КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра технологій управління

Спеціальність 122 «Комп'ютерні науки»

Освітньо-наукова програма «Управління проектами»

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему:

"Дослідження процесів управління проєктом з розробки візуального редактора дерев поведінки для ігрових систем ШІ в середовищі Unity."

Студента 2-го курсу групи УП-22	**Науковий керівник:**
Михайла МАЙСАКА	Кандидат економічних наук, доцент кафедри технологій управління
(ім'я, прізвище)	*(науковий ступінь, вчене звання)*
	Анна КОЛОМІЄЦЬ
	(ім'я, прізвище)
(підпис студента)	*(дата)* *(підпис)*

(Висновок: "До захисту в Екзаменаційній комісії")

Завідувач кафедри технологій управління Віктор МОРОЗОВ

(підпис) *(ім'я, прізвище)* *(дата)*

Київ – 2025

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій

Кафедра технологій управління
Освітній рівень Магістр
Спеціальність 122 Комп'ютерні науки
Освітня програма Управління проєктами

ЗАТВЕРДЖУЮ
Завідувач кафедри професор
Морозов В.В.

_____
"26" листопада 2024 року

ЗАВДАННЯ
НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Студент: Михайло МАЙСАК

Група: УП-22

1. Тема кваліфікаційної роботи:
   Дослідження процесів управління проєктом з розробки візуального редактора дерев поведінки для ігрових систем ШІ в середовищі Unity.
   Затверджена наказом по від "26"листопада 2024 р. № 5.
2. Строк подання студентом готової роботи - "12"травня 2025 р.
3. Цільова установка та вихідні дані до роботи: Дослідження методик, технік та інструментів управління проєктами для втілення обраного проєкту, досягнення визначених цілей та здобуття очікуваних результатів в рамках відведеного часу та бюджету.
4. Зміст роботи: формулювання проблемної області, аналіз літературних та інформаційних джерел, формулювання наукової новизни та інноваційності проєкту, створення концепції проєкту, вибір технологій , аналіз цільової аудиторії, розробка бізнес-моделі, формування організаційної структури управління проєктом, визначення ієрархічної структури робіт, розробка термінів виконання робіт, визначення та планування ресурсів, управління ризиками, розробка програмного забезпечення, тестування та оцінка ефективності запропонованого підходу.

5. Перелік графічного матеріалу (слайдів): Карта емпатії клієнта, Картки персон, SWOT-аналіз, Lean Canvas, Ієрархічна схема організаційної структури проєкту, Ілюстрації роботи з плануванням у Jira, Сюжетна карта беклогу, Загальна структура витрат, Графік вартості, Схема інформаційних потоків, Реалізація поширених вузлів BT, Реалізація візуального редактора через UI Builder, Приклад створеного BT для агресивного ШІ, Концептуальна модель ігрового AI, Структура ConfigService, Статичні дані про AI у GoogleSheet.
6. Календарний план виконання роботи:

№ п/п	Назва частин роботи	План виконання роботи
1.	Вибір теми кваліфікаційної роботи магістра (КРМ)	26.11.24
2.	Підготовка вступу	До 24.04.25
3.	Написання І розділу квалфікаційної роботи	15.02.25
4.	Написання ІІ розділу квалфікаційної роботи	28.02.25
5.	Написання ІІІ розділу квалфікаційної роботи	29.03.25
6.	Написання IV розділу квалфікаційної роботи	29.04.25
7.	Підготовка висновків і пропозицій	29.04.25
8.	Остаточне оформлення кваліфікаційної роботи	08.05.25
9.	Попередній захист кваліфікаційної роботи	12.05.25
10.	Захист магістерської кваліфікаційної роботи	26.05.25-28.05.25

Дата видачі завдання “26” листопада 2024 р.
Керівник роботи:
Кандидат економічних наук, доцент
Анна КОЛОМІЄЦЬ

_____
(підпис)

Завдання прийняв до виконання:
Студент групи УП-22
Михайло МАЙСАК

_____
(підпис)

ЗМІСТ

АНОТАЦІЯ

кваліфікаційної роботи магістра на тему:

«Дослідження процесів управління проєктом з розробки візуального редактора дерев поведінки для ігрових систем ШІ в середовищі Unity»

Студент: Майсак Михайло Костянтинович.

Науковий керівник: Коломієць Анна Степанівна.

Рік захисту - 2025.

Кваліфікаційна робота присвячена дослідженню процесів управління проєктом з розробки інструментарію для ігрового ШІ на основі гібридної архітектури Behavior Trees та Entity-Component-System.

Об'єктом дослідження є система розробки інструментальних засобів для створення ігрового ШІ.

Предметом дослідження є процеси управління проєктом з розробки візуального редактора дерев поведінки на основі гібридної архітектури BT-ECS.

Мета роботи — розробка візуального редактора дерев поведінки для ігрових систем ШІ в середовищі Unity, що забезпечує високу продуктивність, модульність та масштабованість.

Наукова новизна полягає у розробці гібридної архітектури ігрового AI, створенні інноваційного інструментарію для візуального проектування та експериментальному дослідженні запропонованої архітектури.

Дослідження підтвердило, що гібридна архітектура BT-ECS забезпечує підвищення ефективності на 65-93% порівняно з ООП-підходами. Розроблений інструментарій скорочує час створення систем ігрового ШІ та підвищує продуктивність, особливо на мобільних платформах.

Робота містить 113 сторінок, 27 рисунків та 12 таблиць. Додатки складають 10 сторінок.

Ключові слова: дерева поведінки, штучний інтелект, ігрова розробка, Entity-Component-System, Unity, візуальний редактор, проєктне управління, оптимізація продуктивності, модульність, масштабованість.

ВСТУП

Розробка комп'ютерних ігор є однією з найбільш динамічних та інноваційних галузей у сфері інформаційних технологій. Сучасні ігрові проєкти вражають своєю складністю, масштабністю та рівнем технологічної досконалості. Особливо важливу роль у цьому контексті відіграє штучний інтелект (ШІ), який відповідає за поведінку ігрових персонажів та загальну правдоподібність віртуального світу. Створення ефективного та реалістичного ігрового ШІ є складним завданням, яке вимагає застосування передових підходів та технологій.

Дана робота присвячена дослідженню та розробці проєкту зі створення візуального редактора дерев поведінки для ігрових систем ШІ в середовищі Unity. В онові програмного забезпечення лежить інноваційна гібридна архітектура для ігрового ШІ на основі Behavior Trees (BT) та Entity-Component-System (ECS). Актуальність теми зумовлена необхідністю вирішення ключових проблем, з якими стикаються розробники ігрового ШІ, таких як продуктивність, масштабованість, гнучкість та адаптивність. Традиційні підходи, засновані на скінченних автоматах, часто не відповідають вимогам сучасних ігрових проєктів, особливо в контексті розробки для мобільних платформ з обмеженими обчислювальними ресурсами.

Метою дослідження є розробка візуального редактора дерев поведінки для ігрових систем ШІ в середовищі Unity, що поєднує переваги архітектури Behavior Trees та Entity-Component-System (ECS) для забезпечення високої продуктивності, модульності та масштабованості ігрових систем ШІ.

Об'єктом дослідження система розробки інструментальних засобів для створення ігрового ШІ.

Предметом дослідження є процеси управління проєктом з розробки візуального редактора дерев поведінки на основі гібридної архітектури BT-ECS.

У рамках дослідження поставлено та вирішено наступні *завдання*:

1. Аналіз предметної області та обґрунтування доцільності створення спеціалізованого інструментарію для розробки ігрового ШІ.
2. Формування концепції продукту та розробка бізнес-моделі проєкту з урахуванням потреб цільової аудиторії та актуальних тенденцій ринку.
3. Розробка організаційної структури, формування команди та визначення методології управління проєктом.
4. Планування термінів, ресурсів та бюджету проєкту з використанням сучасних підходів до управління IT-проєктами.
5. Ідентифікація та аналіз ризиків проєкту, розробка стратегій їх мінімізації.
6. Проектування та програмна реалізація базових компонентів візуального редактора дерев поведінки та підтвердження концепції на прикладі практичного застосування інструменту.

Методологічною основою дослідження стали принципи проєктного управління, методи системного аналізу, математичного моделювання та програмної інженерії. Для обґрунтування ефективності гібридної архітектури BT-ECS використано методи кількісної оцінки продуктивності та масштабованості програмних систем.

Наукова новизна дослідження полягає у розробці у розробці інноваційної гібридної архітектури ігрового ШІ, яка поєднує переваги Behavior Trees та Entity-Component-System, забезпечуючи принципи гнучкої розробки, компонентно-орієнтованої архітектури та візуального програмування для досягнення оптимального балансу між гнучкістю, масштабованістю та продуктивністю.

Практична цінність отриманих результатів полягає у можливості їх безпосереднього застосування при розробці ігрових проєктів різного масштабу та жанрів. Розроблений інструментарій та програмний прототип можуть бути використані ігровими студіями та незалежними розробниками для створення більш ефективних систем ігрового ШІ, скоротити часові та фінансові витрати на створення інтелектуальних агентів у відеоіграх, а також

забезпечити вищу якість та реалістичність їхньої поведінки. Результати дослідження були впроваджені при розробці реального ігрового проєкту.

Апробація результатів дослідження. Основні положення та результати дослідження представлено на Всеукраїнському конкурсі студентських наукових робіт зі спеціальності "Управління проєктами та програмами" у 2024/2025 н.р., де робота здобула третє місце у першому турі.

Ключові результати, що будуть представлені в роботі, включають:

- Математичну модель оцінки ефективності гібридної архітектури BT-ECS порівняно з традиційними підходами, що демонструє суттєве підвищення продуктивності (до 75%) та скорочення часу розробки.
- Розроблену бізнес-модель проєкту за методологією Lean Canvas з детальним аналізом цільової аудиторії, каналів поширення та потоків доходів.
- Комплексну систему управління проєктом, включаючи організаційну структуру, матрицю відповідальності, календарний план та стратегію управління ризиками.
- Архітектуру та програмну реалізацію візуального редактора дерев поведінки, що поєднує переваги BT та ECS для забезпечення високої продуктивності та гнучкості.
- Результати практичного впровадження розробленого інструментарію, що підтверджують його ефективність у реальних умовах ігрової розробки.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ ТА ЖИТТЄЗДАТНОСТІ ПРОЄКТУ

1.1 Аналіз методів оцінки впливів оточення ІТ проєктів, функціонального призначення окремих частин проєктів, об'єктів, що захищаються. Дослідження існуючих ІТ в предметній галузі

Проектування та реалізація ефективної системи AI вимагає комплексного підходу та врахування багатьох факторів. Зокрема, необхідно ретельно проаналізувати вплив оточення ІТ проєкту, включаючи апаратні обмеження цільових платформ, вимоги до продуктивності та масштабованості, а також очікування гравців щодо інтелектуальності та реалістичності поведінки ігрових агентів.

У випадку мобільних ігор, які є одним з найбільш перспективних та швидкозростаючих сегментів ринку, ці фактори набувають особливого значення. Обмежені обчислювальні ресурси мобільних пристроїв, різноманітність апаратних конфігурацій та високі очікування користувачів створюють унікальні виклики для розробників ігрового AI.

З функціональної точки зору, система AI в ігрових проєктах відповідає за широкий спектр задач, пов'язаних з автономним прийняттям рішень та адаптивною поведінкою ігрових персонажів. Серед ключових компонентів такої системи можна виділити:

1. Модуль сприйняття навколишнього середовища: відповідає за збір та інтерпретацію інформації про ігровий світ.
2. Модуль прийняття рішень: на основі поточного стану гри та заданих цілей визначає оптимальну послідовність дій для ігрових агентів.
3. Модуль виконання дій: реалізує прийняті рішення, контролюючи анімацію, переміщення та взаємодію персонажів з ігровим світом.
4. Модуль адаптації: дозволяє ігровим агентам динамічно коригувати свою поведінку на основі досвіду та зворотного зв'язку, отриманого в процесі гри.

Ефективна взаємодія цих модулів є запорукою створення правдоподібного та захоплюючого ігрового досвіду.

Аналіз існуючих підходів до проектування ігрового AI показує, що найбільш поширеними методами є використання скінченних автоматів, дерев прийняття рішень, навігаційних графів та різноманітних евристичних алгоритмів [10]. Проте, зі зростанням складності та масштабів ігрових проєктів, ці традиційні підходи все частіше демонструють свої обмеження, особливо з точки зору гнучкості, модульності та продуктивності [11].

В останні роки спостерігається активний розвиток та впровадження більш досконалих архітектурних шаблонів та алгоритмічних рішень в сфері ігрового AI. Зокрема, значної популярності набуває використання дерев поведінки (Behavior Trees) для моделювання складних паттернів прийняття рішень [12]. Цей підхід дозволяє декомпозувати поведінку агентів на атомарні дії та умови, які можуть гнучко компонуватись та розширюватись без необхідності вносити зміни в існуючу логіку.

Іншим перспективним напрямком є застосування архітектурного шаблону Entity-Component-System (ECS) для оптимізації продуктивності та масштабованості ігрових систем, включаючи AI. ECS дозволяє відокремити дані від поведінки, що значно спрощує процес розробки, тестування та модифікації окремих компонентів гри.

Окремої уваги заслуговують методи машинного навчання та нейронні мережі, які дозволяють створювати адаптивних та самонавчальних ігрових агентів. Хоча ці підходи все ще знаходяться на стадії активних досліджень та експериментів, вони демонструють значний потенціал для революційних змін в сфері ігрового AI [17].

Підсумовуючи, можна констатувати, що розробка ефективної та інноваційної системи AI є критично важливою для успіху сучасних ігрових проєктів. Врахування факторів оточення, функціональних вимог та обмежень цільових платформ, а також використання передових архітектурних та

алгоритмічних рішень є запорукою створення по-справжньому інтелектуальних та захоплюючих ігрових світів.

1.2 Формулювання проблемної області

Незважаючи на стрімкий прогрес у сфері ігрового AI, розробники все ще стикаються з низкою фундаментальних проблем та викликів. Ці проблеми можна умовно розділити на кілька ключових категорій:

1. Продуктивність та масштабованість. Зі зростанням розмірів та деталізації ігрових світів, а також кількості одночасно активних агентів, різко зростає навантаження на обчислювальні ресурси. Традиційні підходи до проектування AI, які базуються на складних ієрархіях класів та інтенсивних обчисленнях в реальному часі, часто призводять до проблем з продуктивністю та обмежують масштабованість ігрових систем. Особливо гостро ця проблема стоїть для мобільних платформ, які мають порівняно невеликі обсяги оперативної пам'яті та обчислювальні потужності.

2. Гнучкість та модульність. Класичні архітектури ігрового AI, такі як скінченні автомати, часто страждають від надмірної жорсткості та складності внесення змін. З ускладненням ігрової логіки та появою нових вимог, підтримка та розширення таких систем стає справжнім викликом для розробників. Це призводить до збільшення часу та вартості розробки, а також підвищує ризики виникнення помилок та неузгодженостей в роботі AI.

3. Реалістичність та правдоподібність поведінки. Однією з ключових цілей ігрового AI є створення переконливої ілюзії інтелекту та реалістичності поведінки ігрових персонажів. Проте, досягнення цієї мети часто вимагає врахування величезної кількості факторів та розробки складних моделей прийняття рішень. Традиційні підходи, які базуються на жорстко визначених правилах та сценаріях, часто призводять до передбачуваної та неприродньої поведінки агентів, що значно знижує занурення гравця у віртуальний світ.

4. Адаптивність. В реальному світі інтелектуальні агенти здатні динамічно адаптуватись до нових ситуацій. Натомість, переважна більшість

сучасних ігрових AI все ще покладається на статичні, наперед визначені моделі поведінки, які не здатні ефективно реагувати на непередбачувані дії гравця та зміни в ігровому середовищі.

5. Баланс складності та передбачуваності. З одного боку, ігровий AI повинен забезпечувати достатній рівень складності та непередбачуваності, щоб підтримувати інтерес та азарт гравця. З іншого боку, надмірно складний та хаотичний AI може призвести до фрустрації та розчарування, особливо для недосвідчених гравців [31]. Знаходження оптимального балансу між цими двома факторами є складним завданням, яке вимагає глибокого розуміння психології гравців та ретельного тестування ігрових механік [31].

Підсумовуючи, можна констатувати, що розробка ефективного, правдоподібного та адаптивного ігрового AI залишається однією з найбільших викликів для сучасної ігрової індустрії. Вирішення цих проблем вимагає не тільки застосування передових технологічних підходів, але й фундаментального переосмислення самих принципів проектування та реалізації інтелектуальних агентів у віртуальних світах.

1.3 Проведення аналізу літературних та інформаційних джерел щодо можливостей вирішення виявлених проблем

Для вирішення проблем, описаних у попередньому підрозділі, було проведено ґрунтовний аналіз літературних та інформаційних джерел, які висвітлюють сучасні підходи та технології в сфері ігрового AI. Особлива увага була приділена дослідженням, які стосуються застосування архітектури Behavior Trees (BT) та принципів Entity-Component-System (ECS) для підвищення ефективності, гнучкості та масштабованості ігрових систем.

Ключовим джерелом інформації стала фундаментальна праця Millington та Funge "Artificial Intelligence for Games" [33], яка детально описує різноманітні методи та алгоритми, що використовуються в ігровому AI. Автори приділяють значну увагу концепції BT як потужного інструменту для моделювання складної та адаптивної поведінки ігрових агентів. Вони наводять

переконливі аргументи на користь використання BT в порівнянні з традиційними підходами, такими як скінченні автомати.

Ці ідеї знаходять підтвердження в роботах інших дослідників та практиків ігрової індустрії. Зокрема, в своїй лекції "Behavior Trees for Next-Gen Game AI" [34] Champandard розкриває переваги BT з точки зору модульності, гнучкості та зручності налагодження. Він демонструє, як BT дозволяють розробникам створювати складні патерни поведінки з простих, атомарних компонентів, які можуть бути легко модифіковані та повторно використані в різних контекстах.

Ідеї Champandard знаходять практичне підтвердження в роботі Lim et al. "Evolving Behaviour Trees for the Commercial Game DEFCON" [35], яка описує успішний досвід застосування еволюційних алгоритмів для автоматичної генерації BT в комерційній грі. Автори демонструють, як такий підхід дозволив створити різноманітні та ефективні патерни поведінки для ігрових агентів без необхідності ручного програмування.

Окрім BT, значна увага в літературі приділяється концепції ECS як архітектурного шаблону для оптимізації продуктивності та масштабованості ігрових систем. Ґрунтовний опис принципів та переваг ECS можна знайти в роботі Nystrom "Game Programming Patterns" [36], яка є свого роду настільною книгою для розробників ігор. Автор показує, як відокремлення даних від логіки та використання компонентно-орієнтованого підходу дозволяє створювати гнучкі, модульні та високопродуктивні ігрові системи.

Ці ідеї знаходять розвиток в статті Stacey et al. "The Entity-Component-System Pattern for Game Development" [37], яка детально описує практичні аспекти застосування ECS в ігрових проєктах. Автори наводять приклади реальних ігор, в яких використання ECS дозволило досягти вражаючих результатів з точки зору продуктивності та масштабованості.

Цікавим доповненням до теми ECS є дослідження Wiebusch та Latoschik "Decoupling the Entity-Component-System Pattern using Semantic Traits for Reusable Realtime Interactive Systems" [38], в якому пропонується розширення класичної моделі ECS з використанням семантичних характеристик для

підвищення модульності та повторного використання компонентів в інтерактивних системах реального часу. Ця робота демонструє потенціал ECS як потужного архітектурного шаблону, який може бути адаптований та вдосконалений для вирішення специфічних задач ігрової розробки.

Ще одним важливим напрямком досліджень є розробка інструментів та середовищ для спрощення та автоматизації процесу створення ігрового AI. Наприклад, в роботі Grow та Gaudl "A Framework for AI-Based Playtesting" [41] пропонується фреймворк для автоматизованого тестування ігрового AI з використанням методів еволюційного пошуку та навчання з підкріпленням. Такий підхід дозволяє значно скоротити час та зусилля, необхідні для налаштування та збалансування ігрової механіки.

Іншим прикладом інноваційного інструменту для розробки ігрового AI є система Craft, описана в роботі Miller et al. "Craft: An API for Creating AI-Human Interactive Narratives" [42]. Ця система дозволяє створювати інтерактивні наративи з використанням природної мови та автоматично генерувати відповідні репліки та дії ігрових персонажів на основі заданого контексту та цілей. Такий підхід відкриває нові можливості для створення більш природних та захоплюючих ігрових діалогів та сюжетів.

Підсумовуючи результати проведеного аналізу літературних та інформаційних джерел, можна зробити висновок, що комбінація архітектури Behavior Trees та принципів Entity-Component-System є найбільш перспективним напрямком для вирішення ключових проблем ігрового AI, таких як продуктивність, масштабованість, гнучкість та адаптивність. Водночас, розробка та впровадження цих підходів вимагає значних зусиль та інвестицій з боку ігрових студій та дослідницьких центрів.

На основі проведеного аналізу предметної області та огляду сучасних підходів до розробки ігрового AI, можна сформулювати наукову новизну та інноваційність запропонованого проєкту наступним чином:

1. Розробка та дослідження гібридної архітектури ігрового AI, що поєднує переваги Behavior Trees (BT) та Entity-Component-System (ECS) для

створення ефективних, масштабованих та гнучких ігрових систем. На відміну від традиційних підходів, таких як скінченні автомати та дерева прийняття рішень, запропонована архітектура дозволяє моделювати складну та реалістичну поведінку ігрових агентів з використанням модульних, повторно використовуваних компонентів, що значно спрощує процес розробки, налагодження та модифікації AI-системи.

2. Розробка інноваційного інструментарію для візуального проектування, налагодження та оптимізації ігрового AI з використанням BT та ECS. Запропонований набір інструментів дозволяє розробникам та дизайнерам працювати з AI-системою на більш високому рівні абстракції, зосереджуючись на логіці та патернах поведінки, а не на низькорівневих деталях реалізації. Це значно прискорює процес ітеративної розробки та полегшує експериментування з різними конфігураціями та параметрами AI.

3. Проведення експериментального дослідження запропонованої архітектури та методів у контексті розробки реальної гри на ігровому рушії Unity. На відміну від більшості існуючих досліджень, які зосереджуються на теоретичних аспектах або спрощених прототипах, даний проєкт передбачає практичну реалізацію та апробацію розроблених підходів в умовах повноцінного ігрового проєкту. Це дозволяє отримати цінні дані щодо ефективності, продуктивності та обмежень запропонованої архітектури в реальних умовах розробки.

4. Формування науково-обґрунтованих рекомендацій та кращих практик щодо застосування BT та ECS в ігровому AI, з урахуванням специфіки різних ігрових жанрів, платформ та масштабів проєктів. Ці рекомендації можуть стати ціннім внеском у розвиток теорії та практики ігрового AI, допомагаючи розробникам та дослідникам приймати більш обґрунтовані рішення та уникати типових помилок при проєктуванні та реалізації інтелектуальних агентів у відеоіграх.

Таким чином, запропонований проєкт має значну наукову новизну та інноваційність, пропонуючи комплексний підхід до вирішення ключових

проблем ігрового AI з використанням передових архітектурних шаблонів та інструментів розробки.

1.4 Постановка задачі дослідження, формулювання технічного завдання на розробку у вигляді паспорту проєкту

Метою дослідження є розробка та апробація інструменту для створення створення ефективних, масштабованих та адаптивних ігрових агентів з використанням інноваційної гібридної архітектури ігрового AI на основі Behavior Trees (BT) та Entity-Component-System (ECS) у контексті розробки повноцінного ігрового проєкту на рушії Unity.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) Провести аналіз предметної області, дослідити сучасні підходи та технології розробки ігрового AI, виявити їх переваги, недоліки та обмеження.
2) Розробити концептуальну модель гібридної архітектури ігрового AI на основі BT та ECS, визначити ключові компоненти, інтерфейси та механізми взаємодії.
3) Формалізувати математичну модель запропонованої архітектури, провести теоретичний аналіз її ефективності, масштабованості та адаптивності в порівнянні з традиційними підходами.
4) Спроєктувати та реалізувати програмний прототип запропонованої архітектури з використанням ігрового рушія Unity, створити набір модульних BT-компонентів для моделювання різних аспектів ігрової поведінки.
5) Розробити набір інструментів для візуального проектування, налагодження та оптимізації ігрового AI з використанням BT та ECS, забезпечити інтеграцію цих інструментів з ігровим рушієм Unity.
6) Провести експериментальне дослідження ефективності та продуктивності розробленої архітектури в контексті реального ігрового проєкту, зібрати та

проаналізувати дані щодо швидкодії, використання ресурсів та якості ігрової поведінки.

7) Сформувати науково-обґрунтовані рекомендації та кращі практики щодо застосування запропонованих підходів та інструментів в ігровій індустрії, з урахуванням специфіки різних ігрових жанрів, платформ та масштабів проєктів.
8) Оформити результати дослідження у вигляді кваліфікаційної роботи.

Технічне завдання на розробку (паспорт проєкту):

1) Назва проєкту: Розробка гібридної архітектури ігрового AI на основі Behavior Trees та Entity-Component-System зі створенням інструменту візуального проектування.
2) Мета проєкту: Створення ефективної, масштабованої та адаптивної архітектури ігрового AI для реалізації реалістичної ігрових агентів в контексті повноцінного ігрового проєкту на рушії Unity.
3) Очікувані результати:
   - Концептуальна модель гібридної BT-ECS архітектури ігрового AI та математична модель дослідження ефективності цієї архітектури.
   - Програмний прототип архітектури, реалізований з використанням ігрового рушія Unity.
   - Набір модульних BT-компонентів для моделювання різних аспектів ігрової поведінки.
   - Інструментарій для візуального проектування, налагодження та оптимізації ігрового AI з використанням BT та ECS.
   - Результати експериментального дослідження ефективності та продуктивності розробленої архітектури в контексті реального ігрового проєкту.
   - Науково-обґрунтовані рекомендації та кращі практики щодо застосування запропонованих підходів та інструментів в ігровій індустрії.
4) Ключові етапи проєкту:

- Аналіз предметної області та огляд існуючих підходів до розробки ігрового AI.
- Розробка концептуальної моделі гібридної BT-ECS архітектури та математичної моделі дослідження ефективності цієї архітектури.
- Реалізація програмного прототипу архітектури та модульних BT-компонентів на рушії Unity.
- Розробка інструментарію для візуального проектування та налагодження ігрового AI.
- Проведення експериментального дослідження та аналіз результатів
- Формування рекомендацій та кращих практик, оформлення магістерської дисертації та підготовка публікацій

5) Необхідні ресурси:
   - Ігровий рушій Unity, середовище розробки Rider.
   - ПК або робоча станція для розробки та тестування.
   - Доступ до наукових баз даних та джерел інформації.
   - Науковий керівник та консультанти з досвідом в сферах ігрового AI та розробки ігор.
6) Ризики проєкту:
   - Технічні труднощі при реалізації окремих компонентів архітектури та інтеграції з ігровим рушієм.
   - Недостатня ефективність або масштабованість розроблених підходів в умовах реального ігрового проєкту.
   - Потенційні затримки в процесі розробки та тестування через непередбачувані обставини.
7) Критерії успішності проєкту:
   - Розроблена архітектура забезпечує високу ефективність, масштабованість та гнучкість ігрового AI в порівнянні з традиційними підходами.
   - Ігрові агенти демонструють реалістичну, адаптивну та інтелектуальну поведінку, що відповідає очікуванням гравців та вимогам ігрового

жанру.

- Запропоновані інструменти та методи дозволяють значно спростити та прискорити процес розробки, налагодження та оптимізації ігрового AI.
- Результати експериментального дослідження підтверджують ефективність та переваги розробленої архітектури в контексті реального ігрового проєкту.
- Сформовані рекомендації та кращі практики мають практичну цінність для ігрової індустрії та можуть бути застосовані в інших проєктах та дослідженнях.

Представлений паспорт проєкту окреслює ключові аспекти дослідження, визначає основні етапи, необхідні ресурси та критерії успішності. Він може бути використаний як основа для планування та управління процесом розробки та дослідження гібридної архітектури ігрового AI на основі BT та ECS. Варто зазначити, що документ має динамічний характер і буде уточнюватися під час реалізації проєкту на базі зворотного зв'язку від цільової аудиторії та результатів проміжного тестування. Систематичні перегляди та оцінка прогресу відносно визначених у паспорті критеріїв допоможуть забезпечити виконання всіх поставлених завдань у межах встановлених термінів та ресурсних обмежень.

