МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПОЛІСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій,
обліку та фінансів
Кафедра комп'ютерних технологій
і моделювання систем

Кваліфікаційна робота
на правах рукопису

Демченко Юрій Михайлович

(прізвище, ім'я, по батькові здобувача освіти)

УДК 004.8:004.94

КВАЛІФІКАЦІЙНА РОБОТА

«Інформаційна технологія симуляції визначення пріоритетів життя»

(тема роботи)

122 «Комп'ютерні науки»

(шифр і назва спеціальності)

Подається на здобуття освітнього ступеня бакалавр

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

_____

(підпис, ініціали та прізвище здобувача вищої освіти)

Керівник роботи

Николюк Ольга Миколаївна

(прізвище, ім'я, по батькові)

доктор економічних наук, професор

(науковий ступінь, вчене звання)

Житомир – 2026

Висновок кафедри комп'ютерних технологій і моделювання систем

за результатами попереднього захисту: допущений(-на)

Протокол засідання кафедри комп'ютерних технологій і моделювання систем

№ 20 від «05» червня 2026 р.

Завідувач кафедри комп'ютерних технологій і моделювання систем

к.пед.н.,доцент _____ Ковальчук М.О.

(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)

«05» червня 2026 р.

Результати захисту кваліфікаційної роботи

Здобувач вищої освіти Демченко Юрій Михайлович захистив

(прізвище ,ім'я, по батькові)

кваліфікаційну роботу з оцінкою:

сума балів за 100-бальною шкалою _____

за шкалою ECTS _____

за національною шкалою _____

Секретар ЕК

лаборант кафедри _____ Корольчук В. В.

(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)

АНОТАЦІЯ

Демченко Ю. М. Інформаційна технологія симуляції визначення пріоритетів життя. - Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 122 - Комп’ютерні науки. - Поліський національний університет, Житомир, 2026.

Об’єкт дослідження - процеси моделювання тайм-менеджменту та управління балансом між кар'єрою, здоров'ям та особистим життям у формі ігрової симуляції . Мета роботи – програмна реалізація технології симуляції життєвих пріоритетів, створення інтерактивного середовища для дослідження матриці Ейзенхауера . Методи дослідження – об'єктно-орієнтоване проєктування, архітектурний підхід Clean Architecture, алгоритмічне моделювання систем штучного інтелекту.

У кваліфікаційній роботі реалізовано мобільний додаток (2D Sandbox) для платформи Android . Розроблено архітектуру, що базується на строгому розділенні на три незалежні шари (Domain, Data, Presentation) . Реалізовано ігровий цикл, що складається з фаз утреннього планування, денної симуляції та нічного відпочинку. Інтегровано систему відеолупів на базі штучного інтелекту LTX 2.3 з оптимізацією завантаження через Addressables.

Ключові слова: Clean Architecture, інформаційна система, штучний інтелект LTX 2.3, Addressables.

SUMMARY

Demchenko Yu. M. Software implementation of a life priorities simulator for mobile platforms. - Qualification paper in manuscript form.

Qualification paper for obtaining the bachelor's degree in specialty 122 - Computer Science. - Polissia National University, Zhytomyr, 2026.

Object of the study - processes of modeling time management and managing the balance between career, health, and personal life in the form of a game simulation. Aim of the study – software implementation of the life priorities simulation technology, creation of an interactive environment for exploring the Eisenhower matrix. Methods of research – object-oriented design, Clean Architecture approach, algorithmic modeling of artificial intelligence systems.

The qualification paper implements a mobile application (2D Sandbox) for the Android platform. An architecture based on a strict division into three independent layers (Domain, Data, Presentation) has been developed. A game loop consisting of morning planning, daytime simulation, and night rest phases has been implemented. A system of video loops based on LTX 2.3 artificial intelligence with loading optimization via Addressables has been integrated.

