
if (slotsButton != null)
{
slotsButton.ShowWhen = () => panel != PanelType.Servers && panel !=
PanelType.Options;
slotsButton.IsDisabled = () => configurationDisabled() || panel !=
PanelType.Players ||
(orderManager.LobbyInfo.Slots.Values.All(s => !s.AllowBots) &&
!orderManager.LobbyInfo.Slots.Any(s => !s.Value.LockTeam &&
orderManager.LobbyInfo.ClientInSlot(s.Key) != null));
slotsButton.OnMouseDown = _ =>
{
var botTypes = zoneMap.PlayerActorInfo.TraitInfos<IBotInfo>().Select(t => t.Type);
var options = new Dictionary<string, IEnumerable<DropDownOption>>();
var botController = orderManager.LobbyInfo.Clients.FirstOrDefault(c => c.IsAdmin);
if (orderManager.LobbyInfo.Slots.Values.Any(s => s.AllowBots))
{
var botOptions = new List<DropDownOption>()
{
new()
{
Title = TranslationProvider.GetString(Add),
IsSelected = () => false,
OnClick = () =>
{
foreach (var slot in orderManager.LobbyInfo.Slots)
{
var bot = botTypes.Random(Game.CosmeticRandom);
var c = orderManager.LobbyInfo.ClientInSlot(slot.Key);
if (slot.Value.AllowBots && (c == null || c.Bot != null))
orderManager.IssueOrder(Order.Command($"slot_bot {slot.Key}
{botController.Index} {bot}"));
}
}
}
};
if (orderManager.LobbyInfo.Clients.Any(c => c.Bot != null))
```

```
        {
        botOptions.Add(new DropDownOption()
        {
        Title = TranslationProvider.GetString(Remove),
        IsSelected = () => false,
        OnClick = () =>
        {
        foreach (var slot in orderManager.LobbyInfo.Slots)
        {
        var c = orderManager.LobbyInfo.ClientInSlot(slot.Key);
        if (c != null && c.Bot != null)
        orderManager.IssueOrder(Order.Command("slot_open " +
slot.Value.PlayerReference));
        }
        }
        });
        }
        options.Add(TranslationProvider.GetString(ConfigureBots), botOptions);
        }
        var teamCount = (orderManager.LobbyInfo.Slots.Count(s => !s.Value.LockTeam
&& orderManager.LobbyInfo.ClientInSlot(s.Key) != null) + 1) / 2;
        if (teamCount >= 1)
        {
        var teamOptions = Enumerable.Range(2, teamCount - 1).Reverse().Select(d =>
new DropDownOption
        {
        Title = TranslationProvider.GetString(NumberTeams,
Translation.Arguments("count", d)),
        IsSelected = () => false,
        OnClick = () => orderManager.IssueOrder(Order.Command($"assignteams {d}"))
        }).ToList();
        if (orderManager.LobbyInfo.Slots.Any(s => s.Value.AllowBots))
        {
        teamOptions.Add(new DropDownOption
        {
        Title = TranslationProvider.GetString(HumanVsBots),
        IsSelected = () => false,
        OnClick = () => orderManager.IssueOrder(Order.Command("assignteams 1"))
        });
        }
        teamOptions.Add(new DropDownOption
        {
        Title = TranslationProvider.GetString(FreeForAll),
        IsSelected = () => false,
        OnClick = () => orderManager.IssueOrder(Order.Command("assignteams 0"))
        });
        options.Add(TranslationProvider.GetString(ConfigureTeams), teamOptions);
        }
        ScrollItemWidget SetupItem(DropDownOption option, ScrollItemWidget template)
        {
        var item = ScrollItemWidget.Setup(template, option.IsSelected, option.OnClick);
```

```
        item.Get<LabelWidget>("LABEL").GetText = () => option.Title;
        return item;
        }
        slotsButton.ShowDropDown("LABEL_DROPDOWN_TEMPLATE", 175, options,
SetupItem);
        };
        }
        var resetOptionsButton =
lobby.GetOrNull<ButtonWidget>("RESET_OPTIONS_BUTTON");
        if (resetOptionsButton != null)
        {
        resetOptionsButton.ShowWhen = () => panel == PanelType.Options;
        resetOptionsButton.IsDisabled = () => configurationDisabled() ||
!resetOptionsButtonEnabled;
        resetOptionsButton.OnMouseDown = _ =>
orderManager.IssueOrder(Order.Command("reset_options"));
        }
        var optionsBin = Ui.LoadWidget("LOBBY_OPTIONS_BIN",
lobby.Get("TOP_PANELS_ROOT"), new WidgetArgs()
        {
        { "orderManager", orderManager },
        { "getMap", (Func<MapPreview>)(() => zoneMap) },
        { "configurationDisabled", configurationDisabled }
        });
        optionsBin.ShowWhen = () => panel == PanelType.Options;
        var musicBin = Ui.LoadWidget("LOBBY_MUSIC_BIN",
lobby.Get("TOP_PANELS_ROOT"), new WidgetArgs
        {
        { "onExit", DoNothing },
        { "world", worldRenderer.EnvWorld }
        });
        musicBin.ShowWhen = () => panel == PanelType.Music;
        ServerListLogic serverListLogic = null;
        if (!skirmishMode)
        {
        Action<GameServer> doNothingWithServer = _ => { };
        var serversBin = Ui.LoadWidget("LOBBY_SERVERS_BIN",
lobby.Get("TOP_PANELS_ROOT"), new WidgetArgs
        {
        { "onJoin", doNothingWithServer },
        });
        serverListLogic = serversBin.LogicObjects.Select(l => l as
ServerListLogic).FirstOrDefault(l => l != null);
        serversBin.ShowWhen = () => panel == PanelType.Servers;
        }
        var tabContainer = skirmishMode ? lobby.Get("SKIRMISH_TABS") :
lobby.Get("MULTIPLAYER_TABS");
        tabContainer.ShowWhen = () => true;
        var optionsTab = tabContainer.Get<ButtonWidget>("OPTIONS_TAB");
        optionsTab.IsHighlighted = () => panel == PanelType.Options;
        optionsTab.IsDisabled = OptionsTabDisabled;
```

```
optionsTab.OnClick = () => panel = PanelType.Options;
var playersTab = tabContainer.Get<ButtonWidget>("PLAYERS_TAB");
playersTab.IsHighlighted = () => panel == PanelType.Players;
playersTab.IsDisabled = () => panel == PanelType.Kick || panel ==
PanelType.ForceStart;
playersTab.OnClick = () => panel = PanelType.Players;
var musicTab = tabContainer.Get<ButtonWidget>("MUSIC_TAB");
musicTab.IsHighlighted = () => panel == PanelType.Music;
musicTab.IsDisabled = () => panel == PanelType.Kick || panel ==
PanelType.ForceStart;
musicTab.OnClick = () => panel = PanelType.Music;
var serversTab = tabContainer.GetOrNull<ButtonWidget>("SERVERS_TAB");
if (serversTab != null)
{
serversTab.IsHighlighted = () => panel == PanelType.Servers;
serversTab.IsDisabled = () => panel == PanelType.Kick || panel ==
PanelType.ForceStart;
serversTab.OnClick = () =>
{
if (serverListLogic != null && panel != PanelType.Servers)
serverListLogic.RefreshServerList();
panel = PanelType.Servers;
};
}
void StartGame()
{
if (modData.MapCache[zoneMap.Uid].Status == MapStatus.Available)
{
gameStarting = true;
orderManager.IssueOrder(Order.Command("startgame"));
}
else
modData.MapCache.UpdateMaps();
}
bool StartDisabled() => zoneMap.Status != MapStatus.Available ||
orderManager.LobbyInfo.Slots.Any(sl => sl.Value.Required &&
orderManager.LobbyInfo.ClientInSlot(sl.Key) == null) ||
orderManager.LobbyInfo.Slots.All(sl =>
orderManager.LobbyInfo.ClientInSlot(sl.Key) == null) ||
(!orderManager.LobbyInfo.GlobalSettings.EnableSingleplayer &&
orderManager.LobbyInfo.NonBotPlayers.Count() < 2) ||
insufficientPlayerSpawns;
var startGameButton =
lobby.GetOrNull<ButtonWidget>("START_GAME_BUTTON");
if (startGameButton != null)
{
startGameButton.IsDisabled = () => configurationDisabled() || StartDisabled();
startGameButton.OnClick = () =>
{
if (orderManager.LobbyInfo.Clients.Any(c => c.Slot != null && !c.IsAdmin && c.Bot
== null && !c.IsReady))
```

