ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Real Time Strategy з елементами глобальної стратегії»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Ярослав СОЛОДКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42
Ярослав СОЛОДКИЙ

Керівник: Олесь ДІБРІВНИЙ
доктор філософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Солодкому Ярославу Тарасовичу_____

1. Тема кваліфікаційної роботи: «Розробка Real Time Strategy з елементами глобальної стратегії»
керівник кваліфікаційної роботи доктор філософії (PhD) Олесь ДІБРІВНИЙ, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи обробки розробки відеоігор, опис роботи відеоігор, опис методів розробки відеоігор жанру стратегії, технічна документація з описом застосованого ігрового двигуна.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд відеогри з елементами жанрів RTS та глобальної стратегії, та аналіз існуючих аналогів, визначення вимог до програмного забезпечення.
2. Огляд та аналіз засобів реалізації відеогри елементами жанрів RTS та глобальної стратегії.
3. Проектування архітектури та визначення вимог і необхідних ігрових механік до відеоігри, що поєднує в собі елементи жанрів RTS та глобальної стратегії.
4. Програмна реалізація та тестування з елементами жанрів RTS та глобальної стратегії.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення
3. Концепт гри.
4. Короткий огляд ідеї гри.
5. USP (Unique Selling Points)
6. Геймплей, ігровий цикл.
7. Приклади ігрових механік
8. Програмні засоби реалізації
9. Діаграма варіантів використання
10. Діаграма класів
11. Послідовна діаграма зміни станів

12. Екранні форми: стратегічний режим
13. Екранні форми: тактичний режим
14. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд та аналіз відеогри з поєднанням жанрів RTS та глобальної стратегії	13.03-15.03.2025	
4	Огляд та аналіз засобів реалізації відеогри з поєднанням жанрів RTS та глобальної стратегії	16.03-19.03.2025	
5	Проектування архітектури відеогри з поєднанням жанрів RTS та глобальної стратегії	20.03-01.04.2025	
6	Програмна реалізація та тестування відеогри з поєднанням жанрів RTS та глобальної стратегії	02.04-20.04.2025	
7	Огляд та аналіз відеогри з поєднанням жанрів RTS та глобальної стратегії	21.04-26.04.2025	
8	Тестування відеогри	27.04-28.04.2025	
9	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
10	Розробка демонстраційних матеріалів	12.05-18.05.2025	
11	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Ярослав СОЛОДКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Олесь ДІБРІВНИЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 60 стор., 1 табл., 43 рис., 37 джерел.

Мета роботи – знайдення концепції поєднання механік жанрів RTS та глобальної стратегії в межах однієї ігрової системи на основі створеного прототипу гри.

Об'єкт дослідження – процес взаємодії стратегічного та тактичного ігрових режимів у гібридних іграх жанрів Real Time Strategy (RTS) та глобальної стратегії.

Предмет дослідження – ігрові механіки стратегічного та тактичного режимів, а також архітектурні рішення, що забезпечують їхню інтеграцію на основі створеного прототипу гібридної гри.

Короткий зміст роботи: В роботі проаналізовано види та особливості стратегічних ігор представлених на ринку, проаналізовані тренди та цільова аудиторія стратегічних ігор, визначено способи і методи поєднання жанрів RTS та глобальних стратегій. Проаналізовано інструментальні засоби для розробки ігрових продуктів. Розроблено відеогру з елементами двох жанрів та програмно реалізовані ключові функціональні можливості, зокрема: завантаження і вивантаження рівнів у реальному часі, безшовний перехід між режимами, системи економіки та бойових дій. Проведено функціональне та модульне тестування додатку. В роботі використано ігровий двигун Unreal Engine для основного процесу розробки, Blender для редагування 3D моделей, Paint.Net для малювання графічних компонентів та Figma для прототипування інтерфейсу.

Сферою використання відеогри є забезпечення інтелектуального дозвілля гравців та подальше вивчення можливостей реалізації стратегічних ігор.

КЛЮЧОВІ СЛОВА:, СТРАТЕГІЯ, RTS, ГЛОБАЛЬНА СТРАТЕГІЯ, ГІБРИДИЗАЦІЯ ЖАНРІВ, UNREAL ENGINE, ВІДЕОГРА, C++, LEVEL STREAMING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

IDE – Integrated Development Environment

RTS - Real Time Strategy

4X - Explore, Expand, Exploit, Exterminate

MOBA – Multiplayer Online Battle Arena

RPG - Roleplay game

UE - Unreal Engine

UI - User Interface

UX - User Experience

ШІ - Штучний інтелект

HUD - Head-Up Display

RPC - Remote Process Call

BP - Blueprint

WBP - Widget blueprint

BPI - Blueprint interface

BPC - Blueprint component

BPT – Blueprint test

ВСТУП

Актуальність: Розробка відеоігор жанру Real Time Strategy (RTS) з інтеграцією механік глобальної стратегії відповідає сучасним тенденціям у галузі ігрової індустрії, яка демонструє сталий попит на глибокі, комплексні ігрові системи, що поєднують тактичне управління та стратегічне планування. Зростання популярності гібридних ігрових жанрів обумовлюється прагненням гравців до різноманітності досвіду, потребою у розвитку аналітичного мислення та здатності приймати рішення у динамічних умовах [3]. Впровадження елементів глобальної стратегії у форматі RTS дозволяє розширити ігрові можливості, забезпечити вищий рівень занурення у симуляцію складних соціально-економічних і політичних процесів, а також задовольнити вимоги користувачів до гнучкості й варіативності геймплею.

Сучасні технології, такі як потужні ігрові рушії та спеціалізовані фреймворки, надають розробникам інструменти для створення інноваційних продуктів, що можуть використовуватися як для організації інтелектуального дозвілля, так і для моделювання стратегічних сценаріїв та аналізу взаємодії між різними рівнями управління. Саме поєднання стратегічної глибини й динаміки RTS відкриває нові можливості для розвитку ігрових механік та підвищує залученість гравців у ігровий продукт.

Об'єктом дослідження є процес взаємодії стратегічного та тактичного ігрових режимів у гібридних іграх жанрів Real Time Strategy (RTS) та глобальної стратегії.

Предметом дослідження є ігрові механіки стратегічного та тактичного режимів, а також архітектурні рішення, що забезпечують їхню інтеграцію на основі створеного прототипу гібридної гри.

Метою роботи є знайдення концепції поєднання механік жанрів RTS та глобальної стратегії в межах однієї ігрової системи на основі створеного прототипу гри

Методи дослідження: першим кроком було проаналізовано наукові дослідження з теми, для визначення перспективності та поширеності гібридизації ігрових піджанрів. Далі було проаналізовані динаміки розвитку як RTS так і глобальних стратегій. Далі було проведено огляд та аналіз існуючих на ринку стратегій, що мають елементи як RTS, глобальних стратегій одночасно, а також еталонних представників своїх жанрів. На основі отриманого матеріалу, було розпочато формулювання функціональних та нефункціональних вимог до свого програмного забезпечення, в тому числі продумування ігрового циклу, основних ігрових механік та ігрового сетингу.

Далі було проведено огляд стеку технологій реалізації ігрового застосунку, що будуть відповідати вимогам до програмного забезпечення, і було обрано ігровий рушій Unreal Engine 5.5 для збірки гри, Blender для редагування 3D моделей та Paint.Net для створення 2D асетів. Git був застосований для системи контролю версій під час розробки програми. Дослідження реалізації застосунку проводилося під час безпосередньої розробки.

Наукова новизна роботи полягає у вивченні різних рівнів управління ігровими завданнями як зі стратегічного, так і з тактичного рівнів прийняття управлінських рішень, які є перспективним напрямом досліджень в рамках ігрової індустрії, щоб уможливити розробку більш складних і надійних стратегій [15].

Практична значущість результатів полягає у забезпеченні інтелектуального дозвілля гравців та подальше вивчення можливостей реалізації стратегічних ігор.

1 АНАЛІЗ RTS З ЕЛЕМЕНТАМИ ГЛОБАЛЬНОЇ СТРАТЕГІЇ

1.1 RTS з елементами глобальної стратегії

Гібридна стратегічна відеогра типу Real Time Strategy з елементами глобальної стратегії (4X) — це спеціалізований ігровий продукт, який поєднує ключові механіки двох стратегічних піджанрів з метою створення глибшого й більш різноманітного геймплею. У класичних RTS-іграх основна увага приділяється керуванню юнітами та ресурсами в реальному часі, швидкому прийняттю тактичних рішень та динамічному розвитку подій на полі бою. Натомість ігри жанру 4X акцентують на глобальному управлінні, розвитку імперії, дослідженні карти, дипломатії, технологічному прогресі та довгостроковому стратегічному плануванні.

Об'єднання цих двох підходів у межах одного ігрового проєкту дозволяє реалізувати унікальний ігровий цикл. Гравець отримує можливість не лише стратегічно розвивати свою фракцію чи державу, керувати економікою та дипломатією на глобальній карті, але й особисто брати участь у тактичних боях, що проходять у реальному часі. Завдяки такій інтеграції формується новий рівень занурення: глобальні рішення гравця впливають на перебіг локальних битв, а результати боїв у свою чергу визначають подальший розвиток всієї імперії чи фракції.

Основна мета поєднання RTS і 4X-механік — надати гравцю комплексний досвід управління як на макро-, так і на мікрорівні. Це сприяє розвитку аналітичного мислення, вміння планувати на багато ходів вперед і оперативно реагувати на зміну ситуації [15, 24]. До основних компонентів представленої гібридної стратегії належать:

– глобальна карта, де реалізовано дослідження світу, експансію, управління ресурсами та дипломатичну взаємодію;

– тактичний режим реального часу, який дозволяє безпосередньо управляти військами під час битв;

– економічна, наукова та соціальна моделі, які впливають на стратегію розвитку держави або фракції;

– механізми переходу між стратегічним і тактичним рівнями без втрати цілісності геймплею.

Такі ігри надають гравцям змогу обирати власний стиль проходження, комбінуючи довгострокове стратегічне мислення з миттєвим тактичним реагуванням. Завдяки цьому гібридний жанр RTS/4X здатен зацікавити як шанувальників класичних RTS, так і поціновувачів глибоких глобальних стратегій, а також стимулює створення нових сценаріїв для моделювання складних соціально-економічних процесів у віртуальному світі.

1.2 Жанр RTS

Жанр Real Time Strategy (RTS) — це один із найвизначніших і найвпливовіших піджанрів стратегічних відеоігор, що сформувався на початку 1990-х років та досяг піку популярності у кінці 1990-х – початку 2000-х. Класичні RTS-ігри передбачають управління ресурсами, будівництво баз, формування армій і ведення бойових дій у реальному часі, де успіх залежить від поєднання стратегічного мислення, тактичних рішень та оперативного реагування на зміну ситуації на полі бою.

Перші культові представники жанру — Dune II (1992), Warcraft: Orcs & Humans (1994), Command & Conquer (1995), а також StarCraft (1998) та Age of Empires (1997). Вони задали стандарти жанру та сформували велику аудиторію прихильників, а їхні ремастери й сьогодні залишаються взірцями балансу ігрових механік та глибини геймплею [26].

Успіх RTS був зумовлений їх унікальним поєднанням динаміки,

планування і можливості швидко адаптуватися до ситуації. Гравці одночасно керують великою кількістю елементів, розвивають економіку, ведуть бої і досліджують нові технології. Це вимагає високої концентрації, багатозадачності та здатності планувати як короткострокові, так і довгострокові дії.

На піку популярності жанр RTS домінував на ринку стратегічних ігор, але з часом зіштовхнувся із серйозними викликами. Сучасні спроби розвивати або спрощувати (оказуалювати) жанр часто не виправдовують очікувань гравців. Провал Dawn of War 3, останніх Command & Conquer, Supreme Commander 2 і подібних проектів демонструє, що зменшення глибини, спрощення механік та фокус на масового гравця не сприяють відновленню колишньої популярності [25]. На думку аналітиків, повернення жанру до "золотої епохи" можливе лише за умови переосмислення геймплейних основ та впровадження інновацій, а не простого повторення або спрощення старих рішень [13, 15].

Водночас дослідження підтверджують, що ігри жанру RTS позитивно впливають на розвиток когнітивних функцій, зокрема покращують стратегічне мислення, навички вирішення проблем і навіть комунікаційні здібності [24, 13, 15]. Систематичний огляд літератури вказує, що стратегічні ігри можуть використовуватися як інструмент для підвищення когнітивної гнучкості, тренування пам'яті та поліпшення швидкості прийняття рішень у стресових ситуаціях.

Загалом жанр RTS продовжує розвиватися, попри конкуренцію з боку інших типів стратегічних ігор (наприклад, 4X, MOBA) та зміну вподобань аудиторії. Про це свідчить поява низки сучасних ігор, які намагаються не лише наслідувати класиків, але й запропонувати власні унікальні ігрові рішення. Серед успішних новітніх проєктів можна відзначити Age of Empires IV (2021), яка отримала позитивні відгуки за збереження балансу між класичною формулою та сучасними інтерфейсами й механіками. Company of Heroes 3 (2023) продовжує традиції тактичної RTS з акцентом на логістику, реалістичність боїв та

удосконалену систему укриттів. Iron Harvest (2020) вирізняється поєднанням елементів класичних RTS із унікальним візуальним стилем альтернативної історії та мехів.

Окрім згаданих ігор, варто виділити такі сучасні RTS, як WARNO, яка продовжує лінію тактичних воєнних симуляторів з глибокою системою управління військами та серйозною увагою до реалізму. Call to Arms – Ostfront представляє собою продовження традицій тактичних стратегій з можливістю прямого контролю одиниць та широкою підтримкою модифікацій, що розширює ігровий досвід. Starship Troopers: Terran Command (2022) пропонує асиметричний геймплей у науково-фантастичному сетингу з акцентом на оборону та виживання проти хвиль противників.

Проте паралельно з цим на ринку з’являється велика кількість нових RTS, які не здобувають уваги аудиторії через вторинність ідей або недостатню якість виконання. Серед них чимало проєктів, які залишаються маловідомими або швидко забуваються, оскільки не здатні запропонувати гравцеві щось нове чи дійсно захоплююче. Успіх RTS безпосередньо залежить від наявності унікальних механік та глибини геймплею, які створюють мотивацію повертатися до гри знову і знову заради нових викликів та відчуття задоволення від перемоги. Саме ті ігри, які пропонують свіжі ідеї, оригінальні підходи до стратегії чи інноваційні системи управління, формують лояльну спільноту гравців і стають новими еталонами жанру.

Варто зазначити, що українські розробники мають вагомий внесок у розвиток жанру RTS, про що свідчать приклади серії Cossacks та Man of War. Класична гра Cossacks: European Wars ще з початку 2000-х була відзначена глибокою економічною моделлю, масштабними битвами та високим рівнем історичної достовірності. Ремастер Cossacks 3 (2016) має схвальні відгуки у Steam, а переважна більшість негативних оцінок пов’язана не з геймплеєм, а з технічними аспектами гри. Інша відома серія — Man of War — відзначалася

тактичним різноманіттям та реалістичною фізикою, але нещодавній реліз Man of War 2 (2023) зустрів критику спільноти через спрощення механік, відсутність низки функцій та зниження глибини геймплею у порівнянні з оригінальною грою. Це викликало хвилю обурення серед фанатів і є ілюстрацією того, наскільки важливими є автентичність, унікальність і продуманість механік у сучасних RTS, а також підтверджує, що гравці залишаються вірними тим проектам, які пропонують справжню глибину та нові ідеї.

Успіх ремастерів класичних ігор та поява нових проектів свідчить про сталий інтерес до глибокого, складного і динамічного ігрового процесу, що є унікальною рисою жанру RTS [27].

1.3 Жанр глобальна стратегія

Глобальна стратегія — це піджанр стратегічних відеоігор, у якому гравець отримує контроль над управлінням цілою державою, імперією чи іншою масштабною організацією. Відмінною рисою глобальних стратегій є значний акцент на довгостроковому плануванні, багаторівневому управлінні економікою, політикою, дипломатією, технологіями, розвитком суспільства, а також проведенням військових кампаній на глобальній мапі світу.

Часто поняття "глобальна стратегія" перетинається з жанром 4X (explore, expand, exploit, exterminate — досліджуй, розширюйся, експлуатуй, знищуй). Класичний представник 4X — це серія Civilization, де ігровий процес сфокусований на дослідженні карти, поступовому розширенні територій, розвитку ресурсів та веденні воєн або дипломатії. Водночас глобальні стратегії можуть містити не тільки 4X-механіки, а й інші елементи, наприклад, складні соціальні чи економічні системи, гнучкі дипломатичні сценарії та моделювання історичних чи альтернативних процесів, що притаманно іграм Paradox Interactive.

Глобальні стратегії виникли ще у 1980-1990-х роках із появою таких ігор як Balance of Power, Master of Orion, Civilization та Europa Universalis. На відміну від RTS, основна увага в глобалках зосереджена на макрорівневому управлінні, а не на безпосередньому контролі над окремими юнітами у реальному часі. Проте піджанр відрізняється великою різноманітністю підходів до часу: частина ігор (наприклад, Civilization VI) має покрокову структуру, де кожен хід означає рік чи десятиріччя розвитку держави. В іграх Paradox (Hearts of Iron IV, Europa Universalis IV, Crusader Kings III, Victoria 3, Stellaris) використовується дискретний або майже реальний час: гравець керує подіями з можливістю ставити гру на паузу, а внутрішній календар моделює реальні історичні дати, які змінюються кожні декілька секунд чи днів. Деякі сучасні ігри, такі як Terra Invicta, поєднують фазові покрокові рішення із симуляцією часу, що минає в реальному масштабі між окремими фазами.

Притаманні риси глобальних стратегій — це комплексність і глибина механік, багатоваріантність шляхів розвитку, необхідність приймати рішення із відкладеним ефектом, гнучка дипломатія, розвиток внутрішньої та зовнішньої політики, військова експансія або балансування інтересів між державами. Успіх гри часто вимірюється не швидкістю перемоги, а здатністю вибудовувати довгострокову стратегію і реагувати на непередбачувані зміни в ігровому світі. Саме цей піджанр характеризується однією з найбільших реіграбельностей серед відеоігор завдяки процедурній генерації карт, різноманіттю сценаріїв, розгалуженим деревам подій, дипломатичним і військовим конфліктам та багатошаровості геймплею.

На сьогодні серії Total War (Creative Assembly) та глобальні стратегії від Paradox Interactive (Hearts of Iron IV, Europa Universalis IV, Crusader Kings III, Stellaris, Victoria 3) утримують лідерство за масштабами аудиторії та кількістю продажів [31-32]. За даними аналітиків, дохід Paradox Interactive у 2023 році перевищив 2 млрд шведських крон, а гравців серії Hearts of Iron IV, Europa

Universalis IV та Crusader Kings III сумарно нараховується десятки мільйонів [28]. Total War: Three Kingdoms, наприклад, встановила рекорд продажів серії, а інші глобальні стратегії утримують стабільно високу активність у Steam [29].

Зростання популярності цього піджанру підтверджують не лише комерційні результати, а й тенденції до появи нових проектів, зокрема у жанрі sci-fi (Stellaris, Terra Invicta), історичної симуляції (Old World, Humankind) і альтернативних сценаріїв розвитку суспільства (Victoria 3). Глибокі механіки, підтримка модифікацій та активна спільнота забезпечують глобальним стратегіям значний вплив на розвиток індустрії стратегічних ігор.

1.4 Цільова аудиторія та аналіз трендів

Аналіз сучасного ринку стратегічних ігор вказує на суттєві зміни у вподобаннях і поведінці гравців. За останнє десятиліття жанр стратегій — як у класичному, так і у гібридному вигляді — поступово втрачає популярність серед масового гравця. Згідно з дослідженням Quantic Foundry (2024), стратегічні ігри, зокрема RTS та глобальні стратегії, втратили значну частину своєї колишньої аудиторії. Це пов’язано із загальним трендом на зменшення ролі стратегічного мислення в ігровому досвіді, зростанням популярності екшн-орієнтованих ігор та спрощенням геймплею в мобільному сегменті [33].

Сучасна реальність така, що розробникам стратегічних ігор немає сенсу орієнтуватися на широкі маси. Більшість успішних проєктів працюють із лояльною, але більш вузькою спільнотою гравців, які цінують глибину, складність, реіграбельність та унікальні механіки. Саме ця цільова аудиторія — фанати RTS, 4X, глобальних та тактичних стратегій — формує ядро ринку, забезпечуючи попит на нові та інноваційні проєкти. Орієнтація на експертну спільноту дозволяє створювати ігри, які справді знаходять відгук у цільової

аудиторії та стають успішними у своїй ніші.

Водночас варто відзначити, що навіть у періоди падіння популярності, ринок стратегічних ігор залишається достатньо великим та стабільним. За оцінками Verified Market Research, обсяг ринку стратегічних ігор у 2023 році становив понад 14 мільярдів доларів, а до 2030 року прогнозується його зростання до 24 мільярдів доларів [34]. Статистика Statista також підтверджує, що стратегічні ігри мають стійкий попит на світовому ринку, особливо у сегменті мобільних 4X-стратегій, які генерують мільярди доларів доходу щороку [35].

Досвід мобільного ринку демонструє, що створення гібридних стратегій, які поєднують елементи різних жанрів (наприклад, RPG, економіки, головоломок), є одним із ключових трендів останніх років. Саме гібридизація стала рушієм для створення успішних мобільних 4X-стратегій, що дозволило залучити нову аудиторію та забезпечити більшу різноманітність геймплею [11, 30]. Гібридні моделі відкривають нові можливості для розробників, дозволяючи поєднувати глибину традиційних стратегій із динамікою та залученням, притаманними іншим жанрам.

Таким чином, потенційна аудиторія для нової гібридної стратегічної гри — це насамперед досвідчені гравці, які цінують комплексний підхід, багаторівневу взаємодію механік та новизну у геймплеї. Водночас варто враховувати, що навіть серед шанувальників жанру не всі готові сприймати експерименти з поєднанням RTS та глобальних стратегій: лише висока якість, продумана концепція й інноваційний підхід можуть поступово розширити аудиторію за межі ядра спільноти. Створення феноменально успішного продукту у цій ніші — це завдання не однієї гри, а довгострокова стратегія, що передбачає розвиток нових підходів і накопичення довіри гравців.

1.5 Дослідження існуючих аналогів

Ринок RTS та Глобальних стратегій (в тому числі 4X) має цілий ряд легендарних представників, що полюбляються гравцями протягом років. Всі вони поєднують в собі не тільки унікальні особливості, але і спільну риси: такі як стратегічне планування, правильне прийняття рішень, реіграбельність та різноманітність ігрового процесу. В рамках даної кваліфікаційної роботи є доцільним розглянути нижче описаних представників жанру Стратегій.

1.5.1 Серія Total War

Ігрова серія Total War від британської студії Creative Assembly є еталоном поєднання жанрів RTS та глобальної стратегії. Сетинг ігор охоплює історичні періоди, зокрема античність, Середньовіччя, епоху Наполеона, а також фентезі-всесвіт Warhammer. Це дозволяє залучити широку аудиторію – як прихильників історії, так і шанувальників фантастики. У центрі геймплею – складна структура, де гравець керує державою або фракцією на стратегічній карті, плануючи економіку, дипломатію, розвиток міст, а у визначені моменти переходить до тактичних битв у реальному часі з тисячами юнітів на полі бою.

Рис. 1.1 Зображення стратегічного режиму: фізична мапа світу з відображенням армій та міст

Геймплей серії побудовано навколо двох головних циклів: стратегічного управління імперією та тактичного командування арміями. На стратегічному рівні гравець займається розвитком міст, рекрутингом військ, встановленням дипломатичних відносин, збором податків і плануванням загального напряму розвитку фракції. Тактичний рівень реалізується через масштабні битви, де гравець керує розташуванням, рухом та взаємодією підрозділів у реальному часі. Особливою рисою серії є наявність логістики руху армій, значення морального стану військ, терену та погодних умов для результату битв. Інноваційною механікою є також система послідовного розвитку генералів та їх впливу на перебіг боїв.

Рис. 1.2 Зображення тактичного режиму: керування загонами армії на полі бою

Основні механіки Total War включають покрокове управління глобальною картою з можливістю будівництва, дипломатії та дослідження технологій, а

також реальні битви у режимі RTS, де тактичне розташування військ і використання особливих умінь командирів часто визначає результат. Унікальною перевагою серії є поєднання масштабної економічної симуляції та реалістичних боїв, що робить її одним із небагатьох представників гібридних стратегій такого рівня. Це представлено на рисунку 1.2. та 1.1. [5]

Цільова аудиторія Total War складається з гравців, які цінують складне стратегічне планування, історичний або фентезійний антураж, а також можливість контролювати великі армії в реальному часі. Саме симбіоз глибокої стратегії та тактики є основною унікальною пропозицією (USP) цієї серії.

1.5.2 Star Wars: Empire at War

Star Wars: Empire at War – це гра, яка переносить гравців у всесвіт Зоряних війн, дають змогу управляти імперськими або повстанськими силами у масштабній галактичній кампанії. Сеттинг гри охоплює ключові події оригінальної кінотрилогії, що підсилює емоційну залученість фанатів франшизи. Empire at War вдало поєднує покрокове управління галактикою на макрорівні з реальними битвами на поверхні планет або у відкритому космосі.

Рис. 1.3 Зображення стратегічного режиму: карта галактики з відображенням основних систем та наявних армій та флоту

Головний ігровий цикл будується навколо стратегічного розподілу ресурсів, захоплення планет, переміщення флотів і підготовки до битв. Важливою особливістю є те, що при зіткненні флотів чи армій гравець переходить у режим RTS-бою, де в реальному часі командує підрозділами, використовує героїв із унікальними здібностями, вирішує питання логістики та захисту баз. Таким чином, гра дозволяє поєднати довгострокове планування із динамічними тактичними діями, що робить Empire at War представником гібридного підходу до жанру. Це представлено на рисунках 1.3 та 1.4.

Ключові механіки Empire at War включають систему контролю над ресурсами галактики, дипломатію з нейтральними планетами, розвиток технологій та оперативне використання героїв. Значною унікальною перевагою гри є використання культових персонажів і сюжетних арок, що формують її USP і надають грі характерної ідентичності, яка неможлива в інших проектах.

Відзначити слід і просту, але ефективну систему економіки та взаємодії між планетами. [8]

Рис. 1.4 Зображення тактичного режиму: карта самої планети з боями юнітів на землі

Цільова аудиторія – шанувальники Star Wars, а також гравці, які цінують поєднання стратегічного мислення з тактичним командуванням у знайомому тематичному середовищі. Гра приваблює як ветеранів стратегій, так і новачків, завдяки інтуїтивній системі управління та зрозумілим механікам.

1.5.3 Hearts of Iron IV

Hearts of Iron IV – це історична глобальна стратегія від Paradox Interactive, що відтворює події Другої світової війни на глобальному рівні. Сетинг охоплює період із 1936 по 1948 роки, дозволяючи гравцям взяти під контроль будь-яку націю світу й переписати хід історії за власним сценарієм. Особливістю є масштаб симуляції – від виробництва танків до дипломатичних інтриг та

керування політичними режимами.

Рис. 1.5 Зображення стратегічного режиму: політична карта землі з можливістю керувати військами для ведення повномасштабної війни у повітрі, на землі та на морі

Ігровий цикл Hearts of Iron IV сфокусовано на глобальному управлінні всіма аспектами держави: військовою промисловістю, вербуванням і постачанням армій, дипломатією, науковими дослідженнями, і навіть ідеологічною боротьбою. Головний виклик полягає у створенні ефективної стратегії з урахуванням обмежених ресурсів, динамічно змінюваної ситуації на фронті та непередбачених політичних подій. Серед ключових механік – розподіл виробництва, логістика забезпечення військ, вибір стратегічних напрямів розвитку та застосування доктрин. [9]

Унікальність Hearts of Iron IV проявляється у глибокій деталізації та багатошаровості глобальної симуляції: гравець може впливати на хід війни не лише безпосередньо на полі бою, а й через дипломатію, шпигунство та внутрішню політику. Цільова аудиторія – це гравці, які захоплюються історією,

цінують складні симуляції та багатовекторність розвитку, а також фанати Paradox Interactive, для яких важлива можливість альтернативної історії та складного планування на макрорівні. Це можна побачити на рисунку 1.5.

1.5.4 Age of Empires IV

Age of Empires IV – це продовження класичної серії RTS, заснованої на розвитку цивілізацій від Середньовіччя до ранньомодерного періоду. Сетинг охоплює різні епохи, у яких гравець керує однією з націй, розвиваючи її технології, економіку та військову міць. Гра вирізняється акцентом на історичній достовірності та увагою до деталей у представленні архітектури, юнітів і культурних особливостей.

Ігровий цикл базується на традиційному для жанру ланцюгу: збір ресурсів, розвиток бази, дослідження технологій і нарощування військової потужності з наступним веденням битв у реальному часі. Найважливіші механіки Age of Empires IV – це система економіки, унікальні технологічні дерева для кожної цивілізації та асиметричність ігрових фракцій. Гравець мусить враховувати сильні й слабкі сторони обраної нації, балансувати економічний і військовий розвиток та адаптуватися до дій супротивників [10]. Це відображено на рисунку 1.6.

Рис. 1.6 Зображення тактичного режиму: політична карта землі з можливістю керувати військами для ведення повномасштабної війни у повітрі, на землі та на морі

Унікальною перевагою гри є поєднання історичної реконструкції з інноваціями в механіках розвитку цивілізацій, що забезпечує глибоке занурення й реіграбельність. Цільова аудиторія Age of Empires IV – це шанувальники класичних RTS, а також гравці, які цікавляться історією, прагнуть різноманітності стратегій та варіативності підходів у розвитку фракцій.

1.5.5 Supreme Commander

Supreme Commander – масштабна RTS, дія якої відбувається у науково-фантастичному всесвіті, де людство бореться за контроль над галактикою у далекому майбутньому. Гра відома своїми величезними картами, можливістю командувати тисячами юнітів та розгалуженою системою логістики. Сетинг відзначається футуристичними технологіями, унікальними фракціями та наголосом на масових битвах.

Геймплей гри полягає у комплексному управлінні економікою, масовому

виробництві військ та стратегічному використанні ресурсів. Головними механіками Supreme Commander є масштабування ігрового процесу (масштабування від рівня окремого юніта до стратегічної карти), інноваційна система автоматизації наказів, а також модульна побудова баз і виробничих ланцюгів. Кожна з фракцій має унікальні сильні сторони, що змушує гравця адаптувати тактику до ситуації на полі бою [4].