Keywords: Clean Architecture, information system, artificial intelligence LTX 2.3, Addressables.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

ШІ – штучний інтелект

API – Application Programming Interface (програмний інтерфейс додатка)

GDD – Game Design Document (документ гейм-дизайну / технічне завдання гри)

GUI – Graphical User Interface (графічний інтерфейс користувача)

HUD – Heads-Up Display (частина візуального інтерфейсу, що постійно відображається на екрані під час гри)

IDE – Integrated Development Environment (інтегроване середовище розробки)

MVC – Model-View-Controller (архітектурний шаблон проєктування «Модель-Вигляд-Контролер»)

SO – ScriptableObject (серіалізований контейнер даних у рушії Unity, що використовується для збереження конфігурацій)

SRS – Software Requirements Specification (специфікація вимог до програмного забезпечення)

UI – User Interface (інтерфейс користувача)

UX – User Experience (досвід користувача)

Q1–Q4 – Quadrants (квадранти матриці Ейзенхауера, що визначають рівень важливості та терміновості завдань)

ВСТУП

Сучасний темп розвитку цифрового суспільства та підвищення індивідуального психоемоційного навантаження вимагають високого рівня інструментальної підтримки процесів особистої ефективності. Оптимізація розподілу часових ресурсів та управління балансом між професійною діяльністю, здоров'ям, соціалізацією та внутрішнім світом є критично важливим завданням для запобігання емоційному вигоранню та апатії. Одним із найбільш дієвих методологічних інструментів тайм-менеджменту є Матриця Ейзенхауера, яка дозволяє пріоритезувати завдання за критеріями важливості та терміновості. Проблема високого рівня стресу, відсутності оперативного самоконтролю та складності утримання життєвого балансу в умовах сучасного динамічного середовища зумовлює необхідність впровадження спеціалізованих інтерактивних систем моделювання рішень.

Використання сучасних технологій ігрових рушіїв та методів симуляції в реальному часі дозволяє здійснювати наочний моніторинг наслідків щоденного вибору користувача. Проте існуючі рішення в індустрії життєвих симуляторів часто зосереджені на спрощених економічних аспектах або мають надлишково складну архітектуру, що не враховує дуальної природи рішень та їхнього прямого впливу на фізичний і ментальний стан людини. Розробка власного мобільного ігрового додатка для платформи Android, побудованого на базі чистих архітектурних підходів без зайвого ускладнення логіки, дозволить забезпечити точне моделювання життєвого циклу користувача, підвищити наочність розподілу пріоритетів та створити зручний аналітичний інструментарій для дослідження стратегій успіху та виживання. Це підтверджує актуальність обраної теми кваліфікаційної роботи.

Метою роботи є розробка та програмна реалізація мобільного ігрового додатка для автоматизації симуляції життєвих пріоритетів та дослідження стратегій планування на базі Матриці Ейзенхауера.

Для досягнення мети було поставлено такі завдання:

1. Проаналізувати предметну область, теоретичні засади тайм-менеджменту та існуючі програмні рішення у сфері інтерактивних життєвих симуляторів.

2. Обґрунтувати вибір методів, моделей та інструментальних засобів для розробки кросплатформного мобільного додатка.

3. Спроектувати архітектуру інформаційної системи на засадах чистих архітектурних підходів, включаючи структуру даних конфігурацій та інтерфейси взаємодії її шарів.

4. Розробити програмне забезпечення системи для збору статистики планування, автоматичної генерації щоденних маршрутів, обробки випадкових івентів та динамічного відтворення медіаконтенту.

5. Провести тестування розробленого ПЗ та оцінити ефективність і стабільність його функціонування на цільовій мобільній платформі.

Об’єкт дослідження – процес розробки програмного забезпечення системи тайм-менеджменту та управління життєвим балансом у формі інтерактивної симуляції.

Предмет дослідженн**я** – методи, архітектурні патерни та програмні засоби розробки системи життєвих пріоритетів у мобільному середовищі.

Для вирішення поставлених завдань у роботі використано такі методи дослідження:системний аналіз - для дослідження структури взаємозв'язку життєвих сфер та формування функціональних вимог до системи; *об’єктно-орієнтоване моделювання та проектування за допомогою мови UML* - для візуалізації, проектування та побудови гнучкої архітектури програмного забезпечення; *архітектурний розподіл за принципами Clean Architecture* - для чіткого відокремлення бізнес-логіки від технологічного рушія та систем конфігурації;*методи ігрового програмування та мобільної розробки* - для реалізації Core Loop, алгоритмів штучного інтелекту, систем асинхронного завантаження Addressables та оптимізації відтворення відеопотоків на базі ШІ (LTX 2.3); *методи тестування ПЗ* - для інструментальної перевірки надійності, відмовостійкості та швидкодії розробленого додатка.