Реалізація цього амбітного проєкту вимагатиме значних зусиль, знань та навичок в сферах ігрової розробки та програмної інженерії. Водночас, успішне виконання поставлених задач та досягнення очікуваних результатів може стати вагомим внеском у розвиток теорії та практики ігрового AI, відкриваючи нові горизонти для створення більш реалістичних, захоплюючих та інтелектуально насичених ігрових світів.

РОЗДІЛ 2. ОПИС КОНЦЕПЦІЇ ПРОЄКТУ

2.1 Опис продукту проєкту

BehaviorTreeDesigner - візуальний редактор для Unity з інтегрованою системою налагодження та оптимізації дерев поведінки.

Назва проєкту: Проєкт розробки візуального редактора BehaviorTreeDesigner для створення AI в середовищі Unity.

Місія проєкту назовні (повна): BehaviorTreeDesigner надає розробникам ігор потужний інструментарій для створення складних систем AI без необхідності написання складного коду. Наш продукт дозволяє скоротити час розробки AI-систем, забезпечує повний візуальний контроль над поведінкою персонажів та надає гнучкі можливості для налагодження та оптимізації.

Місія проєкту назовні (коротка, слоган): «Створюйте розумних персонажів швидко та інтуїтивно. BehaviorTreeDesigner - ваш особистий AI-архітектор.»

Місія проєкту усередину (програмна заява): Разом ми створюємо інноваційний продукт, який змінить підхід до розробки ігрового AI.

Кожен член команди отримає: Досвід роботи з передовими технологіями в галузі ігрової розробки, можливість створення продукту який використовуватимуть тисячі розробників, розвиток навичок в області архітектури складних систем та UI/UX дизайну, участь у формуванні нового стандарту для розробки ігрового AI, можливість стати експертом у своїй області та ділитися знаннями з спільнотою розробників.

2.2 Вибір технологій

Для реалізації візуального редактора дерев поведінки та супутньої системи виконання AI було необхідно обрати технологічний стек, який би забезпечував необхідну продуктивність, гнучкість та зручність розробки.

Вибір технологій ґрунтувався на детальному аналізі вимог до кінцевого продукту та потенційних сценаріїв використання.

2.2.1 Платформа та середовище розробки

Основою для розробки BehaviorTreeDesigner було обрано Unity 2021.3.42f1 - стабільну LTS версію популярного ігрового рушія. Цей вибір обумовлений такими перевагами: Довготривала підтримка (LTS) - гарантує стабільність та сумісність проєкту протягом тривалого часу що є критичним для інструменту який має інтегруватися з різними проєктами; Велика екосистема та спільнота - наявність великої кількості документації навчальних матеріалів та активна спільнота розробників значно спрощують процес розв'язання потенційних проблем під час розробки; Кросплатформність - можливість використовувати розроблений інструмент на різних платформах без необхідності значних змін у коді; Потужний редактор - вбудовані інструменти Unity для розширення редактора дозволяють створювати власні вікна інспектори та інші елементи інтерфейсу що є критичною вимогою для реалізації візуального редактора дерев поведінки; Asset Store - можливість публікації інструменту в Unity Asset Store що є одним із ключових каналів дистрибуції відповідно до бізнес-моделі проєкту.

2.2.2 Мова програмування та парадигми розробки

Основною мовою програмування для розробки BehaviorTreeDesigner обрано C#, що є стандартною мовою для розробки на Unity. Ключові переваги C# у контексті даного проєкту: Об'єктно-орієнтований підхід - дозволяє створювати чіткі абстракції та застосовувати паттерни проектування що є важливим для розробки гнучкої та розширюваної системи вузлів дерева поведінки; Підтримка узагальнень (generics) - забезпечує створення типобезпечного коду зокрема при реалізації різних типів вузлів та їх параметрів; Рефлексія - дозволяє виявляти та каталогізувати користувацькі

вузли дерева поведінки що спрощує розширення системи кінцевими користувачами; Атрибутний програмний інтерфейс - забезпечує зручне декларативне визначення метаданих для вузлів та параметрів цей аспект активно використовується для автоматичної генерації інтерфейсу редактора

2.2.3 Фреймворки

Entity-Component-System (ECS)

Для забезпечення високої продуктивності системи виконання дерев поведінки було обрано підхід Entity-Component-System з використанням легковагової та швидкої реалізації ECS-архітектури для C#, адаптованої для Unity. Вибір обумовлений наступними факторами: Висока продуктивність - порівняно з традиційним об'єктно-орієнтованим підходом ECS забезпечує значне підвищення швидкодії завдяки оптимізованій роботі з пам'яттю та ефективному механізму обробки компонентів; Низький рівень алокацій пам'яті - система працює на основі пулів об'єктів що мінімізує навантаження на збирач сміття що особливо важливо для мобільних платформ; Проста інтеграція з Unity – власний ECS не залежить від конкретних версій Unity DOTS який ще знаходиться у стадії активної розробки що зменшує ризики сумісності при оновленні Unity; Простота використання - порівняно з офіційним Unity DOTS власний ECS фреймворк має більш прозору та зрозумілу модель програмування що спрощує підтримку та розширення коду; Open-source статус - дозволяє за необхідності модифікувати фреймворк для специфічних потреб проєкту.

Використання ECS-архітектури забезпечує потужні переваги для реалізації системи виконання дерев поведінки: Розділення даних та логіки - чітке відокремлення стану вузлів (компоненти) від логіки їх обробки (системи) що полегшує модульне тестування та налагодження; Висока паралелізація - можливість паралельного виконання незалежних гілок дерев поведінки що критично для проєктів з великою кількістю AI-агентів; кешування даних -

ефективне розміщення компонентів у пам'яті підвищує локальність даних та зменшує кількість cache miss.

UI Toolkit

Для реалізації візуального редактора дерев поведінки обрано Unity UI Toolkit (раніше відомий як UI Elements) - сучасний фреймворк для створення користувацьких інтерфейсів у Unity. Обґрунтування вибору: Декларативний опис інтерфейсу - використання UXML (аналог HTML) та USS (аналог CSS) дозволяє чітко відокремити структуру інтерфейсу від логіки що підвищує модульність та спрощує підтримку; Висока продуктивність - UI Toolkit використовує сучасний рендерер з підтримкою апаратного прискорення та ефективною обробкою подій; Адаптивний дизайн - вбудована підтримка гнучких макетів полегшує адаптацію інтерфейсу під різні розміри вікон; Перспективність - UI Toolkit є стратегічним напрямком розвитку UI в Unity що забезпечує довгострокову підтримку та вдосконалення технології.

2.2.4 Інструменти зберігання та серіалізації даних

Для зберігання даних дерев поведінки обрано механізм ScriptableObjects - вбудований у Unity засіб для створення та керування конфігураційними даними. Цей вибір має натупні переваги: Зручність інтеграції з редактором Unity - ScriptableObjects підтримують стандартний механізм інспекторів Unity, що спрощує відображення та редагування властивостей; Ефективне управління ресурсами - дані, збережені у ScriptableObjects, оптимізовані для завантаження та вивантаження відповідно до потреб проєкту; Підтримка версійного контролю - формат ScriptableObjects добре працює з системами контролю версій, що важливо для командної розробки.

Для зберігання проміжних даних та стану виконання дерев поведінки використовується власна система серіалізації на основі ECS-компонентів, що забезпечує високу продуктивність та мінімальні накладні витрати під час виконання.

Для інтеграції з зовнішніми джерелами конфігураційних даних (зокрема, Google Sheets) реалізовано спеціалізовані адаптери, що використовують: UnityWebRequest - для асинхронного завантаження даних без блокування основного потоку виконання; Json.NET - для ефективного парсингу та перетворення JSON-даних з API Google Sheets; Система кешування - для мінімізації мережевих запитів та забезпечення роботи в офлайн-режимі.

Обраний технологічний стек забезпечує оптимальний баланс між продуктивністю, гнучкістю та зручністю розробки. Поєднання традиційного об'єктно-орієнтованого підходу для редактора та ECS-архітектури для системи виконання дозволяє максимально ефективно використовувати переваги кожної з парадигм: ООП - для забезпечення чіткої структури та розширюваності редактора, де продуктивність не є критичним фактором; ECS - для високопродуктивного виконання дерев поведінки, де важлива ефективна обробка великої кількості AI-агентів.

Використання сучасних технологій Unity (UI Toolkit, ScriptableObjects) забезпечує безшовну інтеграцію з екосистемою Unity та відповідає очікуванням цільової аудиторії. Водночас, вибір на користь стабільних та перевірених технологій (LTS-версія Unity, LeoECS) мінімізує ризики, пов'язані з можливими змінами API та несумісністю в майбутньому.

Розроблений на цій технологічній базі інструмент BehaviorTreeDesigner здатний задовольнити всі функціональні вимоги, визначені в результаті аналізу цільової аудиторії та ринку, забезпечуючи при цьому необхідний рівень продуктивності та зручності використання.

2.3 Аналіз BehaviourTreeDesigner як частини системи ігрового AI

При розробці інструментарію для створення ігрового штучного інтелекту необхідно розглядати його не як ізольоване рішення, а як інтегральний компонент комплексної екосистеми AI. Такий аналіз дозволяє врахувати всі взаємозв'язки, взаємозалежності та потенційні точки інтеграції ще на етапі проєктування.

У рамках роботи інструмент був використаний на практиці для розробки комплексної системи AI. Для ефективного функціонування AI-боти в грі повинні мати чітку модульну структуру, яка відповідає за різні аспекти їхньої поведінки. На основі аналізу вимог до системи AI та особливостей ігрового процесу, було розроблено наступну концептуальну модель (рис. 2.1).

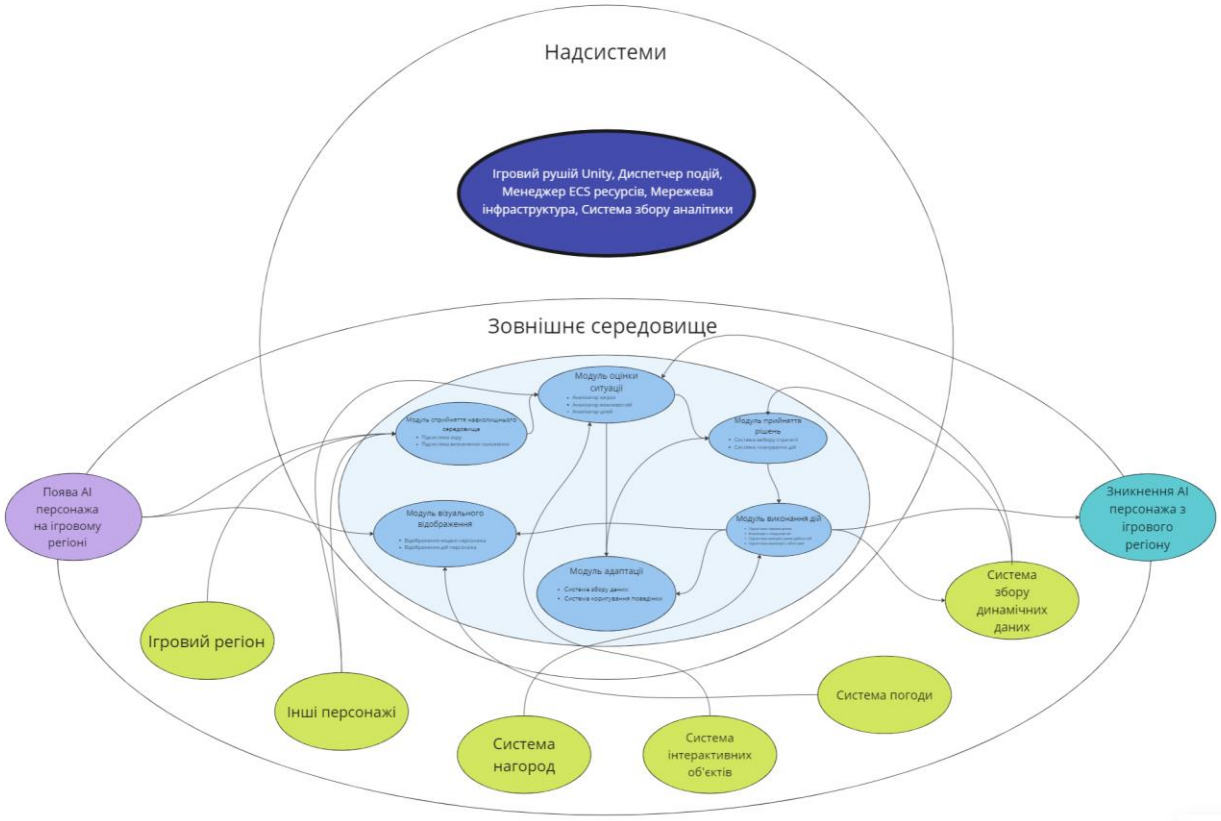


Рис. 2.1. Концептуальна модель ігрового AI

Основні модулі:

Модуль сприйняття навколишнього середовища: підсистема зору відстежує видимі об'єкти персонажів ландшафт, підсистема визначення положення відстежує власну позицію відносно регіону та інших об'єктів.

Модуль оцінки ситуації: аналізатор загроз визначає рівень небезпеки від інших персонажів, аналізатор можливостей оцінює доступні дії в поточній ситуації (атака втеча інтеракція з об'єктами), аналізатор цілей визначає пріоритетність цілей (збір скарбів атака супротивників досягнення точки виходу).

Модуль прийняття рішень: система вибору стратегії обирає загальний шаблон поведінки (агресивний обережний збалансований) базуючись на передналаштуваннях бота, система планування дій обирає послідовність дій для досягнення обраних цілей з урахуванням стратегії.

Модуль виконання дій: підсистема переміщення реалізує пересування персонажа по регіону, взаємодія з ландшафтом (хования в кущах за деревами), підсистема використання здібностей реалізує активні дії персонажа (кидання димової шашки атака грапуном), підсистема взаємодії з об'єктами реалізує підбирання скарбів використання точок виходу.

Модуль адаптації: система збору даних записує інформацію про хід гри дії гравця та інших ботів, система коригування поведінки допускає динамічні зміни параметрів інших модулів на основі виявлених даних.

Модуль візуального відображення: відображення моделі персонажа, відображення дій персонажа.

Ігровий світ (ігровий регіон): статичні об'єкти дерева кущі будинки можуть використовуватись як укриття та як перешкода, ландшафт річка гори впливають на площу пересування, динамічні об'єкти скарби переможені опоненти можуть бути обстежені та підібрані персонажами, точки виходу спеціальні зони через які персонажі можуть зайти у регіон або покинути його.

Інші персонажі: гравець людина яка контролює свого персонажа в грі, інші AI боти автономні персонажі які симулюють поведінку гравців і є суперниками для гравця та один одного.

Система інтерактивних об'єктів: взаємодія зі скарбами, взаємодія з точками інтересу.

Система нагород: видача нагород за обстеження скарбів та переможених опонентів.

Система погоди: зміна погоди на ігровому регіоні.

Система збору динамічних даних: збір та зберігання даних про дії гравця та інших персонажів, збір та зберігання даних про зібрані предмети на ігровому регіоні.

Ігровий рушій Unity: програмне забезпечення яке відповідає за рендеринг графіки фізичні розрахунки управління ігровими об'єктами та сценами.

Система управління ботами: відповідає за створення видалення та загальний верхньорівневий контроль ботів.

Диспетчер подій: відслідковує важливі події в грі (поява нових персонажів зміни в навколишньому середовищі) і повідомляє про них відповідні підсистеми.

Менеджер ECS ресурсів: відповідає за контроль та ефективний розподіл ресурсів між всіма сутностями у ECS світі.

Мережева інфраструктура: забезпечує можливість отримувати дані з backend серверу синхронізує стан гри.

Система збору аналітики: збирає дані про ігрові сесії дії гравців та ботів для подальшого аналізу та покращення гри.

Використання BehaviourTree дозволяє декомпозувати складну поведінку ботів на прості, модульні, багаторазові підзадачі. Дерево поведінки визначає логіку вибору дій в залежності від стану і сприйняття. Це робить систему AI гнучкою, розширюваною і зручною в налагодженні і модифікації порівняно зі складними ООП контролерами.

При розробці та оцінці AI бота важливо враховувати вплив та обмеження, що накладаються цими зовнішніми факторами. Наприклад, складність ландшафту та кількість інших персонажів в регіоні безпосередньо впливають на процес прийняття рішень ботом. А продуктивність ігрового рушія та мережевої інфраструктури встановлює технічні межі для складності алгоритмів AI.

Розглянемо основні системні взаємодії між розроблюваним інструментом та іншими компонентами екосистеми ігрового AI:

1. *Структурна взаємодія з підсистемами сприйняття:* Дерева поведінки, які створюються через BehaviorTreeDesigner, повинні ефективно споживати дані різної природи від підсистем зору, позиціонування та

аналізу ландшафту. Аналіз цієї взаємодії виявив необхідність реалізації універсального механізму отримання даних.

2. *Функціональна взаємодія з аналітичними модулями:* Екосистема ігрового AI включає модулі оцінки ситуації, аналізу загроз та можливостей, які часто використовують складні алгоритми машинного навчання або евристики. Аналіз показав, що BehaviorTreeDesigner має забезпечувати можливість інтеграції з цими модулями через стандартизовані інтерфейси, зберігаючи при цьому низький рівень зв'язності (loose coupling). Така архітектура дозволяє замінювати аналітичні модулі без зміни структури дерев поведінки.
3. *Інтеграційна взаємодія з надсистемами:* Інструмент має взаємодіяти не лише з підсистемами AI, але й з ключовими надсистемами: ігровим рушієм Unity, системою управління ботами, диспетчером подій та менеджером ECS ресурсів. Проведений аналіз цієї взаємодії визначив необхідність розробки легковагових адаптерів, що забезпечують інтеграцію без створення жорстких залежностей.

Результати аналізу свідчать, що для ефективної інтеграції в екосистему ігрового AI, BehaviorTreeDesigner повинен:

- Дотримуватися принципу розділення відповідальності (SRP) при проєктуванні інтерфейсів взаємодії
- Забезпечувати масштабованість через використання ECS-підходу для обробки великої кількості агентів
- Надавати механізми розширення для інтеграції зі специфічними компонентами конкретних проєктів
- Підтримувати можливість інструментування (instrumentation) для відстеження метрик продуктивності та діагностики

Такий системний аналіз місця інструменту в загальній екосистемі дозволяє спроєктувати рішення, яке не тільки вирішує конкретні проблеми створення дерев поведінки, але й органічно інтегрується в складний ландшафт

ігрового AI, забезпечуючи синергетичний ефект від взаємодії різних компонентів.

2.4 Математична модель системи

2.4.1 Формалізація математичних моделей та постановка задачі в математичному вигляді

Припустимо що P_{BT} це ефективність вирішення поставленої задачі за допомоги підходу з використанням BT, а P_{OOP} це ефективність вирішення тої самої задачі за допомоги підходу з використанням традиційного ООП.

Тоді відсоткове співвідношення ефективності двох підходів можна підрахувати наступним чином (Формула 2.1).

$$\Delta P = \left(1 - \frac{P_{BT}}{P_{OOP}}\right) * 100\% \quad (2.1)$$

Для підрахунку ефетивності підходу сформуємо модель на основі важливих критеріїв для виконання задачі (Формула 2.2).

$$P = \frac{1}{w1*R(n)+w2*E(s)+w3*D(s)} \quad (2.2)$$

Де:

$R(n)$ - час виконання (Runtime performance) AI логіки для n активних агентів.

$E(s)$ - час на розширення та підтримку (Extensibility and Maintainability) кодової бази з s поведінкових сценаріїв.

$D(s)$ - час тестування і налагодження (Testing and Debugging) кодової бази з s поведінкових сценаріїв.

n – кількість активних агентів

s – кількість поведінкових сценаріїв

$w1, w2, w3$ - вагові коефіцієнти, що відображають відносну важливість кожного фактору для конкретного проєкту.

Спочатку дамо загальну порівняльну оцінку для стандартного ООП підходу та підходу з використанням BT, використовуючи аналіз асимптотичної складності.

Runtime performance:

При ООП підході, через складні умовні оператори та постійне збільшення їх кількості, час виконання може зростати лінійно чи навіть квадратично: $R_{OOP}(n) = O(n)$ або $O(n^2)$.

BT підхід має правило обходу з ліва на право, та композитні вузли. Ці два фактори повністю замінюють умовні оператори й дозволяють опрацьовувати увесь час виконання через алгоритм обходу дерева, тож: $R_{BT}(n) = O(\log n)$.

Extensibility and Maintainability

При ООП підході, зі зростанням кількості сценаріїв складність розширення і підтримки зростає квадратично: $E_{OOP}(s) = O(s^2)$.

При BT підході, поведінка розбивається на прості умови і дії, які компонуються у різних варіаціях. Тому зі зростанням кількості сценаріїв s, загальна кількість базових вузлів $m \ll s$. Відповідно, складність розробки зростає лінійно: $E_{BT}(s) = O(m), m \ll s$.

Testing and Debugging

При ООП підході, зі зростанням кількості сценаріїв та взаємодії між ними - складність тестування і налагодження зростає квадратично: $D_{OOP}(s) = O(s^2)$.

BT підхід, завдяки модульності та систематизованому графічному відображенню у вигляді дерева, є зачно легшим у тестуванні та нлагодженні. Зі зростанням кількості сценаріїв іде лінійне зростання складості: $D_{BT}(s) = O(s)$.

Отже, узагальнена модель ефективності через асимптотичні складності матиме вигляд формули (2.3) для ООП підходу та формули (2.4) для BT підходу.

$$P_{OOP} = \frac{1}{O(n^2) + O(s^2) + O(s^2) = O(n^2) + 2 * O(s^2)} = \frac{1}{O(\max(n^2, s^2))} \quad (2.3)$$

$$P_{BT} = \frac{1}{O(logn) + O(s) + O(s) = O(\log n) + 2 * O(s)} = \frac{1}{O(\max(\log n, s))} \quad (2.4)$$

Якщо n^2 домінує над s^2 та $\log n$ домінує над s:

$$\Delta P = \left(1 - \frac{O(\log n)}{O(n^2)}\right) * 100\%$$

Якщо s^2 домінує над n^2 та s домінує над $\log n$

$$\Delta P = \left(1 - \frac{O(s)}{O(s^2)}\right) * 100\%$$

В обох випадках за великих значень n та s ΔP тяжіє до 100%, тож попередній аналіз демонструє що використання підіходу з ВТ є набагато ефективнішим.

Візьмемо конкретну кількість активних агентів та необхідних сценаріїв які наразі потрібні за технічним завданням:

$$n = 10$$

$$s = 7$$

$$P_{OOP} = \frac{1}{O(\max(10^2, 7^2))} = \frac{1}{O(\max(100, 49))} = \frac{1}{O(100)}$$

$$P_{BT} = \frac{1}{O(\max(\log 10, 7))} = \frac{1}{O(\max(2.32, 7))} = \frac{1}{O(7)}$$

$$\Delta P = \left(1 - \frac{7}{100}\right) * 100\% = 93\%$$

Отже, за умови 7 необхідних сценаріїв поведінки та 10 активних агентів на ігровому регіоні, підхід з використанням ВТ ефективніший за підхід з використанням ООП приблизно на 93%.

2.4.2 Використання методів моделювання розроблених моделей

Для перевірки адекватності та ефективності розроблених концептуальних та математичних моделей використовувались методи імітаційного моделювання.

Було реалізовано прототип AI-керованих ботів в ігровому рушії Unity з використанням розробленої модульної архітектури на основі BehaviourTree.

Далі було проведено серію експериментів, порівнюючи показники критеріїв з використанням агентів реалізованих за допомогою BehaviourTree, та агентів реалізованих тільки стандартними ООП засобами.

Для вимірювання реальних показників було використано Profiler рушія Unity (рис. 2.2). Таким чином дізнаємося реальний час виконання усіх операцій, пов'язаних з AI.

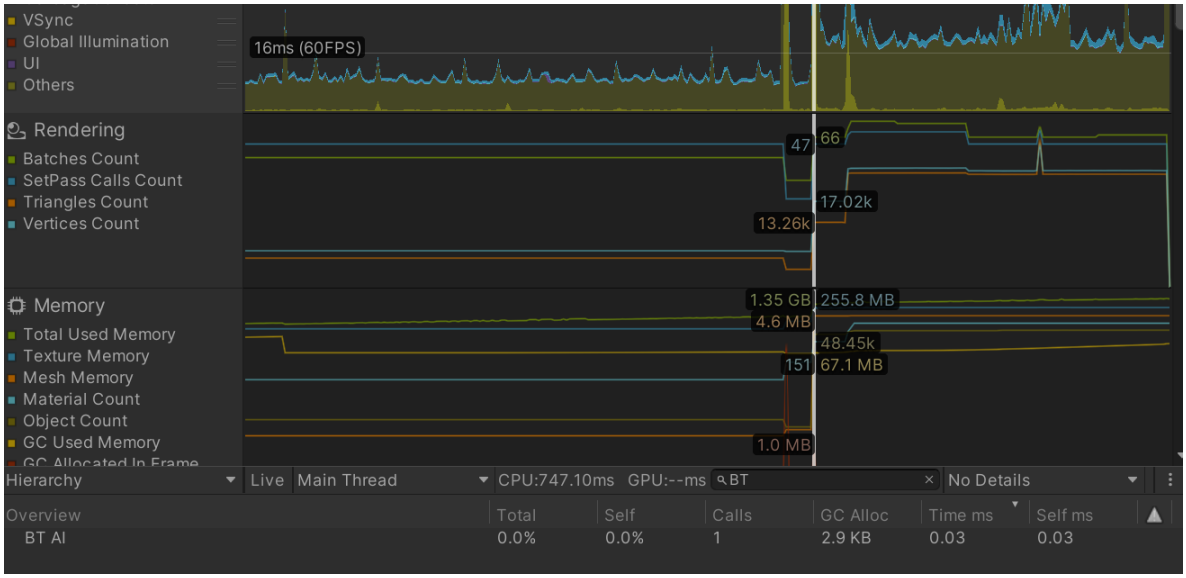


Рис. 2.2. Час виконання AI поведінки у Profiler

1) Час виконання (Runtime performance):
   - Для ООП підходу, середній час виконання AI логіки для 10 активних агентів: $R_{OOP}(10) = 0.2$ сек.
   - Для BT підходу, середній час виконання: $R_{BT}(10) = 0.05$ сек.
2) Розширюваність та підтримка (Extensibility and Maintainability):
   - Для ООП підходу, оцінка складності розширювання та підтримки кодової бази з 7 поведінкових сценаріїв: $E_{OOP}(7) = 0.4$ (середня складність).
   - Для BT підходу, оцінка складності: $E_{BT}(7) = 0.3$ (низька складність).
3) Тестування та налагодження (Testing and Debugging):

- Для ООП підходу, відносний час на тестування та налагодження: $D_{OOP}(7) = 0.6$ (60% від часу розробки).
- Для ВТ підходу, відносний час: $D_{BT}(7) = 0.2$ (20% від часу розробки).

Розподіл коефіцієнтів обумовлений фактичним станом проєкту:

– Проєкт розробляється для мобільних платформ, тож оптимізація часу виконання є критичним: $w1 = 0.8$

– Проєкт є довгостроковим, тож розширюваність і підтримка є важливим критерієм: $w2 = 0.5$

– Проєкт вже випущений для широкого загалу, і знаходиться на стадії активної розробки нового функціоналу, тому стабільність дуже важлива. Критерій тестування та налагодження є критичним. $w3 = 0.7$

Підставивши ці значення у нашу модель, отримаємо:

$$P_{OOP} = \frac{1}{0.8 * 1 + 0.5 * 0.4 + 0.7 * 0.6} = 0.70$$

$$P_{BT} = \frac{1}{0.8 * 0.25 + 0.5 * 0.3 + 0.7 * 0.2} = 2.04$$

Відсоткова перевага ВТ над ООР:

$$\Delta P = \left(1 - \frac{P_{BT}}{P_{OOP}}\right) * 100\% = \left(1 - \frac{0.7}{2.04}\right) * 100\% \approx 65.2\%$$

Тобто, за даних умов, ВТ підхід десонструє на 65.2% кращу ефективність за зваженою комбінацією критеріїв, ніж ООР підхід.

Нарешті, було проаналізовано масштабованість запропонованого рішення при збільшенні кількості агентів та розмірів ігрових сцен.