Рис. 1.7 Зображення тактичного режиму: карта з великою кількістю юнітів та будівель різних спеціалізацій та розміру

Головною унікальною перевагою Supreme Commander є її інженерна масштабованість: гра вперше дозволила реалізувати дійсно епічні битви з тисячами юнітів без втрати контролю. Цільова аудиторія гри – це досвідчені стратеги, які шукають глибокого управління ресурсами та масштабності, а також шанувальники наукової фантастики, яких приваблюють футуристичні технології та велике стратегічне мислення. Відображення геймплею на рисунку 1.7.

1.5.6 Порівняльний аналіз програмних засобів

Вищенаведені приклади демонструють різні підходи до поєднання механік

RTS і глобальної стратегії у сучасних відеоіграх. Total War забезпечує комплексний баланс між покроковим стратегічним управлінням і масштабними тактичними битвами, Empire at War акцентує увагу на динаміці та тематичній глибині, Hearts of Iron IV пропонує унікальний підхід до симуляції політики, економіки й війни, Age of Empires IV – класичну модель розвитку цивілізації, а Supreme Commander – приклад технічної реалізації гіпермасштабних RTS-битв. Порівняльна таблиця буде представлена нижче (Табл. 3.1), де основні параметри, пов'язані з геймплейними циклами, ключовими механіками та орієнтацією на цільову аудиторію, систематизовані для візуального аналізу результатів порівняння.

Таблиця 1.1

Зведена таблиця аналізу існуючих відеоігор з точки зору елементів RTS та глобальної стратегії

Гра / серія	**Сеттинг**	**Основний ігровий цикл**	**Ключові механіки**	**Унікальна перевага (USP)**	**Цільова аудиторія**
Total War	Історія / фентезі	Глобальне управління + тактичні битви	Стратегічна економіка, тактична логістика, розвиток генералів	Поєднання двох рівнів геймплею (глобальний+тактичний)	Любителі історії/фентезі, стратеги, аналітики
Empire at War	Наукова фантастика (Star Wars)	Галактична стратегія + локальні битви	Контроль ресурсів галактики, унікальні герої, тактичні бої	Система героїв та сюжетних подій з культовим сетингом / Поєднання двох рівнів геймплею	Фанати Star Wars, шанувальники гібридних стратегій
Hearts of Iron IV	Історія (Друга світова війна)	Глобальна симуляція управління державою	Виробництво, логістика, політичні/ідеологічні системи	Глибока політична симуляція та альтернативна історія	Любителі історії, фанати симуляцій, Paradox-спільнота

Продовження таблиці 1.1

Зведена таблиця аналізу популярних відеоігор з елементами RTS та глобальної стратегії

Гра / серія	**Сеттинг**	**Основний ігровий цикл**	**Ключові механіки**	**Унікальна перевага (USP)**	**Цільова аудиторія**
Age of Empires IV	Історія (Середньовіччя/Новий час)	Класична RTS: розвиток, економіка, війна	Унікальні дерева технологій, асиметричність фракцій, економіка	Історична достовірність у класичному RTS-циклі	Фанати RTS, історії, новачки та досвідчені гравці
Supreme Commander	Фантастика (майбутнє)	Масштабна RTS: будівництво, ресурси, масові битви	Масштабування управління, автоматизація, модульна економіка	Інженерна масштабованість і контроль сотень юнітів	Досвідчені стратеги, поціновувачі футуристичних сценаріїв

1.6 Визначення вимог до відеогри

Визначення вимог до ігрового продукту потребує глибокого і багаторівневого аналізу як самої предметної галузі, так і сучасних тенденцій у геймдизайні, архітектурі ігор, методах організації штучного інтелекту та інноваційних підходів до взаємодії з користувачем. На думку Ернеста Адамса, викладену у фундаментальній праці "Fundamentals of Game Design" [1], саме ігрові механіки формують ідентичність відеоігри — вони визначають структуру, внутрішню логіку, взаємозв’язки між ігровими системами, а отже, напряму впливають не лише на сам геймплей, але й на залученість та лояльність аудиторії, а також на здатність продукту виділятися серед численних аналогів на ринку. В сучасних умовах розвитку індустрії комп’ютерних ігор провідною тенденцією стає гібридизація — поєднання макро- і мікрорівня управління, що відкриває нові горизонти для поглиблення геймплею, збільшення варіативності і створення унікальних сценаріїв [2]. Унікальною особливістю гібридних стратегічних ігор стає саме безшовний перехід між стратегічними і тактичними режимами, що дає змогу гравцеві гнучко змінювати рівень деталізації управління, зберігаючи при цьому цілісність ігрового світу та ігрового досвіду в цілому.

Окрема роль у цьому належить питанням архітектури ігрового рушія. Як зазначає Джессі Грегорі у книзі "Game Engine Architecture" [6], створення масштабних стратегічних ігор потребує ґрунтовного підходу до побудови внутрішніх систем: ефективне управління ігровими даними, оптимізація продуктивності на різних етапах гри, забезпечення гнучкості систем збережень, відмовостійкості, а також синхронізації станів у багатокористувацькому режимі. Не менш важливим є використання асинхронних обчислень, багатопоточності, підтримки складних алгоритмів ШІ та роботи з великими об’ємами інформації, що стає ключовим для ігор, у яких гравець керує сотнями юнітів, численними територіями, розгалуженими технологічними і соціальними деревами та

складними економічними зв’язками між об’єктами ігрового світу.

Розробка сучасних стратегічних ігор неможлива без впровадження передових технологій у сфері штучного інтелекту. Дослідження Millington & Funge ("Artificial Intelligence for Games", 2009) [7] і Yannakakis & Togelius ("Artificial Intelligence and Games", 2018) [14] наголошують на необхідності створення багаторівневих, адаптивних та реактивних систем ШІ, здатних не лише відповідати на дії гравця, а й самостійно аналізувати зміни на глобальному та локальному рівнях, підлаштовувати власну стратегію до поточних умов, і навіть навчатися від ігрового досвіду користувача. Це дозволяє зробити геймплей справді динамічним, непередбачуваним, підвищити рівень виклику для досвідчених гравців та забезпечити широкий спектр стратегічних сценаріїв. Сучасні підходи до розробки ШІ включають гібридні моделі, евристики, дерева прийняття рішень, навчання з підкріпленням та елементи машинного навчання, що поступово проникають у масові проекти та відкривають нові можливості для реалізації складних і багатовимірних систем поведінки супротивників і союзників у грі.

Важливою складовою успіху будь-якої сучасної гри є продуманий інтерфейс, який забезпечує прозору, гнучку та інтуїтивну взаємодію користувача із системами управління. Розробники мають враховувати не лише базові принципи UI/UX (Rogers, 2014 [11]), а й тенденції до автоматизації рутинних операцій, створення гнучких налаштувань, впровадження розумної системи підказок, адаптивних інформаційних панелей, що допомагають швидко орієнтуватися в складних структурах даних. Надзвичайно актуальним для гібридних стратегій є забезпечення адаптації інтерфейсу до різних рівнів досвіду гравця: новачок має змогу швидко вникнути у базові принципи гри, а досвідчений користувач отримує доступ до розширених функцій, налаштувань та автоматизації частини дій. Окремо варто підкреслити роль процедурної генерації контенту (Sweetser, 2008 [12]), яка не лише підвищує реіграбельність, а

й сприяє формуванню нового досвіду кожного разу — це підтримка унікальних сценаріїв, генерація випадкових карт, подій, завдань та внутрішньоігрових викликів, що сприяє залученню гравців до повторних проходжень гри.

Крім базових функцій, сучасні вимоги до стратегічних ігор охоплюють питання оптимізації ресурсів, гнучкості налаштувань, стабільності багатокористувацької гри, захисту від несанкціонованого доступу та організації надійного збереження прогресу. У гібридних RTS із глобальними елементами особливого значення набуває безшовність переходу між стратегічним і тактичним режимами — це унеможливлює розрив ігрового процесу, дає змогу уникати втрат у логікі й послідовності подій, а також дозволяє гравцеві органічно переміщуватися між різними рівнями керування.

Узагальнюючи проведений аналіз, на основі сучасної літератури з геймдизайну, архітектури ігрових рушіїв та програмної інженерії, можна виділити такі вимоги необхідні до прототипу гібридної RTS з елементами глобальної стратегії.

Функціональні вимоги:

1. Перемикання режимів гри (стратегічний та тактичний RTS-режим) з використанням патерну «Стан».
2. Реалізація стратегічного режиму (управління економікою, переміщення військ по стратегічній карті, розділеної на різні провінції).
3. Реалізація тактичного режиму: Можливість управління військами в RTS-режимі за допомогою миші (віддача наказів для переміщення і атаки).
4. Реалізувати режими кампанії (RTS + Глобальна стратегія).
5. Визначити чіткого списку вимог для отримання перемоги.

Нефункціональні вимоги:

1. Продуктивність**:** Забезпечення стабільного фреймрейту (не нижче 60 FPS)

на більшості комп’ютерів (відповідно до аналітики Steam).

2. Надійність**:** Відсутність критичних помилок (аварійного завершення програми) під час ігрового процесу.
3. Сумісність**:** Працездатність продукту на сучасних версіях Windows (10-11).
4. Локалізація: українська та англійська мови.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Ігровий рушій

Ігровий рушій — це комплексна програмна платформа, яка надає всі базові та спеціалізовані інструменти для розробки інтерактивних додатків, зокрема відеоігор. До функцій рушія належить управління графікою, фізикою, звуком, логікою взаємодії, мережевою взаємодією, анімацією, а також підтримка UI/UX, синхронізації ігрового процесу та оптимізації ресурсів. Саме рушій дозволяє команді розробників створювати ігровий світ із заданою функціональністю, забезпечує кросплатформеність, дає змогу масштабувати проекти та інтегрувати нові технології без повного переписування коду. Вибір рушія — це стратегічне рішення, яке впливає на продуктивність, гнучкість, швидкість розробки, а також можливість адаптації гри до вимог сучасної індустрії.

2.1.1 Вибір Unreal Engine 5.5 для гібридної RTS

Для розробки ігрової логіки та механік гібридної стратегії у цій роботі було обрано Unreal Engine 5.5. Unreal Engine — це сучасний рушій із відкритим вихідним кодом, який став галузевим стандартом для AAA-проєктів і все частіше використовується для складних інді-розробок. Його вибір обумовлений низкою ключових переваг: підтримкою складної тривимірної графіки, потужною

системою візуального скриптингу (Blueprints), розширеним інструментарієм для мережевої взаємодії, а також найсучаснішими технологіями оптимізації (Nanite, Lumen).

Unreal Engine дозволяє розробляти як 3D-, так і 2D-ігри, ефективно працює з великими ігровими світами, підтримує процедурну генерацію контенту, фізичне моделювання, а також розвинуті можливості для створення складних сценаріїв взаємодії між підсистемами (економіка, військові механіки, ШІ). Значна увага приділяється питанням продуктивності: рушій дозволяє працювати з тисячами одночасних об'єктів, динамічно підвантажувати локації, забезпечувати плавність анімацій навіть при високому навантаженні на систему.

2.1.2 Порівняння Unreal Engine із конкурентами: Unity, Godot

Серед альтернативних рушіїв найчастіше розглядаються Unity та Godot. Unity є найбільш розповсюдженим серед інді-розробників завдяки доступності, легкості освоєння, великій кількості плагінів і підтримці мобільних платформ. Однак, у контексті складних гібридних стратегій із масштабними ігровими світами та високими вимогами до графіки та продуктивності, Unity поступається Unreal Engine. UE5 забезпечує вищу якість візуалізації (завдяки Nanite, Lumen), стабільну роботу з великими сценами, кращу оптимізацію рендеру та має більшу підтримку AAA-інструментарію, що підтверджують численні дослідження й практичні кейси з використання рушіїв у сучасній ігровій індустрії [16-17].

Godot, хоча й швидко розвивається, орієнтований передусім на 2D-проєкти або невеликі інді-гри. Його перевагою є повністю відкритий вихідний код, але він не може конкурувати з Unreal щодо оптимізації 3D-графіки, глибини інструментів для ШІ, багатокористувацьких режимів, а також стабільності при роботі з великими проектами [18]. Саме тому вибір Unreal Engine 5.5 є оптимальним для сучасної RTS із елементами глобальної стратегії.

2.2 Середовище розробки

Visual Studio — це одна з найпотужніших інтегрованих середовищ розробки (IDE) для програмування на C++ та ряді інших мов. Воно забезпечує глибоку інтеграцію з Unreal Engine, підтримку автодоповнення, рефакторингу, інтелектуального пошуку помилок, профілювання та налаштування складних build-процесів. Visual Studio дозволяє організувати роботу з великими проектами, підтримує розробку у багатокористувацькому режимі, забезпечує надійну систему тестування та налагодження коду.

Важливою перевагою є можливість розширення за рахунок плагінів, зокрема для інтеграції з Git, налаштування командної роботи, автоматизації повторюваних завдань. Вибір Visual Studio для розробки C++ класів гри у поєднанні з Unreal Engine є практично стандартом у сучасній ігровій індустрії, адже це рішення перевірене у сотнях комерційних та дослідницьких проєктів [19].

2.3 Редактор тривимірної графіки

Blender — це безкоштовний open-source редактор тривимірної графіки, який дає змогу створювати, редагувати, оптимізувати моделі, працювати з матеріалами, текстурами, UV-розгортками, скелетною анімацією та рендерингом. Його головною перевагою є поєднання професійного функціоналу з доступністю для незалежних розробників. Blender активно використовується для створення як персонажів, так і об'єктів середовища, а також для генерації стратегічних карт та елементів інтерфейсу. Глибока інтеграція з Unreal Engine дає змогу легко переносити контент, уникаючи втрат якості, а велика кількість додаткових скриптів і аддонів розширює можливості моделювання та автоматизації робочих процесів [19-20].

У порівнянні з іншими 3D-редакторами (3ds Max, Maya), Blender виділяється відкритістю коду, безкоштовною ліцензією, активною спільнотою та широкими можливостями для кастомізації. Це робить його оптимальним вибором для навчальних та малобюджетних проєктів, а також для прототипування моделей у складних ігрових системах.

2.4 Редактор двовимірної графіки

Paint.NET — це легкий, безкоштовний графічний редактор, який ідеально підходить для швидкої підготовки 2D-ассетів, малювання стратегічних кордонів, редагування текстур та створення елементів інтерфейсу. Його головні переваги — простота освоєння, невибагливість до ресурсів ПК, підтримка шарів, швидке збереження у форматах, сумісних з ігровими рушіями (наприклад, PNG з альфа-каналом).

На відміну від таких професійних інструментів, як Adobe Photoshop чи GIMP, Paint.NET орієнтований саме на прості і швидкі задачі, тому він став невід’ємною частиною пайплайну інді-розробників, які працюють із підготовкою ігрових матеріалів для інтеграції у рушії. Його використання дозволяє суттєво скоротити час на підготовку графічних ресурсів, зберігаючи баланс між якістю і швидкістю розробки [21].

2.5 Система контролю версій

Git — це сучасна розподілена система контролю версій, яка дає змогу ефективно організувати командну роботу над проектом, фіксувати всі зміни у коді та ресурсах, управляти гілками, синхронізувати внесення змін від різних учасників команди. Git дозволяє відстежувати історію проекту, швидко відкотитися до стабільних версій, реалізовувати гнучку стратегію релізів та

організовувати робочі процеси із залученням великої кількості розробників.

В ігровій індустрії Git став стандартом для зберігання коду, blueprint-файлів, документації, а також для управління тестуванням та релізами. У межах цього проекту Git використовується без підключення Large File Storage (LFS) — великі бінарні файли зберігаються окремо, що дозволяє уникнути надлишкового навантаження на систему та забезпечує стабільність роботи навіть із великими обсягами даних [21].

2.6 Інструмент проєктування інтерфейсів

Figma є сучасним хмарним сервісом, орієнтованим на проектування інтерфейсів користувача та створення інтерактивних прототипів. Від моменту запуску у 2012 році, сервіс, розроблений Діланом Філдом та Еваном Воллесом, здобув велику популярність завдяки доступності через веб-браузер, багатому функціоналу й зручності для спільної роботи дизайнерських команд [37].

Важливою перевагою Figma є можливість працювати над макетами та дизайнами без встановлення додаткового програмного забезпечення: для доступу достатньо лише інтернет-з’єднання і сучасного браузера. Це значно полегшує спільну розробку інтерфейсів, особливо у розподілених командах, оскільки всі зміни зберігаються й синхронізуються у реальному часі.

Сервіс також містить розвинені засоби для створення інтерактивних прототипів із різноманітними анімаційними ефектами та моделями взаємодії, що дозволяє проводити швидке тестування та вдосконалення концепцій безпосередньо в середовищі Figma, без необхідності використання сторонніх програм.

Окремо варто відзначити підтримку дизайн-систем: Figma надає інструменти для централізованого зберігання стилів, візуальних компонентів та інших елементів, які можуть бути багаторазово використані у різних проектах.

Це спрощує стандартизацію інтерфейсів та пришвидшує розробку нових функціональних модулів, забезпечуючи цілісність та уніфікований вигляд усіх екранів додатку.

2.7 Організація процесу розробки

Завдяки гнучкій інтеграції між Unreal Engine, Visual Studio, Blender, Paint.NET та Git, вдається побудувати ефективний пайплайн розробки. Моделі, текстури та ассети створюються у Blender і Paint.NET, після чого експортуються у форматах, придатних для імпорту в Unreal Engine. Коди ігрової логіки пишуться у Visual Studio, тестуються і налагоджуються через інтеграцію з рушієм. Проєктування дизайну інтерфейсів було здійснено за допомогою Figma. Усі зміни фіксуються у Git, що дає змогу організувати як індивідуальну, так і командну роботу, забезпечити збереження проміжних та релізних версій проекту. Такий підхід дозволяє гнучко масштабувати проект, підтримувати високий рівень якості та забезпечує швидку ітерацію у циклі розробки.

3 ПРОЄКТУВАННЯ

3.1 Проєктування архітектури застосунку

Будь-який проект на Unreal Engine має спільну базову основу, яка складається з декількох частин - авторитарного сервера, що контролює логіку і розрахунки гри, правила якими будуть слідувати всі під’єднані клієнти в багатокористувацькому режимі. Спеціальні окремі функції які називаються RPC функціями відповідальні за передачу між сервером та клієнтами. Та клієнт, що відповідальний за коректну роботу інтерфейсу та відображення світу гри, який є відображенням того що передає клієнту сервер.

Таким чином відбувається ефективне розділення програмної логіки, що дозволяє не лише забезпечити мультиплеер, але також дозволяє ефективніше

розподілити програмний код, але також убезпечити ігрову сесію від чітерства з боку гравців, так як сама гра буде орієнтуватися лише на дані які створені на сервері.

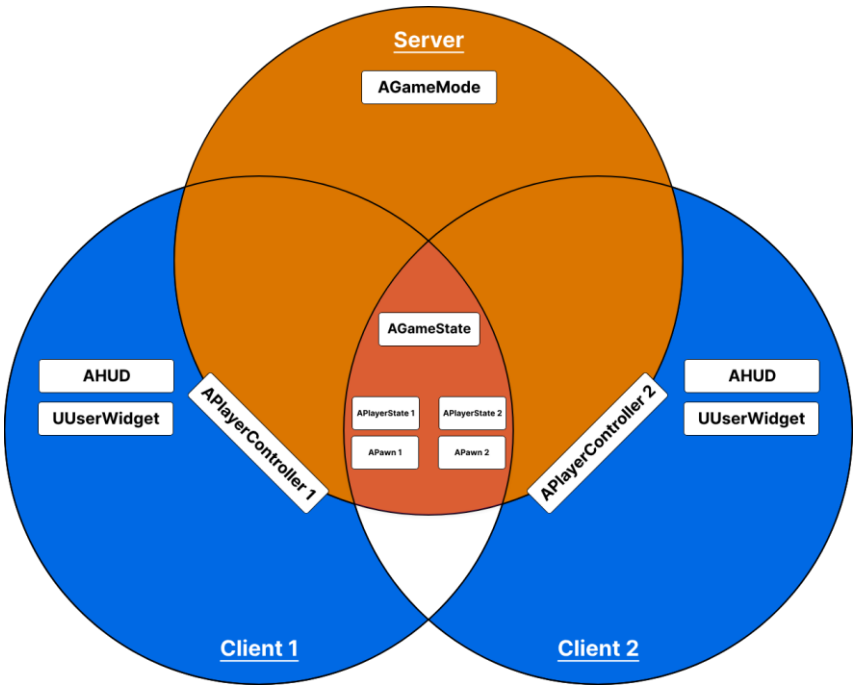


Рис. 3.1 Діаграма відношень між головними класами програми, що відображє взаємодію між сервером та клієнтами [22]

Як видно на рисунку 3.1 класи віджетів - спеціальних класів орієнтованих на малювання інтерфейсу гравця, існують лише на клієнтах. Клас APlayerController є зв'язним елементом між клієнтом на сервером, відносно так званого права Власності сервер визначає, чи буде вислухана команда клієнта, чи проігнорована. Таким чином, наприклад, злочинна спроба викликати позбавити ворога ресурсів викликом команди за допомогою модифікованого клієнта буде відхилена, якщо дану команду застосував клієнт який не має права Власності на клас з держави данного гравця.

Враховуючи ці особливості розробки, програму можна розділити на наступні важливі компоненти:

1. Архітектура ігрового режиму, що відповідальна за логіку та

обрахунки ігрового процесу.

2. Архітектура станів гри, перемикання яких визначає протікання ігрового процесу.

3. Система ігрового часу, що визначає всі темп проходження та специфіку економіки, бойових дій та будівництва.

4. Архітектура бази даних, де відбувається збереження всіх необхідних значень для подальшої роботи гри.

3.2 Архітектура ігрового режиму

“Серверні”, або внутрішні класи представляють собою ті класи, що існують виключно на сервері, або можуть бути редаговані виключно сервером. Такі класи, через їх захищеність від небажаного втручання є фундаментальним елементом програми, адже вони, будучи доступними з усіх інших класів, дозволяють отримати останню актуальну інформацію з мінімумом непотрібних обрахунків або інструкцій.

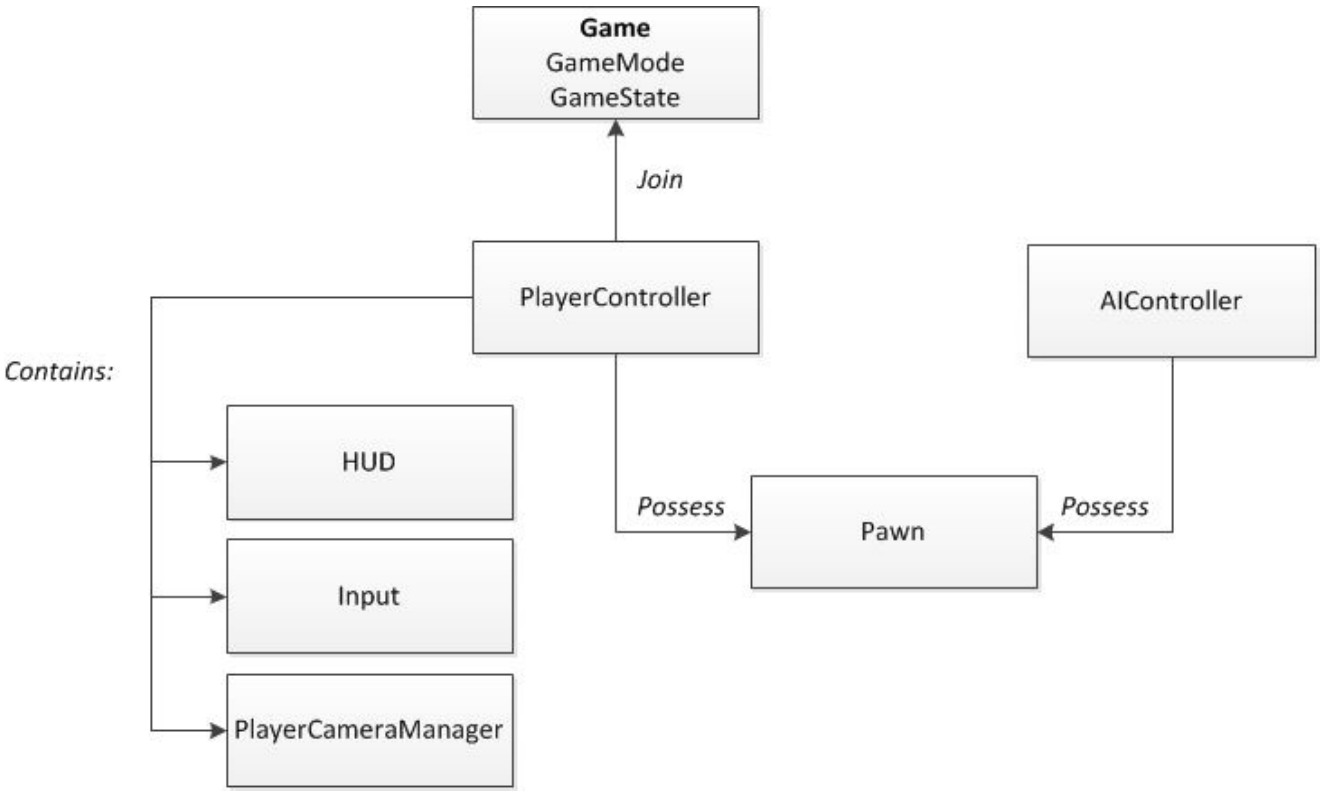


Рис. 3.2 Діаграма зв’язків базових класів у Unreal Engine [23]

Клас ігрового режиму грає ключову роль у програмі, адже саме він керує системи мультиплеєру, створенням персонажа/актора гравця та визначає які ключові класи будуть застосовано - таких як HUD, PlayerPawn, GameState, PlayerState та GameInstance. Значення PlayerController, тобто представлення гравця в одиночній та багатокористувацькій грі відображено на рисунку 3.2.

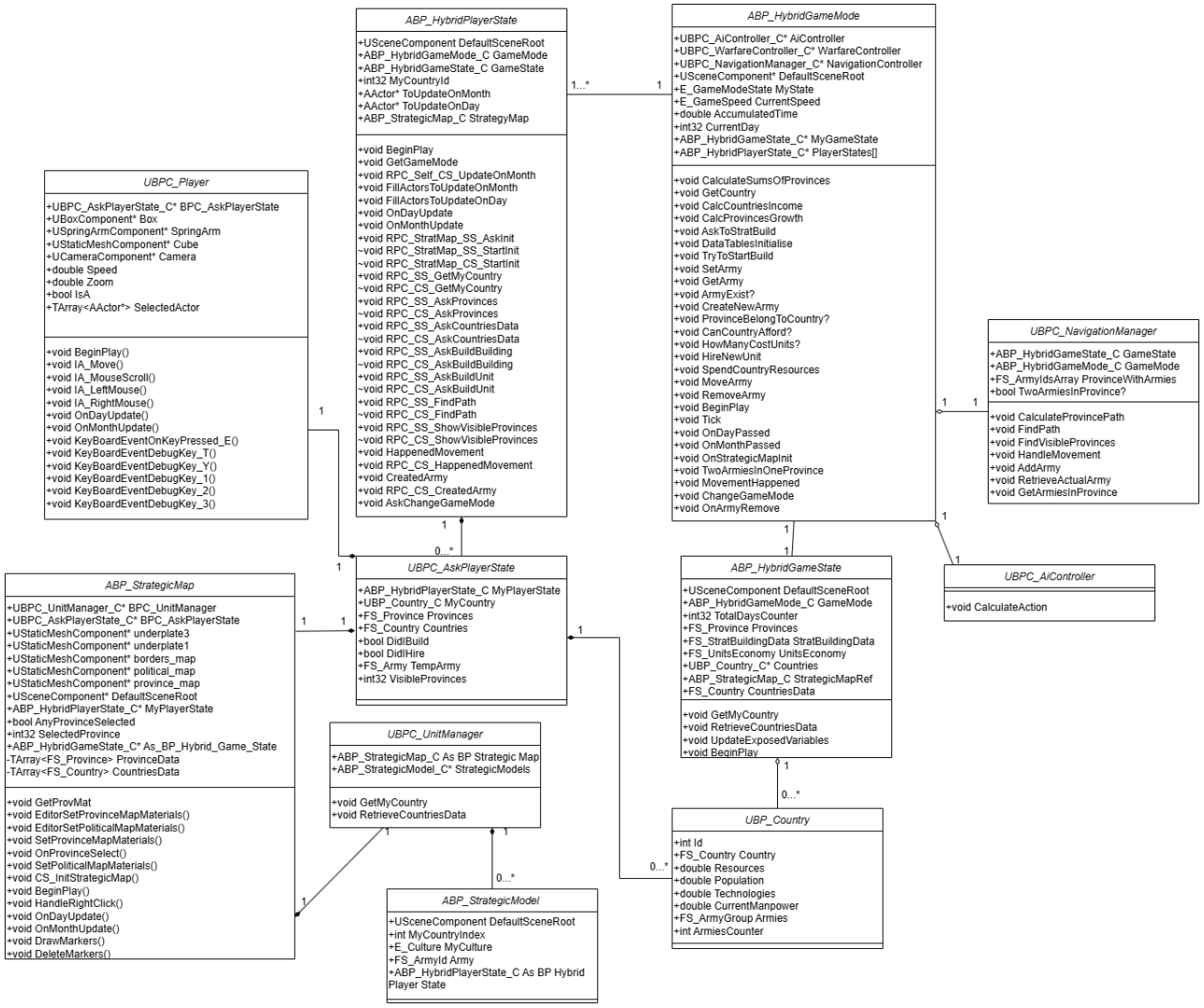


Рис. 3.3 Діаграма класів з найважливішими класами гри та їх членами

Саме тому GameMode є першочерговим місцем де відбуваються обрахунки необхідні для роботи гри, що винесені в окремі функції, класи або події (особливі функції UE, що не повертають значень). Найважливішим аспектом є відсутність реплікації, тобто копіювання всіх значень класу на всі копії цього класу у інших гравців. Діаграма на рисунку 3.3 відображає центральне значення GamMode.

Як єдине місце для збереження всіх необхідних даних для класів гри було використано клас GameState, адже з однієї сторони об’єкт цього класу реплікується на всі об’єкти інших класів, в той час як інші класи не здатні змінювати його версію на сервері - таким чином забезпечується доступ до

спільних даних без ризику небажаних змін критичних даних.

Останнім важливим класом є PlayerState, що представляє реплікований статус гравця у грі. Це дозволяє різним клієнтам мати спільне розуміння відносно даних що мають бути публічними - якому гравцю яка країна належить, які її кордони, тощо.

Взаємодія між GameMode, GameState та PlayerState є ключовим аспектом роботи проекту.