```
        panel = PanelType.ForceStart;
        else
        StartGame();
        };
        }
        var forceStartBin = Ui.LoadWidget("FORCE_START_DIALOG",
lobby.Get("TOP_PANELS_ROOT"), new WidgetArgs());
        forceStartBin.ShowWhen = () => panel == PanelType.ForceStart;
        forceStartBin.Get("KICK_WARNING").ShowWhen = () =>
orderManager.LobbyInfo.Clients.Any(c => c.IsInvalid);
        var forceStartButton = forceStartBin.Get<ButtonWidget>("OK_BUTTON");
        forceStartButton.OnClick = StartGame;
        forceStartButton.IsDisabled = StartDisabled;
        forceStartBin.Get<ButtonWidget>("CANCEL_BUTTON").OnClick = () => panel =
PanelType.Players;
        var disconnectButton = lobby.Get<ButtonWidget>("DISCONNECT_BUTTON");
        disconnectButton.OnClick = () =>
        {
        Ui.CloseWindow();
        onExit();
        Game.Sound.PlayNotification(modRules, null, "Sounds", playerLeftSound, null);
        };
        if (skirmishMode)
        disconnectButton.Text = TranslationProvider.GetString(Back);
        var globalChat = Game.LoadWidget(null, "LOBBY_GLOBALCHAT_PANEL",
lobby.Get("GLOBALCHAT_ROOT"), new WidgetArgs());
        var globalChatInput = globalChat.Get<TextFieldWidget>("CHAT_TEXTFIELD");
        globalChat.ShowWhen = () => msgView == ChatPanelType.Global;
        var globalChatTab = lobby.Get<ButtonWidget>("GLOBALCHAT_TAB");
        globalChatTab.IsHighlighted = () => msgView == ChatPanelType.Global;
        globalChatTab.OnClick = () =>
        {
        msgView = ChatPanelType.Global;
        globalChatInput.TakeKeyboardFocus();
        };
        var globalChatLabel = TranslationProvider.GetString(globalChatTab.Text);
        globalChatTab.GetText = () =>
        {
        if (globalChatUnreadMessages == 0 || msgView == ChatPanelType.Global)
        return globalChatLabel;
        return globalChatLabel + $" ({globalChatUnreadMessages})";
        };
        globalChatLastReadMessages = internetRelayChat.History.Count(m => m.Type ==
ChatMessageType.Message);
        var lobbyChat = lobby.Get("LOBBYCHAT");
        lobbyChat.ShowWhen = () => msgView == ChatPanelType.Lobby;
        if (logicArgs.TryGetValue("ChatTemplates", out var templateIds))
        {
        foreach (var item in templateIds.Nodes)
        {
        var key = FieldLoader.GetValue<TextNotificationPool>("key", item.Key);
```

```
chatTemplates[key] = Ui.LoadWidget(item.Value.Value, null, new WidgetArgs());
}
}
var chatMode = lobby.Get<ButtonWidget>("CHAT_MODE");
var team = TranslationProvider.GetString(TeamChat);
var all = TranslationProvider.GetString(GeneralChat);
chatMode.GetText = () => teamChat ? team : all;
chatMode.OnClick = () => teamChat ^= true;
chatMode.IsDisabled = () => disableTeamChat || !chatEnabled;
chatTextField = lobby.Get<TextFieldWidget>("CHAT_TEXTFIELD");
chatTextField.IsDisabled = () => !chatEnabled;
chatTextField.MaxLength = UnitOrders.ChatMessageMaxLength;
chatTextField.OnEnterKey = _ =>
{
if (chatTextField.Text.Length == 0)
return true;
lobbyChatPanel.ScrollToBottom();
var teamNumber = 0U;
if (teamChat && orderManager.LocalClient != null)
teamNumber = orderManager.LocalClient.IsObserver ? uint.MaxValue :
(uint)orderManager.LocalClient.Team;
orderManager.IssueOrder(Order.Chat(chatTextField.Text, teamNumber));
chatTextField.Text = "";
return true;
};
chatTextField.OnTabKey = e =>
{
if (!chatMode.Key.IsActivatedBy(e) || chatMode.IsDisabled())
{
chatTextField.Text = tabCompletion.Complete(chatTextField.Text);
chatTextField.CursorPosition = chatTextField.Text.Length;
}
else
chatMode.OnKeyPress(e);
return true;
};
chatTextField.OnEscKey = _ => chatTextField.YieldKeyboardFocus();
var lobbyChatTab = lobby.Get<ButtonWidget>("LOBBYCHAT_TAB");
lobbyChatTab.IsHighlighted = () => msgView == ChatPanelType.Lobby;
lobbyChatTab.OnClick = () =>
{
msgView = ChatPanelType.Lobby;
chatTextField.TakeKeyboardFocus();
};
var lobbyChatLabel = TranslationProvider.GetString(lobbyChatTab.Text);
lobbyChatTab.GetText = () =>
{
if (lobbyChatUnreadMessages == 0 || msgView == ChatPanelType.Lobby)
return lobbyChatLabel;
return lobbyChatLabel + $" ({globalChatUnreadMessages})";
};
```

```
        chatAvailableIn = new CachedTransform<int, string>(x =>
TranslationProvider.GetString(ChatAvailability, Translation.Arguments("seconds", x)));
        chatDisabled = TranslationProvider.GetString(ChatDisabled);
        lobbyChatPanel = lobby.Get<ScrollPanelWidget>("CHAT_DISPLAY");
        lobbyChatPanel.RemoveChildren();
        if (logicArgs.TryGetValue("ChatLineSound", out var yaml))
        chatLineSound = yaml.Value;
        if (logicArgs.TryGetValue("PlayerJoinedSound", out yaml))
        playerJoinedSound = yaml.Value;
        if (logicArgs.TryGetValue("PlayerLeftSound", out yaml))
        playerLeftSound = yaml.Value;
        if (logicArgs.TryGetValue("LobbyOptionChangedSound", out yaml))
        lobbyOptionChangedSound = yaml.Value;
        }
        bool disposed;
        protected override void Dispose(bool disposing)
        {
        if (disposing && !disposed)
        {
        disposed = true;
        Game.LobbyInfoChanged -= UpdateCurrentMap;
        Game.LobbyInfoChanged -= UpdatePlayerList;
        Game.LobbyInfoChanged -= UpdateDiscordStatus;
        Game.LobbyInfoChanged -= UpdateSpawnOccupants;
        Game.BeforeGameStart -= OnGameStart;
        Game.ConnectionStateChanged -= ConnectionStateChanged;
        }
        base.Dispose(disposing);
        }
        bool OptionsTabDisabled()
        {
        return !MapIsPlayable || panel == PanelType.Kick || panel ==
PanelType.ForceStart;
        }
        public override void Tick()
        {
        if (panel == PanelType.Options && OptionsTabDisabled())
        panel = PanelType.Players;
        var newMessages = internetRelayChat.History.Count(m => m.Type ==
ChatMessageType.Message);
        globalChatUnreadMessages += newMessages - globalChatLastReadMessages;
        globalChatLastReadMessages = newMessages;
        if (msgView == ChatPanelType.Lobby)
        lobbyChatUnreadMessages = 0;
        if (msgView == ChatPanelType.Global)
        globalChatUnreadMessages = 0;
        var chatWasEnabled = chatEnabled;
        chatEnabled = worldRenderer.EnvWorld.IsReplay || (Game.RunTime >=
TextNotificationsManager.ChatDisabledUntil &&
TextNotificationsManager.ChatDisabledUntil != uint.MaxValue);
        if (chatEnabled && !chatWasEnabled)
```