При $n = 100$ та $s = 30$ отримуємо:

$$R_{OOP}(100) = 0{,}5 \text{ сек.}$$

$$R_{BT}(100) = 0{,}05 \text{ сек.}$$

$$E_{OOP}(30) = 0{,}8$$

$$E_{BT}(30) = 0{,}4$$

$$D_{OOP}(30) = 0{,}7$$

$$D_{BT}(30) = 0{,}2$$

Підставивши ці значення у нашу модель, отримаємо:

$$P_{OOP} = \frac{1}{0.8 * 1 + 0.5 * 0.8 + 0.7 * 0.7} = 0.59$$

$$P_{BT} = \frac{1}{0.8 * 0.1 + 0.5 * 0.4 + 0.7 * 0.2} = 2.38$$

Відсоткова перевага BT над OOP:

$$\Delta P = \left(1 - \frac{P_{BT}}{P_{OOP}}\right) * 100\% = \left(1 - \frac{0.59}{2.38}\right) * 100\% \approx 75.15\%$$

Отже, для заданих значень параметрів та вагових коефіцієнтів, BT підхід показує на 75,15% кращу ефективність за комбінацією критеріїв, ніж OOP підхід. Модульна архітектура та оптимізовані алгоритми BehaviourTree дозволили підтримувати високу продуктивність навіть у складніших сценаріях.

Таким чином, проведене моделювання підтвердило коректність розроблених моделей та ефективність використання BehaviourTree для побудови реалістичної та адаптивної поведінки AI-керованих персонажів у відеогрі.

2.5 Аналіз цільової аудиторії та розробка структури необхідного функціоналу

Комплексний аналіз цільової аудиторії інструменту дозволяє більш якісно визначити обсяг необхідного функціоналу, зрозуміти ціннісну складову проєкту та побудувати більш досконалу бізнес-модель.

Цільова аудиторія інструментарію для створення ігровго AI охоплює дві ключові групи:

- Інді-розробники: шукають прості й доступні рішення.
- Малі та середні студії: потребують потужного інструментарію для команди.

В обох групах кінцевим користувачем є фахівець з розробки ігрового AI, тож створимо карту емпатії користувача (рис. 2.3). Вона має відповідати на

насупні запитання: «Що думає», «Що відчуває?», «Що бачить?», «Що чує?», «Що говорить?», «Що робить?», «Чого прагне?», «Чого остерігається?».

Карта емпатії дозволяє поставити себе на місце клієнта, що дає змогу знайти значно об'єктивніші напрямки необхідного функціоналу.

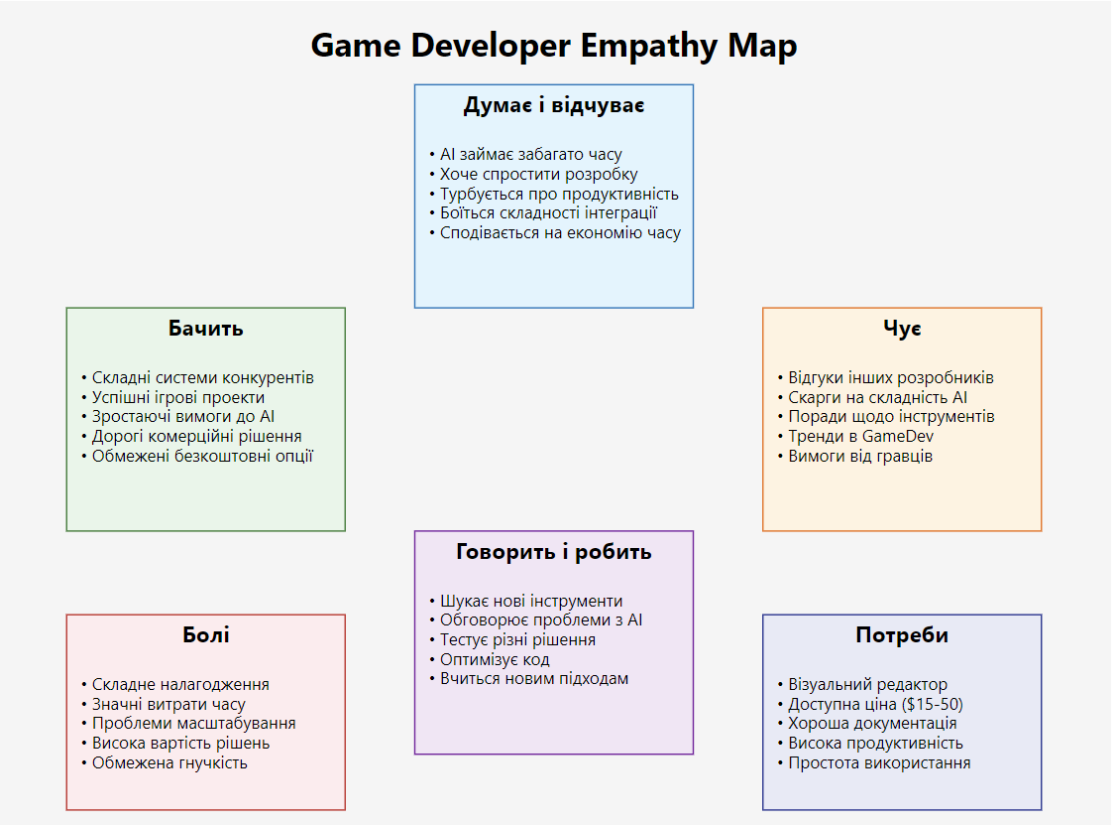


Рис. 2.3. Карта емпатії клієнта

Ще одним важливим етапом є виділення конкретний ролей користувачів та написання їх історій за підходом INVEST, що забезпечує їх незалежність, можливість обговорення, цінність, можливість оцінки, компактність та тестованість.

Основні ролі користувачів:

1) AI-розробник: Створює базову архітектуру AI-систем, розробляє нові компоненти та вузли, Оптимізує продуктивність системи
2) Гейм-дизайнер: Створює та налаштовує поведінку персонажів, тестує та балансує AI, інтегрує AI з ігровою механікою
3) QA-інженер: Тестує поведінку AI в різних сценаріях, знаходить та документує помилки, валідує продуктивність системи

4) Викладач: Викладає курси з розробки ігор та систем штучного інтелекту, використовує наочні інструменти для демонстрації принципів роботи ігрового AI, створює шаблони дерев у якості навчальних матеріалів

User story для Lead AI Developer:

- **Як** Lead AI Developer, **я хочу** бачити стан кожного вузла дерева поведінки в реальному часі, **щоб** швидко знаходити проблеми в логіці AI.
- **Як** Lead AI Developer, **я хочу** мати можливість зберігати часто використовувані піддерева як шаблони, **щоб** забезпечити повторне використання коду між проєктами.
- **Як** Lead AI Developer, **я хочу** мати можливість встановлювати точки зупину на вузлах дерева, **щоб** детально аналізувати процес прийняття рішень AI.

User story для Technical Game Designer:

- **Як** Technical Game Designer, **я хочу** мати візуальний редактор для створення та редагування дерев поведінки, **щоб** налаштовувати AI без написання коду.
- **Як** Technical Game Designer, **я хочу** мати бібліотеку готових патернів поведінки, **щоб** швидко створювати прототипи різних варіантів AI.
- **Як** Technical Game Designer, **я хочу** мати можливість копіювати та модифікувати існуючі дерева поведінки, **щоб** створювати варіації поведінки для різних типів персонажів.

User story для QA Automation Engineer:

- **Як** QA Engineer, **я хочу** мати можливість створювати автоматизовані тести для дерев поведінки, **щоб** перевіряти коректність роботи AI в різних сценаріях.
- **Як** QA Engineer, **я хочу** отримувати детальні логи виконання дерева поведінки, **щоб** мати можливість відтворювати та аналізувати проблеми.

– **Як** QA Engineer, **я хочу** мати інструменти для симуляції різних умов середовища, **щоб** тестувати реакцію AI на різні ситуації.

User story для Викладач:

– **Як** викладач, **я хочу** мати можливість покроково демонструвати роботу дерева поведінки, **щоб** пояснювати студентам принципи роботи AI.
– **Як** викладач, **я хочу** створювати навчальні приклади з різним рівнем складності, **щоб** поступово вводити студентів у тему ігрового AI.
– **Як** викладач, **я хочу** мати можливість експортувати та імпортувати дерева поведінки, **щоб** ділитися прикладами зі студентами.

На основі проведеного аналізу виділимо найважливіші аспекти.

Основні біль-точки користувачів: Витрати значного часу на розробку AI (30-40% часу), Складність налагодження та тестування, Проблеми з масштабуванням рішень, Обмежений бюджет (особливо у інді-розробників).

Ключові потреби: Візуальний інструментарій для розробки AI, Гнучке та масштабоване рішення, Доступна цінова політика, Якісна документація та навчальні матеріали.

Рекомендації для продукту: Зробити акцент на візуальному редакторі та зручності налагодження, Забезпечити гнучку цінову політику з різними планами, Розробити якісну документацію та навчальні матеріали, Приділити особливу увагу продуктивності рішення.

Маркетингові висновки: Фокусуватись на економії часу розробників, Підкреслювати простоту використання та інтеграції, Демонструвати переваги в порівнянні з існуючими рішеннями, Створити різні цінові плани для різних сегментів користувачі.

Ефективним методом аналізу для формування функціоналу продукту є інтерв’ю з клієнтом та створення карток персон. Для проведення дослідження було створено наступну анкету:

1) Загальна інформація: Розмір команди розробки, Досвід роботи з Unity, Типи проєктів, над якими працюєте, Використання AI в проєктах

2) Проблеми та виклики: Як зараз реалізуєте поведінку персонажів?, Скільки часу витрачаєте на розробку AI?, З якими труднощами стикаєтеся при розробці AI?, Що найбільше сповільнює процес розробки AI?
3) Існуючі рішення: Які інструменти зараз використовуєте? Що подобається/не подобається в них? Скільки коштує поточне рішення?
4) Потреби та очікування: Які функції хотіли б бачити в інструменті для розробки AI? Яка цінова політика була б прийнятною? Що могло б спонукати перейти на новий інструмент?

В результаті було проведено 3 інтерв'ю та створено 2 картки персон які відображають два основних архетипи користувачів.

Інтерв'ю #1, Респондент: Олексій К.**,** Посада: Lead Developer в інді-студії

1) Загальна інформація: Команда: 5 розробників, Досвід з Unity: 4 роки, Проєкти: Мобільні RPG ігри, AI: Активно використовують для NPC
2) Проблеми: Використовують власну систему на основі FSM, 30-40% часу розробки йде на AI, Складно налагоджувати складну поведінку, Важко масштабувати рішення
3) Існуючі рішення: Власна розробка + базовий FSM, Не вистачає візуалізації, Витрати на підтримку власного рішення
4) Очікування: Візуальний редактор обов'язковий, Готові платити до $30/місяць на розробника , Важлива можливість відлагодження

Інтерв'ю #2

Респондент: Марія В.

Посада: Indie Game Developer

1) Загальна інформація: Соло-розробник, Досвід з Unity: 2 роки, Проєкти: 2D платформери, AI: Базова поведінка ворогів
2) Проблеми: Використовує перенавантаженіскрипти, Багато часу на тестування, Складно створювати складні патерни поведінки, Обмежений бюджет
3) Існуючі рішення: Безкоштовні ассети, Не вистачає гнучкості, Обмежена функціональність

4) Очікування: Простий інтерфейс, Бюджет до $15/місяць, Важлива наявність навчальних матеріалів

Інтерв'ю #3, Респондент: Ігор П., Посада: Technical Director у середній студії

1) Загальна інформація: Команда: 15 розробників, Досвід з Unity: 7 років, Проєкти: PC/Console Action games, AI: Складні системи поведінки
2) Проблеми: Використовують комбінацію рішень, Складна інтеграція між системами, Проблеми з продуктивністю, Складно підтримувати код
3) Існуючі рішення: Комерційні рішення + власні розробки, Витрачають ~$100/місяць на ліцензії, Не вистачає специфічного функціоналу
4) Очікування: Висока продуктивність, Готові платити до $50/місяць, Важлива можливість кастомізації

Ці інтерв'ю підтверджують попередні гіпотези про потреби ринку та сприяють кращому зрозумінню очікувань користувачів у різних сегментах. На їх базі було створено карти персон (рис. 2.4 та рис 2.5) для кращого розуміння цільової аудиторії серед усієї команди проєкту.

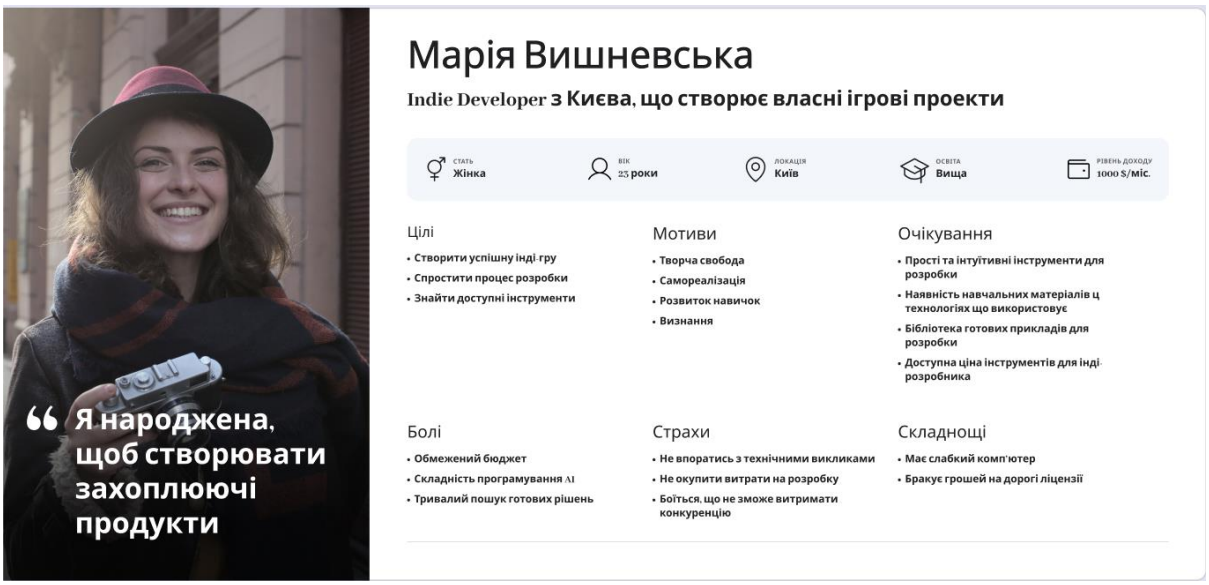


Рис. 2.4. Карта персони Indie Developer

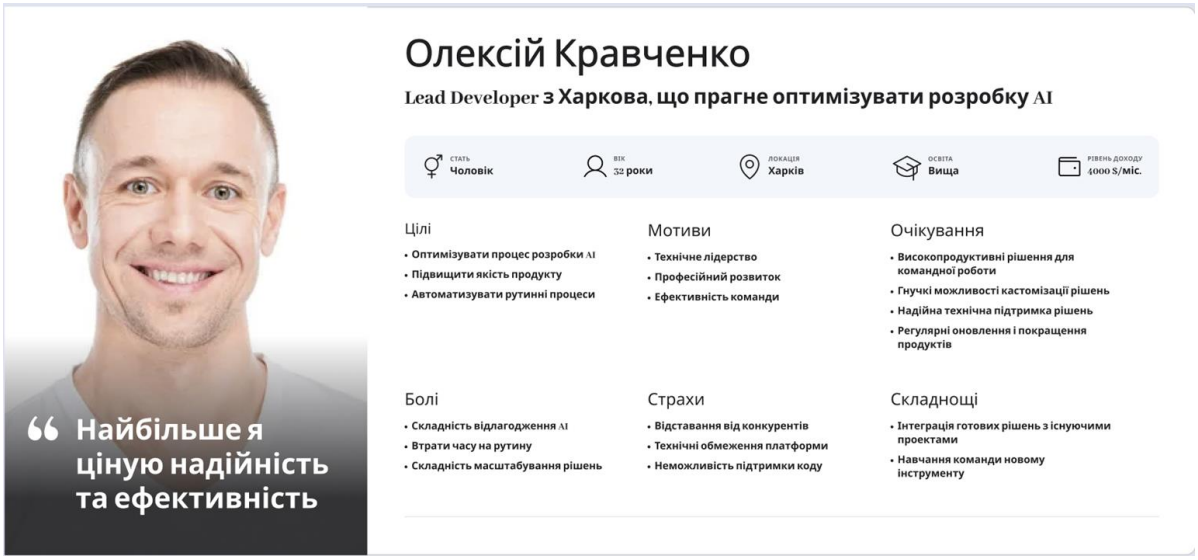


Рис. 2.5. Карта персони Lead Developer

Список основних необхідних функцій продукту формується з врахуванням результатів усього попереднього аналізу цільової аудиторії. Повна структурна ієрархічна модель функціоналу ПЗ міститься у Додатку В.

1) Система виконання: ефективний рушій виконання дерев, підтримка паралельного виконання, система подій та блекбордів, управління пам'яттю та оптимізація, система пріоритетів виконання, кешування результатів.
2) Інтеграція з Unity: компонент BehaviourTreeAgent, інспектор налаштувань у Unity, сумісність з різними версіями Unity, інтеграція з NavMesh системою, підтримка серіалізації, Asset Bundle підтримка.
3) Користувацький інтерфейс: візуальний конструктор дерев з drag-and-drop інтерфейсом, меню стандартних вузлів поведінки, можливість створення користувацьких вузлів, валідація структури дерева, гарячі клавіші для швидкого доступу, підтримка копіювання/вставки піддерев, система шаблонів, пошук по дереву, темна/світла теми оформлення.
4) Інструменти налагодження: візуалізація стану виконання в реальному часі, покрокове виконання дерева та точки зупину на вузлах, інспектор змінних, профілювання продуктивності.
5) Документація та навчання: вбудована довідка, інтерактивні туторіали, API

документація, форум спільноти, база знань.

6) Аналітика та звітність: статистика використання, метрики продуктивності та оптимізації системи, звіти про помилки, користувацькі звіти, візуалізація метрик.
7) Безпека, захист та відмовостійкість: контроль доступу, валідація вхідних даних, резервне копіювання, відновлення даних.

2.6 SWOT-аналіз можливостей та загроз проєкту

У процесі стратегічного планування розробки візуального редактора дерев поведінки для ігрових систем ШІ критично важливо провести всебічний аналіз проєкту з точки зору його життєздатності та конкурентоспроможності. SWOT-аналіз є ефективним інструментом, який дозволяє систематично оцінити внутрішні фактори (сильні та слабкі сторони) та зовнішні чинники (можливості та загрози), що впливають на реалізацію проєкту BehaviorTreeDesigner. Візуалізація проведеного аналізу зображена на рис. 2.6.

Сильні сторони (Strengths): Інноваційна гібридна архітектура - поєднання Behavior Trees (BT) та Entity-Component-System (ECS) забезпечує унікальну технічну перевагу з точки зору продуктивності та гнучкості, висока продуктивність - використання ECS архітектури забезпечує оптимальну роботу з пам'яттю низький рівень алокацій та ефективну паралелізацію що особливо важливо для мобільних платформ, модульність та розширюваність - система дозволяє створювати складні патерни поведінки з простих атомарних компонентів які можуть бути легко модифіковані та повторно використані, зручний візуальний інтерфейс - інтуїтивний редактор спрощує процес створення та налагодження AI без необхідності написання складного коду, комплексна система налагодження - інструменти для візуалізації стану виконання в реальному часі покрокового виконання та встановлення точок зупину, математично обґрунтована ефективність - проведені дослідження показують суттєве покращення (до 65-93%) порівняно з традиційними ООП підходами.

Слабкі сторони (Weaknesses): Залежність від Unity - продукт тісно інтегрований з Unity що обмежує потенційний ринок та створює ризики при оновленні рушія, складність технічної складової - незважаючи на інтуїтивність парадигма BT та ECS може вимагати певного часу для освоєння новими користувачами, обмежена підтримка сторонніх систем - можуть виникати труднощі з інтеграцією з іншими популярними інструментами для Unity, ресурсомісткість розробки та підтримки - необхідність постійної підтримки сумісності з новими версіями Unity та розширення функціоналу може вимагати значних ресурсів, складність монетизації - як зазначено в аналізі ризиків "Складність монетизації інструменту" є одним з найбільш важливих ризиків.

Можливості (Opportunities): Зростання ринку інструментів розробки AI - підвищений інтерес до ігрового AI створює сприятливі умови для виходу нових спеціалізованих інструментів, освітній сектор - можливість співпраці з навчальними закладами для інтеграції продукту в освітні програми з розробки ігор, розширення за межі ігрової індустрії - адаптація технології для інших галузей таких як архітектурна візуалізація симуляції тренувальні програми, спільнота розробників - створення активної спільноти може забезпечити постійний приток ідей для вдосконалення та поширення продукту, інтеграція з машинним навчанням - можливість розширення функціоналу шляхом додавання компонентів машинного навчання для створення адаптивного AI, партнерство з ігровими студіями - стратегічні партнерства з розробниками ігор можуть забезпечити цінний зворотний зв'язок та кейси використання.

Загрози (Threats): Конкуренція від Unity - як зазначено в аналізі ризиків поява конкурентного рішення від самої Unity є однією з найсерйозніших загроз, швидка зміна технологій - ігрова індустрія швидко розвивається що може призвести до застарівання технологічних рішень, низький попит серед розробників - складність переконати розробників перейти з існуючих інструментів на новий, конкуренція з боку інших рішень - наявність альтернативних інструментів для розробки AI таких як Playmaker Bolt

NodeCanvas, проблеми сумісності з новими версіями Unity - як зазначено в аналізі ризиків це може вимагати значного рефакторингу коду, зміни в політиці Unity щодо сторонніх інструментів - потенційні зміни в API або умовах маркетплейсу можуть негативно вплинути на розповсюдження продукту.

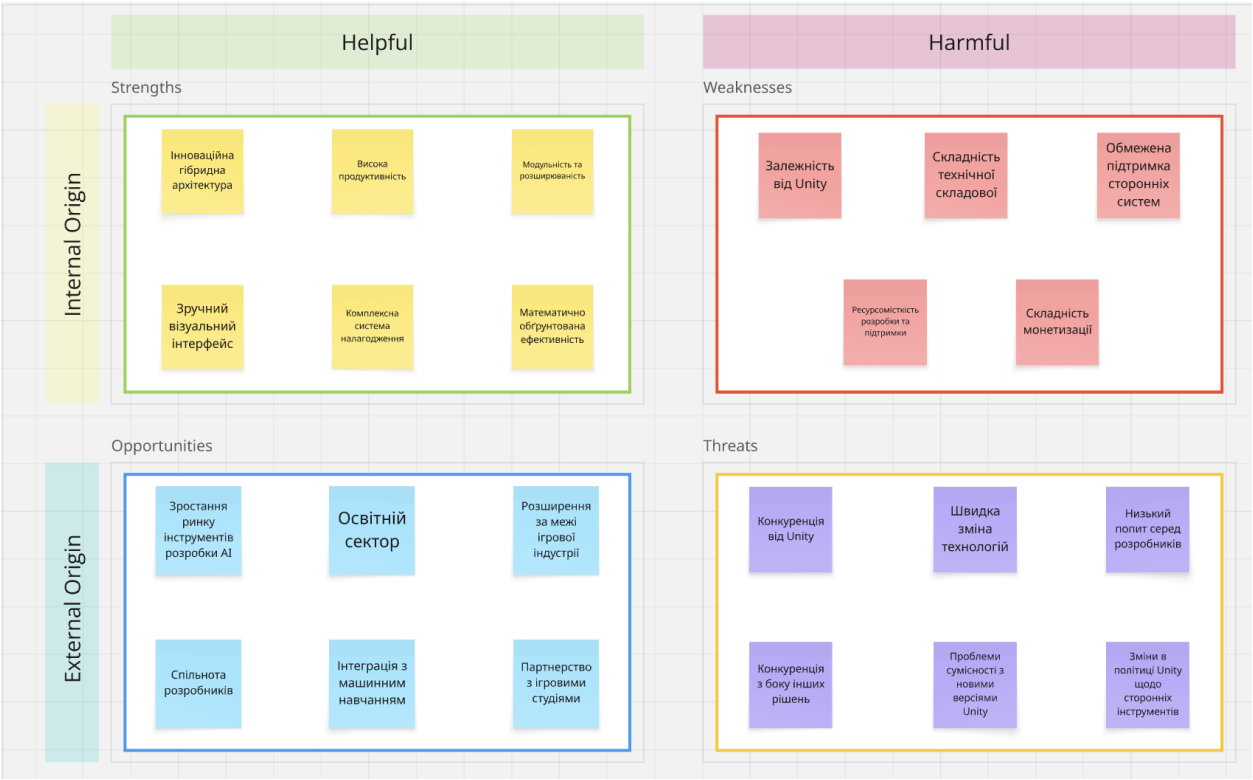


Рис. 2.6. SWOT-аналіз

Аналіз та формування рекомендацій на основі SWOT-аналізу

Використання сильних сторін для реалізації можливостей (S-O): використати високу продуктивність та модульність для створення спеціалізованих рішень для різних сегментів ринку (мобільні ігри VR/AR симуляції), розробити навчальні матеріали та програми для освітнього сектору використовуючи зручний візуальний інтерфейс як конкурентну перевагу.

Усунення слабких сторін для реалізації можливостей (W-O): розробити адаптери для інтеграції з популярними сторонніми системами розширюючи екосистему продукту, створити спрощені шаблони та інтерактивні туторіали для зниження порогу входу для нових користувачів.

Використання сильних сторін для протидії загрозам (S-T): акцентувати увагу на унікальних перевагах гібридної архітектури для диференціації від

потенційного рішення Unity, використати систему налагодження та високу продуктивність як ключові переваги в маркетингових матеріалах для залучення розробників.

Мінімізація слабких сторін та загроз (W-T): розробити стратегію диверсифікації для зменшення залежності від Unity (розширення на інші рушії або створення самостійного рішення), впровадити гнучку цінову політику та пробний період для зниження бар'єру входу та підвищення попиту.

SWOT-аналіз демонструє, що BehaviorTreeDesigner має значний потенціал на ринку інструментів для розробки ігрового AI, але успіх залежатиме від здатності ефективно монетизувати рішення та адаптуватися до змін у технологічному ландшафті. Особливу увагу слід приділити створенню активної спільноти та диференціації від потенційних конкурентних рішень від самої Unity.

2.7 Розробка бізнес-моделі

Успіх будь-якого продукту на ринку значною мірою залежить від правильно розробленої бізнес-моделі. Для візуального редактора BehaviorTreeDesigner важливо не лише створити функціональний інструмент, а й забезпечити його комерційну життєздатність. У цьому розділі представлено аналіз ключових компонентів бізнес-моделі проєкту з розробки візуального редактора для створення AI в середовищі Unity.

2.7.1 Ціннісна пропозиція продукту

Розробка ціннісної пропозиції базується на результатах детального аналізу потреб цільової аудиторії та існуючих рішень на ринку. Проведений попередньо аналіз серед цільової аудиторії показав, що фраза "Створюйте професійний AI без написання коду" демонструє найвищу конверсію серед потенційних користувачів.

Зібрані дані, дозволили виділити та ранжувати ключові переваги для користувачів (табл. 2.1)

Таблиця 2.1

Ціннісні пропозиції

№	Цінність	Рейтинг цінностей
1	Простота використання. Візуальний інтерфейс для розробки AI забезпечує інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам швидко освоїти інструмент навіть без поглиблених технічних знань. Візуальний редактор дозволяє створювати поведінки без необхідності писати складний код.	2
2	Економія часу. Скорочення витрат часу на розробку AI до 50-70%. Завдяки автоматизації процесів та можливості швидкого створення і тестування дерев поведінки, розробники можуть фокусуватися на креативній частині проєкту.	1
3	Гнучкість і масштабованість. Можливість кастомізації для різних типів розробників. Стартап пропонує рішення, яке підходить як для невеликих інді-проєктів, так і для великих студій із потребою в масштабованості. BehaviorTreeDesigner дозволяє користувачам легко адаптувати продукт під свої специфічні потреби.	5
4	Доступна ціна. Диференційовані тарифні плани. BehaviorTreeDesigner надає розробникам ефективний інструмент за доступною ціною. Гнучка модель ціноутворення враховує різні потреби і фінансові можливості клієнтів.	4
5	Інноваційність. Вбудовані інструменти налагодження. Стартап пропонує унікальний продукт, який поєднує у собі передові технології створення AI для ігор та інтуїтивний інтерфейс. Візуальний редактор дозволяє створювати складні системи AI без програмування.	3

Підсумовуючи результати аналізу ціннісних пропозицій, можна зазначити, що економія часу та простота використання є ключовими

конкурентними перевагами BehaviorTreeDesigner. Саме на цих аспектах варто зосередити основну увагу при позиціонуванні продукту на ринку та комунікації з потенційними користувачами. Розроблена система ціннісних пропозицій повністю відповідає виявленим потребам цільової аудиторії та створює міцну основу для формування унікальної ринкової позиції продукту.