3.3 Архітектура станів гри

Так як проєкт має на меті з'єднати одночасно геймплей глобальної стратегії та RTS, це означає серйозну змінну необхідної логіки, адже різні режими потребують різних обчислень. У такому випадку велика кількість провінцій, юнітів та ефектів можуть серйозно навантажити комп’ютер гравця, що буде мати найсерйозніші наслідки - адже саме технічні обмеження впливають на концепцію більшості аналогічних гібридів. Зазвичай для збереження продуктивності відбувається вивантаження глобальної карти та завантаження тактичної (наприклад Total War або Empire At War).

Проте заслуговує на увагу концептуально інше рішення - замість використання кардинально різно реалізованих та існуючих частин гри, було прийнято рішення у даному проекті реалізувати в рамках одного рівня обидва режими за допомогою використання патерну “Стан”. Принцип відображений на рисунку 3.4.

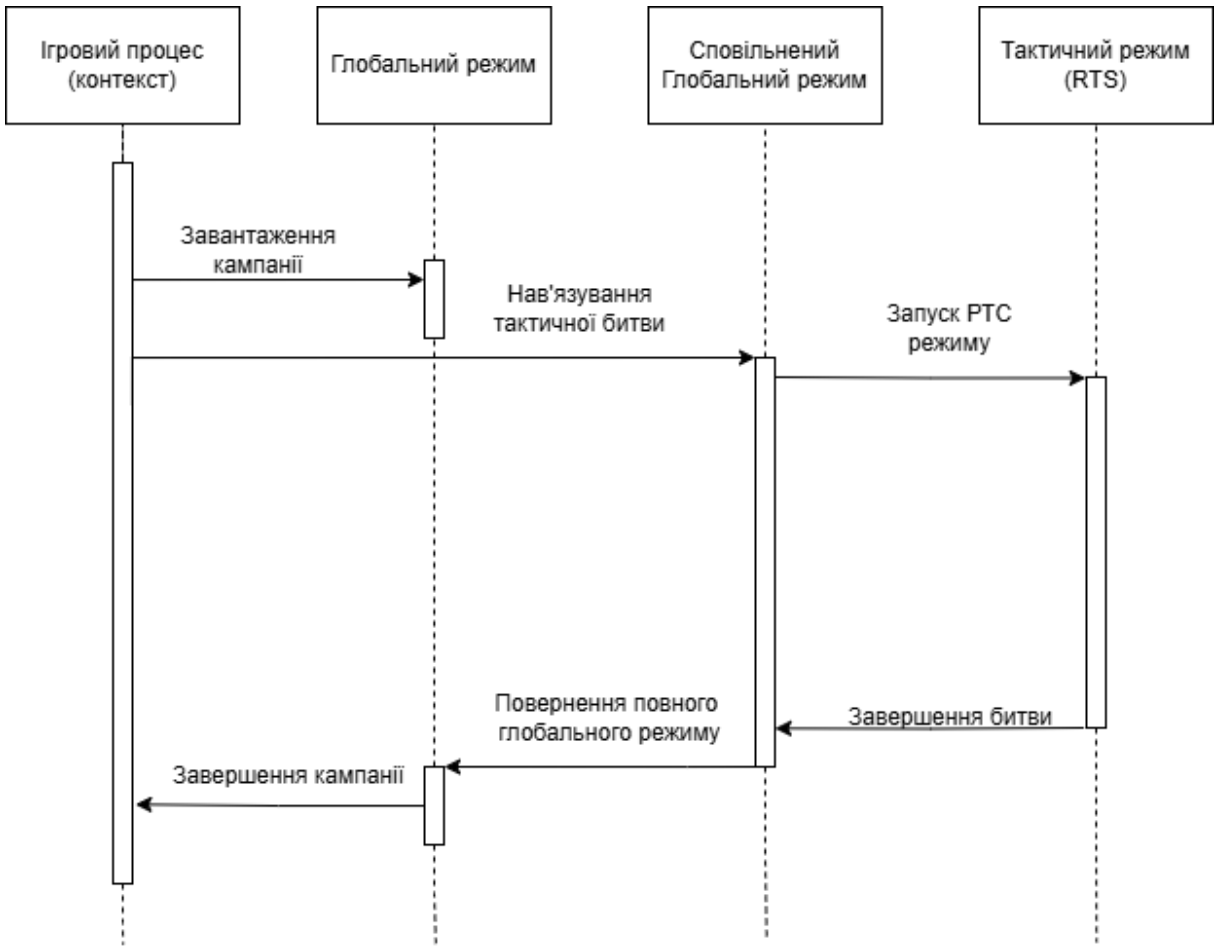


Рис. 3.4 Діаграма послідовності зміни станів

Починаючи на рівні архітектури розділення функціоналу для стратегічного та тактичних режимів, було розроблено концепцію, де в рамках певного стану відбувається певний набір інструкцій. У випадку зміни стану, гра починає діяти за іншими алгоритмом, що дозволяє економити ресурси на безпечному зупиненні непотрібних процесів.

Одночасно з тим заплановано вимикання рендерингу непотрібних частин рівня за допомогою технології Level Streaming. При зміні стану з тактичного на стратегічний, то гра перестає обрахунки та промальовувати RTS мапу. При зворотному обміні відбувається зворотній процес - обрахунок стратегічної логіки призупиняється.

3.4 Система ігрового часу

Ключовим аспектом також є розрахунок на власну систему часу. У будь-якому класі UE існує функція Tick, що викликається кожний фрейм (кадр). Проблема полягає в тому, що в залежності від величини функції, вона довше починає обраховуватися. Так зменшення кадрів в секунду може призвести до неприпустимого змінення логіки. Так можливості гравця відображені на рисунку 3.5, де його можливості використання програми залежать від стану гри.

Щоб запобігти такому розвитку подій, була спланована власна система внутрішньо ігрового часу, яка дозволяє всередині гри орієнтуватися на те, скільки часу має пройти, в незалежності від кількості кадрів. Це зручно необхідно для реалізації фундаментальних механік, таких як переміщення, будівництва чи битв.

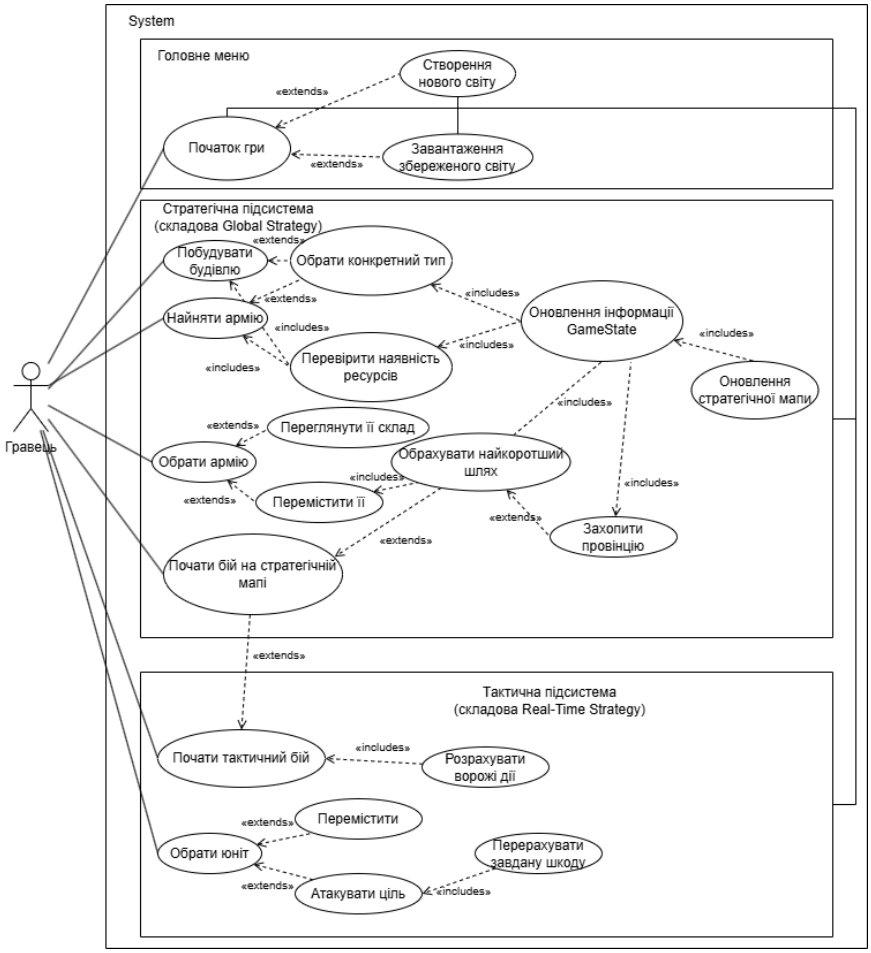


Рис 3.5 Діаграма варіантів використання

Тому раз в кожну чітко визначену кількість реального часу проходить умовна одиниця часу, в цьому конкретному випадку - день. Кожну певну кількість днів відбувається місяць. Кожен день і місяць визначенні значення мають оновлюватися.

Завдяки дискретності ігрового часу є можливість планувати, та надалі реалізувати чітку і зрозумілу систему економіки, будівництва та бойових дій.

3.5 Архітектура бази даних

В UE існує власна система DataTables що здатна задовольнити потреби у базах даних невеликого розміру. Використовуючи цю систему було вирішено записати найважливіші дані, що використовуються для роботи гри.

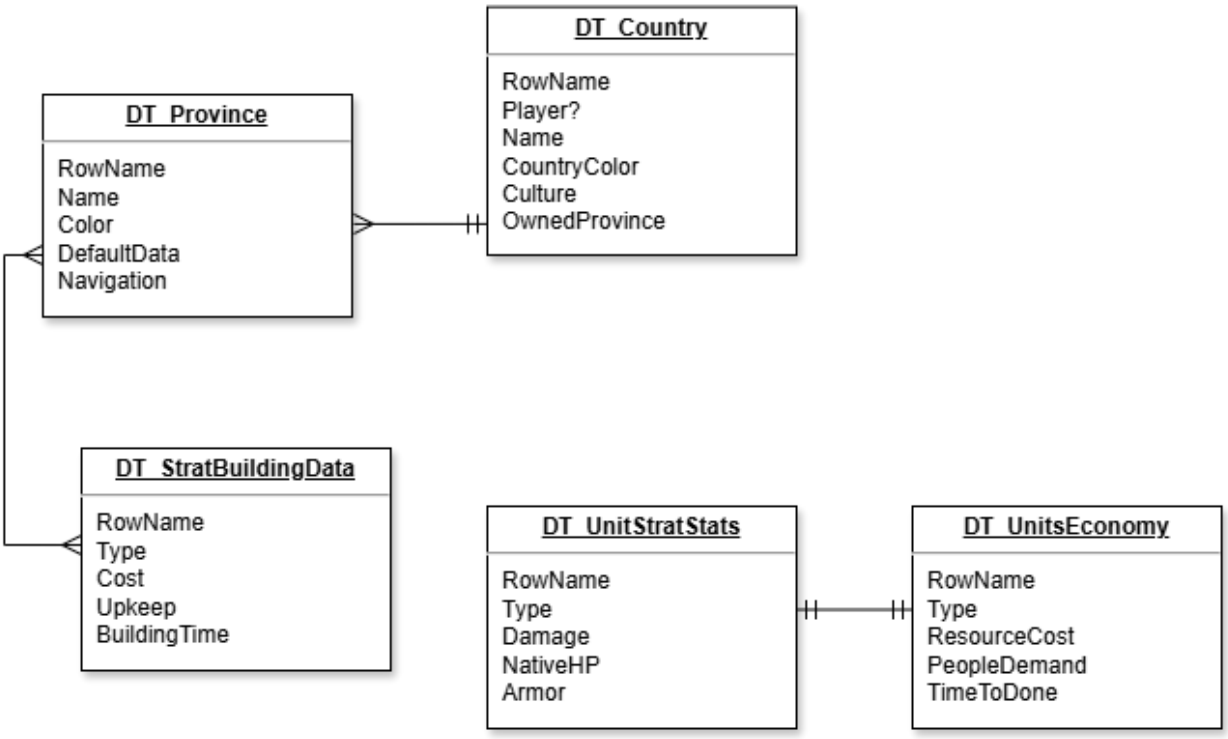


Рис. 3.6 Модель «сутність — зв'язок» бази даних

На рисунку 3.6 можна побачити систему зв’язків між структурами які використовуються в програмі. На старті гри дані з даної бази даних завантажуються в класи гри для подальшого використання.

4 ЗАСОБИ РЕАЛІЗАЦІЇ

4.1 Дизайн інтерфейсу

Figma є сучасною платформою для створення користувацьких інтерфейсів і прототипів, що забезпечує зручне середовище для розробки інтерактивних макетів і колективної роботи над дизайном. Нижче подано опис типових етапів підготовки UI у Figma.

1. Початок роботи з проектом: — Зареєструйтесь у Figma: необхідно створити обліковий запис або виконати вхід, якщо профіль уже існує. — Відкриття нового файлу: на стартовому екрані оберіть опцію створення нового проекту («New File») для ініціалізації робочої сесії.

2. Робота з фреймами. У Figma фрейми виконують роль контейнерів для різних елементів, визначаючи структуру окремих екранів чи блоків інтерфейсу. — Додавання фрейму: скористайтеся інструментом Frame (або клавішею F) і окресліть фрейм у робочій зоні. — Налаштування параметрів: у властивостях фрейму можна встановити стандартний розмір для типових пристроїв (смартфон, планшет, десктоп) або задати власні значення ширини і висоти.

3. Створення елементів інтерфейсу: — За допомогою інструментів Rectangle, Line, Ellipse та інших можна формувати основні UI-компоненти — кнопки, поля, панелі, меню тощо. — Для додавання тексту використовуйте Text Tool (T) і розміщуйте текстові блоки на потрібних ділянках фрейму. — Додавайте зображення та іконки шляхом імпорту із локального ПК або використовуйте вбудовану бібліотеку ресурсів Figma.

4. Використання компонентів: — Компоненти у Figma створюються з груп елементів, які будуть повторюватися у різних частинах інтерфейсу. Щоб

зробити компонент, виділіть потрібну групу, натисніть праву кнопку миші й оберіть «Create Component» (або натисніть Ctrl+Alt+K). — Для багаторазового використання перетягуйте компоненти з бібліотеки у поточний проект, оновлюючи їх за потреби — всі екземпляри будуть синхронізовані.

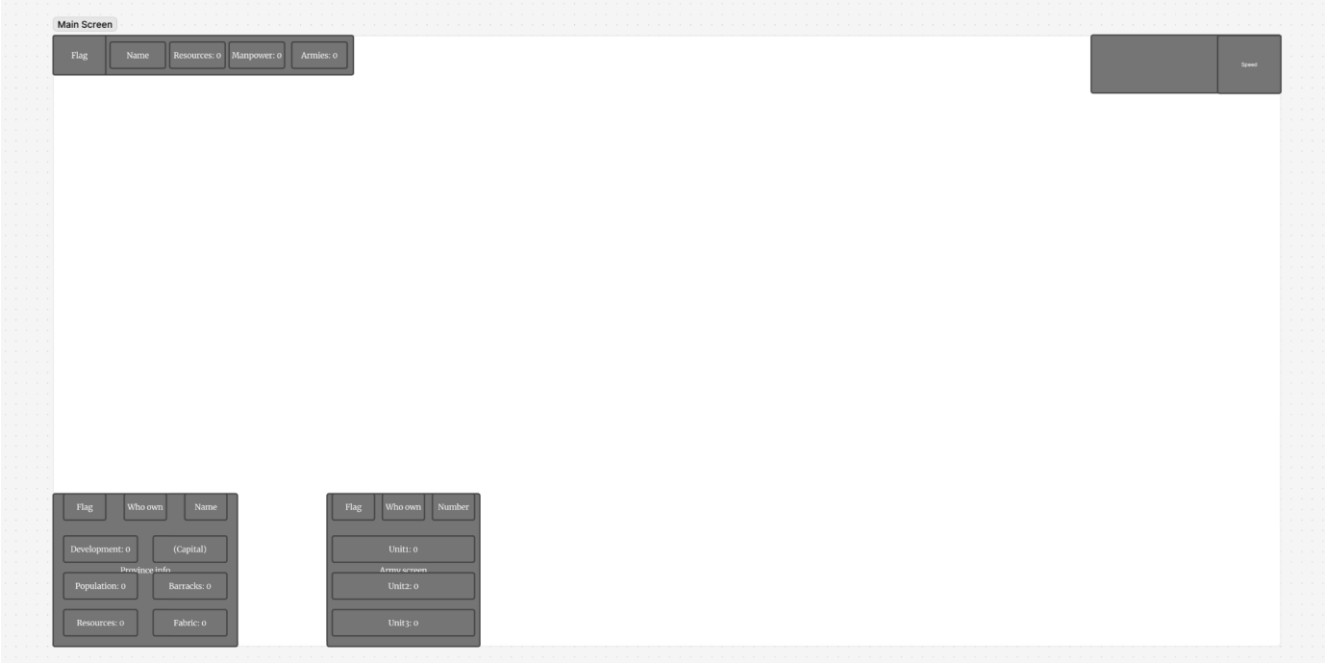


Рис. 4.1 Прототип HUD гравця у Figma

Завдяки використанню Figma дозволило кратно швидше реалізувати інтерфейс для користувача. В Unreal Engine інтерфейси реалізуються за допомогою інтегрованого у рушій фрейморку Slate, який здатний швидко та ефективно створювати складні інтерфейси. Створений прототип є на рисунку 4.1. На рисунку 4.2 видно як було реалізовано віджет у рушії.

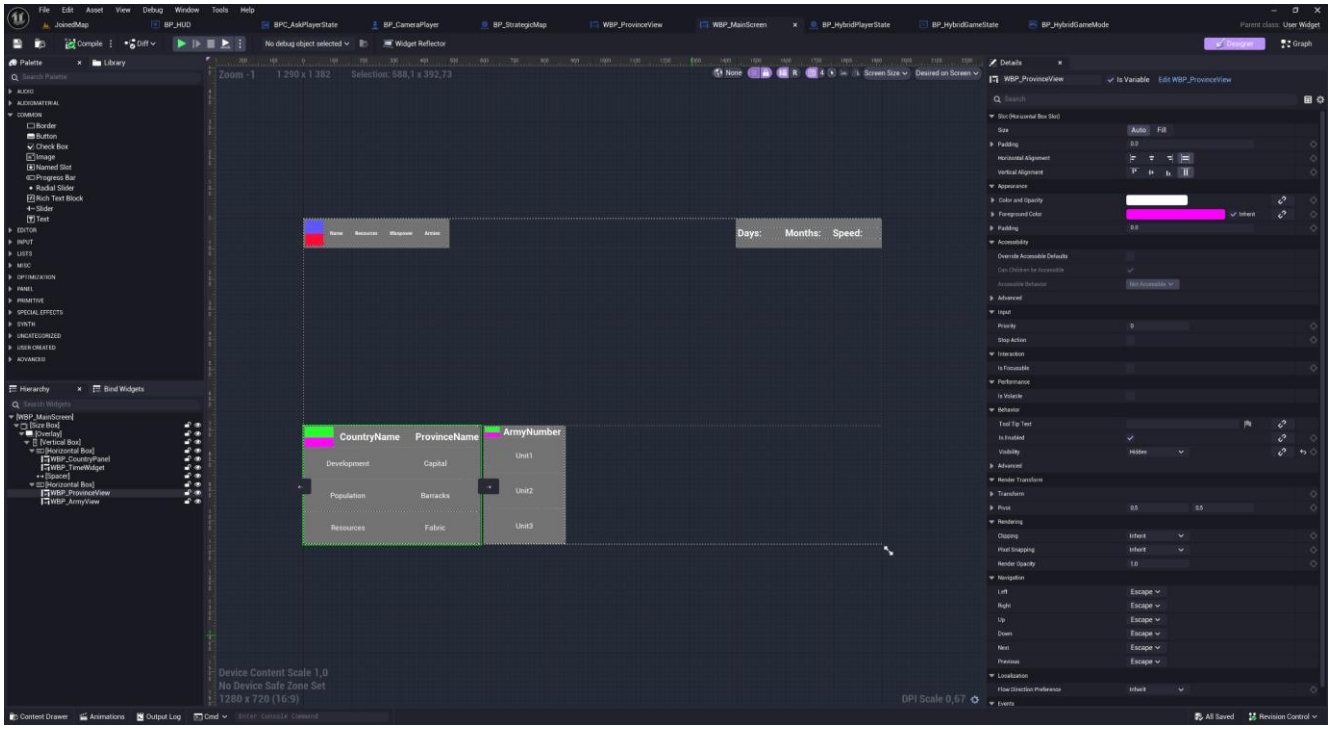


Рис. 4.2 Зображення інтерфейсу у самому Unreal Engine

Для того щоб в Unreal Engine додати віджет на екран користувача, в першу був створений клас BP_HUD, який є частиною PlayerController при створенні, та дозволяє тримати логіку інтерфейсів поза основного класу PlayerController, що і забезпечує всі функції потрібні для представлення гравця.

В BP_HUD ми додаємо функції необхідні для того ініціалізації компонентів, а також в подальшому будемо використовувати клас BP_HUD для зв’язку між віджетами та основною грою. Реалізація BP_HUD на рисунках 4.3 та 4.4.

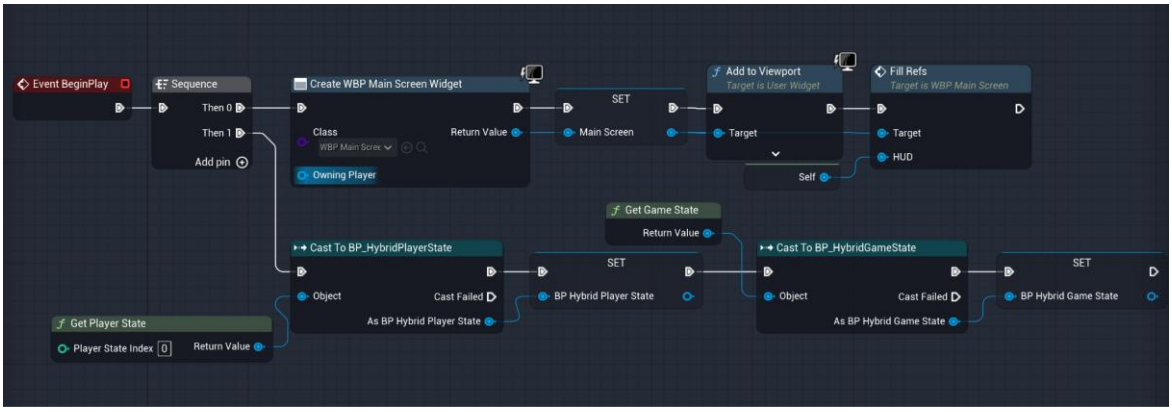


Рис. 4.3 Зображення події BeginPlay всередині BP_HUD

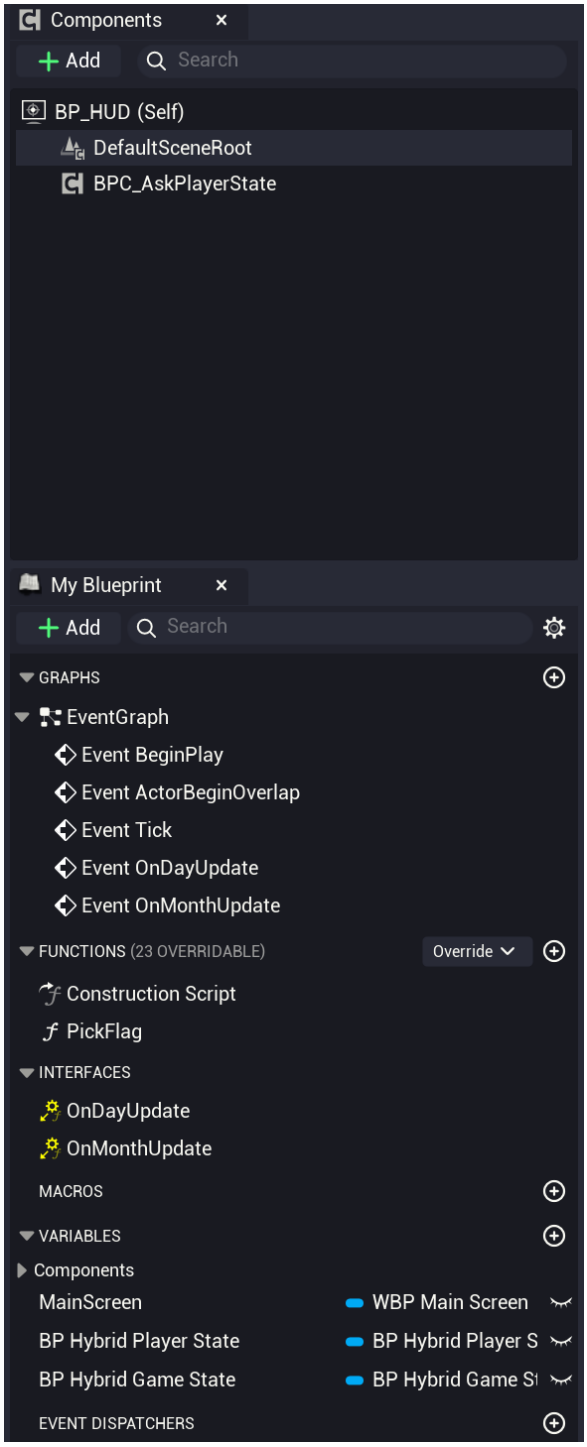


Рис. 4.4 Зображення компонентів, подій, функцій, реалізації інтерфейсів та змінних класу BP_HUD

Кожен з елементів класу WBP_MainScreen - це окремі віджети, що мають власну структуру та методи. Це необхідно для організації коду, та його

перевикористання за потреби. Завдяки розділенню на менші субкласи, з’являються гнучкіші можливості у використанні гарно продуманих окремих елементів коду. Відображення заповненого віджету під час робочого процесу є на рисунках 4.5, 4.6, 4.7

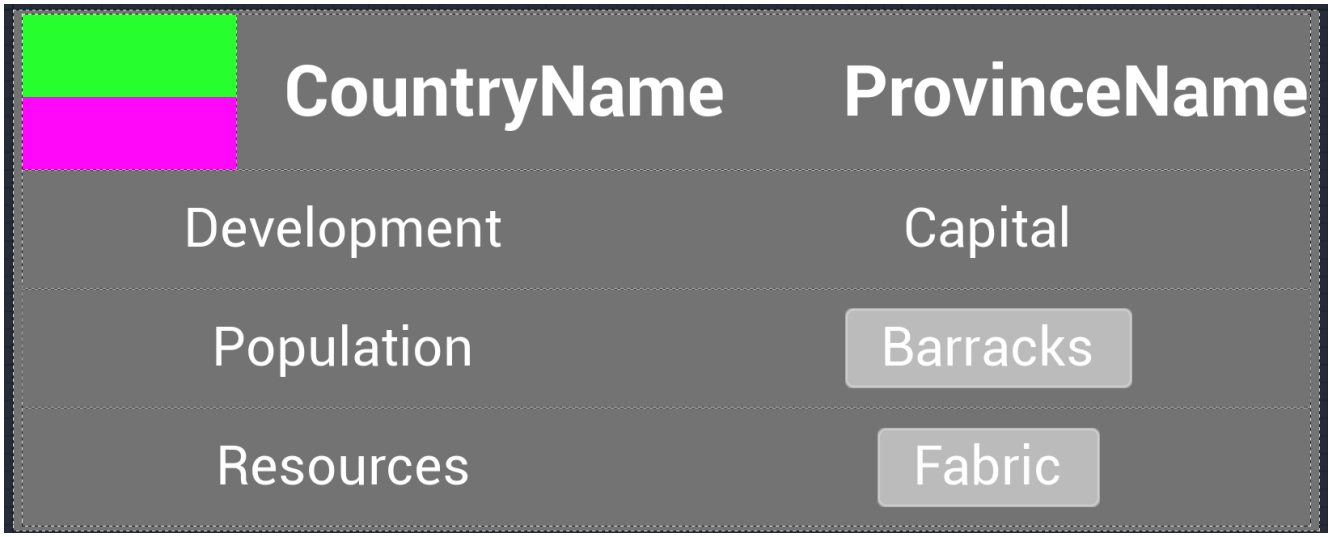


Рис. 4.5 Зображення субвіджета WBP_ProvinceView у дизайнері віджетів

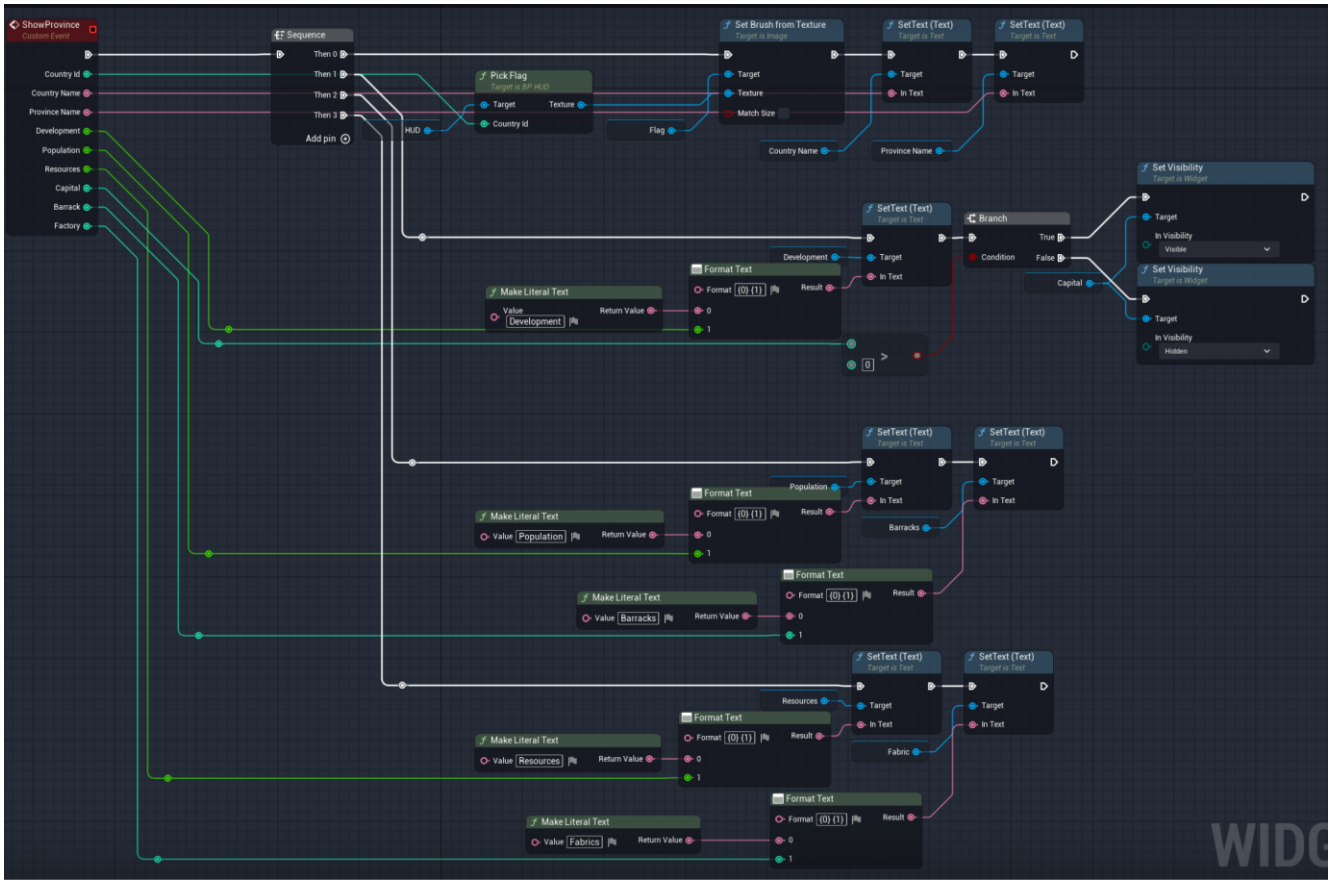


Рис 4.6 Зображення події ShowProvince, що заповнює віджет при натисканні на провінцію стратегічної мапи