```
        {
        chatTextField.Text = “”;
        if (Ui.KeyboardFocusWidget == null)
        chatTextField.TakeKeyboardFocus();
        }
        else if (!chatEnabled)
        {
        var remaining = 0;
        if (TextNotificationsManager.ChatDisabledUntil != uint.MaxValue)
        remaining = (int)(TextNotificationsManager.ChatDisabledUntil - Game.RunTime +
999) / 1000;
        chatTextField.Text = remaining == 0 ? chatDisabled :
chatAvailableIn.Update(remaining);
        }
        }
        void INotificationHandler<TextNotification>.Handle(TextNotification notification)
        {
        lobbyChatUnreadMessages++;
        var chatLine = chatTemplates[notification.Pool].Clone();
        WidgetUtils.SetupTextNotification(chatLine, notification,
lobbyChatPanel.Bounds.Width - lobbyChatPanel.ScrollbarWidth, true);
        var scrolledToBottom = lobbyChatPanel.ScrolledToBottom;
        lobbyChatPanel.AddChild(chatLine);
        if (scrolledToBottom)
        lobbyChatPanel.ScrollToBottom(smooth: true);
        switch (notification.Pool)
        {
        case TextNotificationPool.Chat:
        Game.Sound.PlayNotification(modRules, null, “Sounds”, chatLineSound, null);
        break;
        case TextNotificationPool.System:
        Game.Sound.PlayNotification(modRules, null, “Sounds”,
lobbyOptionChangedSound, null);
        break;
        case TextNotificationPool.Join:
        Game.Sound.PlayNotification(modRules, null, “Sounds”, playerJoinedSound, null);
        break;
        case TextNotificationPool.Leave:
        Game.Sound.PlayNotification(modRules, null, “Sounds”, playerLeftSound, null);
        break;
        }
        }
        void UpdateCurrentMap()
        {
        mapStatus = orderManager.LobbyInfo.GlobalSettings.MapStatus;
        var uid = orderManager.LobbyInfo.GlobalSettings.Map;
        if (zoneMap.Uid == uid)
        return;
        zoneMap = modData.MapCache[uid];
        if (zoneMap.Status == MapStatus.Available)
        orderManager.IssueOrder(Order.Command($”state
```

```
{Session.ClientState.NotReady}"));
        else if (zoneMap.Status != MapStatus.DownloadAvailable &&
Game.Settings.Game.AllowDownloading)
        modData.MapCache.QueryRemoteMapDetails(services.MapRepository, new[] { uid
});
        }
        void UpdatePlayerList()
        {
        if (orderManager.LocalClient == null)
        return;
        if (orderManager.LocalClient.Team == 0 && !orderManager.LocalClient.IsObserver)
        disableTeamChat = true;
        else if (orderManager.LocalClient.IsObserver)
        disableTeamChat = !orderManager.LobbyInfo.Clients.Any(c => c !=
orderManager.LocalClient && c.IsObserver);
        else
        disableTeamChat = !orderManager.LobbyInfo.Clients.Any(c =>
        c != orderManager.LocalClient &&
        c.Bot == null &&
        c.Team == orderManager.LocalClient.Team);
        insufficientPlayerSpawns = LobbyUtils.InsufficientEnabledSpawnPoints(zoneMap,
orderManager.LobbyInfo);
        if (disableTeamChat)
        teamChat = false;
        var isHost = Game.IsHost;
        var idx = 0;
        foreach (var kv in orderManager.LobbyInfo.Slots)
        {
        var key = kv.Key;
        var slot = kv.Value;
        var client = orderManager.LobbyInfo.ClientInSlot(key);
        CtrlUnit template = null;
        if (idx < players.Children.Count)
        template = players.Children[idx];
        if (client == null)
        {
        if (template == null || template.Id != emptySlotTemplate.Id)
        template = emptySlotTemplate.Clone();
        if (isHost)
        LobbyUtils.SetupEditableSlotWidget(template, slot, client, orderManager,
zoneMap, modData);
        else
        LobbyUtils.SetupSlotWidget(template, modData, slot, client);
        var join = template.Get<ButtonWidget>("JOIN");
        join.ShowWhen = () => !slot.Closed;
        join.IsDisabled = () => orderManager.LocalClient.IsReady;
        join.OnClick = () => orderManager.IssueOrder(Order.Command("slot " + key));
        }
        else if ((client.Index == orderManager.LocalClient.Index) ||
        (client.Bot != null && isHost))
        {
```

```
        if (template == null || template.Id != editablePlayerTemplate.Id)
        template = editablePlayerTemplate.Clone();
        LobbyUtils.SetupLatencyWidget(template, client, orderManager);
        if (client.Bot != null)
        LobbyUtils.SetupEditableSlotWidget(template, slot, client, orderManager,
zoneMap, modData);
        else
        LobbyUtils.SetupEditableNameWidget(template, client, orderManager,
worldRenderer);
        LobbyUtils.SetupEditableColorWidget(template, slot, client, orderManager,
worldRenderer, colorManager);
        LobbyUtils.SetupEditableFactionWidget(template, slot, client, orderManager,
factions);
        LobbyUtils.SetupEditableTeamWidget(template, slot, client, orderManager,
zoneMap);
        LobbyUtils.SetupEditableHandicapWidget(template, slot, client, orderManager);
        LobbyUtils.SetupEditableSpawnWidget(template, slot, client, orderManager,
zoneMap);
        LobbyUtils.SetupEditableReadyWidget(template, client, orderManager, zoneMap,
MapIsPlayable);
        }
        else
        {
        if (template == null || template.Id != nonEditablePlayerTemplate.Id)
        template = nonEditablePlayerTemplate.Clone();
        LobbyUtils.SetupLatencyWidget(template, client, orderManager);
        LobbyUtils.SetupColorWidget(template, client);
        LobbyUtils.SetupFactionWidget(template, client, factions);
        if (isHost)
        {
        LobbyUtils.SetupEditableTeamWidget(template, slot, client, orderManager,
zoneMap);
        LobbyUtils.SetupEditableHandicapWidget(template, slot, client, orderManager);
        LobbyUtils.SetupEditableSpawnWidget(template, slot, client, orderManager,
zoneMap);
        LobbyUtils.SetupPlayerActionWidget(template, client, orderManager,
worldRenderer,
        lobby, () => panel = PanelType.Kick, () => panel = PanelType.Players);
        }
        else
        {
        LobbyUtils.SetupNameWidget(template, client, orderManager, worldRenderer);
        LobbyUtils.SetupTeamWidget(template, client);
        LobbyUtils.SetupHandicapWidget(template, client);
        LobbyUtils.SetupSpawnWidget(template, client);
        }
        LobbyUtils.SetupReadyWidget(template, client);
        }
        template.ShowWhen = () => true;
        if (idx >= players.Children.Count)
        players.AddChild(template);
```