2.7.2 Канали просування та взаємодії з клієнтами

Аналіз каналів просування конкурентів та опитування потенційних користувачів щодо джерел інформації про нові інструменти розробки допомогли визначити оптимальні шляхи виходу на ринок:

Unity Asset Store (50% всіх придбань) - основний канал поширення продуктів для розробки на Unity платформа має вбудовану систему оплати рейтингів та відгуків що підвищує довіру користувачів, професійні конференції (20%) - презентації та майстер-класи на GameDev конференціях допомагають демонструвати переваги продукту професійній аудиторії та налагоджувати зв'язки з потенційними великими клієнтами, спільноти розробників (15%) - активна участь у форумах Discord-каналах та інших спільнотах розробників дозволяє отримувати зворотний зв'язок та популяризувати продукт серед цільової аудиторії, навчальні заклади (10%) - співпраця з університетами та школами програмування для інтеграції продукту в навчальні програми, YouTube канали з навчальними матеріалами (5%) - створення освітнього контенту який демонструє можливості продукту та допомагає користувачам освоїти інструмент.

Кожен канал повинен мати окрему стратегію комунікації, що враховує особливості відповідного сегменту аудиторії. Наприклад, для Unity Asset Store акцент має бути на візуальних демонстраціях та порівняннях з конкурентами, в той час як для освітнього сектору фокус має зміщуватись на навчальні матеріали та можливості інтеграції в освітній процес.

Найбільш привабливими сегментами з точки зору потенційного доходу та масштабування є інді-розробники та середні студії, які складають 75% цільової

аудиторії. Саме на ці сегменти має бути спрямована основна маркетингова стратегія.

2.7.3 Потоки доходів

В основі структури потоків доходів лежить аналіз цінової політики конкурентів та опитування щодо бюджетів на інструменти розробки серед геймдев фахівців.

Базова версія $30/місяць: доступ до основного функціоналу редактора базова бібліотека компонентів підтримка через форум спільноти базова документація, Pro версія $50/місяць: повний доступ до всіх функцій редактора розширена бібліотека компонентів підтримка через чат та електронну пошту повна документація та відеоуроки можливість експорту та імпорту дерев поведінки, Enterprise ліцензія $200/місяць: корпоративна ліцензія для команд пріоритетна технічна підтримка можливість кастомізації та інтеграції з власними системами доступ до API для розширення функціоналу командні інструменти для спільної роботи, консультації та технічна підтримка $100/година: індивідуальні консультації з експертами в галузі AI допомога в інтеграції продукту в існуючі проєкти навчання команди роботі з інструментом.

Тестування різних цінових моделей показало, що підписка є оптимальним варіантом для основних сегментів користувачів, оскільки забезпечує стабільний потік доходів та дозволяє користувачам гнучко керувати витратами. Для великих студій передбачено можливість річних контрактів з відповідними знижками.

2.7.4 Ключові метрики

Для ефективного управління бізнесом та оцінки успішності проєкту було визначено наступні ключові метрики:

1) Кількість активних користувачів: Щоденні активні користувачі (DAU), щомісячні активні користувачі (MAU), відношення DAU/MAU для оцінки залученості
2) Середній час використання: Тривалість сесій роботи з інструментом, частота використання різних функцій, найбільш популярні сценарії використання
3) Кількість створених проєктів: Середня кількість проєктів на користувача, складність створюваних дерев поведінки, відсоток завершених проєктів

Система збору та аналізу цих метрик дозволить оперативно реагувати на зміни в поведінці користувачів та приймати обґрунтовані рішення щодо розвитку продукту.

2.7.5 Унікальна перевага

Дослідження унікальних конкурентних переваг продукту включало патентний пошук, аналіз технологічних бар'єрів, оцінку складності відтворення рішення та вивчення унікальних компетенцій команди. За результатами дослідження було визначено наступні унікальні переваги:

Патентоспроможні алгоритми оптимізації - розроблені алгоритми обробки дерев поведінки забезпечують вищу продуктивність у порівнянні з конкурентними рішеннями особливо при роботі з великими проєктами, унікальна експертиза в ігровому AI - команда проєкту має спеціалізований досвід у розробці ігрових систем штучного інтелекту що дозволяє створювати більш ефективні та інтуїтивні рішення, глибока інтеграція з Unity - продукт розроблено з урахуванням специфіки роботи з Unity що забезпечує безшовну інтеграцію та оптимальну продуктивність, оптимізована архітектура для високопродуктивних систем - використання принципів Entity-Component-System (ECS) дозволяє створювати більш ефективні та масштабовані AI-системи, складна для відтворення архітектура - поєднання візуального редактора з потужною системою налагодження в реальному часі створює високий бар'єр для входу конкурентів.

Ці переваги формують міцну основу для конкурентоспроможності продукту на ринку і створюють значні бар'єри для входу потенційних конкурентів.

2.7.6 Побудова Lean Canvas

На основі проведеного аналізу була розроблена бізнес-модель проєкту за методологією Lean Canvas. Цей формат дозволяє наочно представити всі ключові аспекти бізнес-моделі, фокусуючись на цінності для користувачів та шляхах вирішення їхніх проблем. Зручним представленням LeanCanvas моделі є дошка з сегментами для кожного аспекту (рис. 2.7)

1) Проблема: висока складність та часозатратність розробки ігрового AI, складність налаштування та тестування AI-систем, відсутність інтуїтивних інструментів для непрограмістів.
2) Рішення: візуальний редактор поведінкових дерев, система реального часу для налагодження AI, бібліотека готових компонентів та шаблонів.
3) Унікальна ціннісна пропозиція: "Створюйте професійний AI без написання коду", візуальний контроль над поведінкою персонажів, інтуїтивно зрозумілий інтерфейс, швидке прототипування AI-систем.
4) Унікальна перевага: глибока інтеграція з Unity, оптимізована архітектура для високої продуктивності, експертиза в ігровому AI.
5) Сегменти споживачів: інді-розробники та малі студії, середні студії розробки ігор, технічні геймдизайнери, викладачі/студенти ігрової розробки.
6) Ключові метрики: кількість активних користувачів, конверсія trial → paid, retention rate, середній час використання, кількість створених проєктів.
7) Канали: Unity Asset Store, професійні конференції розробників ігор, спільноти розробників (Discord, Reddit), навчальні заклади та курси

розробки ігор, YouTube-канал з туторіалами.

8) Структура витрат: розробка та підтримка продукту (40%), маркетинг та просування (25%), технічна підтримка користувачів (20%), серверна інфраструктура та комісії (15%).
9) Потоки доходів: базова версія: $30/місяць, Pro версія: $50/місяць, Enterprise ліцензія: $200/місяць, консультації та підтримка: $100/година.

Lean Canvas візуального редактора BehaviorTreeDesigner включає:

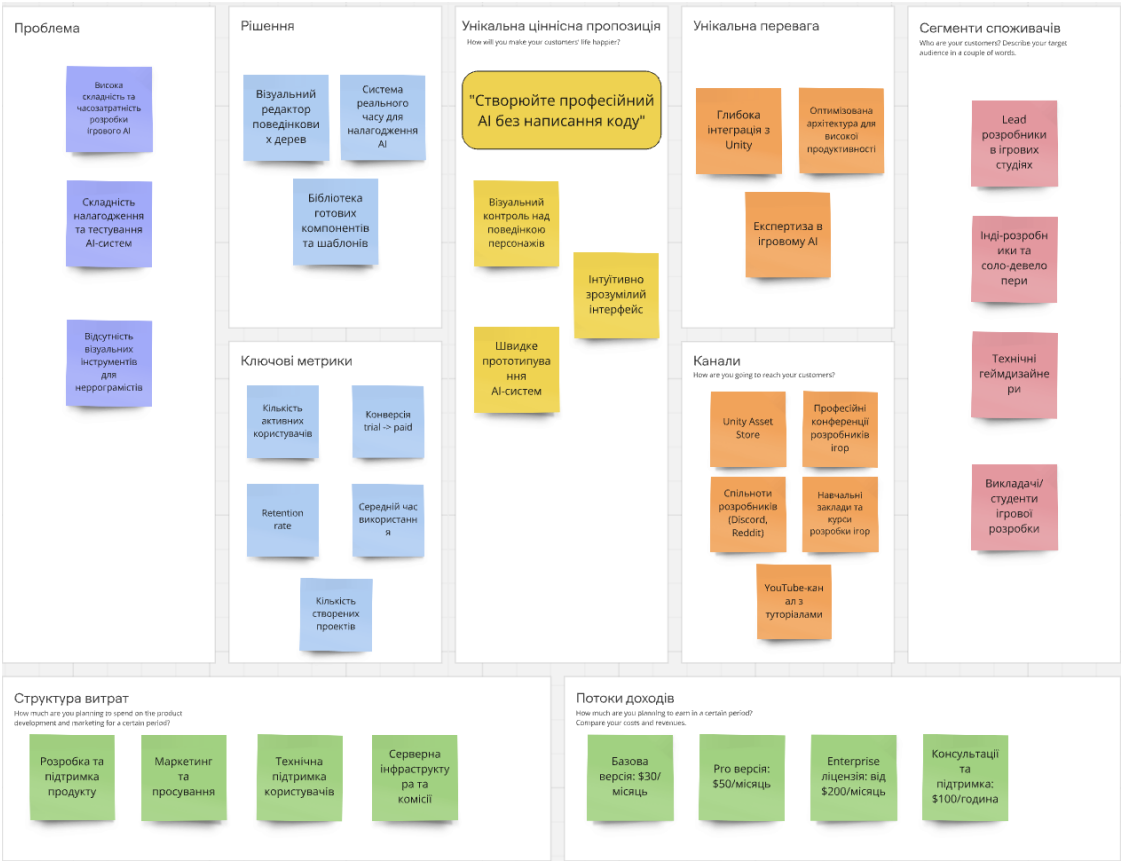


Рис 2.7. Lean Canvas

Розроблена бізнес-модель проєкту BehaviorTreeDesigner базується на глибокому аналізі ринку, потреб цільової аудиторії та конкурентного середовища. Ключовими факторами успіху даної бізнес-моделі є: Фокус на специфічних потребах цільової аудиторії - продукт розроблений з урахуванням реальних проблем та вимог розробників ігор різного масштабу, збалансована цінова політика - диференційовані тарифні плани дозволяють охопити різні

сегменти ринку від інді-розробників до великих студій, багатоканальна стратегія просування - використання різноманітних каналів взаємодії з аудиторією забезпечує широке охоплення та стабільний притік нових користувачів, унікальна технологічна перевага - патентоспроможні алгоритми та оптимізована архітектура створюють значні бар'єри для конкурентів, чіткі метрики оцінки ефективності - система ключових показників дозволяє оперативно оцінювати успішність проєкту та приймати обґрунтовані рішення.

Розроблена бізнес-модель не лише задає напрямок комерційного розвитку проєкту, але й визначає методологію взаємодії з цільовою аудиторією. Ефективність моделі має регулярно аналізуватися через відстеження ключових метрик успіху, таких як зростання кількості активних користувачів, конверсія у платні підписки та рівень утримання клієнтів. Адаптивність бізнес-моделі до динамічних умов ринку забезпечуватиметься через постійний моніторинг зворотного зв'язку від користувачів та аналіз трендів в індустрії ігрової розробки.

Підсумовуючи, можна стверджувати, що розроблена бізнес-модель забезпечує міцну основу для комерційного успіху проєкту та його стійкого розвитку на ринку інструментів для розробки ігор. Вона враховує усі аспекти розвитку проєкту та забезпечує баланс між утриманням високої якості продукту та забезпеченням комерційної привабливості для всіх сегментів цільової аудиторії.

Подальші кроки передбачають практичне впровадження бізнес-моделі з постійним моніторингом ключових метрик та гнучким реагуванням на зміни ринкової ситуації.

РОЗДІЛ 3. УПРАВЛІННЯ ПРОЄКТОМ РОЗРОБКИ ІНСТРУМЕНТАРІЯ СТВОРЕННЯ ІГРОВГО AI

3.1 Розробка організаційної структури управління проєктом. Формування команди проєкту

Для ефективної реалізації проєкту з розробки візуального редактора дерев поведінки для ігрових систем ШІ в середовищі Unity була розроблена збалансована організаційна структура управління, яка враховує специфіку проєкту, його масштаб, технічну складність та особливості ринку ігрової розробки.

3.1.1 Модель організаційної структури проєкту

Для даного проєкту обрана матрична організаційна структура з елементами проєктної. Такий підхід обумовлений потребою в гнучкому управлінні ресурсами, чіткій координації між функціональними напрямками та забезпеченні ефективної комунікації всередині команди.

Основні переваги обраної організаційної структури для нашого проєкту:

- Гнучкість у розподілі людських ресурсів між функціональними напрямками
- Оптимальне використання експертизи членів команди
- Швидке прийняття рішень завдяки прямим комунікаційним каналам
- Можливість оперативного реагування на зміни вимог та ринкові умови
- Чітка відповідальність за досягнення результатів проєкту

Ієрархічна схема організаційної структури проєкту (OBS) представлена на рис. 3.1.

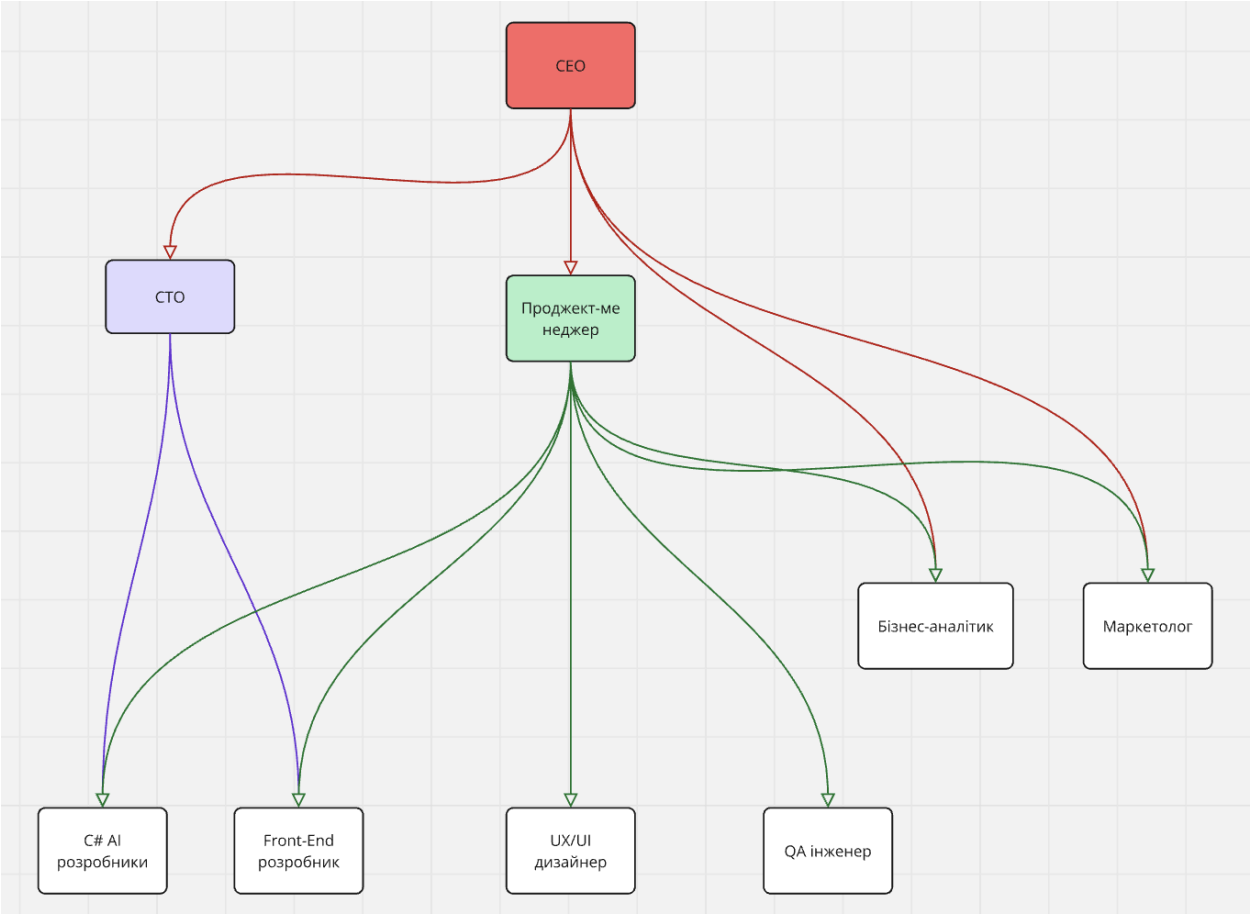


Рис. 3.1. Ієрархічна схема організаційної структури проєкту

Враховуючи специфіку проєкту з розробки інструментарію для ігрового ШІ, була сформована збалансована команда, що забезпечує покриття всіх необхідних компетенцій. Команда проєкту складається з наступних ключових ролей:

CEO *Вимоги:* освіта вища технічна або в галузі управління проєктами, досвід управління IT-проєктами не менше 3 років, знання методологій Agile та Scrum, сертифікація PMP або аналогічна, розуміння специфіки розробки програмного забезпечення для ігрової індустрії, знання процесів управління ризиками та якістю. *Функції:* загальне керівництво проєктом визначення стратегії та тактики, планування та контроль ресурсів проєкту (людських фінансових часових), формування та управління командою проєкту, координація взаємодії між різними функціональними напрямками, комунікація з зацікавленими сторонами проєкту, контроль виконання проєкту відповідно

до затвердженого бюджету та термінів, управління ризиками проєкту, прийняття стратегічних рішень щодо напрямку розвитку продукту.

Технічний директор *Вимоги:* освіта вища технічна в галузі комп'ютерних наук, глибоке знання Unity та C# не менше 5 років практичного досвіду, досвід розробки AI-систем для ігор не менше 3 років, розуміння архітектури програмного забезпечення та патернів проектування, досвід роботи з Entity-Component-System та Behavior Trees, знання принципів оптимізації продуктивності ігрових систем. *Функції:* розробка технічної архітектури продукту, технічне лідерство та прийняття архітектурних рішень, координація роботи розробників, забезпечення якості коду та відповідності технічним стандартам, вирішення складних технічних питань та проблем, визначення технологічного стеку та інструментів розробки, контроль за дотриманням технічних обмежень Unity та цільових платформ, участь у формуванні дорожньої карти розвитку продукту.

Проджект-менеджер *Вимоги:* освіта вища в галузі управління проєктами або технічна з додатковою підготовкою, досвід управління IT-проєктами не менше 2 років, глибоке знання Agile-методологій (Scrum Kanban), впевнене володіння інструментами управління проєктами (Jira), розуміння процесів розробки програмного забезпечення, досвід управління командами розробників. *Функції:* щоденне оперативне управління проєктом, планування та контроль виконання ітерацій (спринтів), організація і проведення нарад планування спринту ретроспектив, відстеження та усунення перешкод для команди, підтримка ведення проєктної документації (беклог звіти статус проєкту), оптимізація процесів розробки, комунікація між командою розробки та іншими стейкхолдерами, моніторинг і контроль виконання задач та дотримання термінів, звітування керівнику проєкту про статус і ризики.

C# AI розробники *Вимоги:* освіта вища технічна в галузі комп'ютерних наук або ігрової розробки, знання C# та Unity не менше 3 років, досвід розробки ігрових систем штучного інтелекту 2+ років, розуміння принципів Behavior Trees State Machines та інших патернів AI, навички оптимізації коду

для забезпечення високої продуктивності, досвід роботи з ECS архітектурою (не обов'язково але бажано). *Функції:* розробка core-функціоналу Behavior Tree системи, імплементація базових та спеціалізованих вузлів дерева поведінки, інтеграція системи з Unity Editor, оптимізація продуктивності AI-системи, створення рішень для типових ігрових ситуацій, розробка API для користувацьких розширень, тестування функціональності в різних сценаріях, документування коду та API.

UI/UX Дизайнер *Вимоги:* освіта вища в галузі дизайну або технічна з додатковою підготовкою, досвід розробки UI/UX для інструментів розробки не менше 2 років, знання Figma, розуміння принципів UX-дизайну, знання принципів візуалізації складних даних, досвід роботи з Unity Editor UI (не обов'язково але буде плюсом). *Функції:* розробка концепції користувацького інтерфейсу BehaviorTreeDesigner, проектування взаємодії користувача з редактором, створення прототипів та макетів інтерфейсу, розробка візуальної мови для елементів дерева поведінки, тестування UX-рішень з користувачами, дизайн навчальних матеріалів та документації.

Front-End розробник *Вимоги:* освіта вища технічна, досвід розробки користувацьких інтерфейсів для Unity Editor не менше 2 років, знання C# та Unity UI Toolkit/IMGUI, досвід створення редакторних розширень для Unity, розуміння принципів UI/UX дизайну, навички оптимізації графічного інтерфейсу. *Функції:* імплементація користувацького інтерфейсу редактора, розробка системи управління вузлами дерева поведінки (drag-and-drop зв'язки), створення інспекторів для налаштування властивостей вузлів, імплементація системи візуалізації стану дерева під час виконання, розробка системи візуального налагодження, інтеграція UI з ядром системи, оптимізація продуктивності інтерфейсу.

QA інженер *Вимоги:* освіта вища технічна або в галузі забезпечення якості, досвід тестування програмного забезпечення не менше 2 років, знання процесів забезпечення якості в ігровій розробці, розуміння принципів роботи AI-систем (бажано), навички автоматизації тестування, досвід з системами

відстеження помилок (JIRA). *Функції:* розробка та виконання тест-планів та тест-кейсів, забезпечення якості продукту на всіх етапах розробки, автоматизація процесів тестування, виявлення та документування дефектів, перевірка продуктивності системи в різних сценаріях, валідація користувацького досвіду, регресійне тестування.

Бізнес-аналітик *Вимоги:* освіта вища економічна або технічна з додатковою економічною підготовкою, досвід роботи з IT-продуктами не менше 2 років, володіння інструментами аналізу даних (Power BI), розуміння ринку ігрової розробки, навички проведення досліджень й аналітики. *Функції:* аналіз вимог користувачів та ринку, проведення дослідження конкурентів, участь у формуванні ціноутворення, аналіз метрик використання продукту, формування рекомендацій щодо розвитку продукту.

Маркетолог *Вимоги:* освіта вища в галузі маркетингу або суміжних областях, досвід створення маркетингових кампаній не менше 3 років, знання SMM аналізу ринку, досвід просування продуктів для розробників, розуміння специфіки ринку ігрової розробки, навички створення контенту. *Функції:* розробка стратегії просування продукту, взаємодія з цільовою аудиторією, створення контенту для соцмереж і платформи Unity Asset Store, організація участі в галузевих заходах, проведення маркетингових кампаній, аналіз ефективності маркетингових активностей, робота з відгуками користувачів.

3.1.3 Матриця відповідальності

Для забезпечення чіткого розподілу обов'язків та відповідальності в рамках проєкту розроблена матриця відповідальності за методологією RACI (Responsible, Accountable, Consulted, Informed), представлена в Таблиці 3.1.

Таблиця 3.1

Матриця відповідальності команди проєкту

Задача/Роль	CEO	PM	CTO	AI-розробники	UI/UX Дизайнер	Front-End розробник	QA інженер	Бізнес-аналітик	Маркетолог
Розробка концепції продукту	A	C	R	C	C	I	I	R	C
Дослідження ринку та конкурентів	A	C	I	I	I	I	I	R	R
Технічна архітектура системи	C	I	A/R	R	C	C	I	I	I
Розробка ядра ВТ системи	I	A	C	R	I	I	C	I	I
Дизайн користувацького інтерфейсу	I	A	C	C	R	C	C	C	I
Імплементація UI	I	A	C	C	C	R	C	I	I
Розробка системи налагодження	I	A	C	R	C	R	C	I	I
Тестування і забезпечення якості	C	A	C	C	C	C	R	I	I
Розробка документації	C	A	C	R	C	C	C	R	C
Маркетингова стратегія	A	C	C	I	C	I	I	C	R
Підготовка до випуску	A	R	C	R	C	R	R	C	R
Управління щоденними процесами	C	A/R	C	I	I	I	I	I	I
Управління спринтами	C	A/R	C	C	C	C	C	I	I

R (Responsible): Відповідальна особа отримує завдання від підзвітної особи і повинна виконати його в межах узгоджених параметрів і в узгоджений термін. Завдання може мати більше однієї відповідальної особи.

A (Accountable): Підзвітна особа гарантує, що всі відповідальні члени команди виконають завдання. Підзвітність не слід делегувати. Найкраще доручити виконання цього завдання одній особі, яка може виступати у ролі особи, що приймає рішення.

C (Consulted): Особа, з якою консультуються. Часто є носієм знань команди. До них можна звернутися за допомогою, додатковим контекстом і порадою щодо завдання. Варто визначити цих людей на ранній стадії, аби одразу залучити їх до проєкту та його робочого процесу.

I (Informed): Поінформована сторона - це, як правило, зацікавлена сторона, керівна рада або особа, яка хоче і потребує інформації про хід проєкту. Наявність проінформованої сторони сприяє внутрішній прозорості, згуртованості команди та дотриманню точних термінів проекту.

3.1.4 Принципи управління командою

Для ефективного функціонування команди проєкту встановлені наступні принципи управління: *Agile-методологія* - використання гнучкого підходу до розробки з короткими ітераціями (спринтами) що дозволяє швидко адаптуватись до змін і забезпечувати поступове нарощування функціоналу, *прозорість процесів* - відкрита комунікація регулярні зустрічі команди використання систем управління задачами для забезпечення видимості прогресу, *автономія та відповідальність* - надання команді достатньої свободи у прийнятті рішень у межах їх компетенції з одночасною відповідальністю за результати, *постійне вдосконалення* - регулярні ретроспективи для аналізу процесів і пошуку шляхів покращення, *крос-функціональна взаємодія* - заохочення співпраці між різними функціональними напрямками для забезпечення інтегрованого підходу до розробки, *менторство та розвиток* - створення умов для професійного зростання членів команди обмін знаннями та досвідом.

Взаємодія між членами команди забезпечується через: Stand-up зустрічі тричі на тиждень (15-20 хвилин), планування і ретроспективи кожні 2 тижня,

демонстрації результатів в кінці кожного спринту, спільне середовище для документації та управління знаннями, онлайн-інструменти комунікації (Discord) для оперативної взаємодії.

3.1.5 Підхід до залучення та розвитку команди

В рамках проєкту визначено наступний підхід до формування та розвитку команди:

Залучення талантів: пошук спеціалістів з досвідом в ігровій індустрії, використання рекомендацій та професійних мереж, залучення фахівців з відкритими джерелами на початкових етапах.

Адаптація нових членів команди: розробка процесу онбордингу, призначення менторів для нових учасників, документування процесів та знань для швидкого старту.

Розвиток компетенцій: виділення ресурсів на навчання та професійний розвиток, участь у профільних конференціях та спільнотах, внутрішні воркшопи та обмін знаннями.

Командний дух: регулярні командні заходи, визнання та відзначення досягнень, формування спільного бачення продукту.

Мотивація: гнучка система винагород, можливість впливати на розвиток продукту, перспективи професійного та кар'єрного зростання.

Такий підхід до формування та управління командою забезпечує оптимальне використання людських ресурсів, створення продуктивного робочого середовища та досягнення цілей проєкту з розробки візуального редактора дерев поведінки для ігрових систем ШІ в середовищі Unity.

3.2 Визначення ієрархічної структури та переліку робіт проєкту.

Частиною ефективного управління проєктом з розробки візуального редактора дерев поведінки є створення чітко визначеної ієрархічної структури робіт (Work Breakdown Structure). WBS дозволяє декомпозувати проєкт на

керовані компоненти, розподілити відповідальність та ресурси, а також підготувати базу для оцінки термінів та вартості.

Структура робіт проєкту BehaviorTreeDesigner розроблена відповідно до основних етапів життєвого циклу програмного забезпечення з урахуванням специфіки розробки інструментарію для ігрового AI. WBS містить шість основних груп робіт, кожна з яких поділяється на конкретні завдання. Повна схема розміщена у Додатку Г.