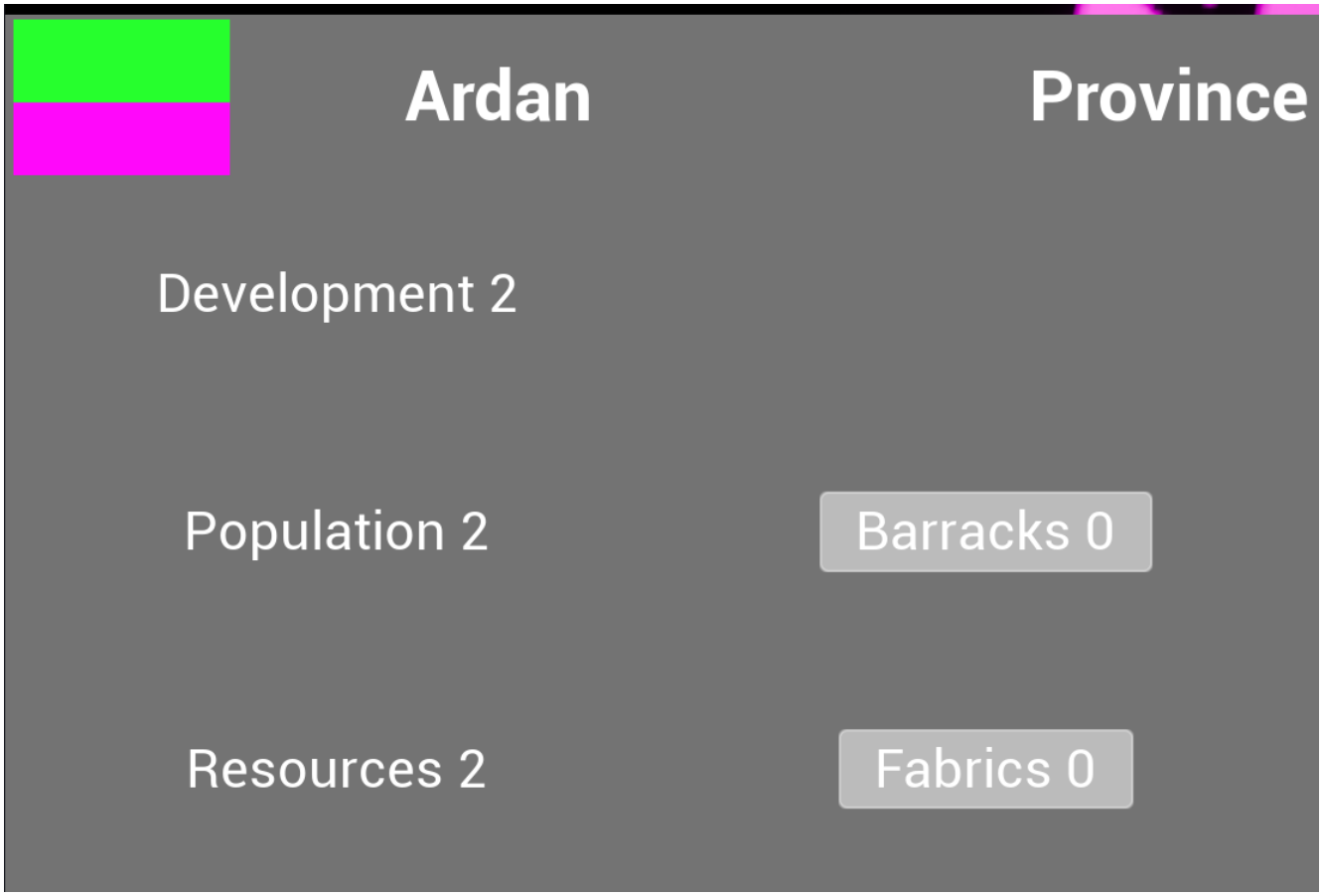


Рис 4.7 Зображення WBP_ProvinceView при обраній провінції під час гри

Одна з найважливіших функцій у віджетах є SetVisibility, що дозволяє обрати правила по яким буде малювати елемент віджета. Це може бути як Hidden, що прибирає малювання елементу, але залишає в розмітці місце під нього. Це може бути Collapsed, що прибирає елемент з розмітки, або Not Hit-Testable - це означає що елемент буде відображатися, але з ним неможливо буде взаємодіяти.

На прикладі з Рис 4.5 та 4.7 - можна побачити що так-як провінція не є столицею, немає відповідного тексту що ідентифікує її як таку. Тобто немає необхідності створювати окремий віджет, чи змінювати навіть напис - достатньо лише його прибрати якщо він не задовольняє фактичному стану речей.

Це робить систему інтерфейсів гнучкою та інтерактивною.

4.2 Стратегічна карта

Внаслідок тривалих роздумів, та аналізу аналогів, було прийнято рішення повторити досвід глобальних стратегічних ігор з картою що має зміну часу у реальному часі. Мова йде про систему провінцій з ігор Paradox Interactive, які і надихнули на концепцію використану нижче. Проте реалізація була виконана абсолютно самостійно, враховуючи кардинально інший двигун від їх ігор.

Першим кроком розробки було намалювання карти у Paint.Net, де після завдяки інструментам редактора зображень був намальований спочатку контур майбутньої карти. Далі він був розділений на багато різноманітних провінцій неправильної форми, для можливості складного пересування, а значить унікальних стратегічних маневрів. Далі кожній провінції був наданий власний колір для їх подальшої обробки. Артефакт розробки на рисунки 4.8.

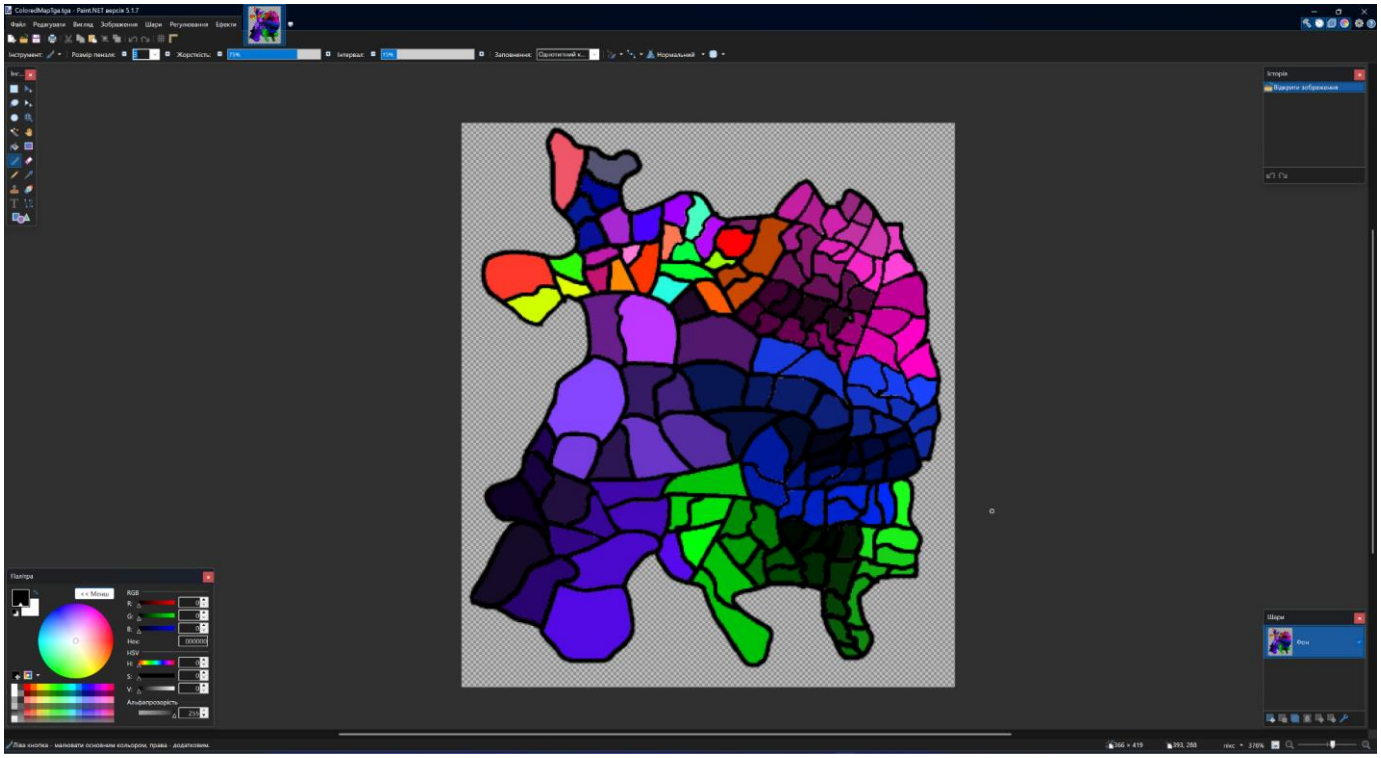


Рис. 4.8 Зображення карти провінцій у Paint.Net

Наступним кроком було створення 3D моделі карти у Blender, де спочатку був взятий звичайний Plane, а після методично в режимі Edit були викарбувані необхідні межі сторін (faces). Для кожної сторони був створений окремий меш, що буде ключовим на наступному етапі в UE. Також у Блендер були налаштовані матеріали для коректного відображення провінцій. Таким чином було створено 158 матеріалів і провінцій, кожна з’єднана з іншими, та кожен має власний колір. Рисунок 4.9 відображає модель у Blender.

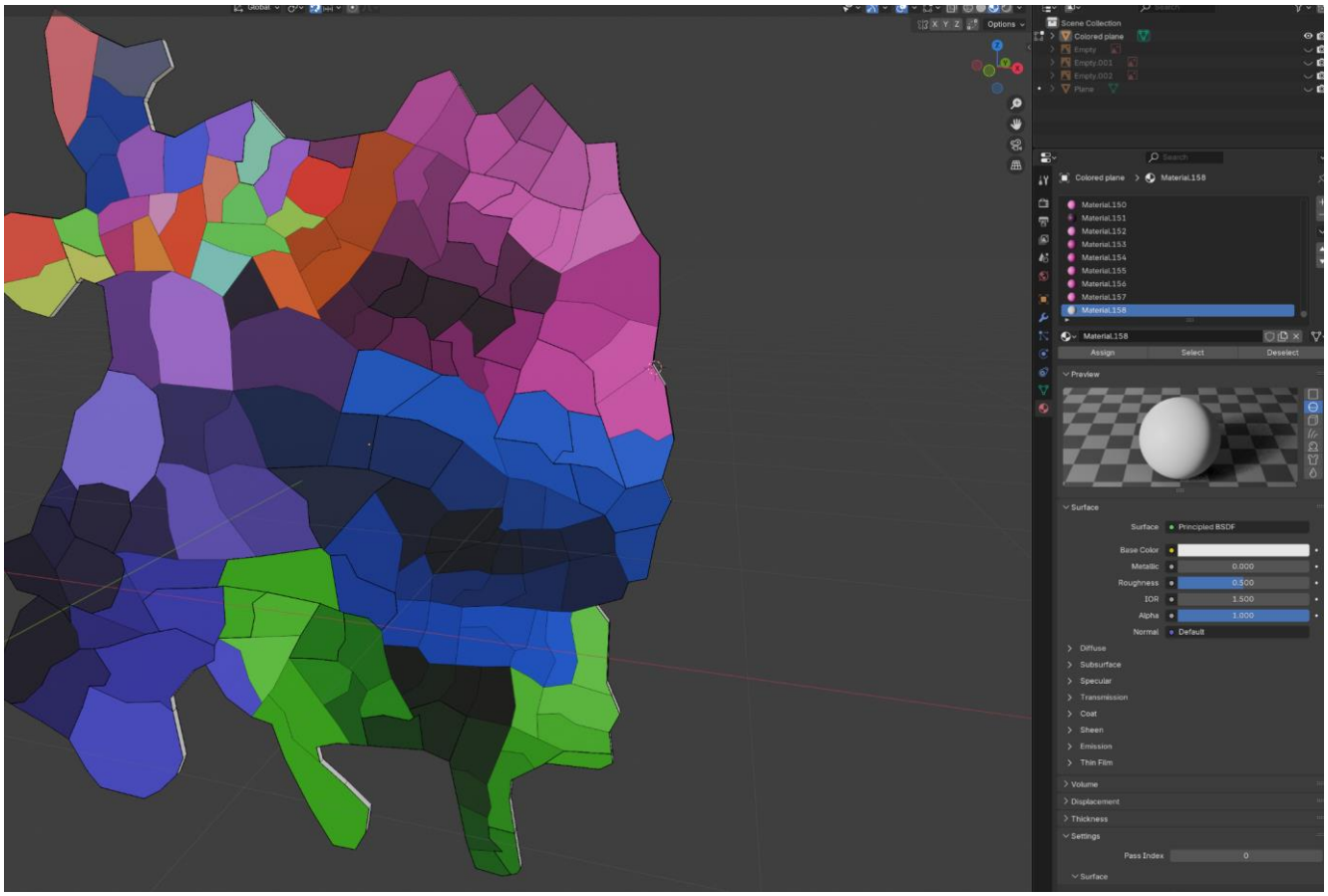


Рис. 4.9 Модель карти провінцій у Blender

Надалі в самому Unreal Engine був використаний DynamicMaterialInstance - він дозволяє створювати велику кількість матеріалів у runtime, що є надзвичайно корисно для розробки. В клас самої стратегічної карти було додано

дві моделі карти - одна це карта провінцій, інша це вже політична карта, яку бачить гравець, та на якій і відбуваються зміни під час процесу гри. Static Mesh відображено на рисунку 4.10.

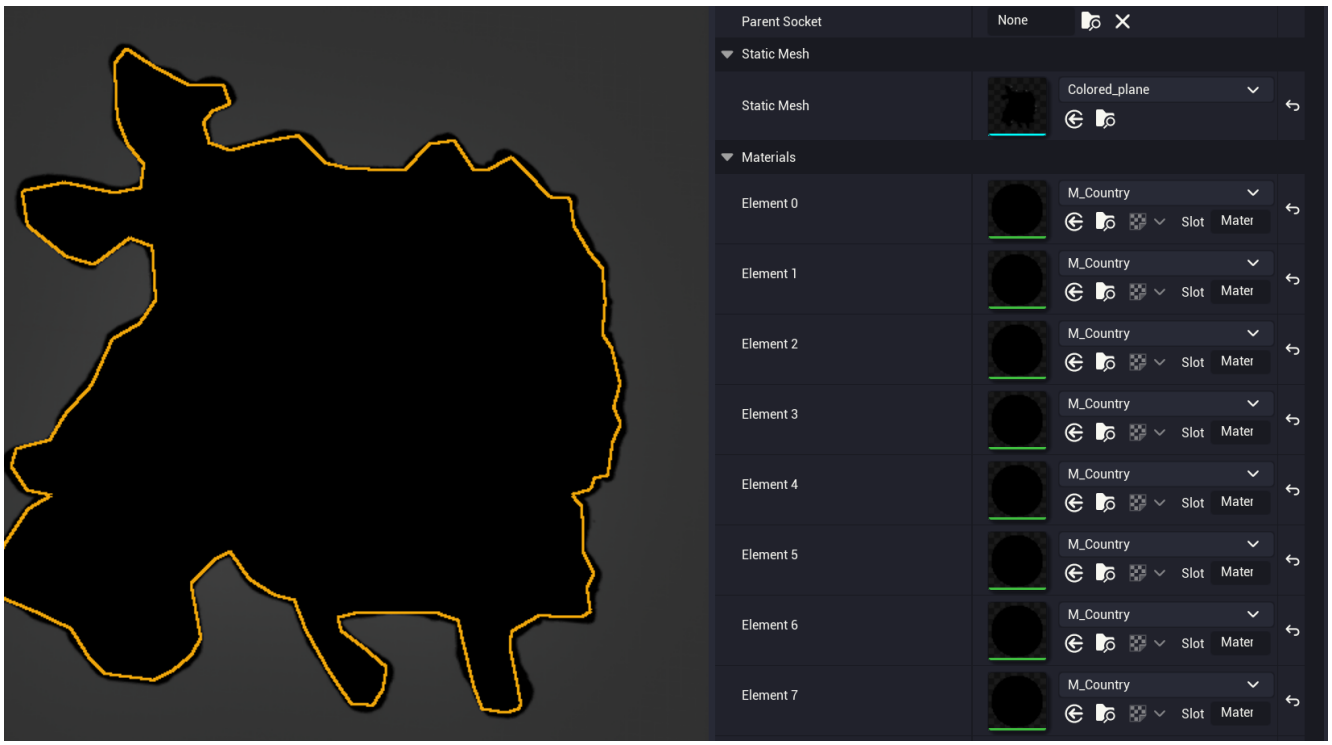


Рис. 4.10 Зображення матеріалів політичної карти - все створена завдяки M_Country

В той же час для правильно відображення юнітів на провінції, модель карти отримала так звані сокети - це точки у просторі які мають певне ім’я та використовують відносне положення від центру моделі, для легшої орієнтації в просторі. Сокети використовуються коли потрібно поставити певного актора у певне місце, та щоб актор коректно знаходився у світі. Зображення сокетів на рисунку 4.11.

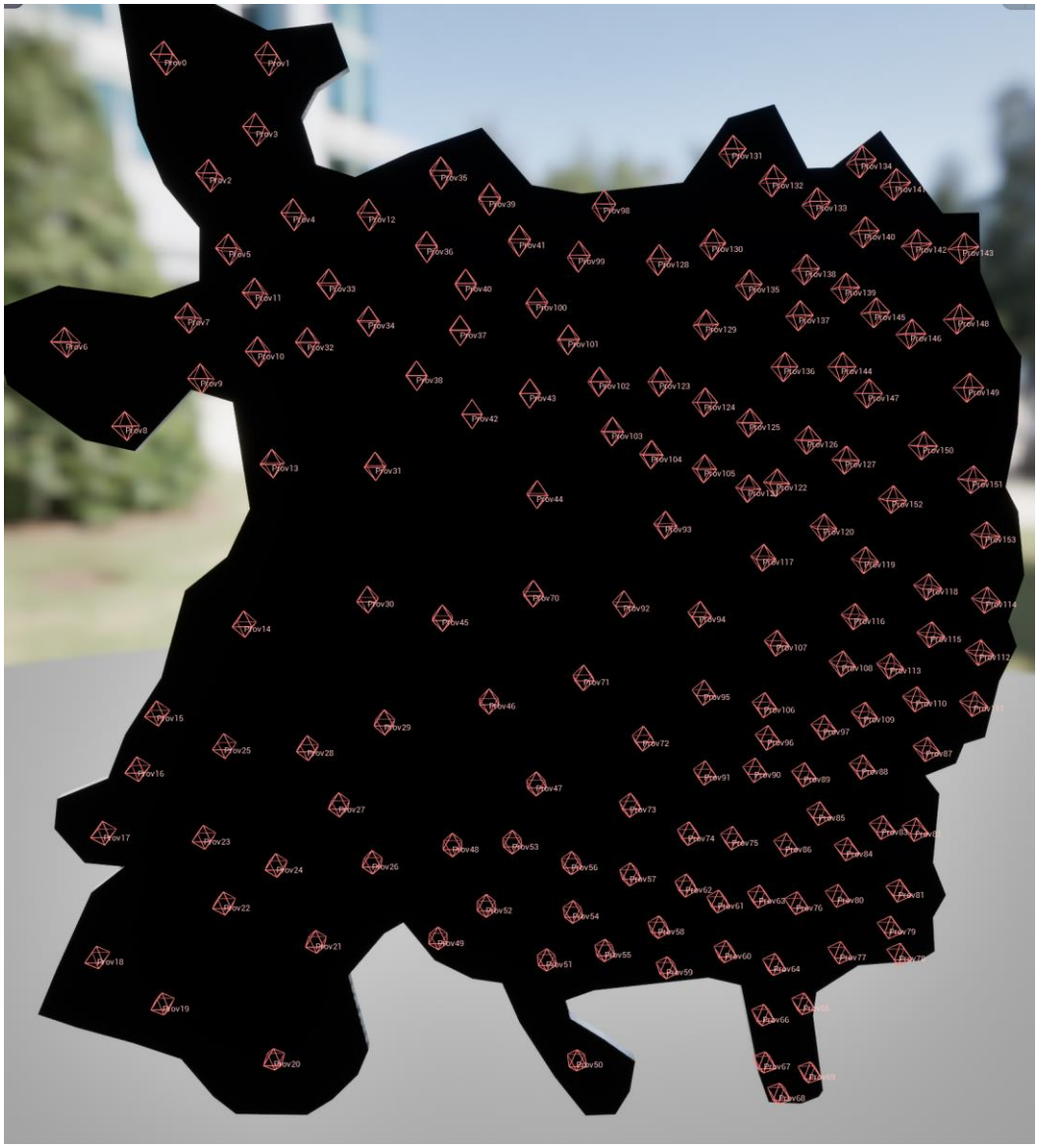


Рис. 4.11 Зображення всіх сокетів на static mesh стратегічної карти

Далі реалізацію було запроваджено через використання line traces, які при знаходженні зіткненням з певним актором намагаються викликати функцію інтерфейсу BPI_HandleMouseClick. У разі влучення по мапі, вона виконає свою імплементацію згаданого інтерфейсу, та відреагує відповідно - обере провінцію та передасть дані до віджету ProvinceInfo. Пайплайн обробки кліку на рисунках 4.12, 4.13. Результат зображено на рисунку 4.14.

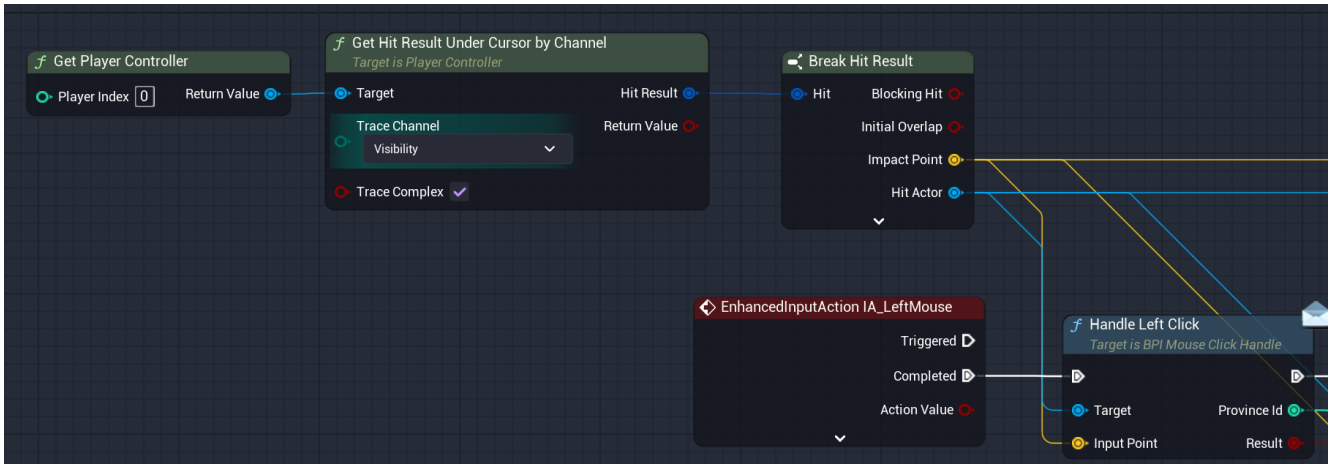


Рис. 4.12 Застосування інтерфейсної функції HandleLeftClick

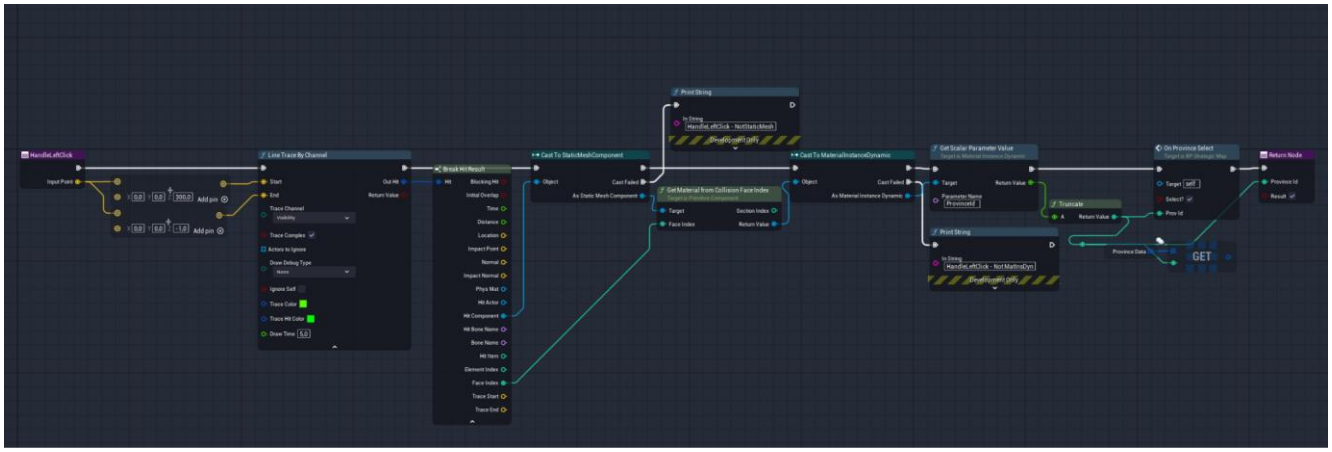


Рис. 4.13 Імплементація інтерфейсу HandleLeftClick у BP_StrategicMap

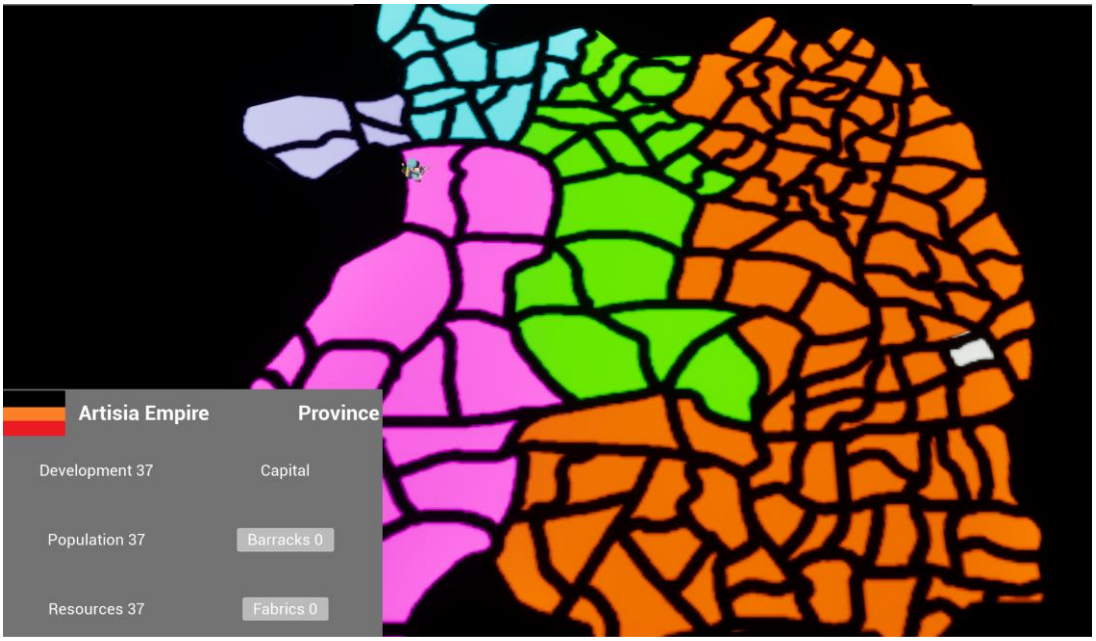


Рис. 4.14 Зображення вибору провінції та інформації у віджеті

Таким чином було реалізована гнучка та глибока система провінцій, що дозволило реалізувати інші фундаментальні механіки на твердій основі.

4.3 Економіка, будівництво

Будь-яка держава складається з власних провінцій, що приносять певних дохід кожен місяць. Кожна провінція має три різних типи властивостей - її ресурси, населення та розвиток, кожна з них несе власну функцію:

- Ресурси визначають скільки в місяць держава отримує ресурсів.
- Населення визначає максимальний розмір армії доступний для створення.
- Розвиток визначає наскільки кожен місяць буде зростати провінція

З іншого боку, як можна побачити з попередніх рисунків, кожна провінція має декілька різних типів будівель, кожна з яких впливає на певним чином на провінцію кожен місяць. Серед будівель існують наступні:

- столиця це унікальна будівля, яка існує лише одна на країну. Якщо столицю втрачено то втрачено і країну. Також столиця надає велике зростання

провінції де вона знаходиться;

- казарми це будівля що збільшує зростання населення кожен місяць;
- фабрика це будівля що збільшує зростання ресурсів кожен місяць.

На рисунку 4.15 зображене відображення інтерфейсом

Рис. 4.15 Відображення побудованих казарм у провінції

Кожна будівля має коштує ресурсів на побудову та на утримання. Всі економічні дії відбуваються за допомогою раніше згаданих RPC функцій у BP_HybridPlayerState, який і відповідає за зв’язок з BP_HybridGameMode, де і відбуваються внутрішні розрахунки.