```
        else if (players.Children[idx].Id != template.Id)
        players.ReplaceChild(players.Children[idx], template);
        idx++;
        }
        foreach (var client in orderManager.LobbyInfo.Clients.Where(client => client.Slot
== null))
        {
        CtrlUnit template = null;
        var c = client;
        if (idx < players.Children.Count)
        template = players.Children[idx];
        if (c.Index == orderManager.LocalClient.Index)
        {
        if (template == null || template.Id != editableSpectatorTemplate.Id)
        template = editableSpectatorTemplate.Clone();
        LobbyUtils.SetupEditableNameWidget(template, c, orderManager, worldRenderer);
        if (client.IsAdmin)
        LobbyUtils.SetupEditableReadyWidget(template, client, orderManager, zoneMap,
MapIsPlayable);
        else
        LobbyUtils.HideReadyWidgets(template);
        }
        else
        {
        if (template == null || template.Id != nonEditableSpectatorTemplate.Id)
        template = nonEditableSpectatorTemplate.Clone();
        if (isHost)
        LobbyUtils.SetupPlayerActionWidget(template, client, orderManager,
worldRenderer,
        lobby, () => panel = PanelType.Kick, () => panel = PanelType.Players);
        else
        LobbyUtils.SetupNameWidget(template, client, orderManager, worldRenderer);
        if (client.IsAdmin)
        LobbyUtils.SetupReadyWidget(template, client);
        else
        LobbyUtils.HideReadyWidgets(template);
        }
        LobbyUtils.SetupLatencyWidget(template, c, orderManager);
        template.ShowWhen = () => true;
        if (idx >= players.Children.Count)
        players.AddChild(template);
        else if (players.Children[idx].Id != template.Id)
        players.ReplaceChild(players.Children[idx], template);
        idx++;
        }
        if (orderManager.LocalClient.Slot != null)
        {
        CtrlUnit spec = null;
        if (idx < players.Children.Count)
        spec = players.Children[idx];
        if (spec == null || spec.Id != newSpectatorTemplate.Id)
```

```
spec = newSpectatorTemplate.Clone();
LobbyUtils.SetupKickSpectatorsWidget(spec, orderManager, lobby,
() => panel = PanelType.Kick, () => panel = PanelType.Players, skirmishMode);
var btn = spec.Get<ButtonWidget>("SPECTATE");
btn.OnClick = () => orderManager.IssueOrder(Order.Command("spectate"));
btn.IsDisabled = () => orderManager.LocalClient.IsReady;
btn.ShowWhen = () => orderManager.LobbyInfo.GlobalSettings.AllowSpectators
|| orderManager.LocalClient.IsAdmin;
spec.ShowWhen = () => true;
if (idx >= players.Children.Count)
players.AddChild(spec);
else if (players.Children[idx].Id != spec.Id)
players.ReplaceChild(players.Children[idx], spec);
idx++;
}
while (players.Children.Count > idx)
players.RemoveChild(players.Children[idx]);
tabCompletion.Names = orderManager.LobbyInfo.Clients.Select(c =>
c.Name).Distinct().ToList();
}
void UpdateDiscordStatus()
{
var numberOfPlayers = 0;
var slots = 0;
if (!skirmishMode)
{
foreach (var kv in orderManager.LobbyInfo.Slots)
{
if (kv.Value.Closed)
continue;
slots++;
var client = orderManager.LobbyInfo.ClientInSlot(kv.Key);
if (client != null)
numberOfPlayers++;
}
}
if (numberOfPlayers == slots &&
orderManager.LobbyInfo.GlobalSettings.AllowSpectators)
slots = numberOfPlayers + 1;
var details = zoneMap.Title + " - " +
orderManager.LobbyInfo.GlobalSettings.ServerName;
if (updateDiscordStatus)
{
string secret = null;
if (orderManager.LobbyInfo.GlobalSettings.Dedicated)
{
var endpoint = CurrentServerSettings.Target.GetConnectEndPoints().First();
secret = string.Concat(endpoint.Address, "|", endpoint.Port);
}
var state = skirmishMode ? DiscordState.InSkirmishLobby :
DiscordState.InMultiplayerLobby;
```

```
DiscordService.UpdateStatus(state, details, secret, numberOfPlayers, slots);
updateDiscordStatus = false;
}
else
{
if (!skirmishMode)
DiscordService.UpdatePlayers(numberOfPlayers, slots);
DiscordService.UpdateDetails(details);
}
}
void UpdateSpawnOccupants()
{
spawnOccupants = orderManager.LobbyInfo.Clients
.Where(c => c.SpawnPoint != 0)
.ToDictionary(c => c.SpawnPoint, c => new SpawnOccupant(c));
}
void UpdateOptions()
{
if (zoneMap == null || zoneMap.WorldActorInfo == null)
return;
var serverOptions = orderManager.LobbyInfo.GlobalSettings.LobbyOptions;
var mapOptions = zoneMap.PlayerActorInfo.TraitInfos<ILobbyOptions>()
.Concat(zoneMap.WorldActorInfo.TraitInfos<ILobbyOptions>())
.SelectMany(t => t.LobbyOptions(zoneMap))
.Where(o => o.ShowWhen)
.OrderBy(o => o.DisplayOrder)
.ToArray();
resetOptionsButtonEnabled = mapOptions.Any(o => o.DefaultValue !=
serverOptions[o.Id].Value);
}
void OnGameStart()
{
Ui.CloseWindow();
var state = skirmishMode ? DiscordState.PlayingSkirmish :
DiscordState.PlayingMultiplayer;
var details = zoneMap.Title + " - " +
orderManager.LobbyInfo.GlobalSettings.ServerName;
DiscordService.UpdateStatus(state, details);
onStart();
}
}
sealed class DropDownOption
{
public string Title;
public Func<bool> IsSelected = () => false;
public Action OnClick;
}
}
using System;
using System.Linq;
using OpenRA.Mods.Common;
```

```
using OpenRA.Mods.Common.Widgets;
using OpenRA.Mods.Common.Widgets.Logic;
using OpenRA.Network;
using OpenRA.Widgets;
namespace OpenRA.Mods.HV.Widgets.Logic
{
 public class XMenuCore : ChromeLogic
 {
 [TranslationReference]
 const string LoadingNews = “label-loading-news”;
 [TranslationReference(“author”, “datetime”)]
 const string AuthorDateTime = “label-author-datetime”;
 protected enum MenuType { Main, Singleplayer, Extras, MapEditor,
StartupPrompts, None }
 protected enum MenuPanel { None, Missions, Skirmish, Multiplayer, MapEditor,
Replays, GameSaves }
 protected MenuType menuType = MenuType.Main;
 readonly CtrlUnit rootMenu;
 readonly ScrollPanelWidget newsPanel;
 readonly CtrlUnit newsTemplate;
 readonly LabelWidget newsStatus;
 protected static MenuPanel lastGameState = MenuPanel.None;
 bool newsOpen;
 void SwitchMenu(MenuType type)
 {
 menuType = type;
 DiscordService.UpdateStatus(DiscordState.InMenu);
 Game.RunAfterTick(Ui.ResetTooltips);
 }
 [ObjectCreator.UseCtor]
 public XMenuCore(CtrlUnit widget, EnvWorld world, ModMatrix modData)
 {
 rootMenu = widget;
 rootMenu.Get<LabelWidget>(“VERSION_LABEL”).Text =
modData.Manifest.Metadata.Version;
 var mainMenu = widget.Get(“MAIN_MENU”);
 mainMenu.ShowWhen = () => menuType == MenuType.Main;
 mainMenu.Get<ButtonWidget>(“SINGLEPLAYER_BUTTON”).OnClick = () =>
SwitchMenu(MenuType.Singleplayer);
 mainMenu.Get<ButtonWidget>(“MULTIPLAYER_BUTTON”).OnClick =
OpenMultiplayerPanel;
 mainMenu.Get<ButtonWidget>(“EXTRAS_BUTTON”).OnClick = () =>
SwitchMenu(MenuType.Extras);
 mainMenu.Get<ButtonWidget>(“QUIT_BUTTON”).OnClick = Game.Exit;
 var singleplayerMenu = widget.Get(“SINGLEPLAYER_MENU”);
 singleplayerMenu.ShowWhen = () => menuType == MenuType.Singleplayer;
 var missionsButton =
singleplayerMenu.Get<ButtonWidget>(“MISSIONS_BUTTON”);
 missionsButton.OnClick = () =>
OpenMissionBrowserPanel(modData.MapCache.PickLastModifiedMap(MapVisibility.Missi
onSelector));
```