Представлена WBS відображає комплексний характер проєкту, поєднуючи в собі дослідницьку складову, технічну розробку, створення користувацького інтерфейсу, тестування та підготовку до комерційного випуску. Також варто зазначити, що в рамках обраної гнучкої методології управління, деталізація робіт може уточнюватись на початку кожної ітерації (спринту) з урахуванням отриманих результатів та зміни пріоритетів.

3.3 Розробка календарного плану. Планування термінів проєкту

Календарне планування є одним із ключових аспектів управління проєктом розробки візуального редактора дерев поведінки для ігрових систем ШІ в середовищі Unity. У цьому розділі представлено процес розробки календарного плану проєкту та основні інструменти що були для цього використані.

3.3.1 Методологія планування термінів проєкту

Для планування термінів проєкту було обрано гнучку методологію Agile з елементами Scrum, що дозволяє ефективно адаптуватися до змін вимог та зменшити ризики, пов'язані з розробкою інноваційного програмного продукту. Основними перевагами такого підходу в контексті даного проєкту є: Можливість ітеративної розробки з поступовим нарощуванням функціоналу, регулярні доставки працюючих інкрементів продукту, гнучке

реагування на зміни вимог та пріоритетів, підвищення прозорості процесу розробки, ефективніше управління ризиками.

Структура планування термінів проєкту була розроблена згідно з принципами ітеративної розробки та включає наступні рівні: *Рівень релізів* - визначає основні етапи випуску продукту з фіксованими датами. *Рівень спринтів* - двотижневі ітерації з конкретними цілями та результатами. *Рівень завдань* - щоденні задачі команди в рамках кожного спринту.

Для зручного оперування процесом розробки планів та управління термінами використовуються наступні інструменти: *Jira* - основна система управління проєктом що включає: панель Timeline для візуалізації всього беклогу та розподілу робіт по часу, інструменти для планування спринтів і відстеження їх виконання, інтеграцію з системами контролю версій для відстеження прогресу розробки. *Confluence* - для ведення документації проєкту включаючи: технічні специфікації, проєктну документацію, навчальні матеріали, протоколи зустрічей. *GitLab* - для управління версіями коду та CI/CD процесами.

3.3.2 Основні віхи проєкту

На основі аналізу обсягу робіт та розподілу функціоналу за пріоритетами, було визначено основні віхи проєкту відображені у табл. 3.2.

Таблиця 3.2

Віхи проєкту

Віха	Назва	Дата завершення	Ключові результати
B1	Початок проєкту	15.09.2024	Сформована команда, затверджений паспорт проєкту
B2	Випуск MVP	15.12.2024	Базовий функціонал редактора BT, включаючи ядро, візуальний редактор та основні інструменти налагодження
B3	Випуск версії 1.0	15.03.2025	Повний функціонал автоматизації тестування, розширені інструменти розробки та оптимізації
B4	Випуск версії 2.0	15.08.2025	Освітні функції, інтеграція з іншими системами, розширений API

Проєкт розділено на три основні релізи, кожен з яких додає певний набір функціональності до продукту:

1) **Реліз 1 (MVP) - 15.12.2024**
   - BT Core Framework - ядро системи виконання дерев поведінки
   - Візуальний редактор - базовий UI для створення та редагування дерев
   - Налагодження - основні інструменти для відлагодження AI
2) **Реліз 2 - 15.04.2025**
   - Автоматизація тестування - системи для автоматичного тестування AI
   - Розширені інструменти розробки - додаткові компоненти для створення AI
   - Оптимізація - підвищення продуктивності системи
3) **Реліз 3 - 15.08.2025**
   - Освітні функції - навчальні матеріали та приклади використання
   - Інтеграція з іншими системами - коннектори для популярних рішень
   - API для розширення - публічний API для створення власних компонентів

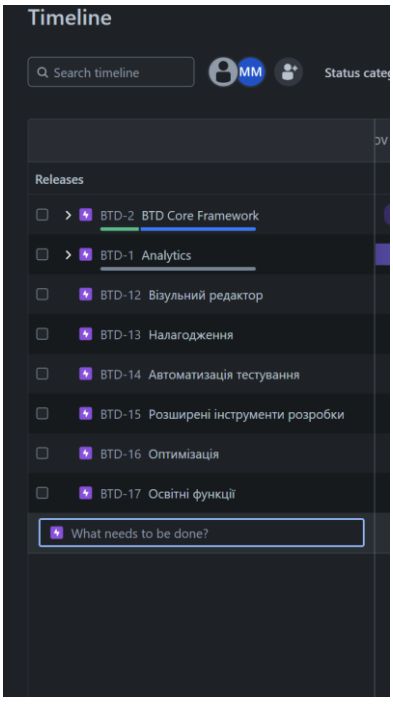


Рис. 3.2. Створення Epic у Jira

У створеному проєкті у Jira було визначено набір так званих Epic, які позначають категорії робіт глобального рівня, визначені у віхах (рис. 3.2).

3.3.3 Планування спринтів

Робота в рамках кожного релізу організована у двотижневі спринти. Для першого релізу (MVP) заплановано 6 спринтів відображених у табл. 3.3.

Таблиця 3.3

MVP Sprints

Спринт	Дати	Основні задачі
Sprint 1	15.09.2024 - 29.09.2024	BehaviourTree core архітектура, Налагодження середовища розробки, Імплементація стандартних типів вузлів
Sprint 2	30.09.2024 - 13.10.2024	Unit тести core функціоналу, Аналітика Core Framework
Sprint 3	14.10.2024 - 27.10.2024	Базовий UI для створення/редагування дерев, Система візуального з'єднання вузлів, Базова панель Inspector для налаштування параметрів
Sprint 4	28.10.2024 - 10.11.2024	Система збереження/завантаження дерев, Базова система логування, Візуальна індикація стану виконання
Sprint 5	11.11.2024 - 24.11.2024	Система breakpoints, Інструменти покрокового виконання
Sprint 6	25.11.2024 - 08.12.2024	Інтеграційне тестування, Підготовка документації, Фінальне тестування релізу MVP

У Jira були створені історії та задачі для виконання, розподілені по Epic-ам (рис. 3.3).

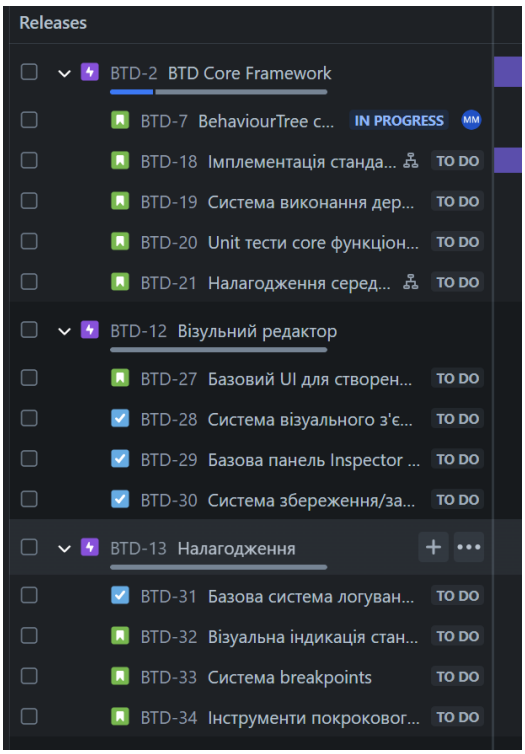


Рис. 3.3 Створення задач та історій у Jira

Кожна історія у свою чергу має список підзадач для виконання (рис. 3.4).

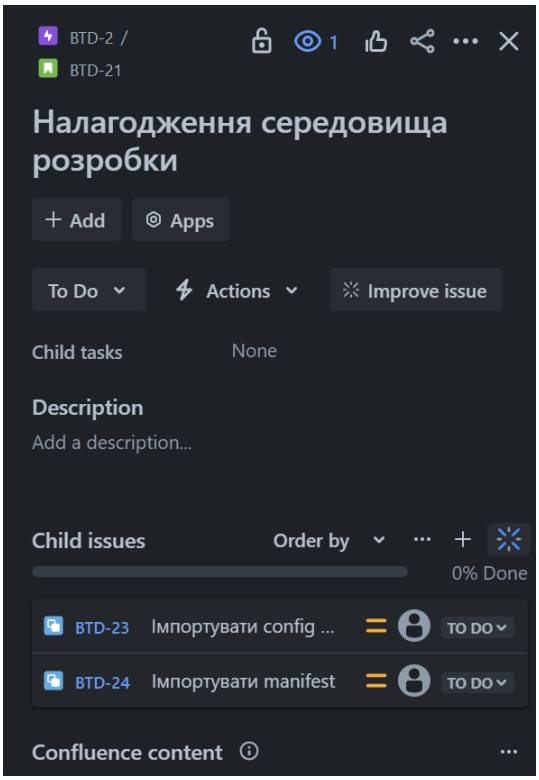


Рис. 3.4. Структура історій у Jira

Планування спрінту включає у себе перелік історій та задач до виконання, список артефактів прикріплених до задач, оцінка кожного блоку роботи за технологією Scrum Pocker та чіткі терміни завершення спрінту (2 тижні). Перші заплановані спрінти зображені на рис. 3.5.

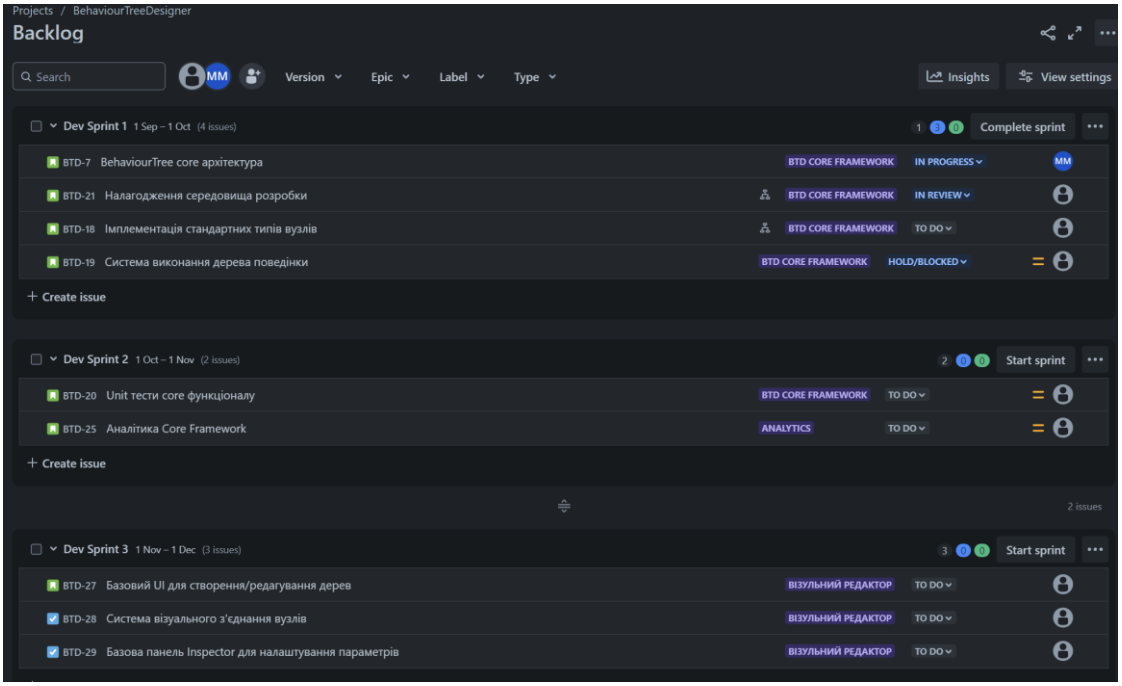


Рис. 3.5. Заплановані спрінти у Jira

Під час роботи в межах спрінта, використовується дошка спрінту, де легко відслідкувати процес виконання окремих блоків роботи, витрачений час та статус (рис. 3.6).

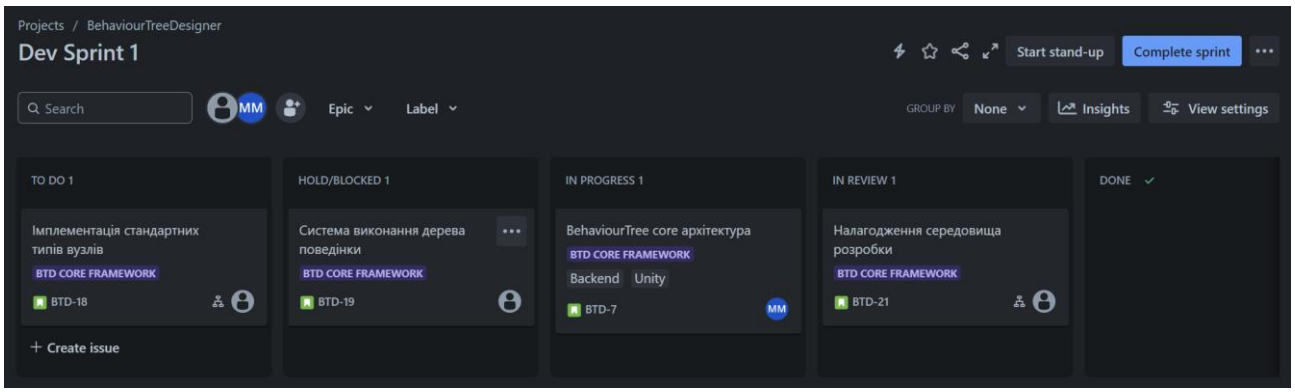


Рис. 3.6. Дошка спрінту у Jira

Також Jira дозволяє запланувати релізи (рис. 3.7), які включають у себе перелік Epic-ів та автоматично даються на панель Timeline.

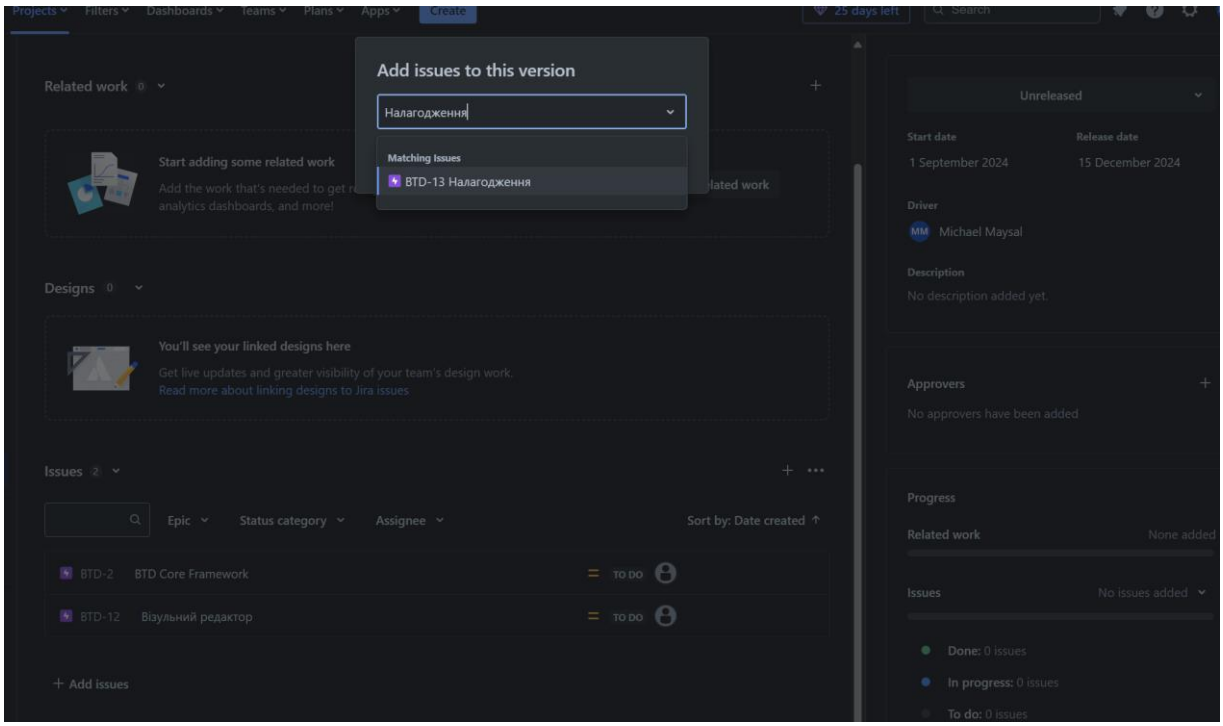


Рис. 3.7. Створення релізу у Jira

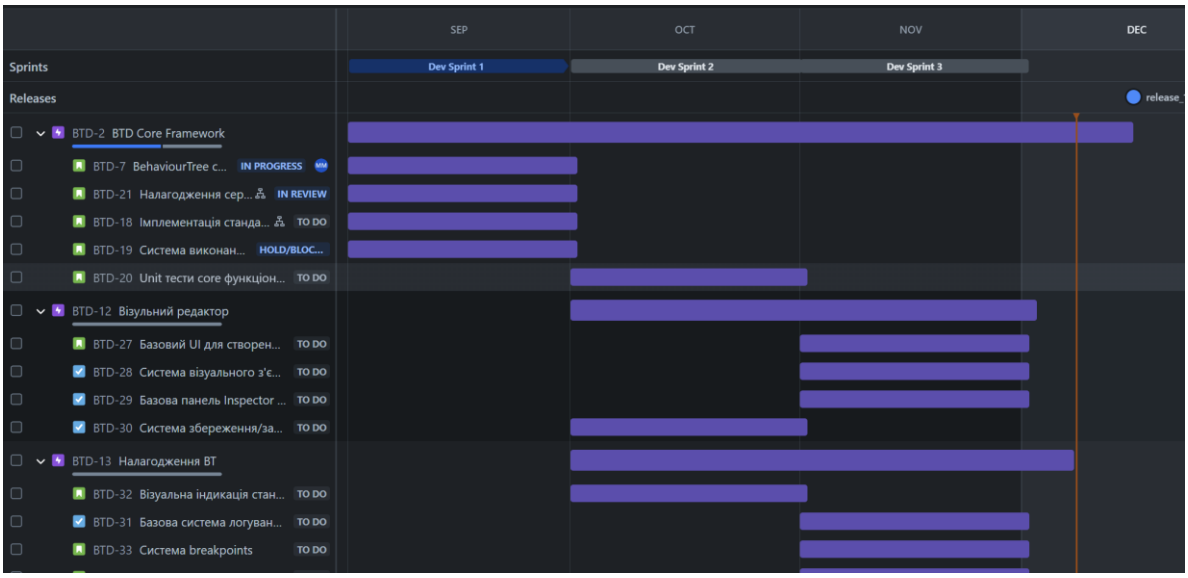


Рис. 3.8. Сюжетна карта беклогу

Панель Timeline у свою чергу наочно демонструє сюжетну карту усього беклогу проєкту (рис. 3.8). Це дає змогу відстежити загальний прогрес виконання проєкту. Тут легко налаштувати дати початку й завершення окремих частин проєкту, залежності між завданнями та паралельність їх виконання. На панелі відображені назначені релізи та обсяг робіт по кожному з них.

Додатково Jira автоматично повідомляє про розбіжності між запланованим та фактичним, тож легко помітити аспекти яким треба приділити додаткову увагу.

3.3.7 Моніторинг і контроль термінів проєкту

Для забезпечення дотримання запланованих термінів проєкту впроваджено наступні процеси: *Планування спринту* - двогодинна зустріч на початку кожного спринту для планування робіт та встановлення цілей, *ретроспектива спринту* - зустріч в кінці кожного спринту для аналізу результатів виявлення проблем та покращення процесів, *демонстрація результатів* – коротка презентація готових функцій стейкхолдерам в кінці кожного спринту, *Stand-up зустрічі* - короткі 15-хвилинні зустрічі для обговорення прогресу та блокерів, *аналіз burndown chart* - регулярний аналіз графіків виконання робіт для раннього виявлення відхилень від плану.

Такий комплексний підхід до планування термінів проєкту дозволяє ефективно управляти розробкою візуального редактора дерев поведінки, забезпечуючи своєчасне виконання робіт та досягнення поставлених цілей. Гнучка методологія та сучасні інструменти управління проєктами дають можливість швидко адаптуватися до змін вимог та мінімізувати ризики затримок у розробці.

3.4 Визначення та планування ресурсів.

Ефективне управління ресурсами є критичним фактором успіху проєкту. Розглянемо процес визначення необхідних ресурсів, їх планування та розподіл.

3.4.1 Класифікація ресурсів проєкту

Для реалізації проєкту BehaviorTreeDesigner необхідні такі категорії ресурсів: *Людські ресурси:* технічні спеціалісти (розробники дизайнери тестувальники), управлінський персонал (керівник проєкту проджект-

менеджер), підтримуючий персонал (бізнес-аналітик маркетолог). *Технічні ресурси:* програмне забезпечення (Unity IDE системи управління проєктами), апаратне забезпечення (робочі станції сервери), інфраструктура (хмарні сервіси системи контролю версій). *Фінансові ресурси:* бюджет на заробітну плату, бюджет на ліцензії та технічні засоби, бюджет на маркетинг та просування продукту, бюджет на підтримку продукту, резерви. *Часові ресурси.*

3.4.2 Планування людських ресурсів

Таблиця 3.4

Планування людських ресурсів

Роль	Кількість	Зайнятість	Необхідні навички	Період залучення
Керівник проєкту	1	50%	Управління IT-проєктами, знання Unity	Весь проєкт
Технічний директор	1	100%	Досвід розробки AI-систем, архітектура ПЗ	Весь проєкт
Проджект-менеджер	1	100%	Agile/Scrum, інструменти управління проєктами	Весь проєкт
C# AI розробники	3	100%	C#, Unity, ігровий ШІ, патерни проектування	Весь проєкт
UI/UX Дизайнер	1	75%	Дизайн інтерфейсів, Unity Editor UI	Релізи 1-2
Front-End розробник	1	100%	C#, Unity UI Toolkit/IMGUI	Релізи 1-2
QA інженер	1	75%	Тестування ПЗ, автоматизація тестів	Весь проєкт
Бізнес-аналітик	1	50%	Аналіз даних, ринок ігрової розробки	Весь проєкт
Маркетолог	1	50%	Просування продуктів для розробників	Починаючи з Релізу 1

У табл. 3.4 наведені вимоги до людських ресурсів та їх загальна зайнятість та період залучення.

Таблиця 3.5

Матриця розподілу людських ресурсів за фазами проєкту

Роль	Фаза ініціації	Фаза MVP (Реліз 1)	Фаза розвитку (Реліз 2)	Фаза розширення (Реліз 3)
Керівник проєкту	80%	40%	40%	60%
Технічний директор	70%	100%	100%	80%

Продовження табл. 3.5

Проджект-менеджер	100%	100%	100%	100%
C# AI розробники	50%	100%	100%	100%
UI/UX Дизайнер	100%	80%	60%	20%
Front-End розробник	30%	100%	100%	50%
QA інженер	20%	80%	100%	100%
Бізнес-аналітик	100%	30%	30%	70%
Маркетолог	20%	50%	70%	100%

Розподіл навантаження людських ресурсів між фазами проєкту представлено в табл. 3.5 (у відсотках від повної зайнятості).

3.4.3 Планування технічних ресурсів

У табл. 3.6 наведені технічні ресурси, необхідні для успішної реалізації проєкту.

Таблиця 3.6

Технічні ресурси

Ресурс	Кількість	Призначення	Період використання
Ліцензії Unity Pro	5	Розробка core функціоналу та візуального редактора	Весь проєкт
Ліцензії JetBrains Rider	5	IDE для C# розробки	Весь проєкт
Робочі станції високопродуктивні	5	Для розробників та технічного директора	Весь проєкт
Робочі станції середньої продуктивності	5	Для решти членів команди	Весь проєкт
Ліцензія Jira + Confluence	11	Для управління проєктом та документації	Весь проєкт

Продовження табл. 3.6

GitLab Premium	1	Система контролю версій з CI/CD	Весь проєкт
Сервер для CI/CD	1	Автоматизоване тестування та збірка	Починаючи з Релізу 1
Хмарний хостинг для документації	1	Розміщення документації та API	Починаючи з Релізу 1
Ліцензії для графічних редакторів	2	Для UI/UX дизайнера та маркетолога	Весь проєкт

Ефективне використання та управління цими технічними ресурсами є критичним фактором для дотримання термінів розробки та забезпечення високої якості кінцевого продукту. Планується регулярний аудит використання ресурсів для оптимізації витрат та своєчасного масштабування інфраструктури при необхідності.

3.5 Визначення вартості проєкту.

Визначення вартості проєкту розробки візуального редактора дерев поведінки для ігрових систем ШІ є важливим аспектом планування, що дозволяє ефективно розподілити фінансові ресурси, встановити бюджетні рамки та контролювати витрати протягом життєвого циклу проєкту. У цьому розділі представлено детальний аналіз структури витрат, методи оцінки вартості та базовий графік вартості проєкту.

3.5.1 Структура витрат

Структура витрат проєкту BehaviorTreeDesigner розподілена на чотири основні категорії, які відображають специфіку проєкту розробки програмного забезпечення для ігрової індустрії (рис. 3.9):

1. Розробка (40%).
2. Маркетинг та просування (25%).
3. Підтримка користувачів (20%).
4. Інфраструктура та комісії (15%).

Також додатково виділяються резерви в обсязці 15% від загального бюджету.

Категорія витрат на розробку становить найбільшу частку бюджету проєкту та включає:

Витрати на персонал розробки: заробітна плата C# AI розробників 3 фахівці × $3,000/місяць × 12 місяців = $108,000, заробітна плата Front-End розробника $2,500/місяць × 12 місяців = $30,000, заробітна плата QA інженера $2,000/місяць × 9 місяців = $18,000, оплата праці технічного директора $4,000/місяць × 12 місяців = $48,000. *Технічні ресурси для розробки:* ліцензії Unity Pro 5 × $1,800/рік = $9,000, ліцензії JetBrains Rider 5 × $500/рік = $2,500, хостинг та інфраструктура $400/місяць × 12 місяців = $4,800. *Витрати на розширення функціоналу:* придбання сторонніх ассетів та компонентів $4,000, інтеграція з іншими системами $6,000. *Витрати на оптимізацію та виправлення помилок:* тестове обладнання для різних платформ $3,500, інструменти для профілювання та діагностики $2,000.

Загальні витрати на розробку та підтримку: $235,800 (40% загального бюджету)

Категорія витрат на маркетинг спрямована на забезпечення успішного виходу продукту на ринок та формування його позитивного сприйняття серед цільової аудиторії: *Рекламні кампанії:* таргетована реклама в спеціалізованих ресурсах $2,000/місяць × 6 місяців = $12,000, контекстна реклама в пошукових системах $1,500/місяць × 6 місяців = $9,000. *Участь у галузевих заходах:* виставки та конференції розробників ігор $20,000, спонсорство професійних заходів $12,000. *Створення маркетингових матеріалів:* розробка веб-сайту продукту $6,000, створення демонстраційних відео та матеріалів $10,000, підготовка технічних статей та документації $7,000. *Партнерські програми та інтеграції:* співпраця з навчальними закладами $12,000, партнерські програми з розробниками ігор $15,000. *Персонал для маркетингу:* заробітна плата маркетолога $2,500/місяць × 12 місяців = $30,000, послуги зовнішніх консультантів з маркетингу $15,000.

Загальні витрати на маркетинг та просування: $148,000 (25% загального бюджету)

Категорія витрат на підтримку користувачів охоплює всі витрати, пов'язані з забезпеченням технічної підтримки користувачів продукту: *Персонал підтримки:* фахівці технічної підтримки 2 × $1,500/місяць × 9 місяців = $27,000, технічні консультанти $80/година × 200 годин = $16,000. *Розробка навчальних матеріалів:* створення відеоуроків $10,000, розробка інтерактивних посібників $8,000, підготовка документації для користувачів $6,000. *Платформа для підтримки користувачів:* система управління тікетами $300/місяць × 12 місяців = $3,600, форум спільноти $200/місяць × 12 місяців = $2,400, база знань $250/місяць × 12 місяців = $3,000. *Витрати на комунікацію з клієнтами:* системи відеоконференцій $50/місяць × 12 місяців = $600, програмне забезпечення для демонстрацій $100/місяць × 12 місяців = $1,200, засоби для збору відгуків користувачів $150/місяць × 12 місяців = $1,800.