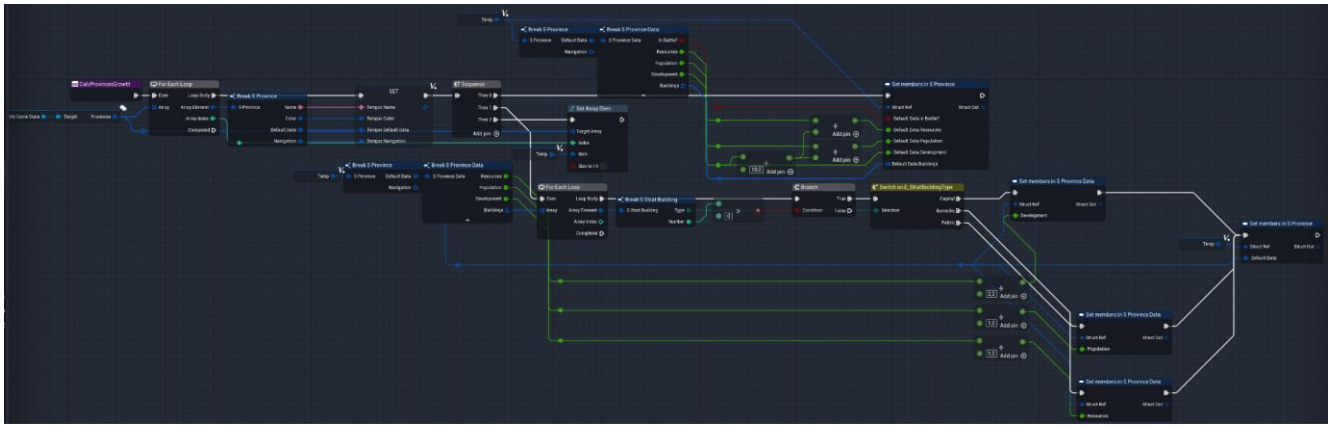


Рис. 4.15 Функція розрахунку зростання провінцій

У випадку якщо країна намагається побудувати щось на іншій території, або якщо вона намагається щось побудувати без потрібних ресурсів - ця країна не може цього зробити, бо BP_HybridGameMode проігнорує тоді її команду.

4.4 Армії

Армії це певне зібрання військ що можна обирати та переміщати вздовж кордонів. При зіткненні з супротивником починається бій, де визначається переможець коли одна з армій повністю гине. Логіка боїв зображена на рисунку 4.16.

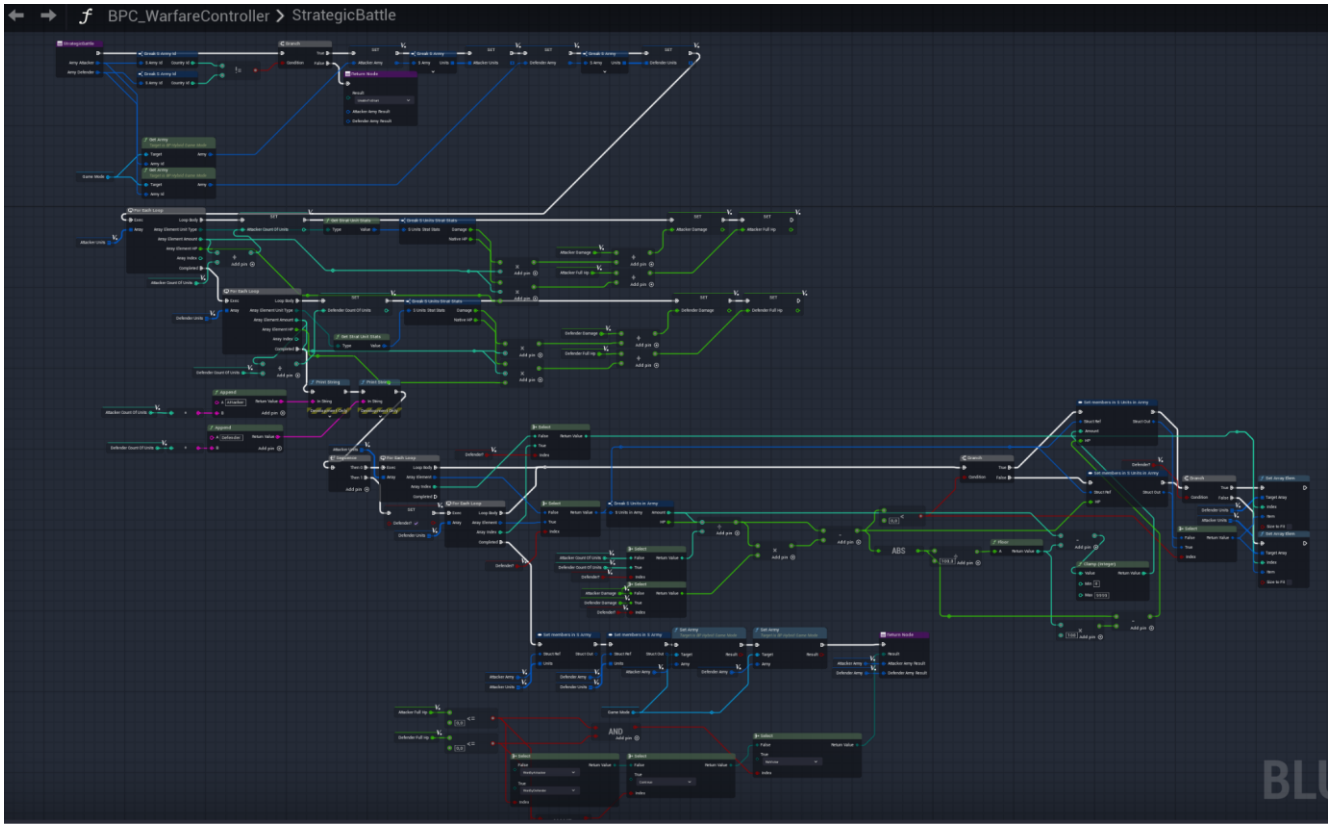


Рис. 4.16 Функція розрахунку битви між арміями

Через великий об’єм, механіки бойових дій та переміщення були переміщені в окремі компоненти: BPC_WarfareController та BPC_MovementController. Компоненти в UE відіграють важливу роль у організації коду та у функціональності. Можливість додати компоненти до будь-якого класу надає можливість їх повторного використання. Приклад переміщення на рисунках 4.17 та 4.18.

Рис. 4.17 Новостворена армія гравця

Рис. 4.18 Переміщена армія після натискання правої клавіші миші

Для створення шляху армій був використаний алгоритм A*, який обирає переміщення в сегмент який найближчий до точки призначення [36]. В нашому випадку використовуються позиції сокетів у світі, які і визначають відстані, між якими обирає алгоритм. Варто, згадати що в даній реалізації алгоритм обирає будь-яку провінцію окрім попередньої та обраної. Таким чином армії здатні обходити вірно обходити непроходиму місцевість, а вигини мапи не приводять рух до нескінченного повернення вперед і назад. Сам метод був зроблений через рекурсію. Реалізація алгоритму на рисунку 4.19.

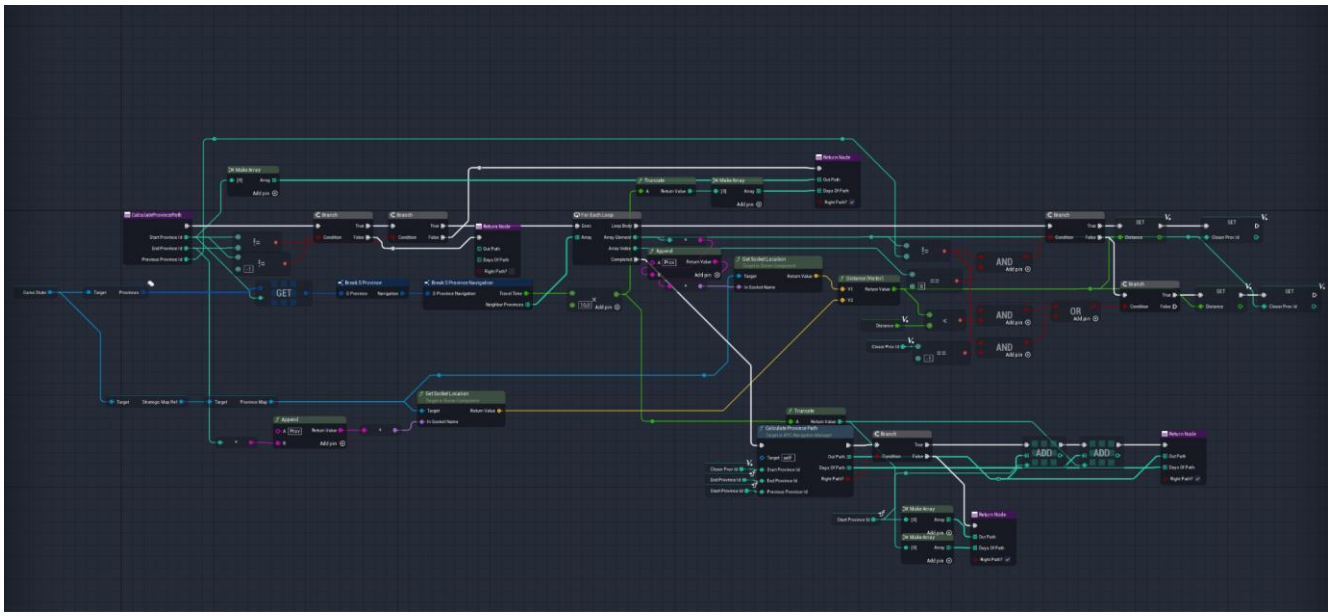


Рис. 4.19 Зображення алгоритму пошуку шляху

4.5 Зміна станів та система часу

Зміна станів та система часу зав’язана на Tick BP_HybridGameMode, як центру який визначає що відбувається у грі у певний момент часу. Швидкості від 0 до 5 визначають швидкість з якою проходить один день. Кожні 30 днів наступає новий місяць. Кожен місяць означає розрахунки зростання провінцій,

накопичення ресурсів держав, тощо. Реалізація зображена на рисунку 4.20.

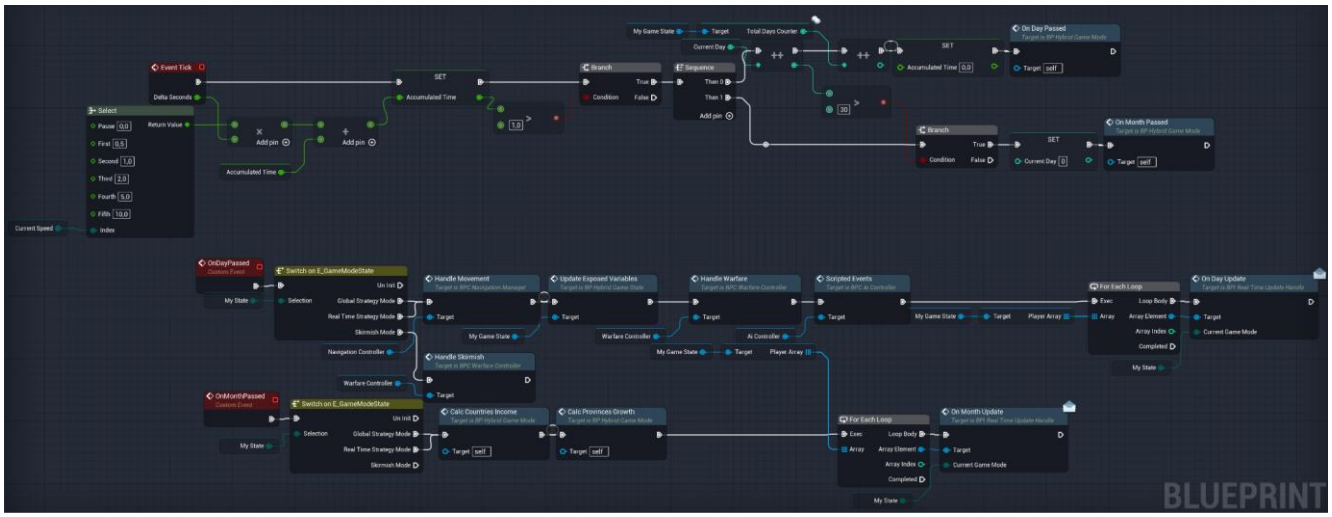


Рис. 4.20 Зображення логіки часу та зміни стану гри

Події OnDayPassed та OnMonthPassed викликають певний набір унікальних функцій, які потрібні для тривалих розрахунків, таких як бої між арміями, переміщення чи звичайне оновлення змінних у BP_HybridGameState. Кожне з існуючих PlayerState володіє масивом елементів які вона має оновити разом з собою. Як видно, в залежності від триваючого стану ігрового режиму, визначається яка логіка обраховується, а яка - ні. Приклад на рисунку 4.21.

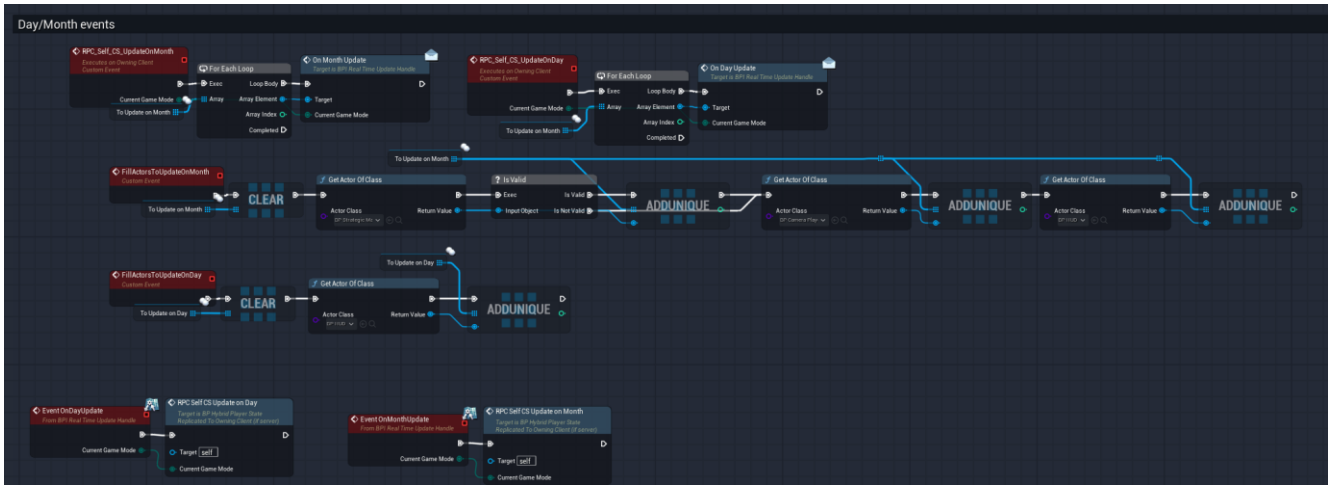


Рис. 4.21 Оновлення акторів всередині BP_HybridPlayerState

В результаті виходить система де оновлення даних відбувається стабільно і постійно, дозволяючи гравцям в реальному часі бачити зміни в грі. Також важливим є використання Level Streaming, що при використанні прибирає з обрахунку непотрібні частини рівня. Робота технології відображена на рисунках 4.22 та 4.23. Реалізація на рисунку 4.24.

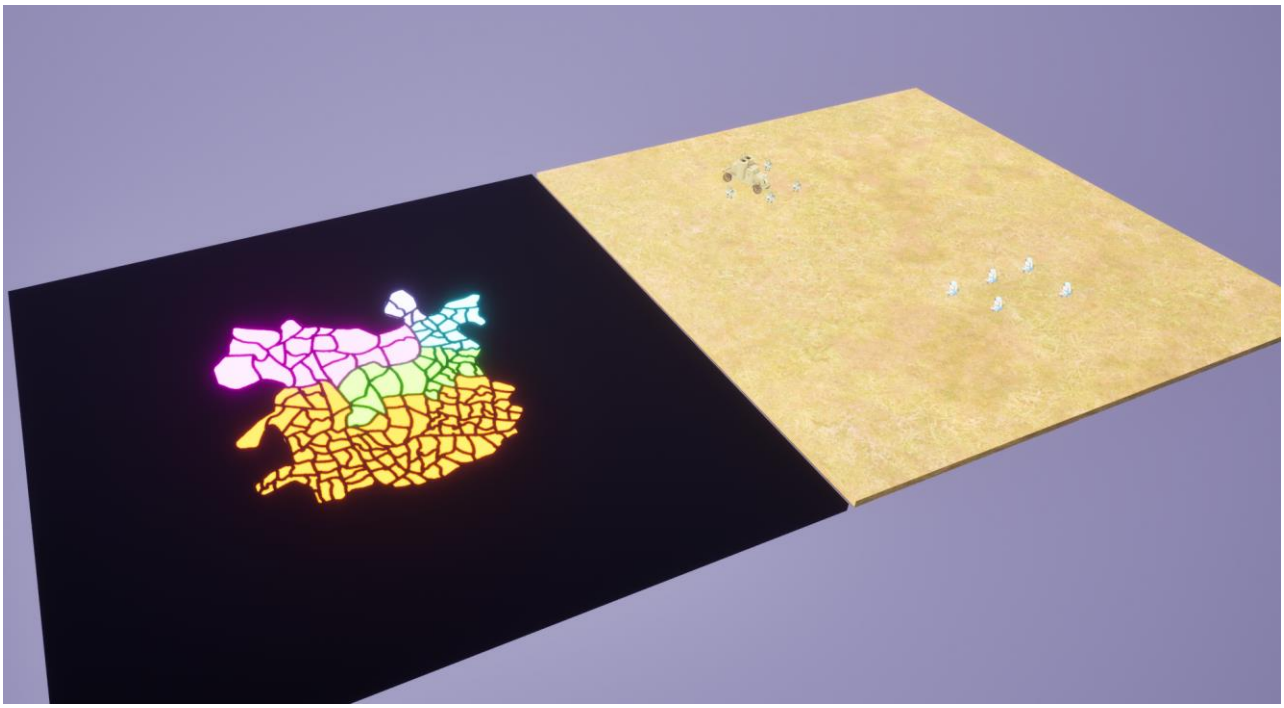


Рис. 4.22 Світ гри без використання Level Streaming

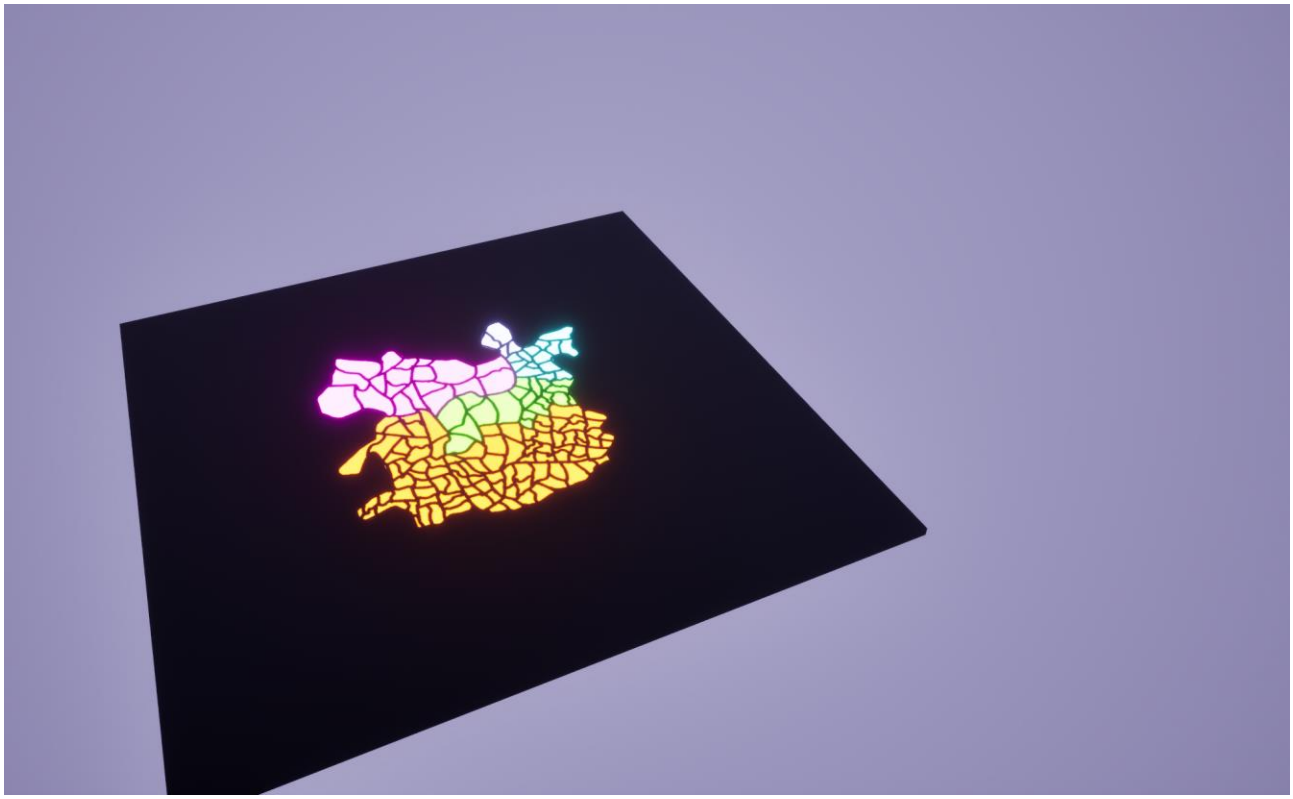


Рис. 4.22 Світ гри при режимі Global Strategy - кількість об’єктів впливаючих на працездатність зменшилося в два рази

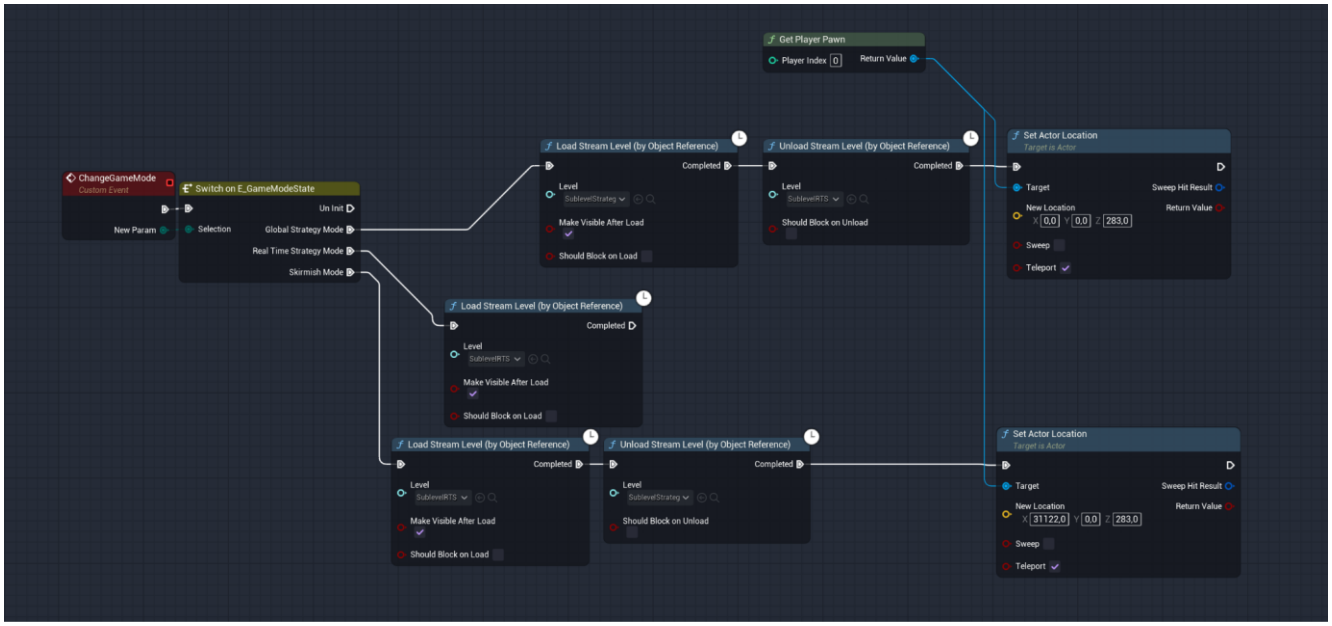


Рис. 4.24 Реалізація Level Streaming всередині BP_HybridGameMode

Таким чином досягається безшовний перехід між режимами, пов’язаними

в достатній мірі для впливу на події один на одного, але недостатньо щоб постраждати від занадто високих технічних вимог. Це означає знайдення концепції гібридної стратегії з елементами RTS та глобальної стратегії.

4.6 Завантаження проєкту на Github

Unreal Engine підтримує цілий ряд різноманітних систем контролю версій. Для великих проектів найбільш підходять платні рішення, проте відеогра розроблена в рамках цієї дипломної роботи має достатнім використання Git для контролю версій та аварійного сховища попередніх версій. В ході розробки продукту Git дозволив відновити декілька надзвичайно важливих файлів, що на практиці підтверджує його значущість для розробки подібних проєктів. Інтерфейс на рисунку 4.25.

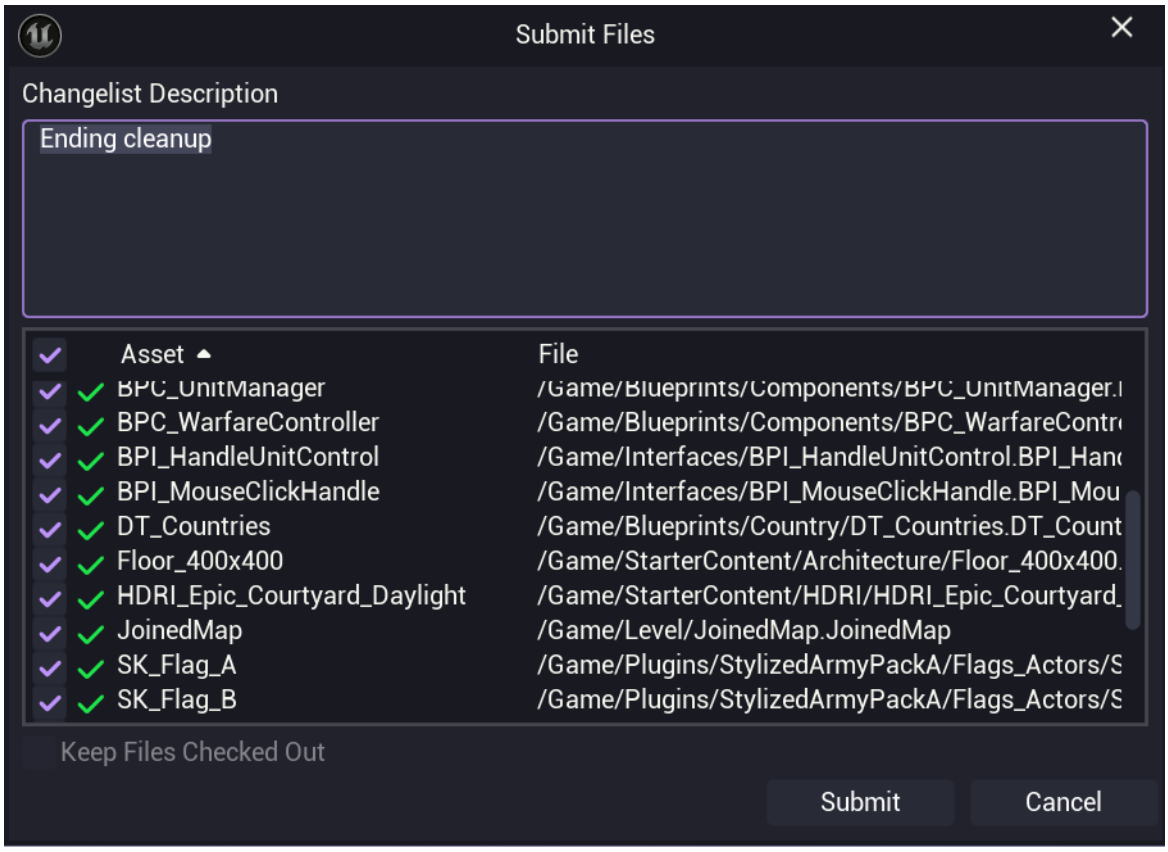


Рис. 4.25 Інтерфейс контролю версій у Unreal

Через те що проекти Unreal Engine великі за розміром та мають велику кількість допоміжних другочергових файлів, надзвичайно важливим є файл .gitignore, що має для даної програми наступний вигляд:

- Binaries
- DerivedDataCache
- Intermediate
- Saved
- .vscode
- .vs
- *.VC.db
- *.opensdf
- *.opendb
- *.sdf
- *.sln
- *.suo
- *.xcodeproj
- *.xcworkspace
- Content/StarterContent

Процес завантаження проекту на Гітхаб зображено на рисунку 4.26.