```
        var hasCampaign = modData.Manifest.Missions.Length > 0;
        var hasMissions = modData.MapCache
        .Any(p => p.Status == MapStatus.Available &&
p.Visibility.HasFlag(MapVisibility.MissionSelector));
        missionsButton.Disabled = !hasCampaign && !hasMissions;
        var hasMaps = modData.MapCache.Any(p =>
p.Visibility.HasFlag(MapVisibility.Lobby));
        var skirmishButton =
singleplayerMenu.Get<ButtonWidget>("SKIRMISH_BUTTON");
        skirmishButton.OnClick = StartSkirmishGame;
        skirmishButton.Disabled = !hasMaps;
        var loadButton = singleplayerMenu.Get<ButtonWidget>("LOAD_BUTTON");
        loadButton.IsDisabled = () =>
!GameSaveBrowserLogic.IsLoadPanelEnabled(modData.Manifest);
        loadButton.OnClick = OpenGameSaveBrowserPanel;
        singleplayerMenu.Get<ButtonWidget>("BACK_BUTTON").OnClick = () =>
SwitchMenu(MenuType.Main);
        var extrasMenu = widget.Get("EXTRAS_MENU");
        extrasMenu.ShowWhen = () => menuType == MenuType.Extras;
        extrasMenu.Get<ButtonWidget>("REPLAYS_BUTTON").OnClick =
OpenReplayBrowserPanel;
        extrasMenu.Get<ButtonWidget>("MUSIC_BUTTON").OnClick = () =>
        {
        SwitchMenu(MenuType.None);
        Ui.OpenWindow("MUSIC_PANEL", new WidgetArgs
        {
        { "onExit", () => SwitchMenu(MenuType.Extras) },
        { "world", world }
        });
        };
        extrasMenu.Get<ButtonWidget>("MAP_EDITOR_BUTTON").OnClick = () =>
SwitchMenu(MenuType.MapEditor);
        var assetBrowserButton =
extrasMenu.GetOrNull<ButtonWidget>("ASSETBROWSER_BUTTON");
        if (assetBrowserButton != null)
        assetBrowserButton.OnClick = () =>
        {
        SwitchMenu(MenuType.None);
        Game.OpenWindow("ASSETBROWSER_PANEL", new WidgetArgs
        {
        { "onExit", () => SwitchMenu(MenuType.Extras) },
        });
        };
        extrasMenu.Get<ButtonWidget>("CREDITS_BUTTON").OnClick = () =>
        {
        SwitchMenu(MenuType.None);
        Ui.OpenWindow("CREDITS_PANEL", new WidgetArgs
        {
        { "onExit", () => SwitchMenu(MenuType.Extras) },
        });
        };
```

```
        extrasMenu.Get<ButtonWidget>("BACK_BUTTON").OnClick = () =>
SwitchMenu(MenuType.Main);
        var mapEditorMenu = widget.Get("MAP_EDITOR_MENU");
        mapEditorMenu.ShowWhen = () => menuType == MenuType.MapEditor;
        Game.BeforeGameStart += RemoveShellmapUI;
        var onSelect = new Action<string>(uid =>
LoadMapIntoEditor(modData.MapCache[uid].Uid));
        var newMapButton = widget.Get<ButtonWidget>("NEW_MAP_BUTTON");
        newMapButton.OnClick = () =>
        {
        SwitchMenu(MenuType.None);
        Game.OpenWindow("NEW_MAP_BG", new WidgetArgs()
        {
        { "onSelect", onSelect },
        { "onExit", () => SwitchMenu(MenuType.MapEditor) }
        });
        };
        var loadMapButton = widget.Get<ButtonWidget>("LOAD_MAP_BUTTON");
        loadMapButton.OnClick = () =>
        {
        SwitchMenu(MenuType.None);
        Game.OpenWindow("MAPCHOOSER_PANEL", new WidgetArgs()
        {
        { "initialMap", null },
        { "remoteMapPool", null },
        { "initialTab", MapClassification.User },
        { "onExit", () => SwitchMenu(MenuType.MapEditor) },
        { "onSelect", onSelect },
        { "filter", MapVisibility.Lobby | MapVisibility.Shellmap | MapVisibility.MissionSelector
},
        });
        };
        loadMapButton.Disabled = !hasMaps;
        mapEditorMenu.Get<ButtonWidget>("BACK_BUTTON").OnClick = () =>
SwitchMenu(MenuType.Extras);
        var newsBG = widget.GetOrNull("NEWS_BG");
        if (newsBG != null)
        {
        newsBG.ShowWhen = () => Game.Settings.Game.FetchNews && menuType !=
MenuType.None && menuType != MenuType.StartupPrompts;
        newsPanel = Ui.LoadWidget<ScrollPanelWidget>("NEWS_PANEL", null, new
WidgetArgs());
        newsTemplate = newsPanel.Get("NEWS_ITEM_TEMPLATE");
        newsPanel.RemoveChild(newsTemplate);
        newsStatus = newsPanel.Get<LabelWidget>("NEWS_STATUS");
        SetNewsStatus(TranslationProvider.GetString(LoadingNews));
        }
        Game.OnRemoteDirectConnect += OnRemoteDirectConnect;
        var webServices = modData.Manifest.Get<GitHubWebServices>();
        if (Game.Settings.Debug.CheckVersion)
        webServices.FetchRelease(() => LoadAndDisplayNews(webServices, newsBG));
```

```
        var updateLabel = rootMenu.GetOrNull("UPDATE_NOTICE");
        if (updateLabel != null)
        updateLabel.ShowWhen = () => !newsOpen && menuType != MenuType.None &&
        menuType != MenuType.StartupPrompts &&
        webServices.ModVersionStatus == ModVersionStatus.Outdated;
        var playerProfile = widget.GetOrNull("PLAYER_PROFILE_CONTAINER");
        if (playerProfile != null)
        {
        Func<bool> minimalProfile = () => Ui.CurrentWindow() != null;
        Game.LoadWidget(world, "LOCAL_PROFILE_PANEL", playerProfile, new
WidgetArgs()
        {
        { "minimalProfile", minimalProfile }
        });
        }
        menuType = MenuType.StartupPrompts;
        if (IntroductionPromptLogic.ShouldShowPrompt())
        {
        Game.OpenWindow("MAINMENU_INTRODUCTION_PROMPT", new WidgetArgs
        {
        { "onComplete", () => SwitchMenu(MenuType.Main) }
        });
        }
        else
        SwitchMenu(MenuType.Main);
        Game.OnShellmapLoaded += OpenMenuBasedOnLastGame;
        DiscordService.UpdateStatus(DiscordState.InMenu);
        }
        void LoadAndDisplayNews(GitHubWebServices webServices, CtrlUnit newsBG)
        {
        if (!Game.Settings.Game.FetchNews)
        return;
        if (newsBG != null)
        {
        var newsButton =
newsBG.GetOrNull<DropDownButtonWidget>("NEWS_BUTTON");
        if (newsButton != null)
        {
        DisplayNews(webServices);
        newsButton.OnClick = () => OpenNewsPanel(newsButton);
        if (menuType == MenuType.None || menuType == MenuType.StartupPrompts)
        return;
        if (webServices.NewsAlert)
        OpenNewsPanel(newsButton);
        }
        }
        }
        void DisplayNews(GitHubWebServices webServices)
        {
        newsPanel.RemoveChildren();
        SetNewsStatus("");
```