Загальні витрати на технічну підтримку: $118,600 (20% загального бюджету)

Остання категорія витрат включає всі технічні та операційні витрати, пов'язані з функціонуванням продукту та його дистрибуцією: *Хостинг та технічне обслуговування:* сервери для CI/CD та тестування $500/місяць × 12 місяців = $6,000, хмарні сервіси для демонстрацій $300/місяць × 12 місяців = $3,600, системи резервного копіювання $200/місяць × 12 місяців = $2,400. *Комісії платіжних систем:* щомісячні платежі за використання платіжних шлюзів $100/місяць × 12 місяців = $1,200. *Комісії маркетплейсів:* комісія Unity Asset Store $55,500, інші маркетплейси $15,000. *Системи моніторингу та аналітики:* сервіси аналітики використання продукту $200/місяць × 12 місяців = $2,400, інструменти моніторингу продуктивності $150/місяць × 12 місяців = $1,800, системи збору та аналізу відгуків $100/місяць × 12 місяців = $1,200, інструменти для A/B тестування $80/місяць × 6 місяців = $480.

Загальні витрати на серверну інфраструктуру та комісії: $89,580 (15% загального бюджету)

Сумарний бюджет проєкту розробки візуального редактора дерев поведінки складає: $235,800 + $148,000 + $118,600 + $89,580 = $591,980.

Рис. 3.9. Загальна структура витрат

Аналіз структури витрат показує, що найбільша частка витрат припадає на розробку та підтримку продукту, що є типовим для високотехнологічних стартапів. Поступове масштабування бізнесу дозволить оптимізувати витрати на розробку за рахунок зростання ефективності та автоматизації процесів.

3.5.2 Базовий графік вартості

Графік вартості проєкту дозволяє відстежувати розподіл витрат у часі та контролювати відповідність фактичних витрат запланованим. Для BehaviorTreeDesigner розроблено базовий розподіл вартості з розбивкою по місяцях (табл. 3.8) та релізах (табл. 3.7).

Таблиця 3.7

Розподіл вартості по релізах

Реліз	**Період**	**Планові витрати**	**Відсоток від загального бюджету**
Фаза ініціації	1 місяць (Вересень 2024)	$29,600	5%
Реліз 1 (MVP)	3 місяці (Жовтень-Грудень 2024)	$207,200	35%
Реліз 2	4 місяці (Січень-Квітень 2025)	$236,800	40%
Реліз 3	4 місяці (Травень-Серпень 2025)	$118,380	20%
Загальний бюджет	**12 місяців**	**$591,980**	**100%**

Таблиця 3.8

Розподіл вартості помісячно

Місяць	Дата	Планові витрати	Накопичувальний підсумок	Відсоток від загального бюджету
M1	Вересень 2024	$29,600	$29,600	5.0%
M2	Жовтень 2024	$59,000	$88,600	15.0%
M3	Листопад 2024	$72,000	$160,600	27.1%
M4	Грудень 2024	$76,200	$236,800	40.0%
M5	Січень 2025	$59,000	$295,800	50.0%
M6	Лютий 2025	$63,500	$359,300	60.7%
M7	Березень 2025	$59,000	$418,300	70.7%
M8	Квітень 2025	$55,500	$473,800	80.0%
M9	Травень 2025	$33,000	$506,800	85.6%
M10	Червень 2025	$30,000	$536,800	90.7%
M11	Липень 2025	$29,180	$565,980	95.6%
M12	Серпень 2025	$26,000	$591,980	100.0%

Базовий графік вартості (або S-крива) візуально представляє розподіл кумулятивних витрат проєкту в часі (рис. 3.10). Цей графік використовується як основа для порівняння фактичних витрат із запланованими в процесі моніторингу проєкту.

Рис. 3.10. Графік вартості

S-крива демонструє класичну S-подібну форму, що відображає типовий розподіл витрат у проєктах розробки програмного забезпечення:

– Поступове зростання на початковому етапі (фаза ініціації)
– Стрімке зростання під час активної розробки (Реліз 1 та 2)
– Сповільнення темпів витрат на завершальному етапі (Реліз 3)

3.5.3 Стратегія резервування коштів

Для управління невизначеністю та ризиками проєкту розроблено стратегію резервування коштів:

1) **Резерв на непередбачені обставини (contingency reserve):**
   - 10% від вартості кожного релізу на покриття ідентифікованих ризиків
   - Загальна сума: $59,198 (10% від загального бюджету)
2) **Управлінський резерв (management reserve):**
   - 5% від загального бюджету на покриття невідомих ризиків
   - Сума: $29,599 (5% від загального бюджету)
3) **Правила використання резервів:**
   - Резерв на непередбачені обставини може бути використаний керівником проєкту за узгодженням з технічним директором
   - Управлінський резерв може бути використаний лише за рішенням керівного комітету проєкту

3.6 Управління ризиками

Розглянемо важливий аспект управління проєктом створення BehaviourTreeDesigner – ідентифікація та методи управління ризиками. Ця компонента є критичною, оскільки дозволяє мінімізувати потенційні затримки та втрати і збільшує ймовірність успішного завершення проєкту.

3.6.1 Аналіз відомих стандартів та визначення стратегії управління ризиками

Одним з першочергових завдань управління ризиками є побудова стратегії їх мінімізації та усунення. Проаналізуємо основні стандарти індустрії в області стратегій управління ризиками.

MSF (Microsoft Solutions Framework) *Підхід:* спеціалізований для розробки програмного забезпечення з особливим фокусом на інструментальних засобах розробки. *Переваги:* спеціально адаптований для IT-проєктів, приділяє особливу увагу тригерам ризикових подій, включає формування бази знань про ризики, добре підходить для ітеративної розробки. *Недоліки:* менш формалізований порівняно з іншими стандартами, вимагає значного досвіду команди в IT-розробці.

ISO 31000 *Підхід:* загальний стандарт управління ризиками для будь-яких організацій та проєктів. *Переваги:* універсальність та гнучкість, чітка структура процесів, міжнародне визнання. *Недоліки:* занадто загальний для специфічних IT-проєктів, потребує значної адаптації.

PMI Risk Management Framework *Підхід:* комплексна методологія управління ризиками на всіх етапах життєвого циклу проєкту. *Переваги:* системний підхід до управління ризиками, охоплює всі аспекти технічні ринкові організаційні фінансові, чіткі процеси планування ідентифікації аналізу та реагування, інтеграція управління ризиками з іншими областями управління проєктом, фокус на бізнес-цінності та ROI проєкту. *Недоліки:* може бути надмірним для малих проєктів, вимагає формального документування.

Було прийнято рішення використовувати PMI Risk Management. Основні причини вибору даної стратегії наступні:

Проєкт має комплексні ризики різних категорій: технічні (архітектура продуктивність інтеграція), ринкові (конкуренція попит), фінансові (монетизація інвестиції), організаційні (команда процеси).

Необхідність структурованого підходу до: оцінки життєздатності проєкту, планування виходу на ринок, управління очікуваннями стейкхолдерів, забезпечення якості продукту.

3.6.2 Ідентифікація та оцінка ризиків

Проведемо ідентифікацію ризиків проєкту розробки інструменту BehaviourTreeDesigner та дамо їм якісну та кількісну оцінку.

Класифікацію ризиків проведемо за наступними типами: Технічні ризики, Ринкові ризики, Організаційні ризики, Операційні ризики, Форс-мажорні ризики.

Далі визначимо критерії ризиків для оцінювання:

Сила впливу:

1 - Мінімальний вплив;

2 - Низький вплив;

3 - Середній вплив;

4 - Високий вплив;

5 - Дуже високий вплив.

Керованість ризику:

1 - Легко керований;

2 - Можливий, але потребує зусиль;

3 - Середньо важко керований;

4 - Важко керований;

5 - Практично неможливо керований.

Ймовірність ризику:

1 - Дуже низька ймовірність;

2 - Низька ймовірність;

3 - Середня ймовірність;

4 - Висока ймовірність;

5 - Дуже висока ймовірність.

Таблиця 3.9

Ідентифікація та оцінка ризиків

№	Тип Ризику	Ризикова подія	Сила Впливу		Керованість		Ймовірність		Затримки у часі		Фінансові втрати		Важливість ризику
			Якісна	К	Якісна	К	Якісна	К	Якісна	К	Якісна	К	
1	Технічні	Помилки в логіці виконання ВТ	Дуже високий	5	Легко керований	1	Висока ймовірність	4	Значуща (11-20 днів)	4	Помірні	3	12
2		Проблеми з продуктивністю при великих деревах	Високий	4	Можливий, але потребує зусиль	2	Висока ймовірність	4	Помірна (6-10 днів)	3	Помірні	3	12
3		Несумісність з новими версіями Unity	Дуже високий	5	Важко керований	4	Середня ймовірність	3	Критична (більше 20 днів)	5	Значущі	5	15
4		Проблеми з масштабуванням AI-систем	Високий	4	Середньо важко керований	3	Середня ймовірність	3	Помірна (6-10 днів)	3	Помірні	3	9
5	Ринкові	Поява конкурентного рішення від Unity	Дуже високий	5	Практично неможливо керований	5	Середня ймовірність	3	Незначна (3-5 днів)	2	Критичні	5	15
9	Організаційні	Нестача експертизи в AI-системах	Високий	4	Середньо важко керований	3	Середня ймовірність	3	Значуща (11-20 днів)	4	Помірні	3	9
10		Втрата ключових розробників	Високий	4	Можливий, але потребує зусиль	2	Низька ймовірність	2	Критична (більше 20 днів)	5	Значущі	4	8
11		Затримки через складність завдань	Середній	3	Середньо важко керований	3	Висока ймовірність	4	Значуща (11-20 днів)	4	Помірні	3	12
12		Проблеми з документуванням функціоналу	Середній	3	Легко керований	1	Висока ймовірність	4	Помірна (6-10 днів)	3	Незначні	2	8
13		Проблеми з відлагодженням	Високий	4	Можливий, але потребує зусиль	2	Середня ймовірність	3	Помірна (6-10 днів)	3	Помірні	3	9
14		Складність оновлення існуючих проєктів	Високий	4	Середньо важко керований	3	Середня ймовірність	3	Значуща (11-20 днів)	4	Значущі	4	12
15	Форс-мажори	Зміни в політиці Unity щодо сторонніх інструментів	Дуже високий	5	Практично неможливо керований	5	Низька ймовірність	2	Помірна (6-10 днів)	3	Критичні	5	10
16		Проблеми з хостингом Asset Store	Середній	3	Практично неможливо керований	5	Дуже низька ймовірність	1	Незначна (3-5 днів)	2	Помірні	3	3
17		Втрата доступу до сервісів розробки	Високий	4	Важко керований	4	Дуже низька ймовірність	1	Значуща (11-20 днів)	4	Значущі	4	4

Порахуємо кофіцієнт важливості для кожного ризику та проведемо ранжування переліку ризикових подій з сортуванням по трьом категоріям:

- найбільш впливові (15-20);
- середнього впливу (8-14);
- найменш впливові (1-7).

Таблиця 3.10

Ранжування ризиків

№	Тип Ризику	Ризикова подія	Важливість ризику
7	Ринкові	Складність монетизації інструменту	16
3	Технічні	Несумісність з новими версіями Unity	15
5	Ринкові	Поява конкурентного рішення від Unity	15
1	Технічні	Помилки в логіці виконання BT	12
2	Технічні	Проблеми з продуктивністю при великих деревах	12
6	Ринкові	Низький попит серед розробників	12
11	Організаційні	Затримки через складність завдань	12
13	Операційні	Складність підтримки користувачів	12
15	Операційні	Складність оновлення існуючих проєктів	12
16	Форс-мажори	Зміни в політиці Unity щодо сторонніх інструментів	10
4	Технічні	Проблеми з масштабуванням AI-систем	9
8	Ринкові	Негативні відгуки користувачів	9
9	Організаційні	Нестача експертизи в AI-системах	9
14	Операційні	Проблеми з відлагодженням	9
10	Організаційні	Втрата ключових розробників	8
12	Організаційні	Проблеми з документуванням функціоналу	8
18	Форс-мажори	Втрата доступу до сервісів розробки	4
19	Форс-мажори	Природні катастрофи	4
17	Форс-мажори	Проблеми з хостингом Asset Store	3

Отже аналіз таблиці 3.10 демонструє що найбільшу загрозу становлять ринкові ризики, пов'язані з монетизацією та конкуренцією, та технічні ризики, пов'язані з сумісністю інструменту. Натомість форс-мажорні ризики отримали низькі оцінки важливості через малу ймовірність їх настання.

3.6.3 Розробка протиризикових заходів

Ефективне управління ризиками в проєкті з розробки візуального редактора дерев поведінки потребує не лише ідентифікації та оцінки ризиків, але й розробки конкретних заходів для запобігання їх виникненню та мінімізації негативних наслідків у разі їх настання. Розробка протиризикових заходів є критично важливим елементом системи управління ризиками, що дозволяє перетворити реактивний підхід до вирішення проблем на проактивний, спрямований на запобігання та раннє виявлення потенційних загроз.

Для кожного з ідентифікованих високопріоритетних ризиків було розроблено три рівні протиризикових заходів: Превентивні заходи (профілактика), Заходи раннього реагування (при появі симптомів), Заходи пом'якшення наслідків (при виникненні проблеми). Запропоновані протиризикові заходи відображають збалансований підхід до управління загрозами проєкту. Превентивні дії, хоча й потребують початкових інвестицій, у довгостроковій перспективі демонструють найвищу економічну ефективність, особливо для ризиків з високим потенційним впливом. Для забезпечення оптимального співвідношення витрат та результатів у контексті реалізації протиризикових заходів, прийнято рішення сконцентрувати ресурси на зниженні ймовірності та впливу п'яти найкритичніших ризиків, застосовуючи для решти загроз стратегію моніторингу та швидкого реагування. Така пріоритизація дозволяє ефективно розподілити обмежені ресурси проєкту, максимізуючи загальний рівень безпеки.

Таблиця 3.11

Протиризикові заходи

Ризикова подія	**ПРЗ 1 (Профілактика)**	**Симптоми (рання ознака)**	**ПРЗ 2 при симптомі**	**ПРЗ 3 при приблемі**
Складність монетизації інструменту	- Проведення маркетингового дослідження цін на подібні інструменти - Розробка різних планів підписки (безкоштовний з обмеженнями, професійний, enterprise) - Створення системи метрик для відслідковування користування функціями	- Низький рівень конверсії безкоштовних користувачів у платних - Негативні відгуки про ціноутворення - Користувачі використовують тільки безкоштовні функції	- Перегляд цінової політики - Впровадження тимчасових акцій та знижок - Додавання нових преміум-функцій - Покращення демонстрації цінності платних функцій	- Повна зміна бізнес-моделі - Перехід на інший тип монетизації (наприклад, консалтинг та підтримка) - Можливий перехід у open-source з монетизацією через послуги
Несумісність з новими версіями Unity	- Створення системи автоматичного тестування на різних версіях Unity - Участь у бета-тестуванні нових версій Unity - Розробка архітектури з мінімальною залежністю від специфічних Unity API - Документування всіх залежностей від Unity	- Помилки компіляції в бета-версіях - Deprecation warnings у нових версіях - Баг-репорти від бета-тестерів	- Швидке виправлення критичних несумісностей - Комунікація з користувачами про відомі проблеми - Тимчасове обмеження підтримуваних версій - Створення hotfix-релізів	- Масштабний рефакторинг кодової бази - Переписування проблемних модулів - Випуск нової мажорної версії з оновленою архітектурою

Продовження табл. 3.11

Поява конкурентного рішення від Unity	- Активна участь у спільноті Unity - Моніторинг roadmap Unity та їх experimental features - Розробка унікальних функцій, яких немає в стандартних рішеннях - Створення міцної спільноти користувачів	- Чутки про розробку подібного інструменту - Поява схожих експериментальних фіч у Unity - Зменшення інтересу інвесторів	- Прискорення розробки диференціюючих функцій - Посилення маркетингової активності - Поглиблення інтеграції з іншими популярними інструментами	- Pivot у нішеве рішення - Можливий продаж технології Unity - Злиття з іншими інструментами для створення - комплексного рішення

Аналіз демонструє значну перевагу профілактичних заходів у співвідношенні вартості до ефективності, особливо у випадку монетизації та конкуренції. При цьому технічні ризики, пов'язані з несумісністю Unity, вимагають більших початкових інвестицій у профілактику, але це виправдано, враховуючи високу вартість їх вирішення на пізніх етапах. Особливу увагу слід приділити ранньому виявленню симптомів та оперативній реакції на них, оскільки це дозволить уникнути необхідності впровадження найбільш дорогих заходів реагування.

Впровадження запропонованої системи управління ризиками передбачає не лише застосування описаних протиризикових заходів, але й формування культури ризик-орієнтованого мислення в команді проєкту. Регулярні обговорення потенційних загроз та способів їх подолання будуть інтегровані в процес розробки через спеціальні сесії при плануванні спринтів та ретроспективах. Такий підхід дозволить забезпечити проактивне виявлення та оперативне реагування на ризики, гарантуючи стабільне просування проєкту до успішного завершення.

РОЗДІЛ 4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РЕАЛІЗАЦІЇ ІТ ПРОЄКТУ

4.1 Опис структури програмного забезпечення

В основі інструменту лежить концепція BehaviourTree з набором основних поширених логічних функцій та обробників, притаманних BehaviourTree, графічного інтерфейсу для зручної роботи та тестування реалізованих дерев поведінок та API для реалізації свого унікального функціоналу під власний проєкт.

З архітектурної точки зору інструмент створений на базі суміші підходів ECS та ООП, що дозволяє проводити обчислення поведінки кількох AI агентів одночасно зі значною ступінню паралелізації.

Для розуміння архітектури та принципів функціонування візуального редактора дерев поведінки важливо проаналізувати структуру інформаційних потоків системи. Ключовим аспектом ефективної роботи BehaviorTreeDesigner є чітка організація взаємодії між компонентами редактора та системою виконання ігрового ШІ. Розроблена архітектура забезпечує безшовний перехід від візуального представлення логіки поведінки до її ефективної реалізації в рантаймі гри за допомогою ECS-підходу.

На схемі інформаційних потоків (Додаток Д) представлено основні компоненти системи та шляхи передачі даних між ними, що демонструє загальну організацію розробленого інструменту. Від коректності цієї організації безпосередньо залежить продуктивність, масштабованість та зручність використання всього рішення. Інформаційна модель розкриває процес перетворення візуальних конструкцій, створених розробником у редакторі, в ефективні структури даних, що забезпечують виконання логіки поведінки ігрових агентів у реальному часі.

Як видно зі схеми, центральним елементом архітектури є БТ-двигун (BT Engine), що забезпечує перетворення візуально створених дерев поведінки в

ефективні структури даних для ECS. Інформаційні потоки в системі можна розділити на три ключові групи:

Потоки проектування — взаємодія між користувачем та редактором для створення редагування та налаштування дерев поведінки ці потоки включають маніпуляції з графічним інтерфейсом створення та з'єднання вузлів налаштування параметрів.

Потоки серіалізації — перетворення візуальних конструкцій у структуровані дані (ScriptableObjects) що зберігаються як ассети Unity ці потоки забезпечують стійкість даних між сесіями редагування та можливість багаторазового використання розроблених дерев.

Потоки виконання — трансформація серіалізованих даних у ECS-сутності та компоненти їх обробка системами відповідно до логіки дерев поведінки та взаємодія з ігровим світом ці потоки активні під час виконання гри та забезпечують високопродуктивну роботу ШІ.

4.2 Реалізація ядра системи

Центральним компонентом розробленого інструментарію є ядро системи, що реалізує фундаментальні механізми роботи з деревами поведінки та їх інтеграцію з Entity-Component-System архітектурою.

Архітектурно ядро системи побудовано з дотриманням принципів гібридного підходу, що поєднує переваги об'єктно-орієнтованого програмування для представлення даних у редакторі та Entity-Component-System для високопродуктивного виконання логіки поведінки в рантаймі. Такий підхід забезпечує чітке розділення відповідальностей між компонентами системи та дозволяє мінімізувати накладні витрати під час виконання.

При реалізації ядра системи були застосовані паттерни проектування, що підвищують модульність та розширюваність кодової бази: Strategy (для різних типів вузлів), Composite (для структурування дерева), Observer (для системи подій), Factory (для створення вузлів) та Visitor (для обходу дерева). Особлива

увага приділялася оптимізації використання пам'яті через впровадження систем об'єктних пулів та ефективних структур даних.

Кожен вузол дерева представлений окремою сутністю у ECS світі. Також кожен вузол містить екземпляр INodeController, який є класом-помічником для зміни стану вузла (рис. 4.2).

Набір реалізованих стандартних вузлів:

Root node: Composite nodes, Sequence, Selector, Parallel, RandomSelector.

Decorator nodes: Repeat, Failer, Succeeder, Timeout, Inverter.

Action nodes: Wait, IsLucky, Failure, Success, RandomFailure, Log, Breakpoint.

Системи обробки цих вузлів представлені на рисунку 4.1.

```
public class BehaviourTreeFeature : IEcsFeature
{
    private readonly IBotDifficultyProvider _botDifficultyProvider;

    ◇  2 usages
    public BehaviourTreeFeature(IBotDifficultyProvider botDifficultyProvider){...}

    ◇  0+1 usages
    public void Build(EcsSystems systems)
    {
        systems
            .Add(new BehaviourTreeCreatorSystem(_botDifficultyProvider))

            //Iteration Systems
            .Add(new SequenceIterationSystem())
            .Add(new SelectorIterationSystem())
            .Add(new RandomSelectorIterationSystem())

            //Activation systems
            .Add(new RootNodeActivationSystem())
            .Add(new ParallelActivationSystem())
            .Add( system: new DefaultCompositeActivationSystem<SequenceNode>())
            .Add( system: new DefaultCompositeActivationSystem<SelectorNode>())
            .Add( system: new DefaultCompositeActivationSystem<RandomSelectorNode>())

            .Add(new RepeatActivationSystem())
            .Add(new TimeoutActivationSystem())
            .Add( system: new DefaultDecoratorActivationSystem<FailerNode>())
            .Add( system: new DefaultDecoratorActivationSystem<SucceederNode>())
            .Add( system: new DefaultDecoratorActivationSystem<InverterNode>())

            //Composite Node Systems
            .Add(new SequenceNodeSystem())
            .Add(new SelectorNodeSystem())
            .Add(new RandomSelectorNodeSystem())
            .Add(new ParallelNodeSystem())

            .Add(new RepeatNodeSystem())
            .Add(new FailerNodeSystem())
            .Add(new SucceederNodeSystem())
            .Add(new TimeoutNodeSystem())
            .Add(new InverterNodeSystem())

            // Common Action Node Systems
            .Add(new BreakpointSystem())
            .Add(new WaitSystem())
            .Add(new IsLuckySystem())
            .Add(new LogSystem())
            .Add(new RandomFailureSystem())
            .Add(new FailureSystem())
            .Add(new SuccessSystem());
    }
}
```

Рис. 4.1. Реалізація поширених вузлів BT

```
namespace Client.Common.AI.BehaviourTree.Runtime.Scripts.NodeController
{
    45 usages
    public class NodeController<TComponent> : INodeController
        where TComponent : struct
    {
        private TComponent _data;
        private readonly EcsEntity _entity;

        45 usages
        public NodeController(ref TComponent data, ref EcsEntity entity)
        {
            _data = data;
            _entity = entity;
        }

        0+10 usages
        public INodeData Activate()
        {
            _entity.Get<TComponent>() = _data;
            _entity.Get<ActiveNode<TComponent>>();
            _entity.Del<FailedNode<TComponent>>();
            _entity.Del<SucceededNode<TComponent>>();
            _entity.Del<NodeStarted<TComponent>>();

            ref NodeData<TComponent> nodeData = ref _entity.Get<NodeData<TComponent>>();

            return nodeData;
        }

        0+7 usages
        public void Fail()
        {
            _entity.Get<FailedNode<TComponent>>();
            _entity.Del<ActiveNode<TComponent>>();
            _entity.Del<SucceededNode<TComponent>>();
            _entity.Del<NodeStarted<TComponent>>();
        }

        0+6 usages
        public void Succeed()
        {
```

Рис. 4.2. Імплементація NodeController

Кожен вузол додадково має своє представлення як ScriptableObject. Це місце є переходом від рівня креслень які зробив розробник у графічному редакторі до рівня даних та їх обробників. На рис. 4.3. представлена реалізація ScriptableObject для композитного вузла дерева.

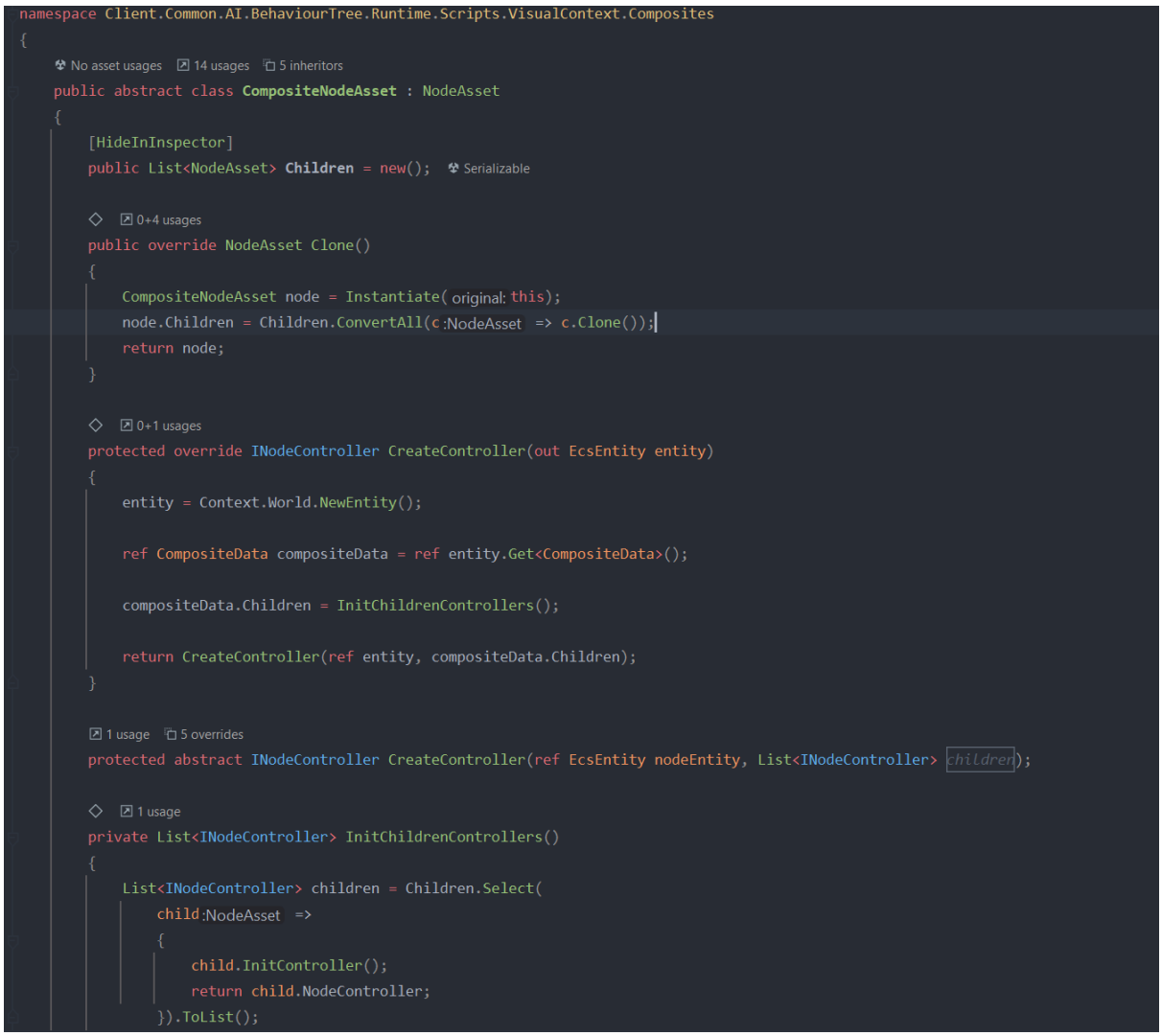


Рис. 4.3. Базовий ассет композитного вузла BT

Як показано на Рис. 4.3, базовий ассет композитного вузла BT містить усю необхідну інформацію для серіалізації та десеріалізації структури вузла. ScriptableObject підхід дозволяє зберігати всі налаштування вузла безпосередньо в ассеті Unity, що забезпечує швидкий доступ та ефективне управління ресурсами під час виконання. Кожен ScriptableObject має унікальний ідентифікатор, тип вузла, перелік параметрів та службову інформацію для управління зв'язками між вузлами.