Репозиторій зображено на рисунку 4.27.

```
D:\Diplom\StellarCampaign>git push origin main
Uploading LFS objects: 100% (322/322), 163 MB | 941 KB/s, done.
Enumerating objects: 377, done.
Counting objects: 100% (377/377), done.
Delta compression using up to 12 threads
Compressing objects: 100% (370/370), done.
Writing objects: 100% (377/377), 57.63 KiB | 1.80 MiB/s, done.
Total 377 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/Goodwin251/InvaisionCampaign.git
 * [new branch]      main -> main

D:\Diplom\StellarCampaign>
```

Рис. 4.26 Завантаження проекту на GitHub

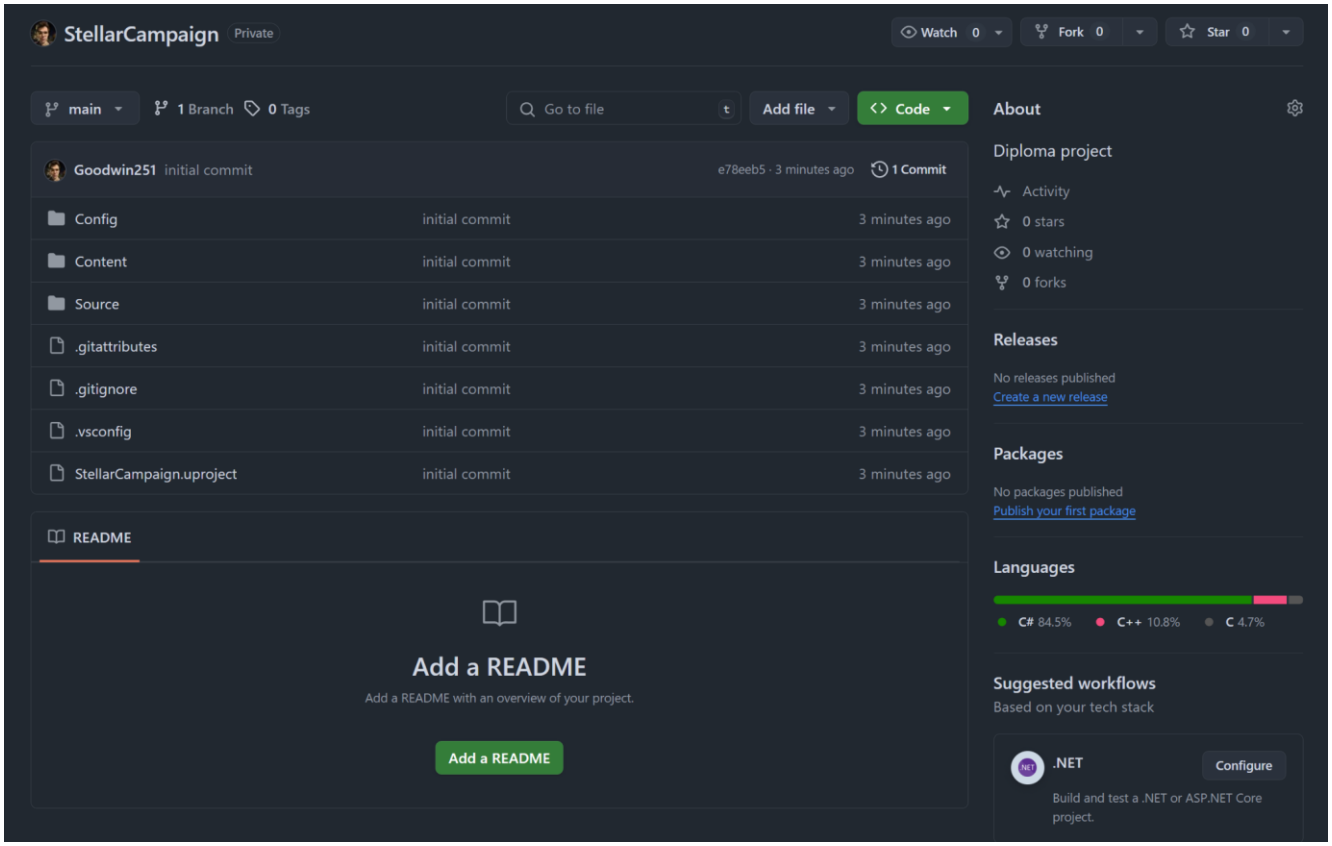


Рис. 4.27 Проект на GitHub

Як можна побачити, система контролю версій дозволяє безпечно зберігати та розробляти програмні проекти.

4.7 Тестування відеогри

Тестування великих проектів зазвичай вимагає колосальну кількість людського часу, щоб перевірити всі можливі ситуації та функції. З метою економлення часу розробники програмного забезпечення створюють функціональні та інтегровані тести, які дозволяють визначити наскільки вірно працюють функції, які сценарії необхідно перевірити.

З цією метою був застосовані інструменти Unreal Engine для автоматизації проекту тестування. У окремо створеному світі були підготовлені умови, які дозволяють провести тестування у різних сценаріях. Приклад реалізованого тесту на рисунку 4.28.

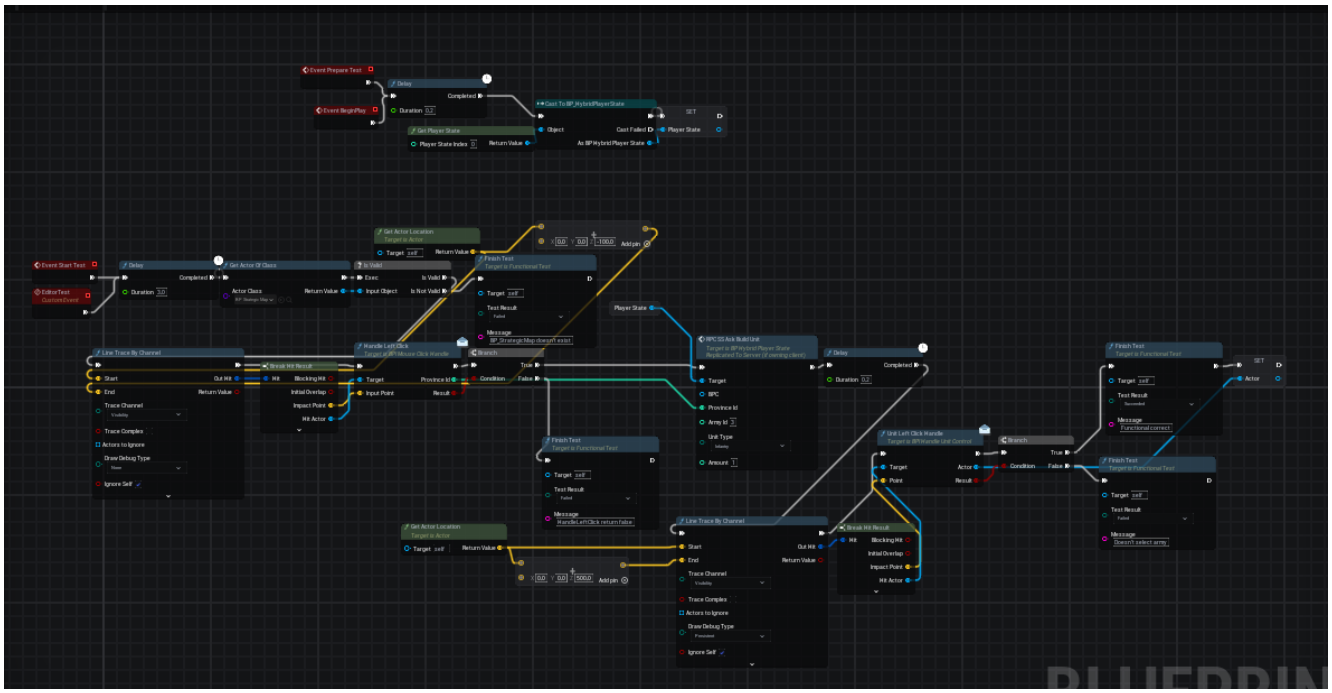


Рис. 4.28 Тестовий клас BPT_TestUnitSelection

У всіх тестів існують спільні події - Prepare Test викликається перед самим початком запуску тесту. Його можна використати для ініціалізації інших класів

підготовлення триваючого. Наступною подією є StartTest - вона дозволяє запустити логіку функції яка нас цікавить. На Рис. 4.28 представлено як імітується клік по провінції через line trace, далі використання отриманого ID для створення армії. Кількість необхідних ресурсів для держави щоб створити армію вже задана завчасна у тестовому рівні. У випадку якщо все правильно, викликається інтерфейс кліку по юніту - якщо функція повертає true, то відповідним функція здійснила перевірку правильно.

Екран автоматичного тестування зображений на рисунку 4.29.

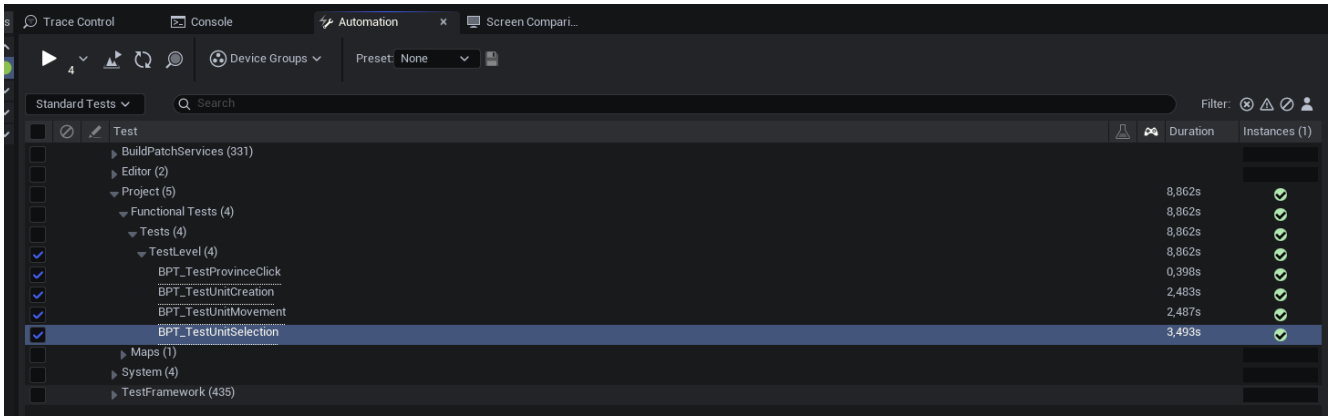


Рис. 4.29 Екран автоматичної перевірки тестів

На Рис. 4.29 видно інтерфейс SessionFrontend, що відповідає за дебаг та тестування проєкту. Тест вивів успішні результати, час виконання всіх тестів зайняв 4 секунди, що означає правильну роботу програми.

ВИСНОВОК

У результаті виконання кваліфікаційної роботи було розроблено прототип відеогри, що поєднує механіки жанрів Real Time Strategy (RTS) та глобальної стратегії. Основною метою дослідження було знайдення ефективної концепції поєднання стратегічних і тактичних механік у межах єдиної ігрової системи, що дозволяє створити більш глибокий, різноманітний і динамічний геймплей. Протягом роботи були вирішені наступні задачі:

– проведено ґрунтовний аналіз предметної області: визначено специфіку стратегічних ігор, розглянуто відмінності та спільні риси RTS та глобальних стратегій, проаналізовано сучасні тенденції розвитку жанру, особливості цільової аудиторії та тренди ринку. Окрему увагу приділено дослідженню перспектив гібридизації ігрових піджанрів та визначенню вимог до сучасних стратегічних ігор на основі аналізу літератури, наукових публікацій і огляду успішних комерційних проектів.

– проаналізовано існуючі засоби реалізації програмних продуктів для розробки стратегічних ігор, зокрема проведено порівняння рушіїв Unreal Engine, Unity та Godot, а також графічних редакторів і систем контролю версій. На основі дослідження обґрунтовано вибір технологічного стеку для розробки: Unreal Engine 5.5 як основний ігровий рушій, Visual Studio для створення C++ класів, Blender для роботи з 3D-моделями, Paint.NET для підготовки 2D-ассетів та Git для організації контролю версій. Вихідний код завантажений до GitHub.

– виконано проєктування архітектури майбутньої гри: визначено ключові компоненти, сформульовано мінімальні вимоги до програмного продукту, розроблено основний ігровий цикл та обґрунтовано доцільність використання унікальних ігрових механік для забезпечення безшовної інтеграції стратегічного і тактичного рівнів.

– реалізовано функціональний прототип гри, що містить стратегічну карту з можливістю управління територіями й ресурсами, економічну систему, модулі тактичного управління бойовими підрозділами та безшовний перехід між стратегічним і тактичним режимами. Здійснено програмну інтеграцію ключових ігрових механік та проведено функціональне тестування основних сценаріїв геймплею.

– проведено тестування та аналіз працездатності розробленого програмного продукту. Виявлені недоліки та складнощі реалізації дозволили уточнити вимоги до архітектури та визначити напрямки для подальшого розвитку.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Солодкий Я.Т., Дібрівний О.А. Концепція поєднання стратегій реального часу з елементами глобальних стратегій. Матеріали четвертої Всеукраїнської науково-технічної конференції конференції “Застосування програмного забезпечення в інформаційно-комунікаційних технологіях”, 24.04.2025, ДУІКТ, м.Київ. К.: ДУІКТ, 2025. С. 236-238

2. Солодкий Я.Т. Захист даних під час ігрового процесу у багатокористувацькій стратегії у реальному часі на Unreal Engine від зловмисних втручань. Матеріали дванадцятої Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.2025, Університет Грінченка, м. Київ. К.: Університет Грінченка, 2025. С. 328-330

3. Солодкий Я.Т. Інтеграція алгоритмів штучного інтелекту у гібридні стратегії реального часу та глобальні стратегії для підвищення навчальної ефективності й залучення користувачів. Матеріали шостої науково-технічної конференції «Сучасний стан та перспективи розвитку IoT», 15.04.2025, ДУІКТ, м. Київ, с. (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Adams, E. Основи проєктування ігор [Fundamentals of Game Design]: підручник. 3-тє вид. Сан-Франциско: New Riders, 2014. 576 с.

2. Adams, E.; Dormans, J. Ігрова механіка: поглиблене проєктування ігор [Game Mechanics: Advanced Game Design]. Сан-Франциско: New Riders, 2012. 353 с.

3. Boot, W. R. et al. The Changing Face of Video Games and Video Gamers: Future Directions in the Scientific Study of Video Game Play and Cognitive Performance. Журнал когнітивних наук (Journal of Cognitive Enhancement), 2017. Т. 1, № 4. С. 280–294.

4. BradyGames. Supreme Commander Official Strategy Guide. Нью-Йорк: BradyGames, 2007. 288 с.

5. Creative Assembly. Total War: Warhammer III. Official Guide. Готем: SEGA, 2022. 214 с.

6. Gregory, J. Архітектура ігрового рушія [Game Engine Architecture]: підручник. 3-тє вид. Бока-Ратон: CRC Press, 2018. 1040 с.

7. Millington, I.; Funge, J. Штучний інтелект для ігор [Artificial Intelligence for Games]. 2-ге вид. Бока-Ратон: CRC Press, 2009. 896 с.

8. Petroglyph. Star Wars: Empire at War. Strategy Guide. Сан-Франциско: LucasArts, 2006. 160 с.

9. Paradox Interactive. Hearts of Iron IV. Game Manual. Стокгольм: Paradox Interactive, 2016. 178 с.

10. Relic Entertainment. Age of Empires IV. Official Manual. Редмонд: Xbox Game Studios, 2021. 195 с.

11. Rogers, S. Покращуйся! Посібник зі створення відмінних відеоігор [Level Up! The Guide to Great Video Game Design]. 2-ге вид. Нью-Джерсі: Wiley, 2014. 516 с.

12. Sweetser, P. Емергентність у іграх [Emergence in Games]. Бостон: Cengage

Learning, 2008. 290 с.

13. Tran, K. M. et al. Using Strategy Video Games to Improve Problem Solving and Communication Skills: A Systematic Literature Review. Міжнародний журнал STEM-освіти (International Journal of STEM Education), 2023. Т. 10, № 1. С. 1–18.

14. Yannakakis, G. N.; Togelius, J. Штучний інтелект та ігри [Artificial Intelligence and Games]. Нью-Йорк: Springer, 2018. 337 с.

15. Zahed, B. T. et al. A Review of How Artificial Intelligence and Learning Analytics Are Transforming Educational Games. IEEE Access, 2021. Т. 9. С. 134 928–134 944.

16. Documentation for Unreal Engine 5.2. [Електронний ресурс]. Режим доступу: https://docs.unrealengine.com/5.2/en-US/ (дата звернення: 28.05.2025).

17. Godot Engine Documentation. [Електронний ресурс]. Режим доступу: https://docs.godotengine.org/ (дата звернення: 28.05.2025).

18. Visual Studio Documentation. [Електронний ресурс]. Режим доступу: https://learn.microsoft.com/en-us/visualstudio/ (дата звернення: 28.05.2025).

19. Blender Manual. [Електронний ресурс]. Режим доступу: https://docs.blender.org/manual/en/latest/ (дата звернення: 28.05.2025).

20. Paint.NET. [Електронний ресурс]. Режим доступу: https://www.getpaint.net/ (дата звернення: 28.05.2025).

21. Git – Documentation. [Електронний ресурс]. Режим доступу: https://git-scm.com/ (дата звернення: 28.05.2025).

22. Neukirchen, C. Multiplayer Compendium: Framework and Network. [Електронний ресурс]. Режим доступу: https://cedric-neukirchen.net/docs/multiplayer-compendium/framework-and-network/ (дата звернення: 28.05.2025).

23. Prins, J. Notes on Unreal Engine. [Електронний ресурс]. Режим доступу: https://jip.dev/notes/unreal-engine/ (дата звернення: 28.05.2025).

24. Green, C. S. et al. The Cognitive Effects of Video Games: A New Meta-

Analysis. [Електронний ресурс]. 2017. Режим доступу: https://greenlab.psych.wisc.edu/wp-content/uploads/sites/280/2017/07/cogBat_2017.pdf (дата звернення: 28.05.2025).

25. Parkin, S. The Fall and Rise of Real-Time Strategy Games. Wired, 2021. [Електронний ресурс]. Режим доступу: https://www.wired.com/story/fall-and-rise-real-time-strategy-games/ (дата звернення: 28.05.2025).

26. Parkin, S. How Age of Empires IV Grapples With Real History. Wired, 2021. [Електронний ресурс]. Режим доступу: https://www.wired.com/story/age-of-empires-iv-real-time-strategy-games-history/ (дата звернення: 28.05.2025).

27. TLVTech. How Strategy Games are Transforming. [Електронний ресурс]. 2023. Режим доступу: https://www.tlvtech.io/post/strategy-games-transforming (дата звернення: 28.05.2025).

28. GameDiscoverCo. Big Get Bigger: Why Paradox is Refocusing. [Електронний ресурс]. 2023. Режим доступу: https://newsletter.gamediscover.co/p/big-get-bigger-why-paradox-is-refocusing (дата звернення: 28.05.2025).

29. GameRefinery. Hybridization is Key for 4X Strategy Success. [Електронний ресурс]. 2023. Режим доступу: https://www.gamerefinery.com/hybridization-is-the-key-for-the-latest-4x-strategy-success-stories (дата звернення: 28.05.2025).

30. GameRefinery. What Drives Success in 4X Strategy Games? Part 1. [Електронний ресурс]. 2023. Режим доступу: https://www.gamerefinery.com/what-drives-success-in-4x-strategy-slg-games-part-1/ (дата звернення: 28.05.2025).

31. PCGamesN. Europa Universalis IV Sells 2 Million Copies. [Електронний ресурс]. 2022. Режим доступу: https://www.pcgamesn.com/europa-universalis-iv/million-sales (дата звернення: 28.05.2025).

32. PC Gamer. Hearts of Iron 4 Sales Surpass 3 Million. [Електронний ресурс]. 2023. Режим доступу: https://www.pcgamer.com/hearts-of-iron-4-has-sold-over-3-million-copies/ (дата звернення: 28.05.2025).

33. Quantic Foundry. The Decline of Strategy Games. [Електронний ресурс].

2024. Режим доступу: https://quanticfoundry.com/2024/05/21/strategy-decline/ (дата звернення: 28.05.2025).

34. Verified Market Research. Strategy Games Market Report. [Електронний ресурс]. 2023. Режим доступу: https://www.verifiedmarketresearch.com/product/strategy-games-market/ (дата звернення: 28.05.2025).

35. Statista. Strategy Games – Worldwide. [Електронний ресурс]. 2023. Режим доступу: https://www.statista.com/outlook/amo/app/games/strategy-games/worldwide (дата звернення: 28.05.2025)

36. Гарт П.Е., Нільссон Н.Й., Рафаель Б. Формальні основи евристичного визначення найкоротших шляхів // *IEEE Transactions on Systems Science and Cybernetics*. 1968. Т. 4, № 2. С. 100–107.

37. Figma. [Електронний ресурс]. Режим доступу: https://www.figma.com/ (дата звернення: 28.05.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Real Time Strategy з елементами глобальної стратегії

Виконав студент 4 курсу
групи ПД-42
Солодкий Ярослав Тарасович
Керівник роботи
Д.ф. (PhD), доц, доцент кафедри ІПЗ, Дібрівний Олесь Андрійович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - знайдення концепції поєднання механік жанрів RTS та глобальної стратегії в межах однієї ігрової системи на основі створеного прототипу гри.
- **Об’єкт дослідження -** процес взаємодії стратегічного та тактичного ігрових режимів у гібридних іграх жанрів Real Time Strategy (RTS) та глобальної стратегії.
- **Предмет дослідження -** ігрові механіки стратегічного та тактичного режимів, а також архітектурні рішення, що забезпечують їхню інтеграцію на основі створеного прототипу гібридної гри.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Ознайомитись з ситуацією на ринку RTS та Глобальних стратегій.
2. Провести порівняльний аналіз з аналогічними продуктами.
3. Визначити функціональні та нефункціональні вимоги продукту.
4. Спроектувати архітектуру програмного продукту.
5. Реалізувати модуль глобальної стратегії.
6. Реалізувати модуль стратегії в реальному часі.
7. Розробити інтеграцію обох основних модулів.
8. Перевірити працездатність розробленого продукту на відповідність визначеним вимогам.

3

АНАЛІЗ АНАЛОГІВ

Характеристика	Керування часом гри	Наявність тактичного режиму	Керування економікою	Система провінцій	Час переходу між режимами
Серія Total War	+	+	+	+	**2-5хв**
Empire At War	-	+	-	+	**1хв**
Europa Universalis IV	+	-	+	+	-
Stellar Campaign	+	+	+	+	**<1с**

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Перемикання режимів гри (стратегічний та тактичний RTS-режим) з використанням патерну «Стан».
2. Реалізація стратегічного режиму (управління економікою, переміщення військ по стратегічній карті, розділеної на різні провінції).
3. Реалізація тактичного режиму: Можливість управління військами в RTS-режимі за допомогою миші (віддача наказів для переміщення і атаки).
4. Реалізувати режими кампанії (RTS + Глобальна стратегія).
5. Визначити чіткого списку вимог для отримання перемоги.

Нефункціональні вимоги:

1. Продуктивність: Забезпечення стабільного фреймрейту (не нижче 60 FPS) на більшості комп'ютерів (відповідно до аналітики Steam).
2. Надійність: Відсутність критичних помилок (аварійного завершення програми) під час ігрового процесу.
3. Сумісність: Працездатність продукту на сучасних версіях Windows (10-11).
4. Локалізація: українська та англійська мови.

5

Концепт гри

- **Назва - Stellar Campaign.**
- **Жанр** – гібрид глобальної стратегії та RTS.
- **Цільова аудиторія** – першочергово фанати ігор Paradox Interactive та Empire At War, вони же молоді люди 18-25 років, орієнтованих на виклик та планування від глобальних стратегій, але також бажаючих динаміки RTS.
- **Сеттинг** – вигадані події на іншій планеті з технологічним рівнем Першої Світової війни. Гравець керує експедицією, яка вторгається на інший континент, з чого і починаються події гри.
- **Ключові особливості** – вертикальна безшовна синхронізація двох режимів гри - стратегічна та тактична складові впливають на один-одного в реальному часі, без додаткових завантажень та очікувань.
- **Платформа -** Windows.
- **Інші характеристики** - керування завдяки клавіатурі та миші, низькі вимоги до технічного забезпечення.

6

Короткий огляд ідеї гри

1. Гравець грає за завойовника, що вторгається на інший конинент з іншою культурою.
2. Ворожі країни є роздріблені та різні за своїм потенціалом - грамотне розширення є ключовим для війни з найсильнішим з ворогів.
3. Гравець має приймати адекватні ситуації рішення для виживання, адже ворожі сили є численними, нехай гравець і має найважливішу перевагу - власний розум.
4. Гравець або перемагає після захоплення всіх столиць противника, або гине коли втрачає власну.
5. Гравець може вирішувати які битви він воює особисто, а які дозволяє провести локальним командирам - в різних ситуаціях різне рішення може бути доцільним.

USP (Unique Selling Points)

1. Безшовна інтеграція стратегічного та тактичного режимів.
2. Реалізація найцікавіших елементів обох піджанрів стратегій (тактичні, стратегічні битви, будівництво, економіка)

Геймплей, ігровий цикл

Геймплей: орієнтований на ведення війни з кількома країнами на спільному континенті, да задача гравця є їх перемогти, незважаючи на їх переважаючі кількість ресурсів. Маневри, розбудова армії на стратегічній та тактичних мапах визначають які будуть результати війни.

Ігровий цикл: Гравець розбудовує країну будівлями, збільшує розвиток власних провінцій, розбудовує армію та воює з ворогом, захоплює провінції.

Приклади ігрових механік

Провінційні характеристики - кожна провінція має власну користь, у вигляді добуваємих ресурсів та її населення. Не всі провінції рівноцінні, що робить деякі з важливіше за інші.

Механіка будівництва - всі провінції гравця мають можливість будувати фабрики та казарми, для збільшення вироблення ресурсів та збільшення потенційної армії в майбутньому.

Найм військ - кількість населення в країні та наявність ресурсів визначає можливість найняти бажану кількість військ.

Типи військ - кожна армія може мати унікальний тип військ - піхота не рівнозначна броньованій машині, що відображається як у розрахунку на глобальні мапі, так і при битві на полі бою.

Туман війни - гравець може бачити землі тільки навколо власних армій або сусідніх з власними провінціями

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Для написання класів на C++

Ігровий двигун, основне середовище розробки

Для редагування 3D моделей

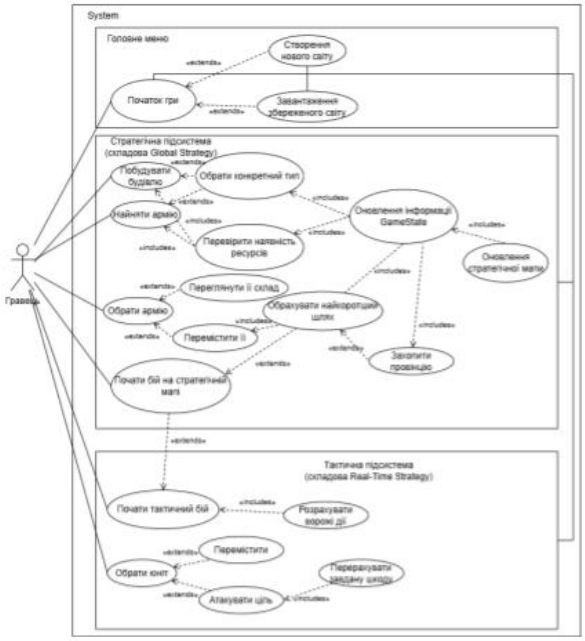
Для створення стратегічної карти

Контроль версій проекту

Проектування інтерфейсів користувача

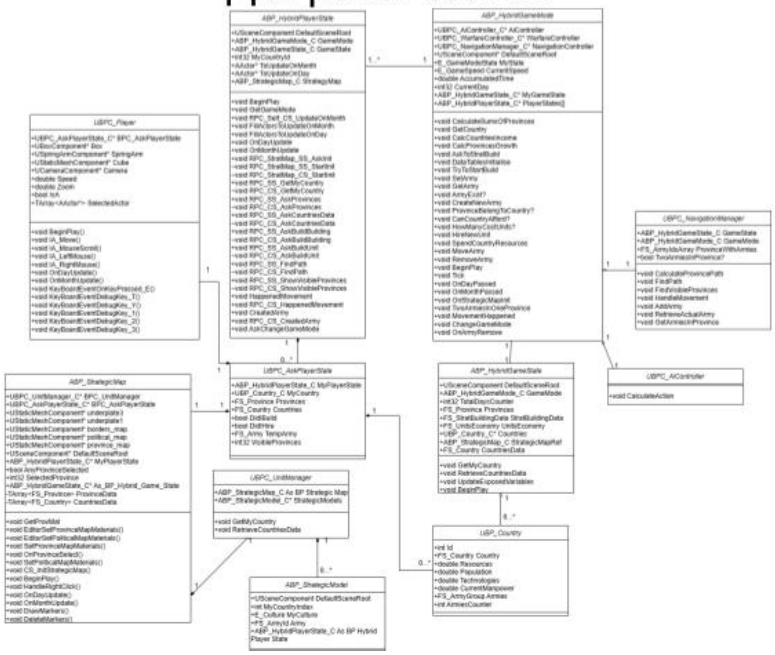
11

Діаграма варіантів використання



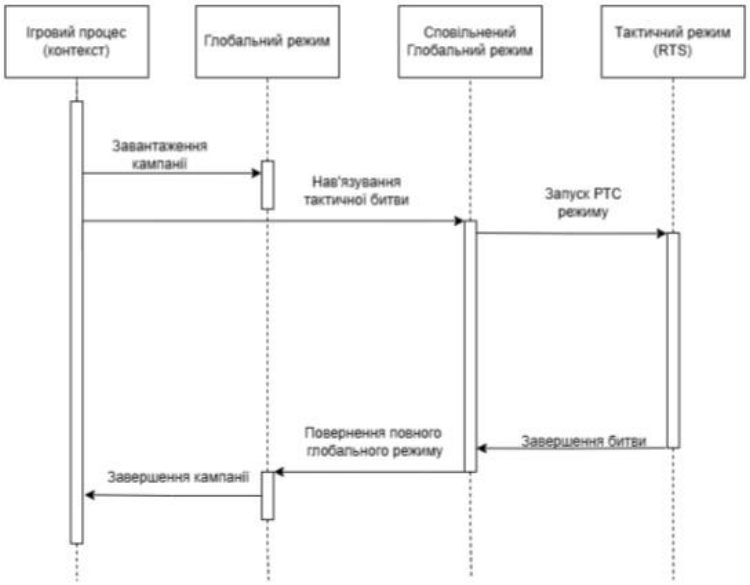
12

Діаграма класів



13

Послідовна діаграма зміни станів



14

ЕКРАННІ ФОРМИ

Екран стратегічного

15

ЕКРАННІ ФОРМИ

Екран тактичного режиму

16

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Солодкий Я.Т., Дібрівний О.А. Концепція поєднання стратегій реального часу з елементами глобальних стратегій. Матеріали четвертої Всеукраїнської науково-технічної конференції конференції “Застосування програмного забезпечення в інформаційно-комунікаційних технологіях”, 24.04.2025, ДУІКТ, м.Київ. К.: ДУІКТ, 2025. С. 236-238
2. Солодкий Я.Т. Захист даних під час ігрового процесу у багатокористувацькій стратегії у реальному часі на Unreal Engine від зловмисних втручань. Матеріали дванадцятої Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.2025, Університет Грінченка, м. Київ. К.: Університет Грінченка, 2025. С. 328-330
3. Солодкий Я.Т. Інтеграція алгоритмів штучного інтелекту у гібридні стратегії реального часу та глобальні стратегії для підвищення навчальної ефективності й залучення користувачів. Матеріали шостої науково-технічної конференції «Сучасний стан та перспективи розвитку IoT», 15.04.2025, ДУІКТ, м. Київ, с. (подано до друку).

ВИСНОВКИ

1. Проаналізовано ситуацію на ринку стратегій, що показало великий потенціал на глобальні стратегії з елементами інших жанрів.
2. Внаслідок порівняльного аналіз конкурентів на ринку, визначено найбільшу слабкість гібридів - розірваний ігровий процес, розробники не здатні одночасно реалізувати потенціал через технічні проблеми.
3. Був спроектована архітектура, визначені необхідні технології продукту, що оминає обмеження завдяки реалізації різних станів гри та швидких переходів між її частинами.
4. В результаті розробки вдалося зробити досконалий ігровий продукт з великою кількістю реалізованих складних механік, які вдалось реалізувати в рамках концепції.
5. Автоматизоване функціональне та ручне тестування показало успішне дотримання ПЗ всіх визначених вимог.