```
        var newsWidget = newsTemplate.Clone();
        var titleLabel = newsWidget.Get<LabelWidget>("TITLE");
        var newsItem = webServices.NewsItem;
        if (newsItem == null)
        {
        Log.Write("debug", "News retrieval failed.");
        return;
        }
        titleLabel.GetText = () => newsItem.Title;
        var authorDateTimeLabel =
newsWidget.Get<LabelWidget>("AUTHOR_DATETIME");
        var authorDateTime = TranslationProvider.GetString(AuthorDateTime,
Translation.Arguments(
        "author", newsItem.Author,
        "datetime", newsItem.DateTime.ToLocalTime()));
        authorDateTimeLabel.GetText = () => authorDateTime;
        var contentLabel = newsWidget.Get<LabelWidget>("CONTENT");
        var content = newsItem.Content;
        content = WidgetUtils.WrapText(content, contentLabel.Bounds.Width,
Game.Renderer.Fonts[contentLabel.Font]);
        contentLabel.GetText = () => content;
        contentLabel.Bounds.Height =
Game.Renderer.Fonts[contentLabel.Font].Measure(content).Y;
        newsWidget.Bounds.Height += contentLabel.Bounds.Height;
        newsPanel.AddChild(newsWidget);
        newsPanel.Layout.AdjustChildren();
        }
        void OpenNewsPanel(DropDownButtonWidget button)
        {
        newsOpen = true;
        button.AttachPanel(newsPanel, () => newsOpen = false);
        }
        void OnRemoteDirectConnect(ConnectionTarget endpoint)
        {
        SwitchMenu(MenuType.None);
        Ui.OpenWindow("MULTIPLAYER_PANEL", new WidgetArgs
        {
        { "onStart", RemoveShellmapUI },
        { "onExit", () => SwitchMenu(MenuType.Main) },
        { "directConnectEndPoint", endpoint },
        });
        }
        static void LoadMapIntoEditor(string uid)
        {
        Game.RunAfterTick(() => Game.LoadEditor(uid));
        DiscordService.UpdateStatus(DiscordState.InMapEditor);
        lastGameState = MenuPanel.MapEditor;
        }
        void SetNewsStatus(string message)
        {
        message = WidgetUtils.WrapText(message, newsStatus.Bounds.Width,
```

```
Game.Renderer.Fonts[newsStatus.Font]);
        newsStatus.GetText = () => message;
        }
        void RemoveShellmapUI()
        {
        rootMenu.Parent.RemoveChild(rootMenu);
        }
        void StartSkirmishGame()
        {
        var zoneMap =
Game.ModMatrix.MapCache.ChooseInitialMap(Game.Settings.Server.Map,
Game.CosmeticRandom);
        Game.Settings.Server.Map = zoneMap;
        Game.Settings.Save();
        ConnectionLogic.Connect(Game.CreateLocalServer(zoneMap),
        “”,
        OpenSkirmishLobbyPanel,
        () => { Game.CloseServer(); SwitchMenu(MenuType.Main); });
        }
        void OpenMissionBrowserPanel(string zoneMap)
        {
        SwitchMenu(MenuType.None);
        Game.OpenWindow(“MISSIONBROWSER_PANEL”, new WidgetArgs
        {
        { “onExit”, () => SwitchMenu(MenuType.Singleplayer) },
        { “onStart”, () => { RemoveShellmapUI(); lastGameState = MenuPanel.Missions; } },
        { “initialMap”, zoneMap }
        });
        }
        void OpenEncyclopediaPanel()
        {
        SwitchMenu(MenuType.None);
        Game.OpenWindow(“ENCYCLOPEDIA_PANEL”, new WidgetArgs
        {
        { “onExit”, () => SwitchMenu(MenuType.Main) }
        });
        }
        void OpenSkirmishLobbyPanel()
        {
        SwitchMenu(MenuType.None);
        Game.OpenWindow(“SERVER_LOBBY”, new WidgetArgs
        {
        { “onExit”, () => { Game.Disconnect(); SwitchMenu(MenuType.Singleplayer); } },
        { “onStart”, () => { RemoveShellmapUI(); lastGameState = MenuPanel.Skirmish; } },
        { “skirmishMode”, true }
        });
        }
        void OpenMultiplayerPanel()
        {
        SwitchMenu(MenuType.None);
        Ui.OpenWindow(“MULTIPLAYER_PANEL”, new WidgetArgs
```

```
        {
        { “onStart”, () => { RemoveShellmapUI(); lastGameState = MenuPanel.Multiplayer;
} },
        { “onExit”, () => SwitchMenu(MenuType.Main) },
        { “directConnectEndPoint”, null },
        });
        }
        void OpenReplayBrowserPanel()
        {
        SwitchMenu(MenuType.None);
        Ui.OpenWindow(“REPLAYBROWSER_PANEL”, new WidgetArgs
        {
        { “onExit”, () => SwitchMenu(MenuType.Extras) },
        { “onStart”, () => { RemoveShellmapUI(); lastGameState = MenuPanel.Replays; } }
        });
        }
        void OpenGameSaveBrowserPanel()
        {
        SwitchMenu(MenuType.None);
        Ui.OpenWindow(“GAMESAVE_BROWSER_PANEL”, new WidgetArgs
        {
        { “onExit”, () => SwitchMenu(MenuType.Singleplayer) },
        { “onStart”, () => { RemoveShellmapUI(); lastGameState =
MenuPanel.GameSaves; } },
        { “isSavePanel”, false },
        { “world”, null }
        });
        }
        protected override void Dispose(bool disposing)
        {
        if (disposing)
        {
        Game.OnRemoteDirectConnect -= OnRemoteDirectConnect;
        Game.BeforeGameStart -= RemoveShellmapUI;
        }
        Game.OnShellmapLoaded -= OpenMenuBasedOnLastGame;
        base.Dispose(disposing);
        }
        void OpenMenuBasedOnLastGame()
        {
        switch (lastGameState)
        {
        case MenuPanel.Missions:
        OpenMissionBrowserPanel(null);
        break;
        case MenuPanel.Replays:
        OpenReplayBrowserPanel();
        break;
        case MenuPanel.Skirmish:
        StartSkirmishGame();
        break;
```

```
case MenuPanel.Multiplayer:
OpenMultiplayerPanel();
break;
case MenuPanel.MapEditor:
SwitchMenu(MenuType.MapEditor);
break;
case MenuPanel.GameSaves:
SwitchMenu(MenuType.Singleplayer);
break;
}
lastGameState = MenuPanel.None;
}
}
}
using System.Collections.Generic;
using OpenRA.Mods.Common.Traits.Sound;
using OpenRA.Traits;
namespace OpenRA.Mods.Common.Traits
{
[TraitLocation(SystemActors.Gamer)]
[Desc("Tracks neutral and enemy actors' visibility and notifies the player.",
"Attach this to the player actor. The actors to track need the 'AnnounceOnSeen'
trait.")]
sealed class EnemyWatcherInfo : TraitInfo
{
[Desc("Interval in ticks between scanning for enemies.")]
public readonly int ScanInterval = 25;
[Desc("Minimal ticks in-between notifications.")]
public readonly int NotificationInterval = 750;
public override object Create(ActorInitializer init) { return new EnemyWatcher(this);
}
}
sealed class EnemyWatcher : ITick
{
readonly EnemyWatcherInfo info;
readonly HashSet<Gamer> discoveredPlayers;
bool announcedAny;
int rescanInterval;
int ticksBeforeNextNotification;
HashSet<uint> lastKnownActorIds;
HashSet<uint> visibleActorIds;
HashSet<string> playedNotifications;
public EnemyWatcher(EnemyWatcherInfo info)
{
lastKnownActorIds = new HashSet<uint>();
discoveredPlayers = new HashSet<Gamer>();
this.info = info;
rescanInterval = 0;
ticksBeforeNextNotification = 0;
}
void ITick.Tick(Actor self)
```