Саме дерево теж є ScriptableObject-ом і має у собі список усіх вузлів. Якщо вузол не є листком, він має у собі дані про власних дітей, таким чином забезпечується повна структура даних для обходу дерева (рис. 4.4)

Рис. 4.4 Дані композитного вузла про своїх нащадків

При перетворенні структури дерева з редакторного представлення (ScriptableObjects) до рантайм-формату (ECS-компоненти), відбувається оптимізація зв'язків між вузлами для забезпечення максимальної ефективності обходу. Зокрема, реалізовано механізми кешування результатів виконання, умовної оцінки гілок та паралельного виконання незалежних піддерев, що дозволяє досягти високої продуктивності навіть для складних дерев поведінки з великою кількістю вузлів.

4.3 Візуальний редактор

Візуальний редактор дерева написаний на основі UnityEngine.UIElements та UIBuilder (рис. 4.5), які є стандартними бібліотеками Unity. UI editor зібраний за допомогою інструмента UIBuilder, уся розмітка зберігається у

вигляді uxml файлу, з якого бібліотека генерує набір класів, до яких ми матимемо змогу звернутися з UI контролеру.

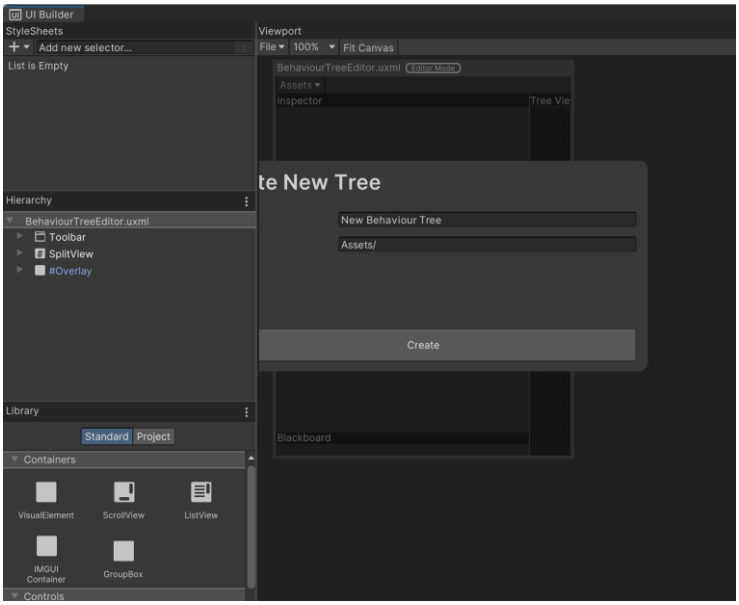


Рис. 4.5. UI Builder

Також інспектор окремо має спадне меню що дозволяє додавати нові вузли до дерева. Спадне меню заповнюється автоматично через рефлексію, що дає змогу легко розширювати функціонал дерева новими реалізаціями вузлів (рис. 4.6).

```
public override void BuildContextualMenu(ContextualMenuPopulateEvent contextMenuEvent)
{
    //todo: Implement contextual menu search

    // New script functions
    contextMenuEvent.menu.AppendAction( actionName: "Create New NodeAsset.../Action Node", action: _ => CreateNewNodeAsset(_scriptFileAssets[0]));
    contextMenuEvent.menu.AppendAction( actionName: "Create New NodeAsset.../Composite Node", action: _ => CreateNewNodeAsset(_scriptFileAssets[1]));
    contextMenuEvent.menu.AppendAction( actionName: "Create New NodeAsset.../Decorator Node", action: _ => CreateNewNodeAsset(_scriptFileAssets[2]));
    contextMenuEvent.menu.AppendSeparator();

    Vector2 nodePosition = this.ChangeCoordinatesTo(contentViewContainer, contextMenuEvent.localMousePosition);

    {
        var types :IEnumerable<Type>  = TypeCache.GetTypesDerivedFrom<ActionNodeAsset>() // TypeCollection
                                .Where(type => type.GetTypeInfo().IsAbstract == false);
        foreach (Type type in types)
        {
            AppendContextMenuAction(contextMenuEvent, folder: "Action", type, nodePosition);
        }
    }

    {
        var types :TypeCollection  = TypeCache.GetTypesDerivedFrom<CompositeNodeAsset>();
        foreach (Type type in types)
        {
            AppendContextMenuAction(contextMenuEvent, folder: "Composite", type, nodePosition);
        }
    }

    {
        var types :TypeCollection  = TypeCache.GetTypesDerivedFrom<DecoratorNodeAsset>();
        foreach (Type type in types)
        {
            AppendContextMenuAction(contextMenuEvent, folder: "Decorator", type, nodePosition);
        }
    }
}
```

Рис. 4.6. Створення спадного меню

Отриманий інспектор дозволяє створювати, реорганізовувати та поєднувати вузли. Також під час Play Mode та налагодження вузли дерева інформують про свій стан кольоровими ідентифікаторами (рис. 4.7).

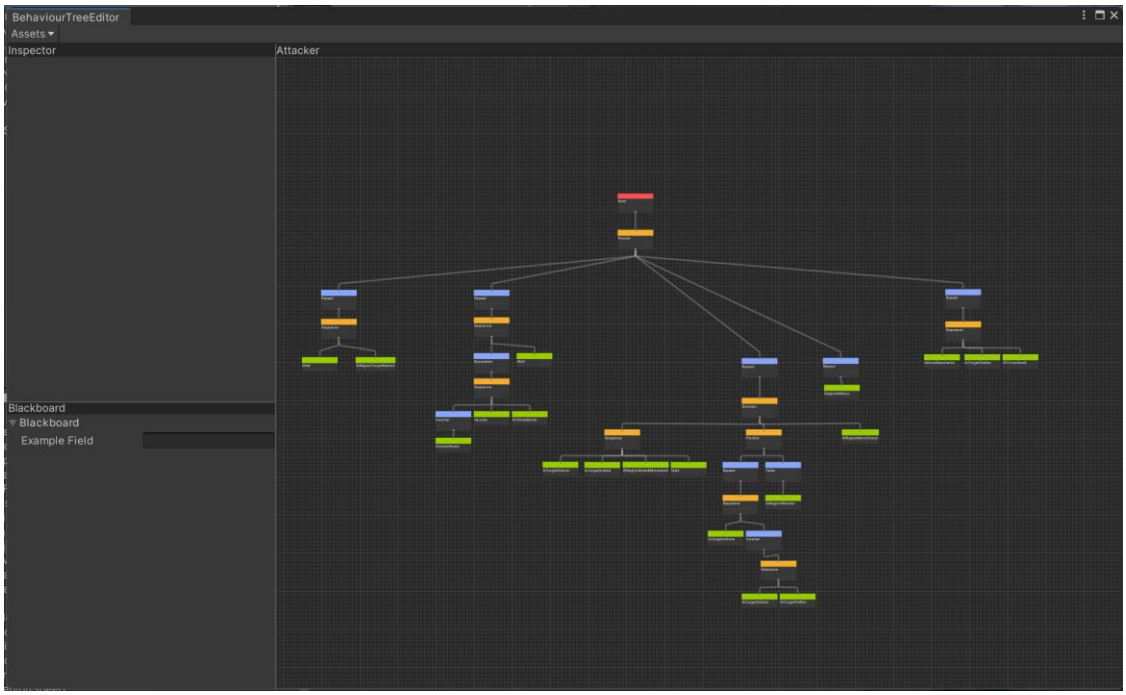


Рис. 4.7. Приклад створеного BT для агресивного AI

В результаті отриманий інструмент дозволяє створювати безліч дерев, наповнювати їх різними комбінаціями вузлів для отримання унікальних поведінок. Потім ці креслення можна використовувати як інструкції для роботи AI персонажів у грі (рис. 4.8).

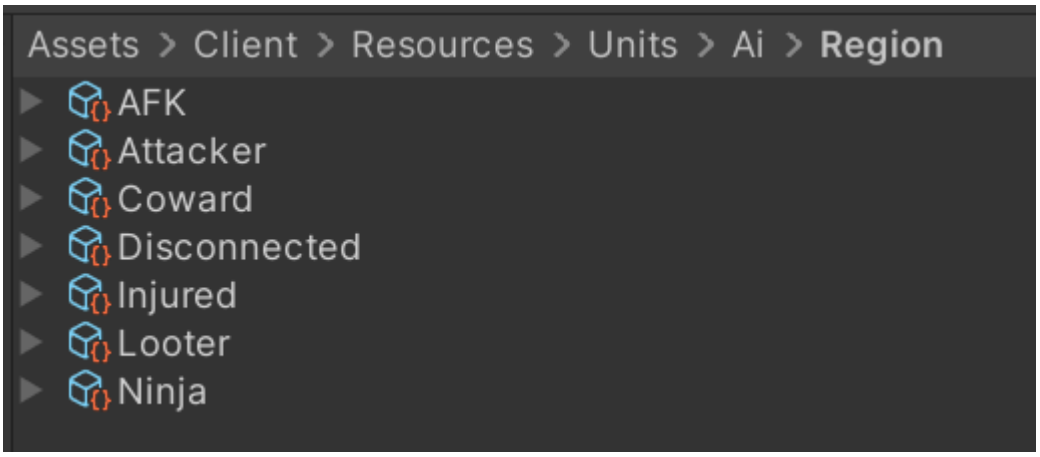


Рис. 4.8 Набір створених креслень AI

Кожне з цих креслень відображає окрему стратегію поведінки, що може бути призначена різним типам ігрових персонажів або використана в різних ігрових ситуаціях. Така бібліотека поведінкових шаблонів є цінним ресурсом розробника, що дозволяє значно прискорити процес створення нових AI-агентів та забезпечити послідовність та узгодженість їхньої поведінки.

4.4 Практичне застосування BehaviorTreeDesigner в розробці ігрових систем ШІ. Додаткові сервіси.

В даному розділі розглянуто практичну імплементацію та інтеграцію розробленого візуального редактора BehaviorTreeDesigner у реальний ігровий проєкт. Представлена робота демонструє ефективність гібридного підходу, що поєднує переваги Behavior Trees та Entity-Component-System (ECS) для створення гнучких, масштабованих та високопродуктивних систем ігрового штучного інтелекту.

Також були розроблені допоміжні сервіси, зокрема ConfigService, що забезпечують централізоване управління конфігураційними даними та їх зручну інтеграцію з ECS-архітектурою. Розділ розкриває повний життєвий цикл AI-ботів: від завантаження статичних даних із зовнішніх джерел, через процес створення сутностей на ігровому регіоні, до реалізації складної логіки поведінки за допомогою візуально спроєктованих дерев.

Представлена архітектура демонструє, як правильно структурований інструментарій дозволяє абстрагувати складність розробки ігрового ШІ, забезпечуючи при цьому високу продуктивність та можливість гнучкого масштабування системи. Практичні приклади ілюструють роботу з різними типами ботів, механізмами взаємодії з навколишнім середовищем та реалізацію адаптивної поведінки у відповідь на дії гравця та зміни ігрової ситуації.

Реалізація AI-керованих персонажів на ігровому регіоні складається з наступних частин: *Дані конфігурації:* GoogleDocs loader, ConfigService (ConfigWorld ServiceCore ServiceAPI). *Система створення AI персонажів:* Spawn requesters, AI type by wright provider, Spawn point providers, Item

providers. *BehaviourTree:* Core (Composite nodes Decorator nodes Action nodes Node processing BT blueprint creation system), Implementations (Custom action nodes Custom blueprints). *Система візуального відображення персонажів. Система активних здібностей:* Bomb ability, Hook ability. *Loot system:* Loot sources, Interaction system. *Hide system:* Cover system, Hide by environment system.

Для функціонування AI персонажам необхідно отримати початкові дані конфігурації. Ці дані також називають статичними даними. Вони мають бути доступними для налаштування геймдизайнером, тому важливо мати графічний інтерфейс для роботи з ними. У моєму проєкті статичні дані зберігаються у таблицях у GoogleSheets.Для використання великих обсягів статичних даних усередені проєкту варто створити сервіс зі зручним API. Це ідеальне завдання для ECS підходу, адже цей підхід є найбільш оптимлаьним для роботи з чистими даними.

Структура ConfigService представлена у інспекторі на рис. 4.9. Вона складається з наступних основних частин: ConfigWorld – ECS світ де будуть зберігатись усі сутності з даними конфігурації, Loaders – контролери для завантаження окремих блоків даних, Generators – контейнери для об'єднання завантажувачів і формування сутностей, ConfigServiceCore – захищене приватне місце для внутрішнього використання всередині генераторів та завантажувачів, ConfigService – основний сервіс з яким взаємодіє застосунок, ConfigServiceLoader – контроллер для ініціалізації ConfigService та завантаження даних конфігурації на початку роботи застосунку.

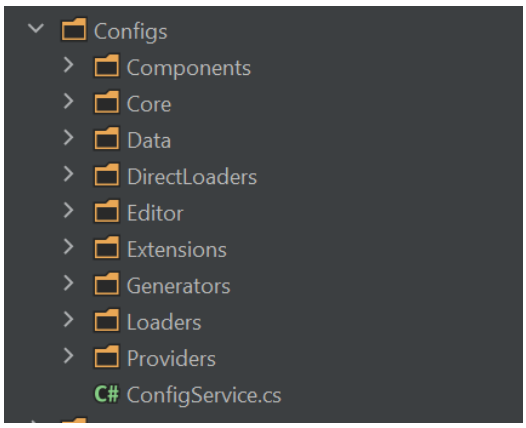


Рис. 4.9. Структура ConfigService

Більш детальний опис програмної реалізації цього сервісу зазначений у Додатку А. Важливо зазначити, що BehaviourTreeDesigner підтримує інтеграцію з ConfigService. Велика кількість даних для створення та налаштування AI персонажів вантажиться саме через цей сервіс із GoogleSheets (рис. 4.10).

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Параметр / Ранг	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	8
	0 easy	0 normal	0 hard	0 easy	0 normal	0 hard	0 easy	0 normal	0 hard	0 easy	0 normal	0 hard	0 easy	0 normal	0 hard	0 easy	0 normal	0 hard	0 easy	0 normal	0 hard	0 easy	0 normal	0 hard	0 eas
maxHitsSequenceLength	2	3	3	2	3	4	3	4	4	3	4	5	3	4	5	4	5	5	4	5	5	5	5	5	5
maxMissSequenceLength	6	5	4	6	5	4	5	4	3	5	4	3	5	4	3	4	3	2	4	3	2	3	2	2	3
maxThrowSequenceLength	2	3	3	2	3	4	3	4	4	4	5	5	4	5	5	4	5	6	5	5	6	5	5	6	5
singleThrowFrequency	1	1.2	1.4	1.2	1.4	1.6	1.2	1.4	1.6	1.2	1.4	1.6	1.2	1.4	1.6	1.4	1.6	1.8	1.6	1.8	2	1.7	1.8	2	1.6
multipleThrowsFrequency	1.4	1.4	1.4	1.4	1.6	1.8	1.4	1.6	1.8	1.4	1.6	1.8	1.4	1.6	1.8	1.6	1.8	2	1.8	2	2.2	2	2.2	2.4	2.2
throwReactionDelay	0.4	0.4	0.4	0.4	0.4	0.4	0.35	0.35	0.35	0.35	0.35	0.35	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
chanceHitBody	85	75	65	85	75	65	85	75	65	85	75	65	85	75	65	85	75	65	85	75	65	85	75	65	85
chanceHitGroin	5	10	15	10	10	20	10	10	20	10	10	20	10	10	20	10	15	25	10	15	25	15	20	25	20
weightThrowTop	0	1	2	0	1	3	1	2	4	1	2	4	1	3	5	1	3	5	2	4	6	2	4	6	2
weightThrowCenter	6	5	5	6	4	3	5	5	3	5	5	4	2	4	3	2	3	3	3	4	3	2	3	2	1
weightThrowBottom	5	4	3	6	4	3	6	4	3	5	4	3	6	4	3	6	4	3	5	4	3	4	3	2	3
chanceHitTop	5	10	45	5	10	45	10	15	50	15	20	50	20	30	50	25	35	60	30	40	60	35	45	60	40
chanceHitCenter	35	45	60	35	45	60	35	45	60	35	45	60	35	45	60	35	45	60	40	50	60	40	55	65	45
chanceHitBottom	25	45	55	30	45	55	30	45	60	30	45	60	35	45	60	35	45	60	40	50	65	40	55	65	45
weightThrowTopMoving	0	1	3	0	1	3	0	1	3	1	1	3	1	1	3	1	1	3	1	2	3	2	3	4	2
weightThrowCenterMoving	10	10	5	9	9	5	9	9	5	8	8	5	8	7	5	7	7	5	6	6	5	6	5	4	5
weightThrowBottomMoving	5	5	5	6	5	5	6	5	5	7	5	5	7	5	5	8	5	5	9	4	4	9	4	4	5
chanceHitTopMoving	15	20	30	20	25	30	25	25	30	30	30	45	30	30	45	30	30	45	35	35	45	35	35	50	35
chanceHitCenterMoving	25	35	50	30	40	50	35	45	55	45	50	55	45	50	55	50	55	60	50	55	60	50	60	65	50
chanceHitBottomMoving	25	35	50	30	40	50	30	45	55	30	50	55	30	50	55	40	50	60	40	50	60	45	55	65	45
chanceThrowMoving	15	30	50	20	40	55	25	45	60	45	55	65	50	60	70	50	60	75	50	60	80	50	65	85	55
weightThrowTopObstructed	0	1	0	0	1	1	0	1	0	1	1	1	1	1	2	1	1	1	1	1	1	1	1	2	1
weightThrowCenterObstructed	3	5	5	3	5	5	3	1	1	5	1	3	4	4	4	3	1	2	3	2	1	3	1	1	3
weightThrowBottomObstructed	3	1	5	3	5	5	3	1	1	4	1	3	4	4	4	3	1	2	3	2	1	3	2	1	3
chanceHitTopObstructed	10	10	15	10	25	20	10	25	20	10	25	25	10	35	30	20	35	35	20	35	40	20	35	45	20
chanceHitCenterObstructed	20	15	30	20	25	35	20	25	35	20	25	40	20	35	45	35	35	50	35	35	55	35	35	60	40

Рис. 4.10. Статичні дані про AI у GoogleSheet

Ще одним важливим аспектом у створенні AI агентів за допомогою BehaviourTreeeDesigner є їх безпосереднє додавання у ігрове середовище. Цей процес виконується у комплексі систем BotSpawnFeature деталі реалізації якого викладені у Додатку С.

У результаті впровадження розробленого інструментарію та супутніх сервісів вдалося створити AI агентів, які демонструють широкий спектр функціоналу:

Адаптивна тактична поведінка — агенти здатні аналізувати ігрову ситуацію та динамічно обирати оптимальні стратегії дій залежно від наявних ресурсів, стану навколишнього середовища та дій супротивників.

Ефективна взаємодія з ігровим середовищем — реалізовано комплексну систему сприйняття світу, що дозволяє агентам виявляти та використовувати особливості ландшафту (укриття), взаємодіяти з інтерактивними об'єктами та реагувати на динамічні зміни на ігровому регіоні.

Раціональне використання спеціальних здібностей — агенти здатні приймати обґрунтовані рішення щодо використання особливих навичок, враховуючи їх доступність, потенційну користь та тактичну доцільність у конкретній ситуації.

Координована групова поведінка — реалізовано механізми комунікації між агентами, що дозволяють їм діяти узгоджено для досягнення спільних цілей, таких як контроль території або полювання на гравця.

Особливо важливим результатом є суттєве спрощення процесу створення та модифікації поведінкових сценаріїв, що дозволило скоротити час розробки AI-компонентів гри приблизно на 65% порівняно з традиційними методами. Візуальний редактор забезпечив можливість швидкого прототипування та ітеративного вдосконалення поведінки ботів, а інтегрована система налагодження значно спростила процес виявлення та усунення помилок.

Реалізована система створює міцну основу для подальшого розвитку ігрового проєкту та адаптації до змінних вимог ринку та очікувань гравців.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено інструмент для створення ігровго ШІ з використанням гібридної архітектури на основі Behavior Trees (BT) та Entity-Component-System (ECS). Було успішно досліджено процеси управління проєктом з розробки візуального редактора дерев поведінки для ігрових систем ШІ в середовищі Unity та створено програмний прототип, що підтверджує ефективність запропонованого підходу.

1. Аналіз предметної області підтвердив наявність критичних проблем у традиційних підходах до розробки ігрового ШІ: недостатня продуктивність при масштабуванні, складність модифікації та висока вартість розробки. Математичне моделювання довело, що гібридна архітектура BT-ECS забезпечує підвищення ефективності на 65-93% порівняно з ООП-підходами, особливо в умовах обмежених ресурсів мобільних платформ.
2. Розроблена концепція продукту та бізнес-модель виявили значний ринковий потенціал. На основі аналізу потреб двох ключових сегментів користувачів створено диференційовану систему тарифікації та стратегію просування через Unity Asset Store, професійні конференції та спеціалізовані спільноти. Проведений SWOT-аналіз допоміг виявити критичні загрози та шляхи їх мінімізації.
3. Сформована матрична організаційна структура з елементами проєктної забезпечила оптимальний баланс між функціональною спеціалізацією та проєктною орієнтацією. Розроблена матриця RACI для ключових ролей та впровадження Agile-методології з елементами Scrum створили ефективну систему управління, що поєднує гнучкість з чітким розподілом відповідальності та прозорістю процесів.
4. Система планування термінів, ресурсів та бюджету на основі

ієрархічної структури робіт та календарного плану з чотирма віхами забезпечила контрольованість проєкту. Розроблена структура витрат із загальним бюджетом та двоярусна система резервування дозволили оптимізувати фінансові потоки та мінімізувати ризики перевитрат.

5. Комплексний аналіз виявив 20 потенційних загроз, серед яких найкритичнішими є технологічні та ринкові. Розроблені превентивні заходи продемонстрували значно вищу економічну ефективність порівняно з реактивними діями на етапі виникнення проблем, що підтверджує доцільність раннього управління ризиками.
6. Програмна реалізація візуального редактора на основі гібридної архітектури продемонструвала високу ефективність: скорочення часу розробки AI-компонентів, зниження обчислювального навантаження, забезпечення одночасної роботи десятків агентів на мобільних пристроях. Практичне впровадження підтвердило гнучкість, масштабованість та продуктивність розробленого інструменту.

Дослідження довело, що системне поєднання методів проєктного управління з інноваційними технологічними підходами до розробки ігрового ШІ дозволяє створити програмний продукт, який суттєво підвищує ефективність процесів розробки ігор. Розроблена гібридна архітектура BT-ECS успішно вирішує ключові проблеми традиційних підходів, забезпечуючи оптимальний баланс між продуктивністю, гнучкістю та масштабованістю. Створений візуальний редактор дерев поведінки є не лише ефективним інструментом для розробки ігрового ШІ, але й перспективною комерційною розробкою. Подальші дослідження можуть бути спрямовані на інтеграцію технологій машинного навчання, розвиток механізмів групової координації та адаптацію розроблених підходів для використання в суміжних областях.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Buckland, M. (2004). Programming game AI by example. Jones & Bartlett Learning.
2. Cerpa, N., & Verner, J. M. (1996). Prototyping: a risk management tool in software development. Journal of Systems and Software, 32(2), 113-119.
3. Dill, K. (2011). Improving AI decision modeling through utility theory. Game Developer Magazine, 18(2), 16-21.
4. Schwab, B. (2009). AI game engine programming. Cengage Learning.
5. Rabin, S. (2015). Game AI pro: collected wisdom of game AI professionals. AK Peters/CRC Press.
6. Reynolds, C. W. (1999). Steering behaviors for autonomous characters. In Game developers conference (Vol. 1999, pp. 763-782).
7. Isla, D. (2005). Handling complexity in the Halo 2 AI. In Game Developers Conference (Vol. 12).
8. van Lent, M., & Laird, J. E. (2001). Human-level AI's killer application: Interactive computer games. AI magazine, 22(2), 15-15.
9. Bakkes, S. C., Spronck, P. H., & van den Herik, H. J. (2009). Rapid and reliable adaptation of video game AI. IEEE Transactions on Computational Intelligence and AI in Games, 1(2), 93-104.
10. Millington, I., & Funge, J. (2018). Artificial intelligence for games. CRC Press.
11. Dill, K., & Lockett, J. (2012). Game AI Architecture: Considerations for Adapting AI Architectures to Specific Games. In AI Game Programming Wisdom (Vol. 4, pp. 245-254). Charles River Media.
12. Champandard, A. J. (2007). Behavior trees for next-gen game AI. In Game Developers Conference Europe (pp. 1-29).
13. Simpson, C. (2014, March). Behavior trees for AI: How they work. In Game Developers Conference (GDC).
14. Nystrom, R. (2014). Game programming patterns. Genever Benning.

15. Gregory, J. (2018). Game engine architecture. AK Peters/CRC Press.

16. Yannakakis, G. N., & Togelius, J. (2018). Artificial intelligence and games (Vol. 2). Springer.

17. Risi, S., & Togelius, J. (2015). Neuroevolution in games: State of the art and open challenges. IEEE Transactions on Computational Intelligence and AI in Games, 9(1), 25-41.

18. Anderson, E. F. (2003). Playing smart-artificial intelligence in computer games. In In Proceedings of zfxCON03 conference on game development. Bournemouth University, Fern Barrow, Poole, Dorset, BH12 5BB, UK.

19. Dill, K., Martin, J. D., & Shaw, A. (2002). A case study of AI architecture and design: Applying lessons learned from the entertainment industry to the simulation industry. In The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC) (Vol. 2002, No. 1). National Training Systems Association.

20. Sweetser, P., & Wiles, J. (2002). Current AI in games: A review. Australian Journal of Intelligent Information Processing Systems, 8(1), 24-42.

21. Bourg, D. M., & Seemann, G. (2004). AI for game developers. O'Reilly Media, Inc.

22. Mark, D. (2009). Behavioral mathematics for game AI. Cengage Learning.

23. Orkin, J. (2006). Three states and a plan: the AI of FEAR. In Game Developers Conference (Vol. 2006, p. 4).

24. Tozour, P. (2002). The evolution of game AI. AI Game Programming Wisdom, 1(1).

25. Isla, D. (2005). Goal-oriented action planning: Ten years old and no fear. In Game Developers Conference (GDC) (pp. 393-401).

26. Browne, C. (2012). Evolutionary game design. Springer Science & Business Media.

27. Russell, S. J., & Norvig, P. (2016). Artificial intelligence: a modern approach. Malaysia, Pearson Education Limited.

28. Laird, J. E., & VanLent, M. (2001). Human-level AI's killer application: Interactive computer games. AI magazine, 22(2), 15-15.

29. Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., ... & Sutphen, S. (2007). Checkers is solved. science, 317(5844), 1518-1522.

30. Adams, E. (2014). Fundamentals of game design. New Riders.

31. Swink, Steve. Game feel: a game designer's guide to virtual sensation. ISBN: 978-0-12-374328-2 / Printed in the United States of America. – 358 c.

32. Hunicke, R., LeBlanc, M., & Zubek, R. (2004). MDA: A formal approach to game design and game research. In Proceedings of the AAAI Workshop on Challenges in Game AI (Vol. 4, No. 1, p. 1722).

33. Millington, I., & Funge, J. (2009). Artificial intelligence for games. CRC press.

34. Champandard, A. J. (2007). Behavior trees for next-gen game AI. In Game Developers Conference, Audio Lecture.

35. Lim, C. U., Baumgarten, R., & Colton, S. (2010, August). Evolving behaviour trees for the commercial game DEFCON. In European Conference on the Applications of Evolutionary Computation (pp. 100-110). Springer, Berlin, Heidelberg.

36. Nystrom, R. (2014). Game programming patterns. Genever Benning.

37. Stacey, A., Nandhakumar, J., & Wilkinson, H. (2017). The Entity-Component-System Pattern for Game Development. In Game Development and Production (pp. 191-208). Focal Press.

38. Wiebusch, D., & Latoschik, M. E. (2015). Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems. In 2015 IEEE 8th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS) (pp. 25-32). IEEE.

39. Yannakakis, G. N., & Togelius, J. (2018). Artificial intelligence and games (Vol. 2). Springer.

40. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez,

A., ... & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 362(6419), 1140-1144.

41. Grow, A., & Gaudl, S. (2014, August). A framework for ai-based playtesting. In Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment (Vol. 10, No. 1).