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ

Вихідний код файлів проекту:

StellarCampaign.uproject:

```
{
"FileVersion": 3,
"EngineAssociation": "5.5",
"Category": "",
"Description": "",
"Modules": [
	{
		"Name": "StellarCampaign",
		"Type": "Runtime",
		"LoadingPhase": "Default"
	}
],
"Plugins": [
	{
		"Name": "ModelingToolsEditorMode",
		"Enabled": true,
		"TargetAllowList": [
			"Editor"
		]
	},
	{
		"Name": "ElectronicNodes",
		"Enabled": true,
		"MarketplaceURL":
"com.epicgames.launcher://ue/marketplace/content/5cb2a394d0c04e73891762be4cbd7216"
	},
	{
		"Name": "DarkerNodes",
		"Enabled": true,
		"MarketplaceURL":
"com.epicgames.launcher://ue/marketplace/content/4b3441f0228a40ec9ca986489a5bd682"
```

```
},
{
        "Name": "AutomatedPerfTesting",
        "Enabled": true
},
{
        "Name": "AutomationDriverTests",
        "Enabled": true
},
{
        "Name": "CQTest",
        "Enabled": true
},
{
        "Name": "EditorTests",
        "Enabled": true
},
{
        "Name": "CQTestEnhancedInput",
        "Enabled": true
},
{
        "Name": "FbxAutomationTestBuilder",
        "Enabled": true
},
{
        "Name": "FunctionalTestingEditor",
        "Enabled": true
},
{
        "Name": "Gauntlet",
        "Enabled": true
},
{
        "Name": "PythonAutomationTest",
        "Enabled": true
```

```
		},
		{
			"Name": "RHITests",
			"Enabled": true
		},
		{
			"Name": "TestFramework",
			"Enabled": true
		},
		{
			"Name": "RuntimeTests",
			"Enabled": true
		},
		{
			"Name": "TestSamples",
			"Enabled": true
		},
		{
			"Name": "WidgetAutomationTests",
			"Enabled": true
		},
		{
			"Name": "AdvancedSteamSessions",
			"Enabled": false
		},
		{
			"Name": "AdvancedSessions",
			"Enabled": false
		}
],
"AdditionalPluginDirectories": [
		"Content/Plugins"
]
}
```


StellarCampaign.Build.cs:

```
using UnrealBuildTool;

public class StellarCampaign : ModuleRules
{
   public StellarCampaign(ReadOnlyTargetRules Target) : base(Target)
   {
      PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

      PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine",
"InputCore", "EnhancedInput" });

      PrivateDependencyModuleNames.AddRange(new string[] {  });

      // Uncomment if you are using Slate UI
      // PrivateDependencyModuleNames.AddRange(new string[] { "Slate", "SlateCore" });

      // Uncomment if you are using online features
      // PrivateDependencyModuleNames.Add("OnlineSubsystem");

      // To include OnlineSubsystemSteam, add it to the plugins section in your uproject file with the Enabled
attribute set to true
   }
}
```

StellarCampaign.cpp:

```
#include "StellarCampaign.h"
#include "Modules/ModuleManager.h"

IMPLEMENT_PRIMARY_GAME_MODULE( FDefaultGameModuleImpl, StellarCampaign,
"StellarCampaign" );
```

StellarCampaign.h:

```
#pragma once

#include "CoreMinimal.h"
```

StellarCampaign.Target.cs:

```
using UnrealBuildTool;
using System.Collections.Generic;

public class StellarCampaignTarget : TargetRules
{
   public StellarCampaignTarget(TargetInfo Target) : base(Target)
   {
      Type = TargetType.Game;
      DefaultBuildSettings = BuildSettingsVersion.V5;
      IncludeOrderVersion = EngineIncludeOrderVersion.Unreal5_5;
      ExtraModuleNames.Add("StellarCampaign");
   }
}
```

StellarCampaignEditor.Target.cs:

```
using UnrealBuildTool;
using System.Collections.Generic;

public class StellarCampaignEditorTarget : TargetRules
{
```

```
	public StellarCampaignEditorTarget( TargetInfo Target) : base(Target)
	{
		Type = TargetType.Editor;
		DefaultBuildSettings = BuildSettingsVersion.V5;
		IncludeOrderVersion = EngineIncludeOrderVersion.Unreal5_5;
		ExtraModuleNames.Add("StellarCampaign");
	}
}
```

Блюпринтів:

HybridGameMode.h:

```
/** Please add a class description */
UCLASS(Blueprintable, BlueprintType)
class ABP_HybridGameMode : public AGameMode
{
	GENERATED_BODY()
public:
	/** Please add a function description */
	UFUNCTION(BlueprintPure, Category="Units")
	void ArmyExist?(int32 CountryId, int32 ArmyId, bool& Result);

	/** Please add a function description */
	UFUNCTION(BlueprintPure, Category="Provinces")
	void CalculateSumsOfProvinces(UPARAM(ref) TArray<int32>& Indexes,
double& ResourcesSum, double& PopulationSum, double& DevelopmentSum,
double TempResourcesSum, double TempPopulationSum, double
TempDevelopmentSum);
```

```
/** Please add a function description */
UFUNCTION(BlueprintPure, Category="Country")
void GetCountry(int32 CountryId, UBP_Country_C*& Country);

/** Please add a function description */
UFUNCTION(BlueprintCallable, Category="Country")
void CalcCountriesIncome();

/** Please add a function description */
UFUNCTION(BlueprintCallable, Category="Provinces")
void CalcProvincesGrowth(FS_Province Temp);

/** Please add a function description */
UFUNCTION(BlueprintCallable, Category="Building")
void AskToStratBuild(int32 CountryIndex,
TEnumAsByte<E_StratBuildingType> BuildingType, int32 Amount, bool& Build
possible?);

/** Please add a function description */
UFUNCTION(BlueprintCallable, Category="Data")
void DataTablesInitialise(TArray<FS_Province> DefaultProvinceData,
TArray<FS_Country> DefaultCountriesData, TArray<FS_StratBuildingData>
DefaultStratBuildingData);

/** Please add a function description */
UFUNCTION(BlueprintCallable, Category="Building")
void TryToStartBuild(int32 ProvinceId, int32 CountryIndex,
```

TEnumAsByte<E_StratBuildingType> BuildingType, int32 Amount, bool& Success, FS_Province TempProv);

/** Please add a function description */
UFUNCTION(BlueprintCallable, Category="Units")
void SetArmy(FS_Army Army, bool& Result);

/** Please add a function description */
UFUNCTION(BlueprintPure, Category="Units")
void GetArmy(FS_ArmyId ArmyId, FS_Army& Army);

/** Please add a function description */
UFUNCTION(BlueprintCallable, Category="Units")
void CreateNewArmy(int32 CountryId, int32 ProvinceId, bool& Result, int32& NewID, FS_ArmyGroup NewLocalVar_0);

/** Please add a function description */
UFUNCTION(BlueprintPure, Category="Provinces")
void ProvinceBelongToCountry?(int32 CountryId, int32 ProvinceId, bool& Result);

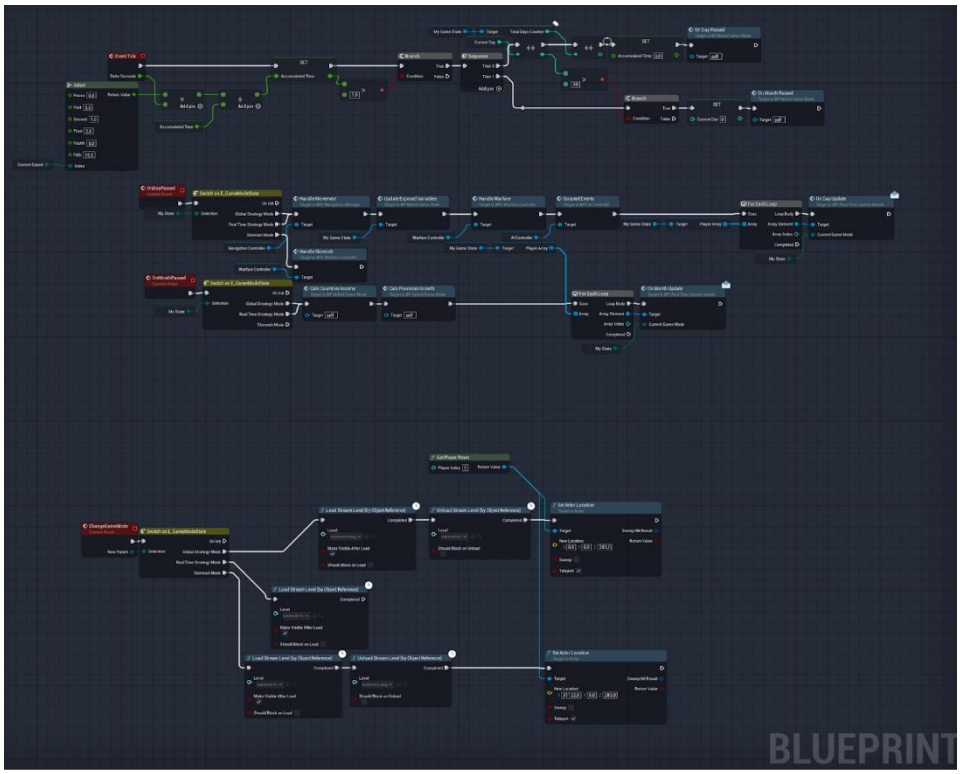
/** Please add a function description */
UFUNCTION(BlueprintPure, Category="Country")
void CanCountryAfford?(int32 CountryId, double ResourcesAmount, double PeopleAmount, bool& Result);

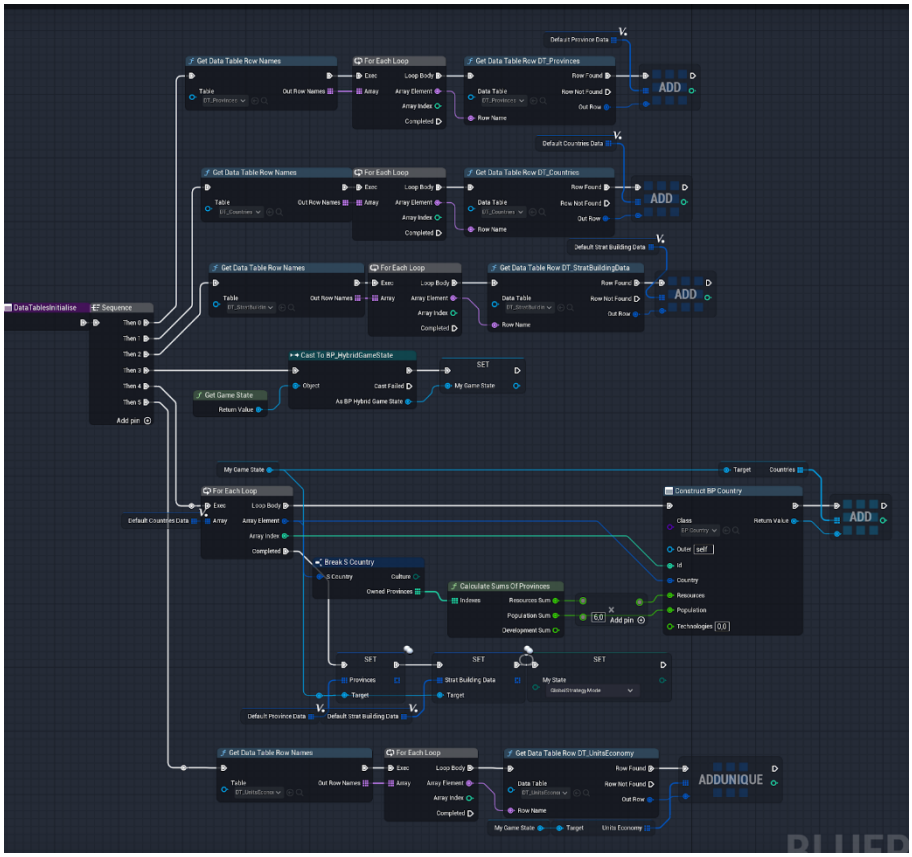
/** Please add a function description */
UFUNCTION(BlueprintPure, Category="Units")

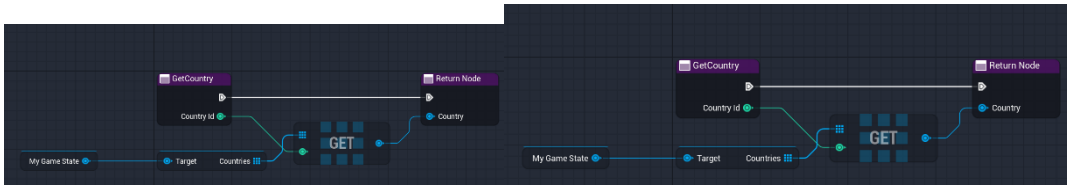
```
	void HowManyCostUnits?(TEnumAsByte<E_UnitType> UnitType, int32 Amount, bool& Possible?, double& ResourcesAmount, double& PopulationAmount, double& TimeAmount);

	/** Please add a function description */
	UFUNCTION(BlueprintCallable, Category="Units")
	void HireNewUnit(int32 CountryId, int32 ProvinceId, int32 ArmyId, TEnumAsByte<E_UnitType> UnitType, int32 Amount, bool& Result, FS_ArmyGroup NewLocalVar_1);

	/** Please add a function description */
	UFUNCTION(BlueprintCallable, Category="Country")
	void SpendCountryResources(int32 CountryId, double ResourcesAmount, double PeopleAmount);

	/** Please add a function description */
	UFUNCTION(BlueprintCallable, Category="Units")
	void MoveArmy(int32 CountryId, int32 Army, int32 EndProvince, TArray<int32>& Path);

	/** Please add a function description */
	UFUNCTION(BlueprintCallable, Category="Units")
	void RemoveArmy(FS_ArmyId ArmyId);

	/** Please add a function description */
	UFUNCTION(BlueprintPure, Category="Provinces")
	void WhoHaveProvince?(int32 ProvinceId, int32& CountryId);
public:
```

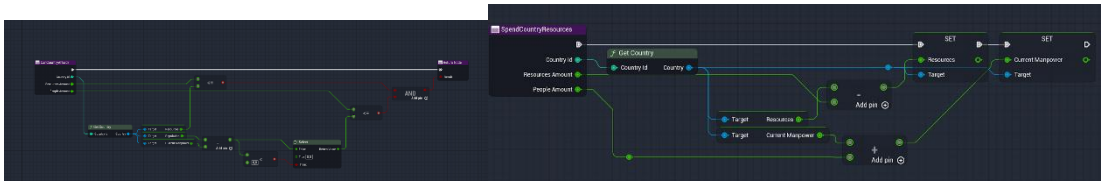
```
	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Date")
	TEnumAsByte<E_GameSpeed> CurrentSpeed;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
	TObjectPtr<UBPC_WarfareController_C> WarfareController;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
	TObjectPtr<UBPC_NavigationManager_C> NavigationController;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
	TObjectPtr<USceneComponent> DefaultSceneRoot;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly,
Category="GameData")
	TEnumAsByte<E_GameModeState> MyState;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
	TObjectPtr<UBPC_AiController_C> AiController;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Date")
	double AccumulatedTime;
```

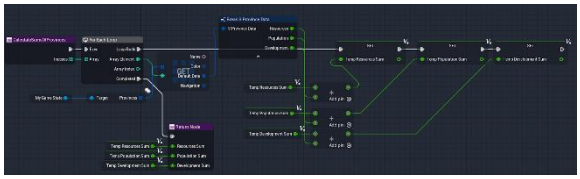
```
	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Date")
	int32 CurrentDay;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TObjectPtr<ABP_HybridGameState_C> MyGameState;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TArray<ABP_HybridPlayerState_C*> PlayerStates;
};
```

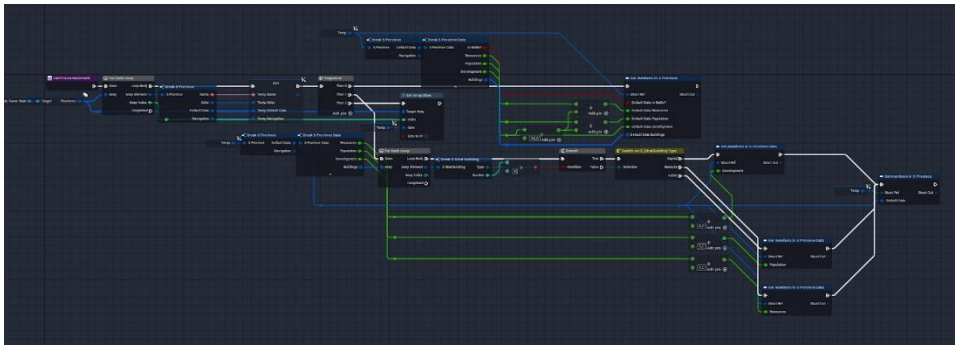


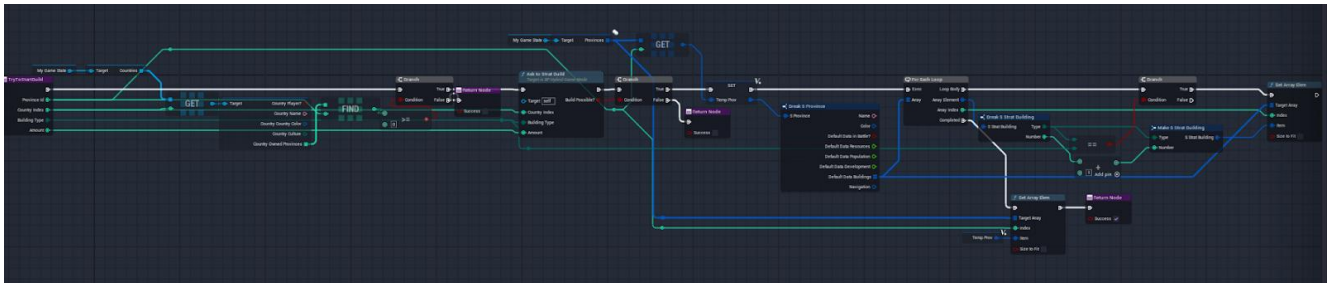

DataTablesInitialise
Sequence
Get Data Table Row Names
For Each Loop
Get Data Table Row DT_Provinces
Get Data Table Row DT_Countries
Get Data Table Row DT_StratBuildingData
Cast To BP_HybridGameState
Get Game State
SET
ADD
For Each Loop
Construct BP Country
Break S Country
Calculate Sums Of Provinces
Get Data Table Row DT_UnitsEconomy
ADDUNIQUE

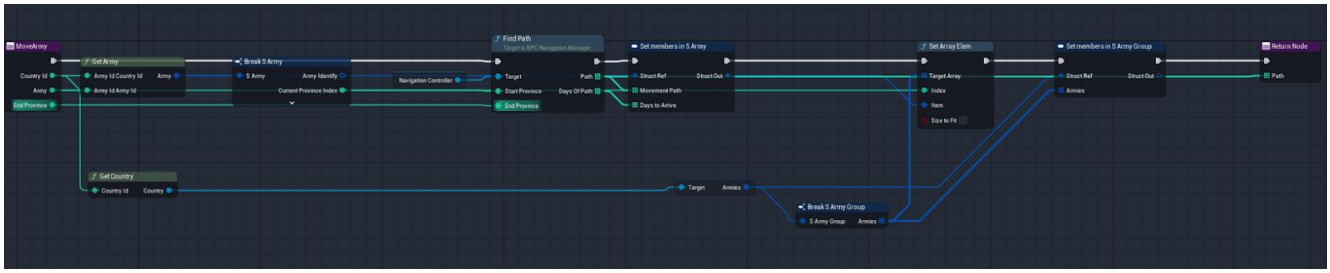

GetCountry
Country Id
Return Node
Country
My Game State
Target
Countries
GET

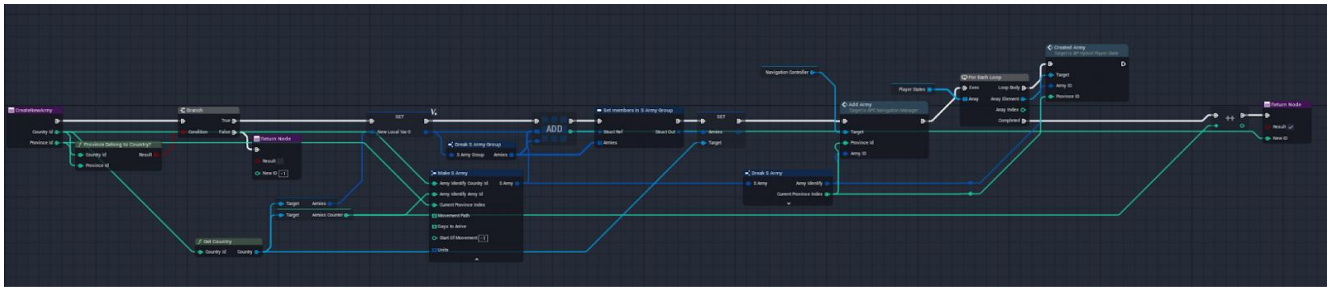

SpendCountryResources
Country Id
Resources Amount
People Amount
Get Country
SET
Add pin











HybridGameState:

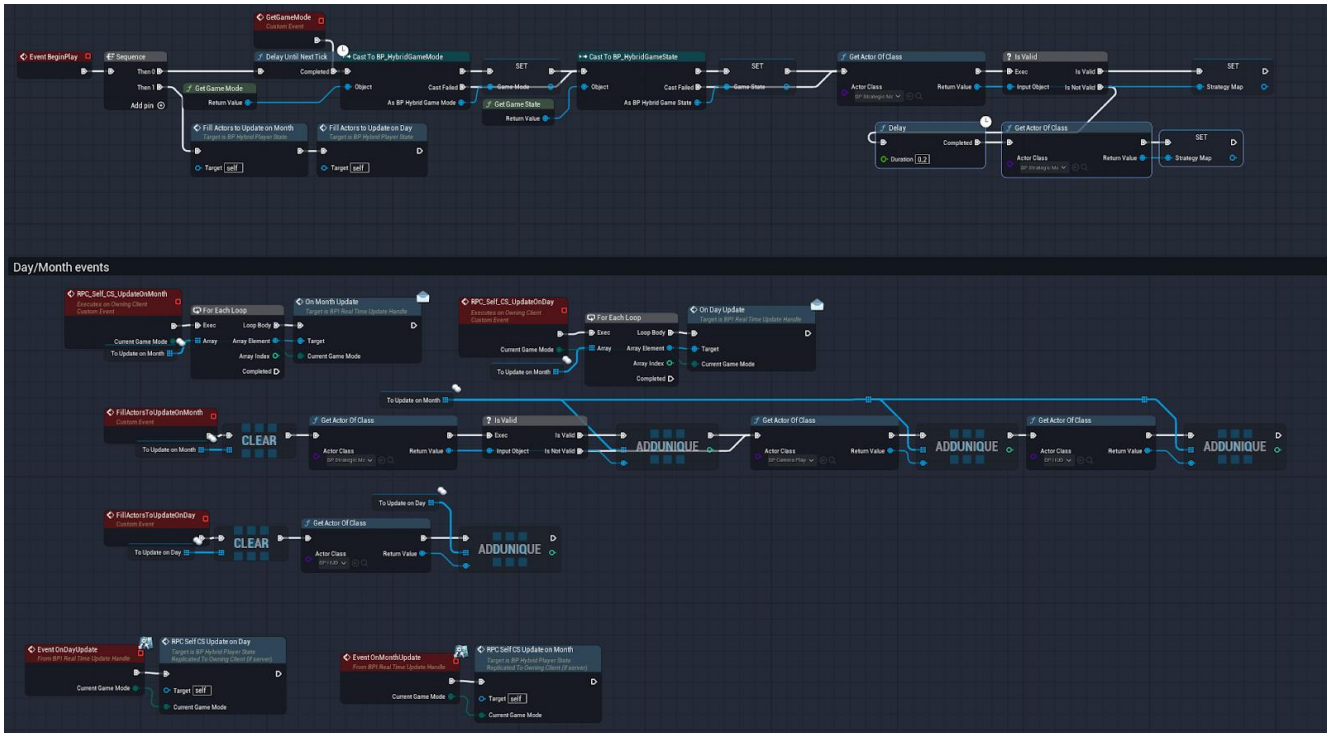
```
/** Please add a class description */
UCLASS(Blueprintable, BlueprintType)
class ABP_HybridGameState : public AGameState
{
	GENERATED_BODY()
public:
	/** Please add a function description */
	UFUNCTION(BlueprintCallable)
	void GetMyCountry(ABP_HybridPlayerState_C* PlayerState,
UBP_Country_C*& Country);

	/** Please add a function description */
	UFUNCTION(BlueprintCallable)
```

```
	void RetrieveCountriesData(TArray<FS_Country>& CountriesData,
TArray<FS_Country> NewLocalVar);
public:
	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TArray<FS_UnitsEconomy> UnitsEconomy;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TObjectPtr<ABP_HybridGameMode_C> GameMode;

	/** Please add a variable description */
	static_assert(false, "You will need to add
DOREPLIFETIME(ABP_HybridGameState, TotalDaysCounter) to
GetLifetimeReplicatedProps");
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default",
Replicated)
	int32 TotalDaysCounter;

	/** Please add a variable description */
	static_assert(false, "You will need to add
DOREPLIFETIME(ABP_HybridGameState, Provinces) to
GetLifetimeReplicatedProps");
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default",
Replicated)
	TArray<FS_Province> Provinces;

	/** Please add a variable description */
```

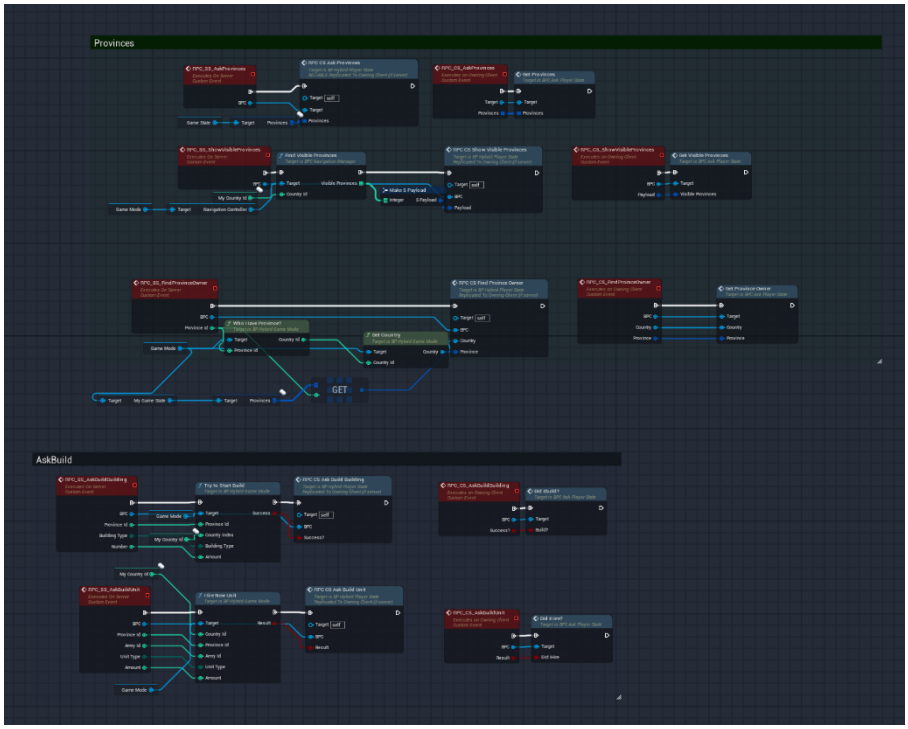
```
	static_assert(false, "You will need to add
DOREPLIFETIME(ABP_HybridGameState, StratBuildingData) to
GetLifetimeReplicatedProps");
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default",
Replicated)
	TArray<FS_StratBuildingData> StratBuildingData;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
	TObjectPtr<USceneComponent> DefaultSceneRoot;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TArray<UBP_Country_C*> Countries;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TObjectPtr<ABP_StrategicMap_C> StrategicMapRef;

	/** Please add a variable description */
	static_assert(false, "You will need to add
DOREPLIFETIME(ABP_HybridGameState, CountriesData) to
GetLifetimeReplicatedProps");
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default",
Replicated)
	TArray<FS_Country> CountriesData;

	/** Please add a variable description */
```

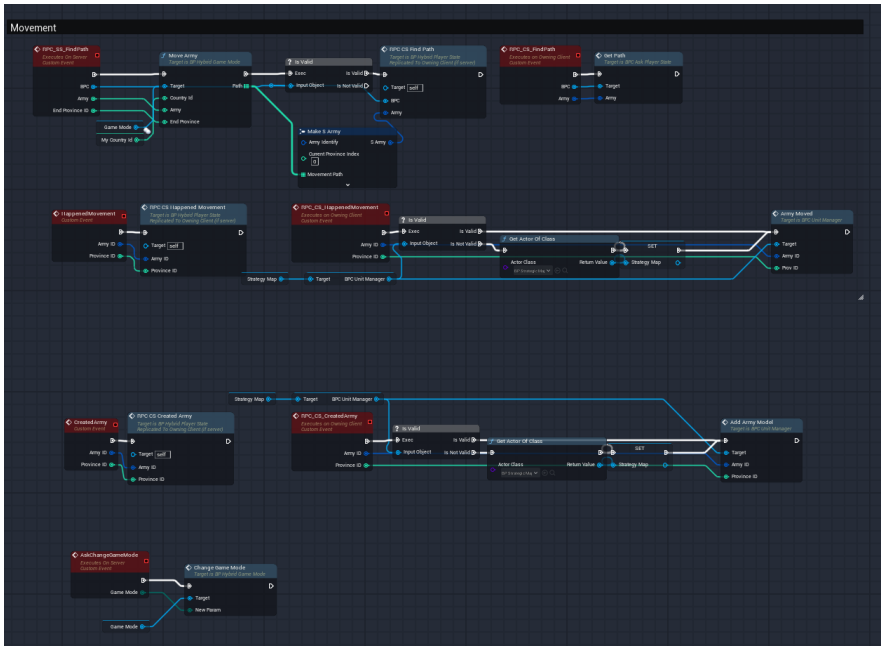
```
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	int32 Current Day;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TEnumAsByte<E_GameSpeed> Current Speed;
};

/** Please add a class description */
UCLASS(Blueprintable, BlueprintType)
class ABP_HybridPlayerState : public APlayerState
{
	GENERATED_BODY()
public:
	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TObjectPtr<ABP_HybridGameMode_C> GameMode;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TObjectPtr<ABP_HybridGameState_C> GameState;

	/** Please add a variable description */
	static_assert(false, "You will need to add
DOREPLIFETIME(ABP_HybridPlayerState, MyCountryId) to
GetLifetimeReplicatedProps");
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default",
Replicated)
```

```
	int32 MyCountryId;

	/** Please add a variable description */
	static_assert(false, "You will need to add DOREPLIFETIME(ABP_HybridPlayerState, ToUpdateOnMonth) to GetLifetimeReplicatedProps");
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default", Replicated)
	TArray<AActor*> ToUpdateOnMonth;

	/** Please add a variable description */
	static_assert(false, "You will need to add DOREPLIFETIME(ABP_HybridPlayerState, ToUpdateOnDay) to GetLifetimeReplicatedProps");
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default", Replicated)
	TArray<AActor*> ToUpdateOnDay;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TObjectPtr<ABP_StrategicMap_C> StrategyMap;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
	TObjectPtr<USceneComponent> DefaultSceneRoot;
};
```