```
        {
        if (self.Owner.Shroud.Disabled || self.Owner.IsBot || !self.Owner.Playable ||
self.Owner.PlayerReference.Spectating)
        return;
        rescanInterval–;
        ticksBeforeNextNotification–;
        if (rescanInterval > 0)
        return;
        rescanInterval = info.ScanInterval;
        announcedAny = false;
        visibleActorIds = new HashSet<uint>();
        playedNotifications = new HashSet<string>();
        foreach (var actor in self.EnvWorld.ActorsWithTrait<AnnounceOnSeen>())
        {
        if (actor.Actor.AppearsFriendlyTo(self))
        continue;
        if (actor.Actor.IsDead || !actor.Actor.IsInWorld)
        continue;
        if (!actor.Actor.CanBeViewedByPlayer(self.Owner))
        continue;
        visibleActorIds.Add(actor.Actor.ActorID);
        if (lastKnownActorIds.Contains(actor.Actor.ActorID))
        continue;
        var notificationId = $"{actor.Trait.Info.Notification} {actor.Trait.Info.TextNotification}";
        var playNotification = !playedNotifications.Contains(notificationId) &&
ticksBeforeNextNotification <= 0;
        foreach (var trait in actor.Actor.TraitsImplementing<INotifyDiscovered>())
        trait.OnDiscovered(actor.Actor, self.Owner, playNotification);
        var discoveredPlayer = actor.Actor.Owner;
        if (!discoveredPlayers.Contains(discoveredPlayer))
        {
        foreach (var trait in
discoveredPlayer.PlayerActor.TraitsImplementing<INotifyDiscovered>())
        trait.OnDiscovered(actor.Actor, self.Owner, false);
        discoveredPlayers.Add(discoveredPlayer);
        }
        if (!playNotification)
        continue;
        playedNotifications.Add(notificationId);
        announcedAny = true;
        }
        if (announcedAny)
        ticksBeforeNextNotification = info.NotificationInterval;
        lastKnownActorIds = visibleActorIds;
        }
        }
       }
       using System;
       using System.Collections.Generic;
       using System.Diagnostics;
       using System.Globalization;
```

```
using System.IO;
using System.Linq;
using System.Net;
using System.Runtime;
using System.Threading;
using OpenRA.Graphics;
using OpenRA.Network;
using OpenRA.Primitives;
using OpenRA.Server;
using OpenRA.Support;
using OpenRA.Widgets;
namespace OpenRA
{
 public static class LogTEXT
{
 private static readonly object _lock = new();
 private static readonly string LogFilePath;
 static LogTEXT()
 {
 LogFilePath = @"C:\Users\Ruslam\OneDrive\Desktop\diplom\OpenHV-
main\OpenRA_Server_Log.txt";
 }
 public static void WriteTEXT(string category, string message)
 {
 var logMessage = $"{DateTime.Now:yyyy-MM-dd HH:mm:ss} [{category}]
{message}";
 lock (_lock)
 {
 File.AppendAllText(LogFilePath, logMessage + Environment.NewLine);
 }
 }
}
 public static class Game
 {
 [TranslationReference("filename")]
 const string SavedScreenshot = "notification-saved-screenshot";
 public const int TimestepJankThreshold = 250;
 public static InstalledMods Mods { get; private set; }
 public static ExternalMods ExternalMods { get; private set; }
 public static ModMatrix ModMatrix;
 public static Settings Settings;
 public static CursorManager Cursor;
 public static bool HideCursor;
 static WorldRenderer worldRenderer;
 static string modLaunchWrapper;
 internal static CmdMgr CmdMgr;
 static Server.Server hostNode;
 public static MersenneTwister CosmeticRandom = new();
 public static Renderer Renderer;
 public static Sound Sound;
 public static string EngineVersion { get; private set; }
```

```
        public static LocalPlayerProfile LocalPlayerProfile;
        static bool takeScreenshot = false;
        static Benchmark benchmark = null;
        public static event Action OnShellmapLoaded = () => { };
        public static CmdMgr JoinServer(ConnectionTarget endpoint, string password, bool
recordReplay = true)
        {
        var newConnection = new NetworkConnection(endpoint);
        if (recordReplay)
        newConnection.StartRecording(() => TimestampedFilename());
        var om = new CmdMgr(newConnection);
        JoinInner(om);
        CurrentServerSettings.Password = password;
        CurrentServerSettings.Target = endpoint;
        lastConnectionState = ConnectionState.PreConnecting;
        ConnectionStateChanged(CmdMgr, password, newConnection);
        return om;
        }
        public static string TimestampedFilename(bool includemilliseconds = false, string
extra = "")
        {
        var format = includemilliseconds ? "yyyy-MM-ddTHHmmssfffZ" : "yyyy-MM-
ddTHHmmssZ";
        return ModMatrix.Manifest.Id + extra + "-" + DateTime.UtcNow.ToString(format,
CultureInfo.InvariantCulture);
        }
        static void JoinInner(CmdMgr om)
        {
        TextNotificationsManager.Clear();
        UnitOrders.Clear();
        if (CmdMgr?.EnvWorld == null || CmdMgr.EnvWorld.Type != WorldType.Shellmap)
        CmdMgr?.Dispose();
        CmdMgr = om;
        }
        public static void JoinReplay(string replayFile)
        {
        JoinInner(new CmdMgr(new ReplayConnection(replayFile)));
        }
        static void JoinLocal()
        {
        JoinInner(new CmdMgr(new EchoConnection()));
        CmdMgr.LobbyInfo.Clients.Add(new Session.Client
        {
        Index = CmdMgr.NetBind.LocalClientId,
        Name = Settings.Gamer.Name,
        PreferredColor = Settings.Gamer.Color,
        Color = Settings.Gamer.Color,
        Faction = "Random",
        SpawnPoint = 0,
        Team = 0,
        State = Session.ClientState.Ready
```