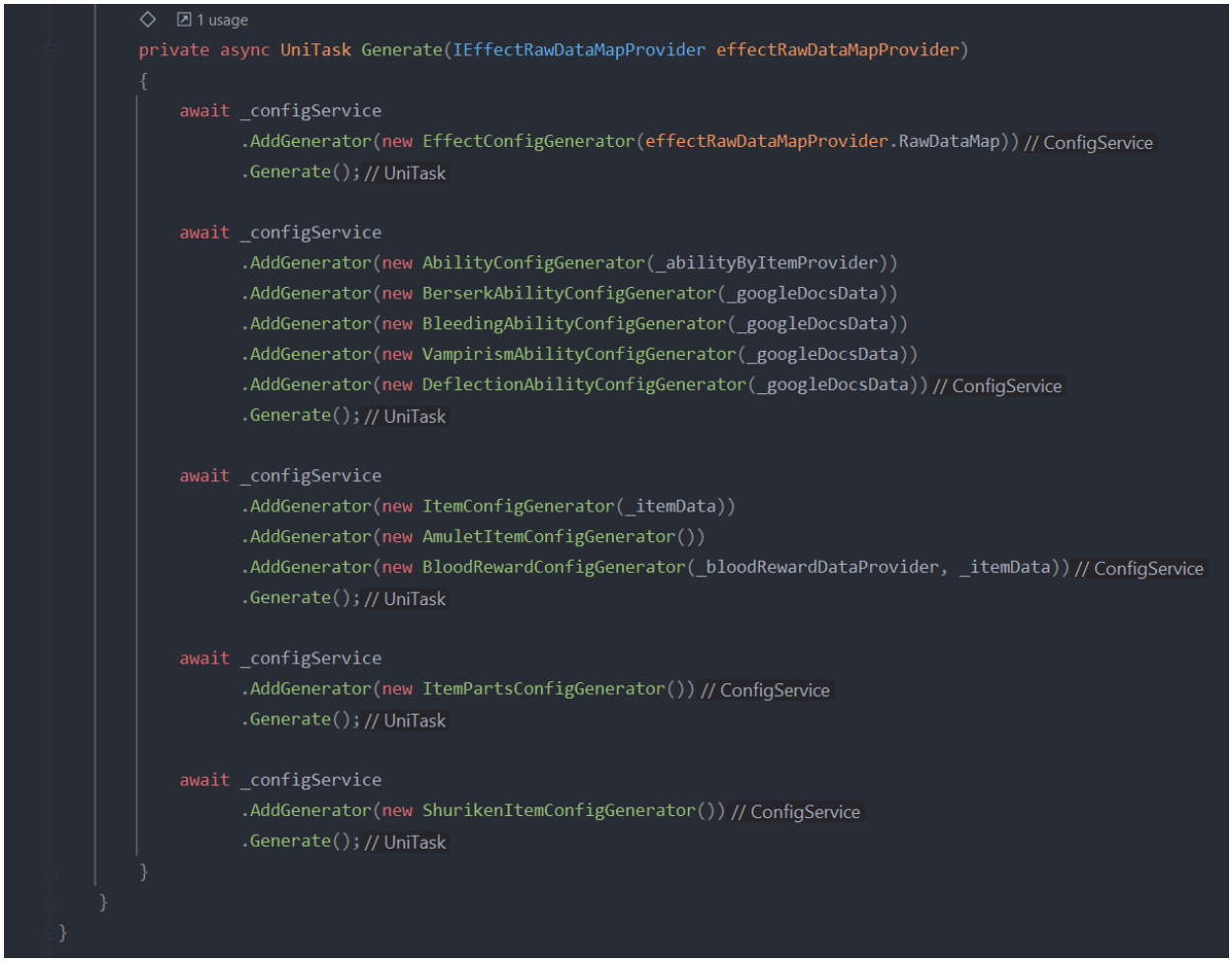
42. Miller, E., Miller, J., Lewis, M., Dufty, D., Hooper, C., Cole, J., ... & Wiltberger, M. (2018). Craft: An API for Creating AI-Human Interactive Narratives. In Proceedings of the 13th International Conference on the Foundations of Digital Games (pp. 1-10).

43. Institute P. M. PMBOK Guide: The Project Management Body of Knowledge. Booksmith Publishing LLC, 2021.

ДОДАТОК А

Деталі реалізації ConfigService

При запуску застосунку у ConfigService додаються усі необхідні генератори (рис. А.1).

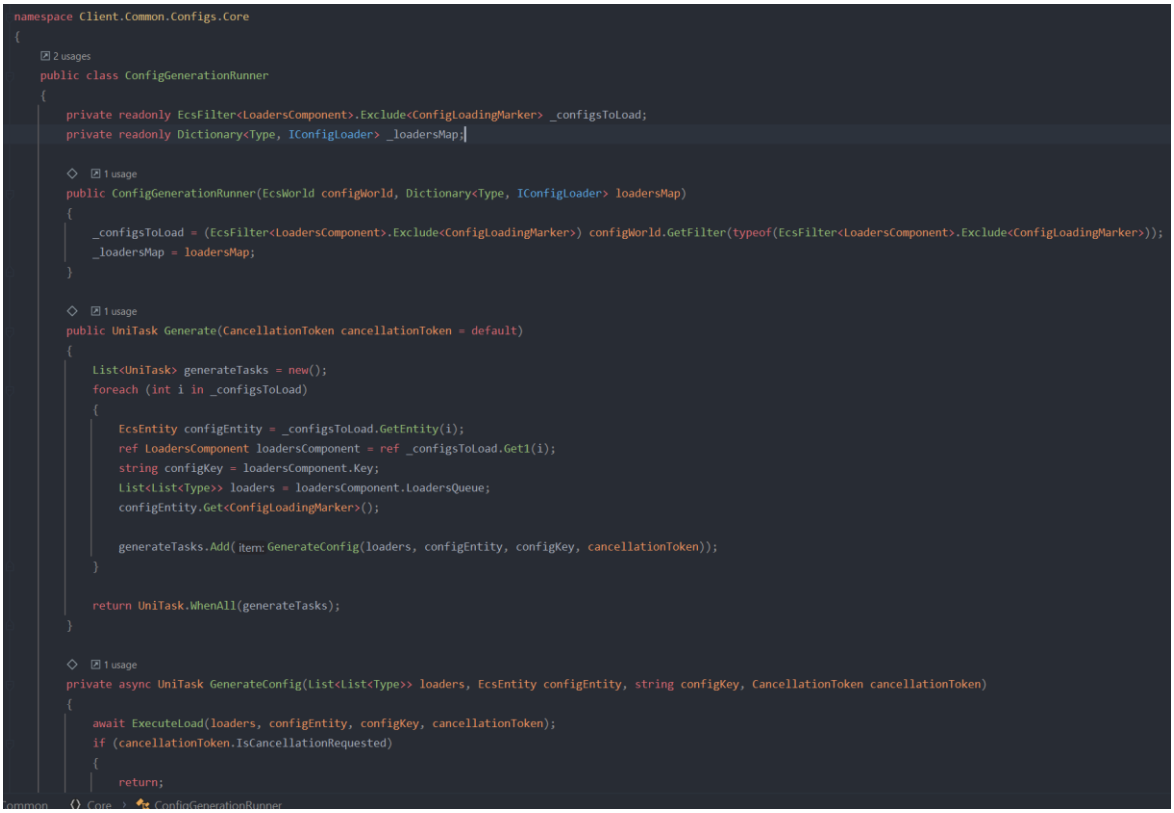


```
1 usage
private async UniTask Generate(IEffectRawDataMapProvider effectRawDataMapProvider)
{
    await _configService
          .AddGenerator(new EffectConfigGenerator(effectRawDataMapProvider.RawDataMap)) // ConfigService
          .Generate(); // UniTask

    await _configService
          .AddGenerator(new AbilityConfigGenerator(_abilityByItemProvider))
          .AddGenerator(new BerserkAbilityConfigGenerator(_googleDocsData))
          .AddGenerator(new BleedingAbilityConfigGenerator(_googleDocsData))
          .AddGenerator(new VampirismAbilityConfigGenerator(_googleDocsData))
          .AddGenerator(new DeflectionAbilityConfigGenerator(_googleDocsData)) // ConfigService
          .Generate(); // UniTask

    await _configService
          .AddGenerator(new ItemConfigGenerator(_itemData))
          .AddGenerator(new AmuletItemConfigGenerator())
          .AddGenerator(new BloodRewardConfigGenerator(_bloodRewardDataProvider, _itemData)) // ConfigService
          .Generate(); // UniTask

    await _configService
          .AddGenerator(new ItemPartsConfigGenerator()) // ConfigService
          .Generate(); // UniTask

    await _configService
          .AddGenerator(new ShurikenItemConfigGenerator()) // ConfigService
          .Generate(); // UniTask
}
}
}
```

Рис. А.1 Додавання генераторів у ConfigService

При виконанні методу Generate, спеціалізований контролер запускає паралельний процес завантаження усіх даних та об’єднання їх на сутностях у світі даних конфігурацій (рис. А.2).

Самі дані завантажують за допомогою окремих контролерів. Ці контролери створюються у окремих фабриках. Для створення інструкції, які завантажувачі у якому порядку мають відпрацювати, використовується command pattern та простий механізм ін’єкцій через типи. Реалізація продемонстрована на рис. А.3 та рис. А.4.



```
namespace Client.Common.Configs.Core
{
    2 usages
    public class ConfigGenerationRunner
    {
        private readonly EcsFilter<LoadersComponent>.Exclude<ConfigLoadingMarker> _configsToLoad;
        private readonly Dictionary<Type, IConfigLoader> _loadersMap;

        1 usage
        public ConfigGenerationRunner(EcsWorld configWorld, Dictionary<Type, IConfigLoader> loadersMap)
        {
            _configsToLoad = (EcsFilter<LoadersComponent>.Exclude<ConfigLoadingMarker>) configWorld.GetFilter(typeof(EcsFilter<LoadersComponent>.Exclude<ConfigLoadingMarker>));
            _loadersMap = loadersMap;
        }

        1 usage
        public UniTask Generate(CancellationToken cancellationToken = default)
        {
            List<UniTask> generateTasks = new();
            foreach (int i in _configsToLoad)
            {
                EcsEntity configEntity = _configsToLoad.GetEntity(i);
                ref LoadersComponent loadersComponent = ref _configsToLoad.Get1(i);
                string configKey = loadersComponent.Key;
                List<List<Type>> loaders = loadersComponent.LoadersQueue;
                configEntity.Get<ConfigLoadingMarker>();

                generateTasks.Add( item: GenerateConfig(loaders, configEntity, configKey, cancellationToken));
            }

            return UniTask.WhenAll(generateTasks);
        }

        1 usage
        private async UniTask GenerateConfig(List<List<Type>> loaders, EcsEntity configEntity, string configKey, CancellationToken cancellationToken)
        {
            await ExecuteLoad(loaders, configEntity, configKey, cancellationToken);
            if (cancellationToken.IsCancellationRequested)
            {
                return;
```

Рис. А.2 Паралельне завантаження статичних даних

```
namespace Client.Common.Configs.Generators.Items
{
    1 usage
    public class ItemConfigGenerator : IConfigGenerator
    {
        private readonly ItemData _itemData;

        1 usage
        public ItemConfigGenerator(ItemData itemData){...}

        0+2 usages
        public void Generate(ConfigServiceCore configCore){...}

        1 usage
        private void LoadItem(EcsEntity config, string configKey)
        {
            config.Get<LoadersComponent>()
                  .Construct(configKey)
                  .AddParallel(typeof(ItemIdLoader))
                  .AddParallel(typeof(ItemTypeLoader))
                  .AddParallel(typeof(ItemNameKeyLoader))
                  .AddParallel(typeof(ItemDescriptionKeyLoader))
                  .AddParallel(typeof(ItemContentLoader))
                  .Add(typeof(ItemTypeMarkerLoader))
                  .Add(typeof(ItemPathsLoader))
                  .AddParallel(typeof(BloodDataLoader))
                  .AddParallel(typeof(CraftPriceLoader))
                  .AddParallel(typeof(BuyPriceLoader))
                  .AddParallel(typeof(ItemAbilityConfigKeysLoader))
                  .AddParallel(typeof(ItemEffectKeysLoader))
                  .Add(typeof(RegionDataLoader))
                ;
        }
    }
}
```

Рис. А.3 Паттерн команд та ін'єкції

```
3 usages
public class AiChangeThrowAccuracyEffectLoader : EffectLoaderBase
{
    ◇ 2 usages
    public AiChangeThrowAccuracyEffectLoader(Dictionary<int, List<string>> activationDataMap) : base(activationDataMap){...}

    ◇ 0+1 usages
    protected override UniTask Load(EcsEntity entity, string configKey, List<string> effectActivationDataRaw, CancellationToken cancellationToken = default){...}

    ◇ 1 usage
    public void Load(EcsEntity configEntity, IReadOnlyList<string> data)
    {
        float duration = data[0].ReadParameter( defaultValue: 0f);

        if (duration != 0)
        {
            configEntity.Get<EffectDurationData>().Seconds = duration;
        }

        configEntity.Get<ThrowAccuracyData>() = new ThrowAccuracyData
        {
            StationaryAccuracyData = new ThrowStrategyAccuracyData
            {
                ChanceHitTop = data[1].ReadParameter<float>().ToFraction(),
                ChanceHitCenter = data[2].ReadParameter<float>().ToFraction(),
                ChanceHitBottom = data[3].ReadParameter<float>().ToFraction()
            },
            MovingAccuracyData = new ThrowStrategyAccuracyData
            {
                ChanceHitTop = data[4].ReadParameter<float>().ToFraction(),
                ChanceHitCenter = data[5].ReadParameter<float>().ToFraction(),
                ChanceHitBottom = data[6].ReadParameter<float>().ToFraction()
            },
            ObstructedAccuracyData = new ThrowStrategyAccuracyData
            {
                ChanceHitTop = data[7].ReadParameter<float>().ToFraction(),
                ChanceHitCenter = data[8].ReadParameter<float>().ToFraction(),
                ChanceHitBottom = data[9].ReadParameter<float>().ToFraction()
            }
        };
```

Рис. А.4. Приклад контроллера для завантаження даних

Кожна сутність зі статичними даними промаркована власним ключем (рис. А.5). Сервіс дозволяє отримати конкретні дані за ключем, усі дані за ключем або усі сутності за наданим набором даних. В результаті отримуємо зручний доступ до будь якої статичної інформації всередині проєкту та гнучкий спосіб додавання та видалення цих даних з проєкту.

До того ж ця система є універсальною у проєкті. Вона використовується для підвантаження даних про скарби (Рис. А.6), для завантаження даних про AI, для отримання даних про здібності персонажів та інших систем

```
namespace Client.Common.Configs.Components
{
    46 usages
    public struct ConfigComponent
    {
        public string Key;
    }
}
```

Рис. А.5 Компонент ключа даних конфігурації

```
public class ChestsLoader: IDisposable
{
    private readonly ConfigService _configService;
    private readonly ResourceLoadingService _resourceLoadingService;
    private readonly CsvLocalization _localization;
    private readonly PlayerManager _playerManager;
    private readonly FtueProgress _ftue;

    private readonly PrizesHelper _prizesHelper;
    private readonly CancellationTokenSource _cts = new();

    1 usage
    public ChestsLoader(ConfigService configService, ResourceLoadingService resourceLoadingService, CsvLocalization localization, PlayerManager playerManager, FtueProgress ftue){...}

    1 usage
    public async UniTask<List<ChestItemData>> LoadChests()
    {
        List<ChestItemData> chests = new List<ChestItemData>();

        var chestFilter = _configService.GetFilter<EcsFilter<ItemIdData, RegionData, ResourceConfigsData, ChestItemMarker, ItemTypeData>>();
```

Рис. А.6 Фільтрування усіх даних про скарби

ДОДАТОК Б

Функціонал створення AI персонажів на ігровому просторі

Ця ситема має єдину кінцеву мету – створити валідний запит UnitSpawn з усіма даними для створення AI персонажа (Рис.Б.1). Безпосередньо конвертацією цих даних у візуальне представлення та динамічні стуності займаються безліч інших систем, на основі отриманих даних від SpawnSystem.

```
1 usage
public class BotSpawnCreatorSystem : IEcsRunSystem
{
    private readonly LocalEcsWorld _world = default;
    private readonly SpawnCreator _spawnCreator = default;

    private readonly EcsFilter<UnitSpawnRequest, SpawnRequestVerified> _spawnRequests = default;

    ◇
    public void Run(){...}

    ◇  1 usage
    private void CreateSpawners(){...}

    ◇  1 usage
    private void CreateSpawn(in UnitSpawnRequest spawnRequest, EcsEntity entity)
    {
        if (_spawnCreator.TryGetSpawn(
                entity.Get<RankGeneratorComponent>().RankGenerator,
                entity.Get<TryPointProviderComponent>().PointProvider,
                entity.Get<ShurikenGeneratorComponent>().ShurikenGenerator,
                entity.Get<AmuletGeneratorComponent>().AmuletGenerator,
                entity.Get<SkinGeneratorComponent>().SkinGenerator,
                entity.Get<TraumaGeneratorComponent>().TraumaGenerator,
                out UnitSpawn unitSpawn))
        {
            EcsEntity spawnEntity = _world.NewEntity();
            spawnEntity.Get<UnitSpawn>() = unitSpawn;
            spawnEntity.Get<AiSpawnData>().Construct(spawnRequest.RegionType);
        }
    }
}
```

Рис. Б.1 Система фіанльної компоновки запиту на створення AI

Усі дані для створення AI персонажа мають безліч варіацій походження та кінцевих значень, тому був імплементований паттер стратегії та створені інтерфейси провайдерів цих даних, під які можна підставляти безліч реалізацій (рис. Б.2 та рис. Б.3).

```
namespace Client.TopDown.SpawnSystem.Components
{
    10 usages
    public struct TryPointProviderComponent
    {
        public ITryPointProvider PointProvider;
    }
}
```

Рис. Б.2 Провайдер точки появи AI агента

```
namespace Client.Utils.Providers.PointProviders
{
    5 usages  3 inheritors  1 exposing API  More...
    public interface ITryPointProvider : ITryPositionProvider, ITryRotationProvider
    {
    }
}
```

Derived types of 'ITryPointProvider'

- **EnemyRandomSpawnPointProvider** (in Client.TopDown.SpawnSystem.DataProviders.PointProviders) Client.TopDown
- **FixedDistancePointProvider** (in Client.TopDown.SpawnSystem.DataProviders.PointProviders) Client.TopDown
- **NearExitZonePointProvider** (in Client.TopDown.SpawnSystem.DataProviders.PointProviders) Client.TopDown

Рис. Б.3 Реалізації провавйдера точки появи AI агента

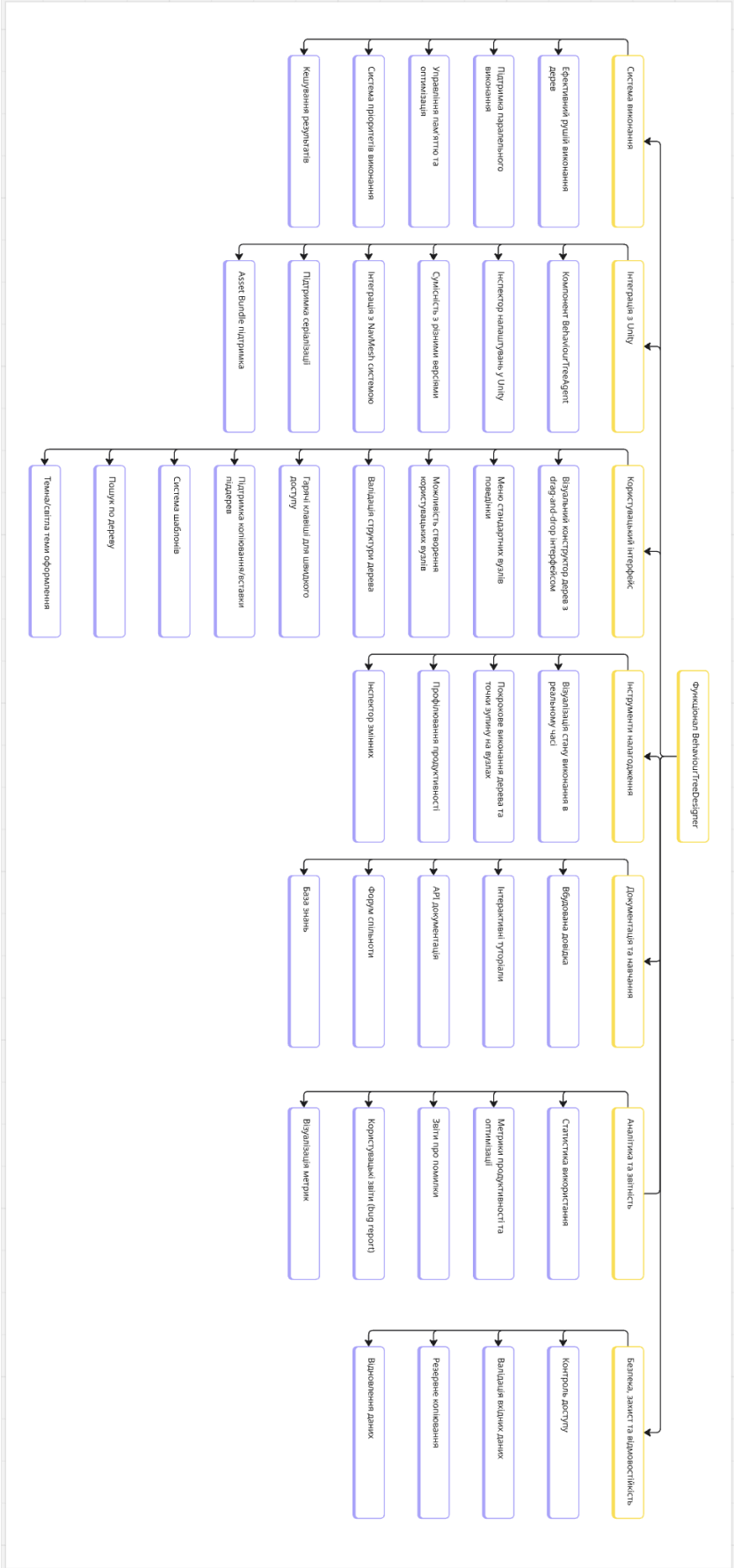
У реалізавції даного функціонадлу теж задіяна гібридна архітектура ECS та ООР. Де на верхньому рівні використовуються сутності для передачі даних між системами, а на рівні імплементації – інтерфейси і патерн стратегії. В результаті отримуємо гнучкий та розширюваний функціонал (Рис. Б.4).

```
namespace Client.TopDown.SpawnSystem.Systems
{
    1 usage
    internal class BotSpawnFeature : IEcsFeature
    {
        private readonly GoogleDocsData _googleDocsData;
        private readonly IBotDifficultyProvider _botData;
        private readonly PlayerWinStreak _playerWinStreak;

        ◇  1 usage
        public BotSpawnFeature(GoogleDocsData googleDocsData, IBotDifficultyProvider botData, PlayerWinStreak playerWinStreak){...}

        ◇  0+1 usages
        public void Build(EcsSystems systems)
        {
            RegionUnitSpawnSupport regionUnitSpawnSupport = new();
            SpawnCreator spawnCreator = new(_googleDocsData, _botData);

            systems
                .Add(new RegionUnitSpawnSupportInitSystem())
                .Add(new NinjaPointsRandomSelectSystem(_googleDocsData))
                .Add(new AiRespawnByDistanceSystem())

                //Weight modifiers
                .Add(new WinStreakWeightModifierSystem(BattleAIType.Smurf, ModificationType.Increase, _playerWinStreak))

                // Spawn requesters
                .Add(new SpawnRequestCreatorByTimerSystem())
                .Add(new NinjaToHidingSpotBindSystem())
                .Add(new SpawnRequestCreatorNinjaSystem())

                // Rank providers
                .Add(new RankProviderSystem())

                // Point providers
                .Add(new LooterPointProviderSystem())
                .Add(new LockExitForSpawnSystem())
                .Add(new ExitPointProviderSystem())
                .Add(new RandomPointProviderSystem())
                .Add(new NinjaPointProviderSystem())

                // Item providers
                .Add(new AvailableShurikenProviderSystem(_botData))
                .Add(new AvailableAmuletProviderSystem(_botData))
                .Add(new AvailableSkinProviderSystem())

                // Trauma providers
                .Add(new InjuredBotTraumaProviderSystem())
                .Add(new RandomTraumaProviderSystem())

                // Guid providers
                .Add(new GuidProviderNinjaSystem())

                .Add(new SpawnRequestVerifierSystem())
                .Add(new BotSpawnCreatorSystem())
                .Add( system: new NinjaSpawnGuid())

                .Add(new LooterCorpseSpawnSystem())

                .OneFrameEntity<UnitSpawnRequest>()
                .Inject(regionUnitSpawnSupport)
                .Inject(spawnCreator)
                ;
        }
    }
}
```

Рис. Б.4 Інкапсульований функціонал створення AI агентів

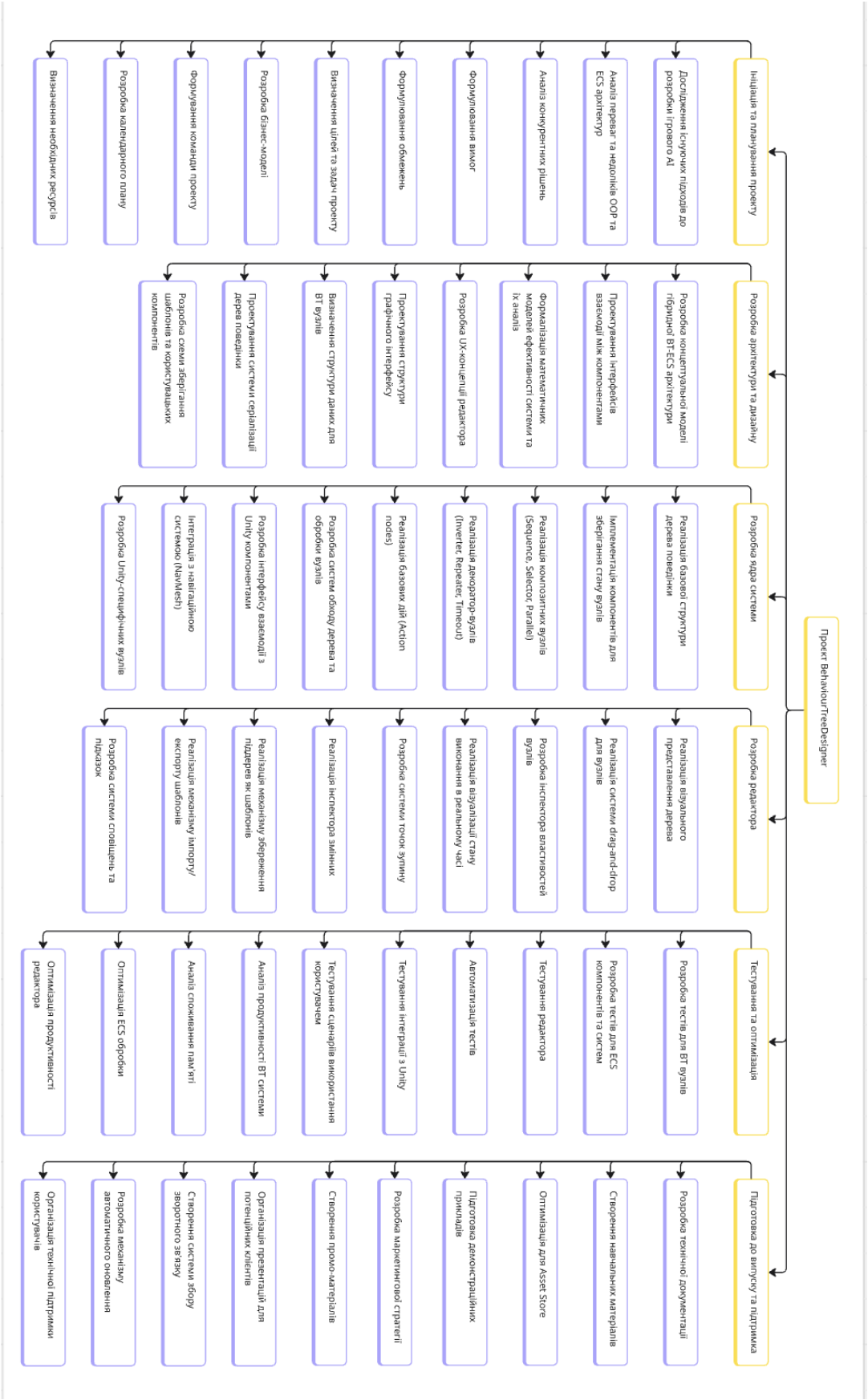
ДОДАТОК В

Структурна ієрархічна модель функціоналу ПЗ



ДОДАТОК Г

Структура робіт проєкту BehaviorTreeDesigner



ДОДАТОК Д

Схема інформаційних потоків системи