Day/Month events
CLEAR
ADDUNIQUE
CLEAR
ADDUNIQUE
ADDUNIQUE
ADDUNIQUE


AskCountry
StratMap





NavigationManager:

/** Please add a class description */

UCLASS(Blueprintable, BlueprintType)

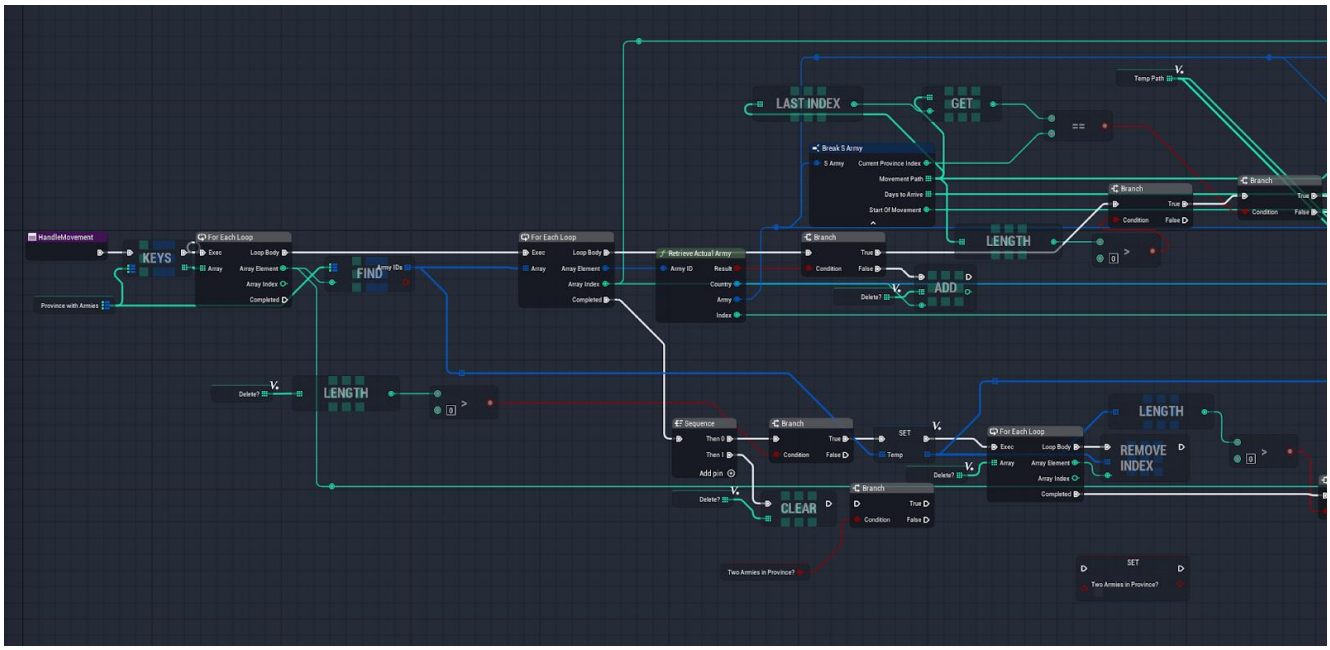
class UBPC_NavigationManager : public UActorComponent

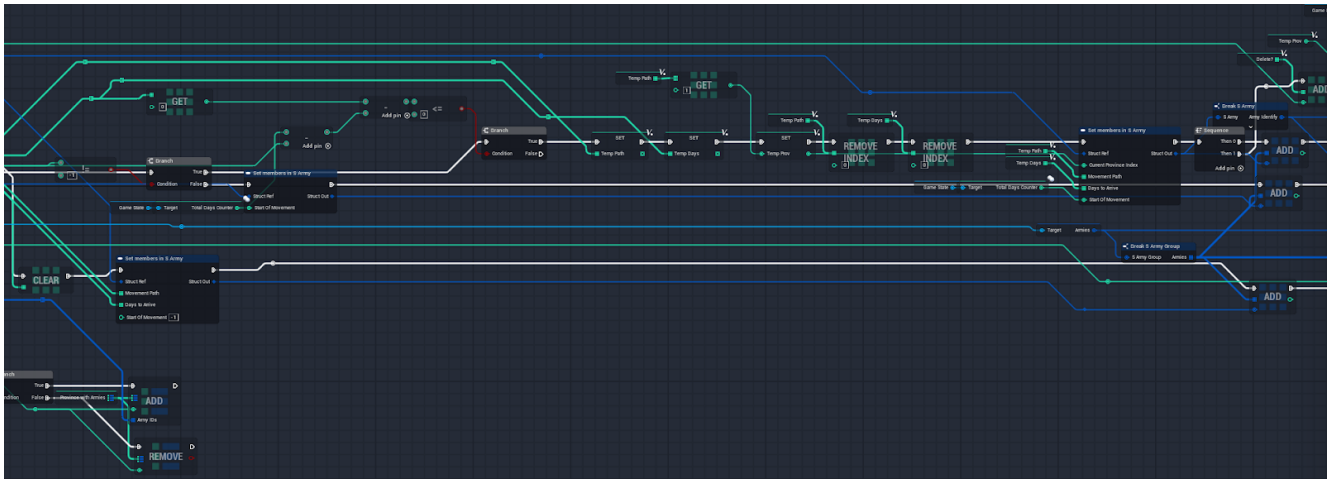
{

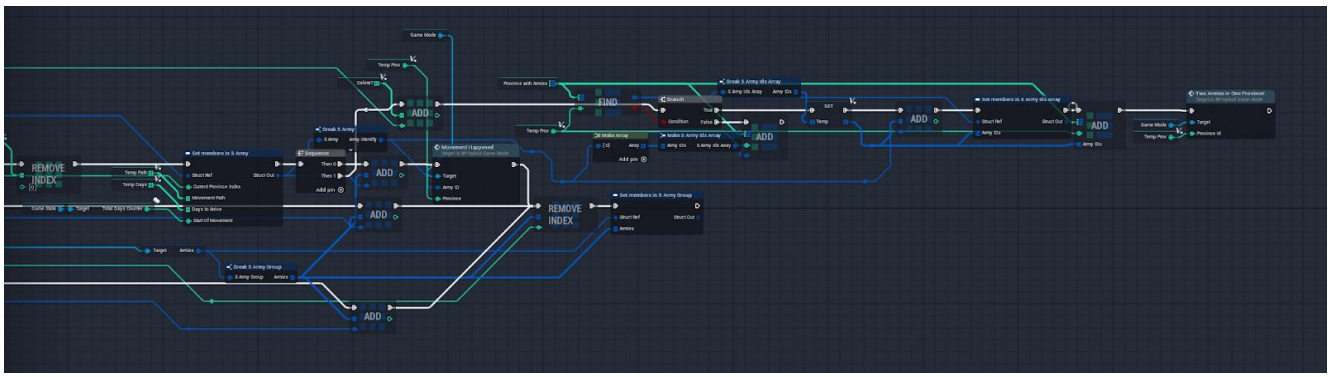
GENERATED_BODY()

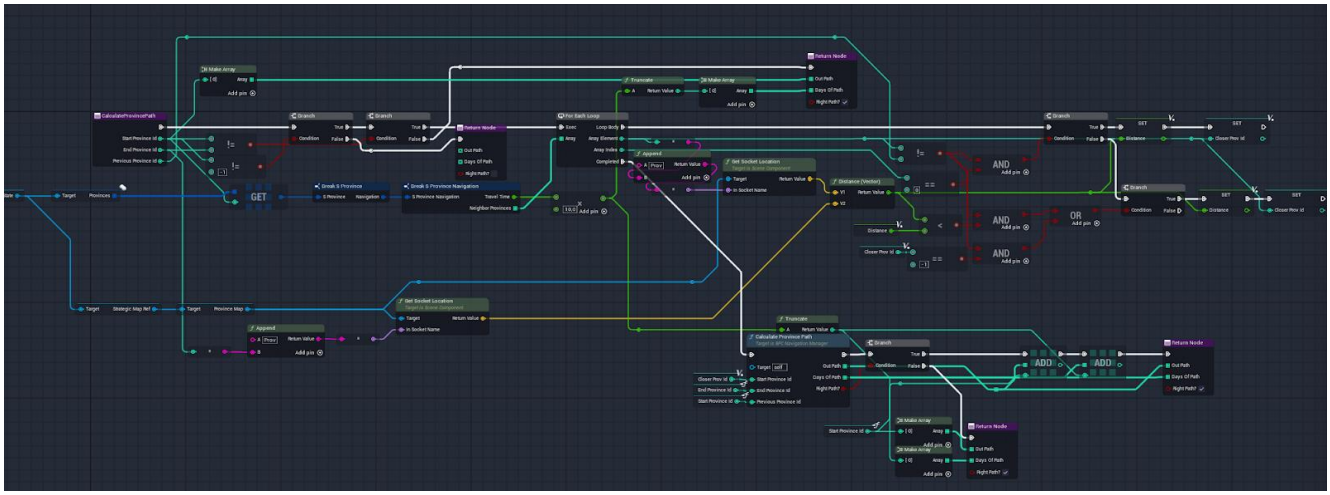
```
public:
	/** Please add a function description */
	UFUNCTION(BlueprintCallable)
	void FindPath(int32 StartProvince, int32 EndProvince, TArray<int32>& Path, TArray<int32>& DaysOfPath);

	/** Please add a function description */
	UFUNCTION(BlueprintCallable)
	void FindVisibleProvinces(int32 CountryId, TArray<int32>& VisibleProvinces, TArray<int32> TempVisibleProvinces);

	/** Please add a function description */
	UFUNCTION(BlueprintCallable)
	void HandleMovement(int32 TempProv, TArray<int32> TempPath, TArray<int32> TempDays, TArray<int32> Delete?, TArray<FS_ArmyId> Temp);

	/** Please add a function description */
	UFUNCTION(BlueprintCallable)
	void AddArmy(int32 ProvinceId, FS_ArmyId ArmyID, FS_ArmyGroup NewLocalVar);

	/** Please add a function description */
	UFUNCTION(BlueprintPure)
	void RetrieveActualArmy(FS_ArmyId ArmyID, bool& Result, UBP_Country_C*& Country, FS_Army& Army, int32& Index);

	/** Please add a function description */
	UFUNCTION(BlueprintPure)
```

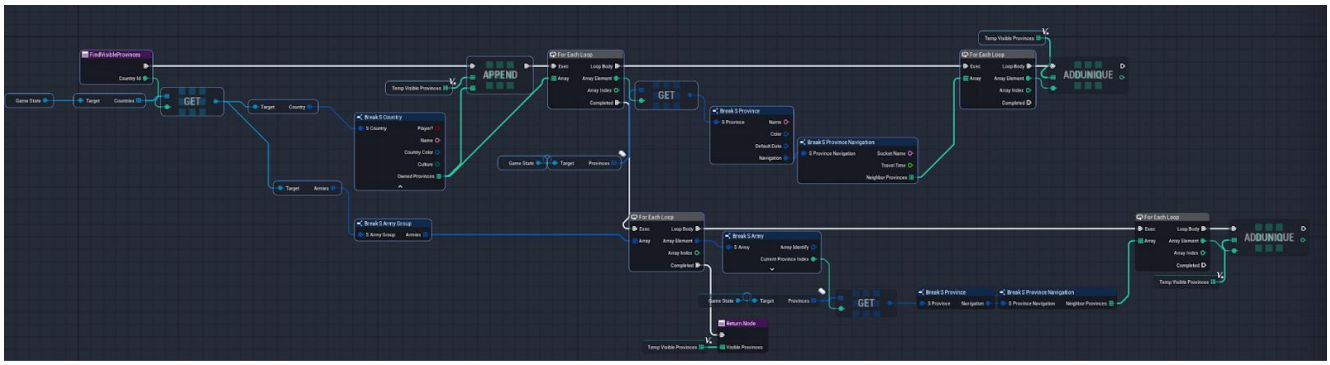
```
	void GetArmiesInProvince(int32 ProvinceId, FS_ArmyIdsArray& ArmyIds);
protected:
	/** Please add a function description */
	UFUNCTION(BlueprintCallable)
	void CalculateProvincePath(int32 StartProvinceId, int32 EndProvinceId, int32 PreviousProvinceId, TArray<int32>& OutPath, TArray<int32>& DaysOfPath, bool& RightPath?, double Distance, int32 CloserProvId);
public:
	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TObjectPtr<ABP_HybridGameMode_C> GameMode;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TMap<int32,FS_ArmyIdsArray> ProvinceWithArmies;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	bool TwoArmiesInProvince?;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	bool NewVar;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TObjectPtr<ABP_HybridGameState_C> GameState;
};
```


HandleMovement
KEYS
For Each Loop
FIND
LAST INDEX
GET
LENGTH
ADD
CLEAR
REMOVE INDEX
Branch
Sequence
SET


GET
CLEAR
ADD
REMOVE
REMOVE INDEX


REMOVE INDEX
ADD
FIND
REMOVE INDEX

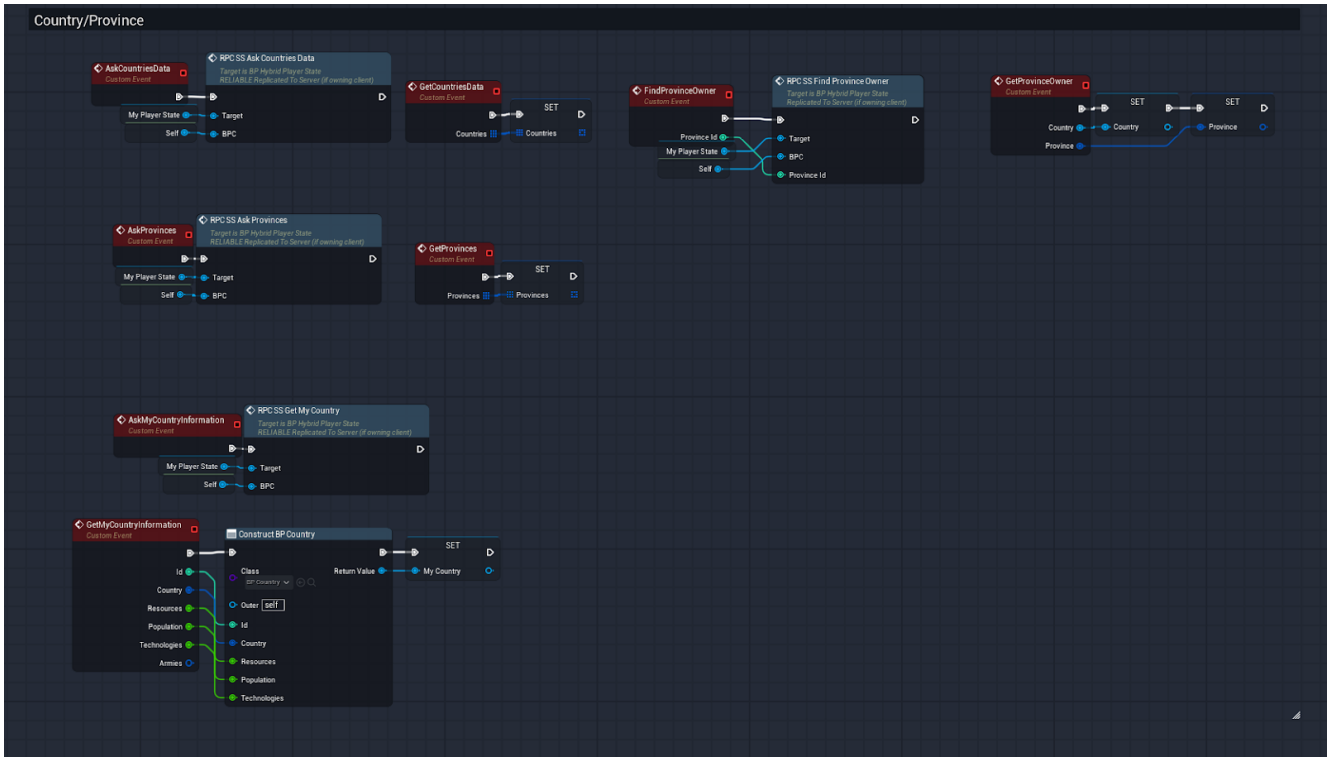


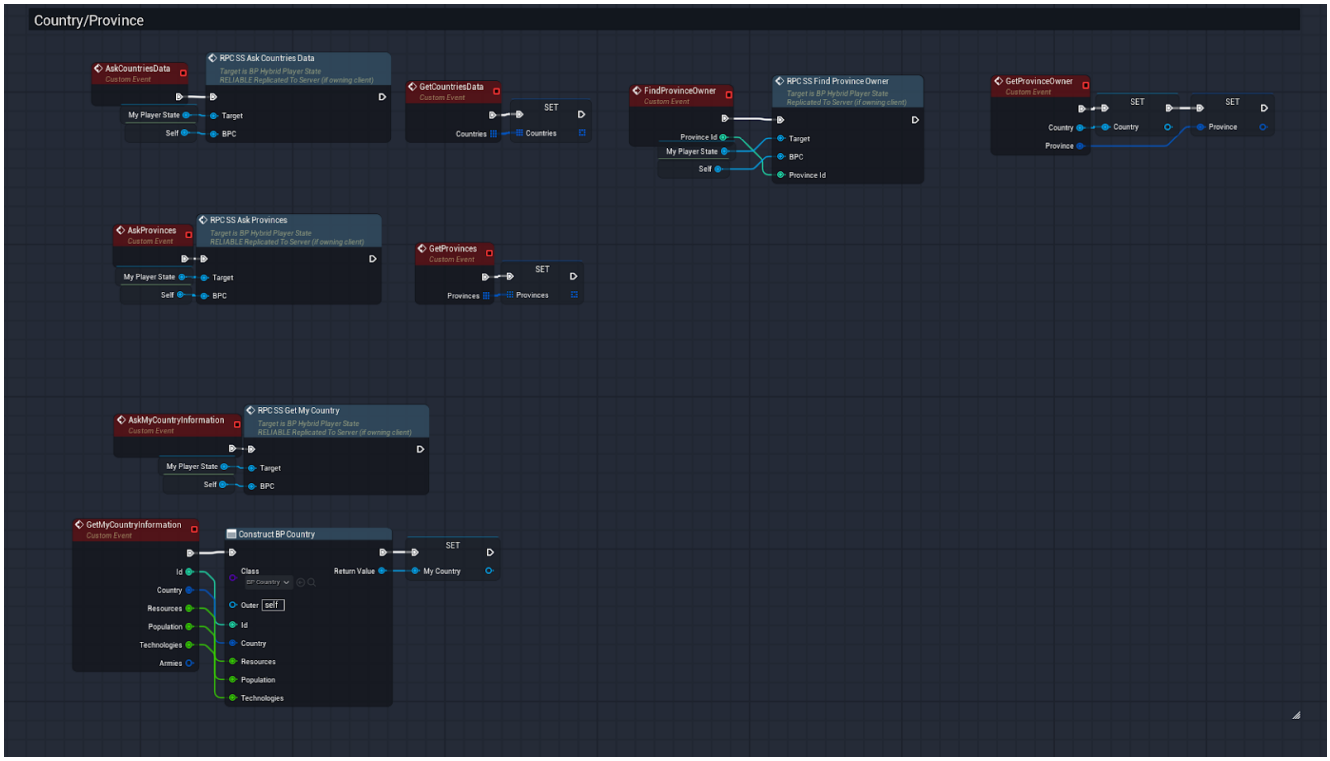


```
/** Please add a class description */
UCLASS(Blueprintable, BlueprintType)
class UBPC_AskPlayerState : public UActorComponent
{
	GENERATED_BODY()
public:
	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	bool DidIHire;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TObjectPtr<UBP_Country_C> MyCountry;
```

```
/** Please add a variable description */
UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
TArray<FS_Province> Provinces;

/** Please add a variable description */
UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
TArray<FS_Country> Countries;

/** Please add a variable description */
UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
bool DidIBuild;

/** Please add a variable description */
UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
TObjectPtr<ABP_HybridPlayerState_C> MyPlayerState;

/** Please add a variable description */
UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
FS_Army TempArmy;

/** Please add a variable description */
UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
TArray<int32> VisibleProvinces;

/** Please add a variable description */
UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
TObjectPtr<UBP_Country_C> Country;
```

```
    /** Please add a variable description */
    UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
    FS_Province Province;
};
```



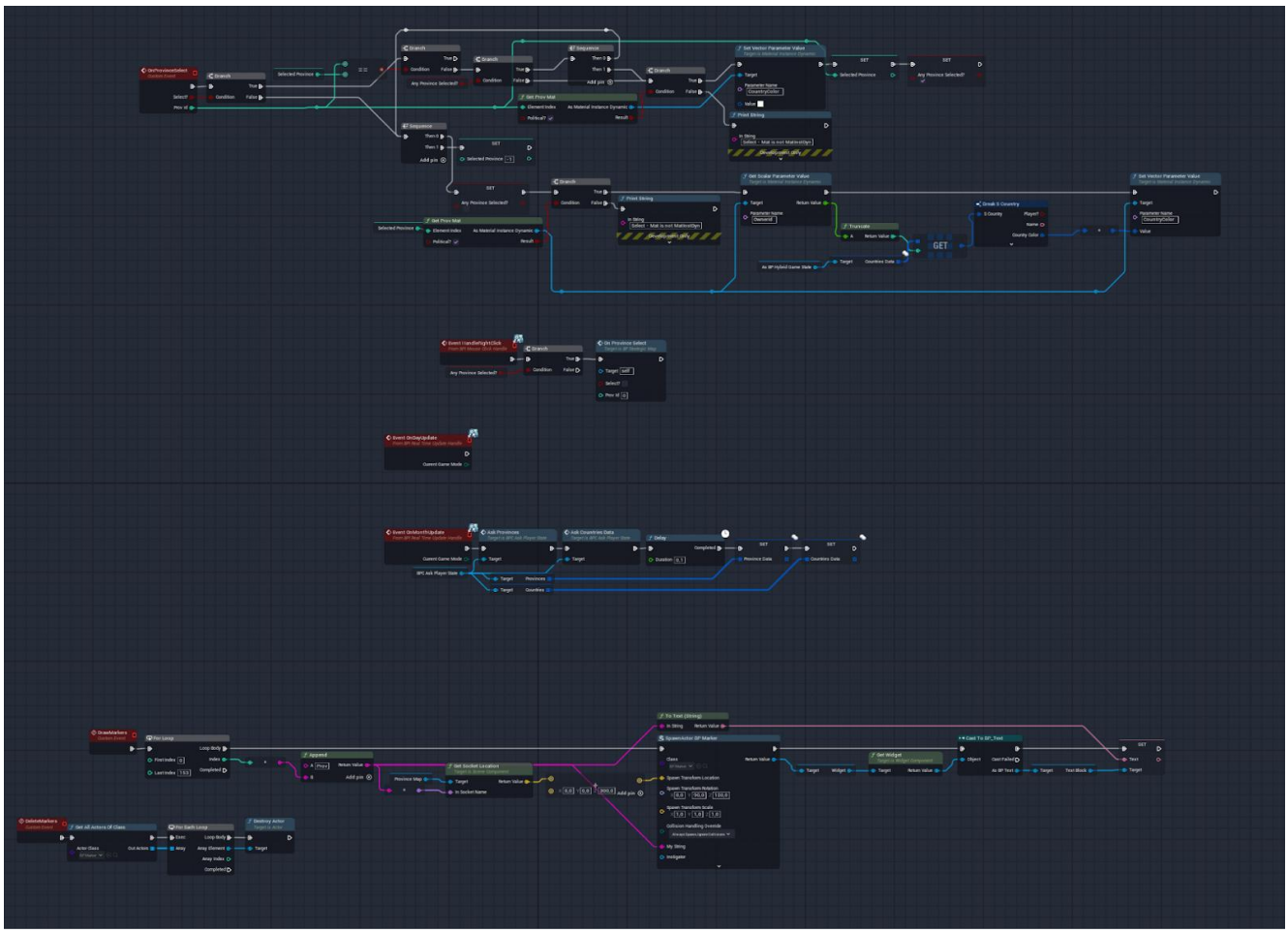


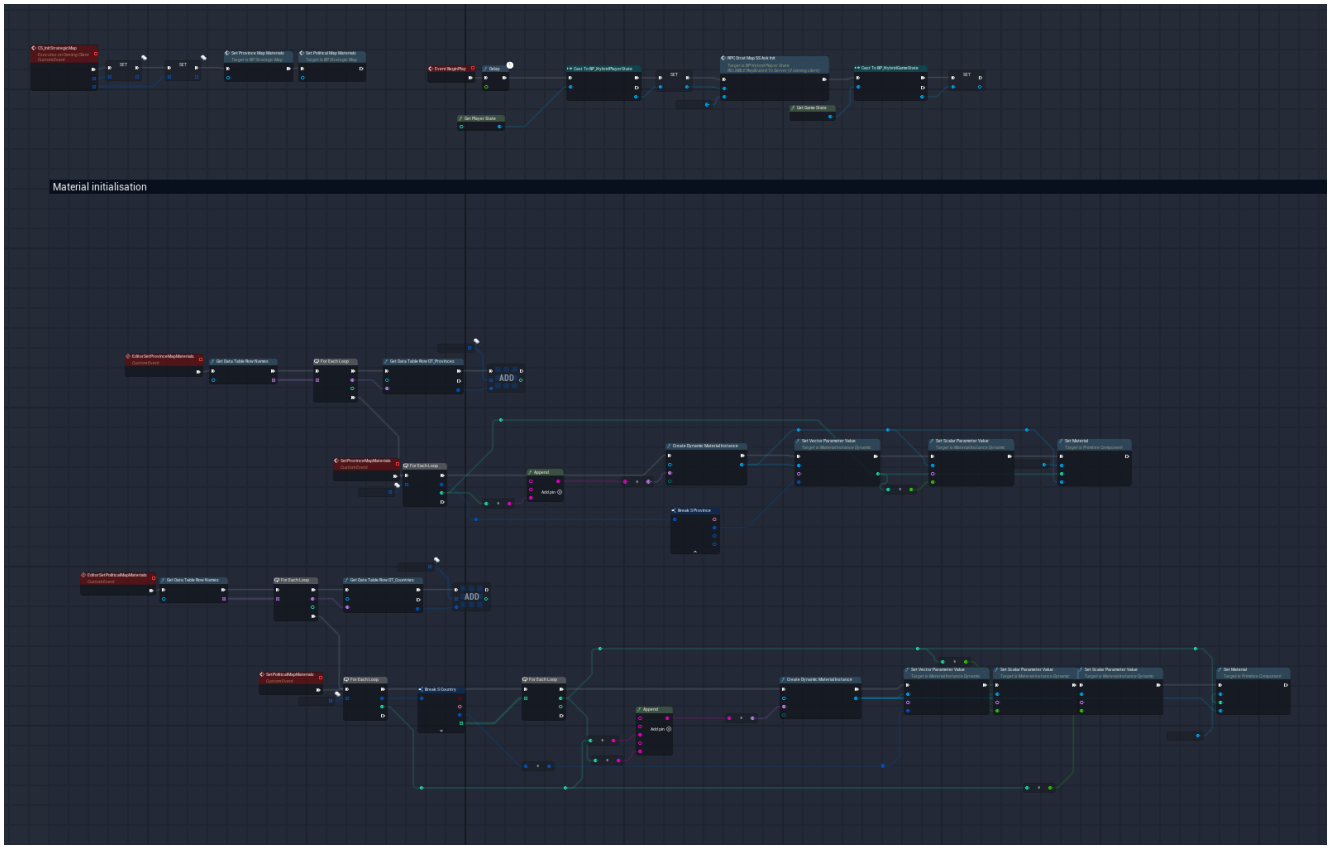

```
/** Please add a class description */
UCLASS(Blueprintable, BlueprintType)
class ABP_StrategicMap : public AActor
{
	GENERATED_BODY()
public:
	/** Please add a function description */
	UFUNCTION(BlueprintPure)
	void GetProvMat(int32 ElementIndex, bool Political?,
UMaterialInstanceDynamic*& AsMaterial Instance Dynamic, bool& Result);
public:
	/** Please add a variable description */
	UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
	TObjectPtr<USceneComponent> DefaultSceneRoot;
```

```
/** Please add a variable description */
UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
TObjectPtr<UBPC_AskPlayerState_C> BPC_AskPlayerState;

/** Please add a variable description */
UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
TObjectPtr<UStaticMeshComponent> underplate3;

/** Please add a variable description */
UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
TObjectPtr<UStaticMeshComponent> underplate1;

/** Please add a variable description */
UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
TObjectPtr<UStaticMeshComponent> borders map;

/** Please add a variable description */
UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
TObjectPtr<UStaticMeshComponent> political map;

/** Please add a variable description */
UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
TObjectPtr<UStaticMeshComponent> province map;

/** Please add a variable description */
UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category="Default")
TObjectPtr<UBPC_UnitManager_C> BPC_UnitManager;
```

```
	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TObjectPtr<ABP_HybridPlayerState_C> MyPlayerState;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	int32 SelectedProvince;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	bool AnyProvinceSelected?;

	/** Please add a variable description */
	UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
	TObjectPtr<ABP_HybridGameState_C> As BP Hybrid Game State;
private:
	/** Please add a variable description */
	static_assert(false, "You will need to add
DOREPLIFETIME(ABP_StrategicMap, CountriesData) to
GetLifetimeReplicatedProps");
	UPROPERTY(EditDefaultsOnly, Category="Default", Replicated)
	TArray<FS_Country> CountriesData;

	/** Please add a variable description */
	static_assert(false, "You will need to add
DOREPLIFETIME(ABP_StrategicMap, ProvinceData) to
GetLifetimeReplicatedProps");
	UPROPERTY(EditDefaultsOnly, Category="Default", Replicated)
```

```
    TArray<FS_Province> ProvinceData;
};
```




Material initialisation