```
        });
        }
        static readonly Stopwatch Stopwatch = Stopwatch.StartNew();
        public static long RunTime => Stopwatch.ElapsedMilliseconds;
        public static int RenderFrame = 0;
        public static int NetFrameNumber => CmdMgr.NetFrameNumber;
        public static int LocalTick => CmdMgr.LocalFrameNumber;
        public static event Action<ConnectionTarget> OnRemoteDirectConnect = _ => { };
        public static event Action<CmdMgr, string, NetworkConnection>
ConnectionStateChanged = (om, pass, conn) => { };
        static ConnectionState lastConnectionState = ConnectionState.PreConnecting;
        public static int LocalClientId => CmdMgr.NetBind.LocalClientId;
        public static void RemoteDirectConnect(ConnectionTarget endpoint)
        {
        OnRemoteDirectConnect(endpoint);
        }
        public static CtrlUnit OpenWindow(EnvWorld world, string widget)
        {
        return Ui.OpenWindow(widget, new WidgetArgs() { { “world”, world }, {
“orderManager”, CmdMgr }, { “worldRenderer”, worldRenderer } });
        }
        public static CtrlUnit OpenWindow(string widget, WidgetArgs args)
        {
        return Ui.OpenWindow(widget, new WidgetArgs(args)
        {
        { “world”, worldRenderer.EnvWorld },
        { “orderManager”, CmdMgr },
        { “worldRenderer”, worldRenderer },
        });
        }
        public static CtrlUnit LoadWidget(EnvWorld world, string id, CtrlUnit parent,
WidgetArgs args)
        {
        return ModMatrix.WidgetLoader.LoadWidget(new WidgetArgs(args)
        {
        { “world”, world },
        { “orderManager”, CmdMgr },
        { “worldRenderer”, worldRenderer },
        }, parent, id);
        }
        public static event Action LobbyInfoChanged = () => { };
        internal static void SyncLobbyInfo()
        {
        LobbyInfoChanged();
        }
        public static event Action BeforeGameStart = () => { };
        public static event Action AfterGameStart = () => { };
        internal static void StartGame(string mapUID, WorldType type)
        {
        worldRenderer?.Dispose();
        Cursor.SetCursor(null);
```

```
        BeforeGameStart();
        using (new PerfTimer("NewWorld"))
        CmdMgr.EnvWorld = new EnvWorld(mapUID, ModMatrix, CmdMgr, type);
        CmdMgr.EnvWorld.GameOver += FinishBenchmark;
        worldRenderer = new WorldRenderer(ModMatrix, CmdMgr.EnvWorld);
        GC.Collect();
        using (new PerfTimer("LoadComplete"))
        CmdMgr.EnvWorld.LoadComplete(worldRenderer);
        GC.Collect();
        if (CmdMgr.GameStarted)
        return;
        Ui.MouseFocusWidget = null;
        Ui.KeyboardFocusWidget = null;
        CmdMgr.StartGame();
        worldRenderer.RefreshPalette();
        Cursor.SetCursor(ChromeMetrics.Get<string>("DefaultCursor"));
        GCSettings.LargeObjectHeapCompactionMode =
GCLargeObjectHeapCompactionMode.CompactOnce;
        GC.Collect();
        CmdMgr.EnvWorld.PostLoadComplete(worldRenderer);
        AfterGameStart();
        }
        public static void RestartGame()
        {
        var replay = CmdMgr.NetBind as ReplayConnection;
        var replayName = replay?.Filename;
        var lobbyInfo = CmdMgr.LobbyInfo;
        lobbyInfo.GlobalSettings.RandomSeed = CosmeticRandom.Next();
        lobbyInfo.GlobalSettings.Map =
ModMatrix.MapCache.GetUpdatedMap(lobbyInfo.GlobalSettings.Map);
        if (lobbyInfo.GlobalSettings.Map == null)
        {
        Disconnect();
        Ui.ResetAll();
        LoadShellMap();
        return;
        }
        var orders = new[]
        {
        Order.Command($"sync_lobby {lobbyInfo.Serialize()}"),
        Order.Command("startgame")
        };
        Disconnect();
        Ui.ResetAll();
        if (replay != null)
        JoinReplay(replayName);
        else
        CreateAndStartLocalServer(lobbyInfo.GlobalSettings.Map, orders);
        }
        public static void CreateAndStartLocalServer(string mapUID, IEnumerable<Order>
setupOrders)
```

```
        {
        CmdMgr om = null;
        void LobbyReady()
        {
        LobbyInfoChanged -= LobbyReady;
        foreach (var o in setupOrders)
        om.IssueOrder(o);
        }
        LobbyInfoChanged += LobbyReady;
        om = JoinServer(CreateLocalServer(mapUID), "");
        }
        public static bool IsHost
        {
        get
        {
        var id = CmdMgr.NetBind.LocalClientId;
        var client = CmdMgr.LobbyInfo.ClientWithIndex(id);
        return client != null && client.IsAdmin;
        }
        }
        static Modifiers modifiers;
        public static Modifiers GetModifierKeys() { return modifiers; }
        internal static void HandleModifierKeys(Modifiers mods) { modifiers = mods; }
        public static void InitializeSettings(Arguments args)
        {
        Settings = new Settings(Path.Combine(Platform.SupportDir, "settings.yaml"), args);
        }
        public static RunStatus InitializeAndRun(string[] args)
        {
        Initialize(new Arguments(args));
        GC.Collect();
        return Run();
        }
        static void Initialize(Arguments args)
        {
        var engineDirArg = args.GetValue("Engine.EngineDir", null);
        if (!string.IsNullOrEmpty(engineDirArg))
        Platform.OverrideEngineDir(engineDirArg);
        var supportDirArg = args.GetValue("Engine.SupportDir", null);
        if (!string.IsNullOrEmpty(supportDirArg))
        Platform.OverrideSupportDir(supportDirArg);
        Console.WriteLine($"Platform is {Platform.CurrentPlatform}
({Platform.CurrentArchitecture})");
        try
        {
        EngineVersion = File.ReadAllText(Path.Combine(Platform.EngineDir,
"VERSION")).Trim();
        }
        catch { }
        if (string.IsNullOrEmpty(EngineVersion))
        EngineVersion = "Unknown";
```

```
        Console.WriteLine($"Engine version is {EngineVersion}");
        Console.WriteLine($"Runtime: {Platform.RuntimeVersion}");
        var modID = args.GetValue("Game.Mod", null);
        var explicitModPaths = Array.Empty<string>();
        if (modID != null && (File.Exists(modID) || Directory.Exists(modID)))
        {
        explicitModPaths = new[] { modID };
        modID = Path.GetFileNameWithoutExtension(modID);
        }
        InitializeSettings(args);
        Log.AddChannel("perf", "perf.log");
        Log.AddChannel("debug", "debug.log");
        Log.AddChannel("hostNode", "hostNode.log", true);
        Log.AddChannel("sound", "sound.log");
        Log.AddChannel("graphics", "graphics.log");
        Log.AddChannel("geoip", "geoip.log");
        Log.AddChannel("nat", "nat.log");
        Log.AddChannel("client", "client.log");
        var platforms = new[] { Settings.Game.Platform, "Default", null };
        foreach (var p in platforms)
        {
        if (p == null)
        throw new InvalidOperationException("Failed to initialize platform-integration
library. Check graphics.log for details.");
        Settings.Game.Platform = p;
        try
        {
        var platform = CreatePlatform(p);
        Renderer = new Renderer(platform, Settings.Graphics);
        Sound = new Sound(platform, Settings.Sound);
        break;
        }
        catch (Exception e)
        {
        Log.Write("graphics", $"{e}");
        Console.WriteLine("Renderer initialization failed. Check graphics.log for details.");
        Renderer?.Dispose();
        Sound?.Dispose();
        }
        }
        Nat.Initialize();
        var modSearchArg = args.GetValue("Engine.ModSearchPaths", null);
        var modSearchPaths = modSearchArg != null ?
        FieldLoader.GetValue<string[]>("Engine.ModsPath", modSearchArg) :
        new[] { Path.Combine(Platform.EngineDir, "mods") };
        Mods = new InstalledMods(modSearchPaths, explicitModPaths);
        Console.WriteLine("Internal mods:");
        foreach (var mod in Mods)
        Console.WriteLine($"\t{mod.Key}: {mod.Value.Metadata.Title}
({mod.Value.Metadata.Version})");
        modLaunchW
```

ДОДАТОК Б

Б.1 Скріншноти програми:

Рисунок Б.1 - Інтерфейс головного меню

Рисунок Б.2 - Інтерфейс ігрових опцій

Рисунок Б.3 - Інтерфейс ігрової карти