Основні результати дослідження були висвітлені у:

Демченко Ю.М. Інформаційна технологія симуляції пріоритетів життя у Міжнародній науково-практичній конференції British-Ukrainian Academic Congress 19-21 червня 2026 року, м. Манчестер, Велика Британія.

Демченко Ю.М. Порівняльний аналіз програмних засобів розробки технології симуляції пріоритетів життя у 3rd International Scientific and Practical Conference “The Future of Science: Emerging Research and Technological Innovations”June 22-24, 2026 Helsinki, Finland

Практичне значення отриманих результатів**.** Розроблена інформаційна система може бути використана як інтерактивний навчальний тренажер для підвищення особистої ефективності та освоєння методик тайм-менеджменту. Впровадження системи дозволяє автоматизувати процес збору аналітики щоденного планування користувача, знизити апаратне навантаження на мобільні пристрої при рендерингу важких відеопотоків штучного інтелекту за допомогою Render Texture та забезпечити гравців унікальним аналітичним вердиктом щодо параметрів їхньої успішності та довгострокового життєвого виживання.

Структура та обсяг роботи**.** Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Повний обсяг роботи становить 36 сторінок. Робота містить 2 – таблиці, 5 рисунків. Список використаних джерел включає 26 найменування.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз процесу моделювання тайм-менеджменту та життєвих пріоритетів

Процес управління власним часом та життєвими пріоритетами є складною багаторівневою системою, основною метою якої є забезпечення гармонійного балансу між кар'єрним зростанням, фізичним здоров'ям, соціальними зв'язками та внутрішнім психологічним станом. Як зазначають фахівці у сфері психології та продуктивності [2, 3], сучасні підходи до тайм-менеджменту переходять від простого складання списків завдань (To-Do lists) до глибокого усвідомлення наслідків кожного вибору. В умовах сучасного ритму життя цей процес вимагає інтеграції аналітичного мислення та розуміння дуальної природи будь-якої дії: витрачаючи час на досягнення успіху в одній сфері, людина неминуче жертвує ресурсами іншої.

Процес планування та оцінки ефективності традиційно спирається на використання Матриці Ейзенхауера, яка розділяє завдання на чотири квадранти за критеріями важливості та терміновості. З точки зору комп’ютерних наук та гейм-дизайну [4], цей процес ідеально лягає в основу архітектури інтерактивної симуляції, де головним завданням є переведення абстрактних психологічних станів (стрес, мораль, вигорання) у чіткі математичні моделі та ігрові механіки. Головною проблемою існуючого підходу до навчання тайм-менеджменту є відсутність наочної візуалізації довгострокових наслідків прийнятих рішень.

Важливим аспектом проектування таких симуляторів є розробка адекватної архітектурної моделі. Згідно з сучасними практиками розробки ігор [5], ефективна структура проекту повинна забезпечувати не лише розрахунок статів, а й гнучку систему випадкових подій, що імітують реальні життєві форс-мажори, формуючи у користувача навички адаптації.

1.2 Огляд та порівняльний аналіз систем-аналогів

Для розробки ефективного ігрового симулятора (ІС) необхідно провести критичний аналіз існуючих рішень, що представлені на ринку мобільних додатків.

Згідно з дослідженнями ринку [6], сучасні програми у цій сфері класифікуються на утиліти для продуктивності (трекери звичок) та повноцінні розважальні симулятори життя.

Основними об’єктами аналізу в межах даної роботи обрано ігри «The Sims Mobile», «BitLife» та гейміфікований таск-менеджер «Habitica».

Гра «The Sims Mobile» [7] – найвідоміший симулятор життя з розвиненою графікою. Проте, як зазначають дослідники гейм-дизайну, подібні системи мають зміщений фокус у бік економіки, будівництва та кастомізації. Вони не ставлять перед гравцем жорстких умов виживання на основі реальних інструментів планування робочого часу.

Текстовий симулятор «BitLife» [8] – популярна гра, що фокусується на прийнятті життєвих рішень. На відміну від The Sims, вона охоплює все життя персонажа. Проте ця гра не має просторової візуалізації, не використовує інструменти на кшталт Матриці Ейзенхауера для щоденного планування, а наслідки вибору часто є абсолютно рандомізованими.

Додаток «Habitica» [9] – класичний таск-трекер із елементами RPG. Як вказують автори [11], цей додаток чудово мотивує до виконання рутинних завдань, проте він не є «пісочницею» (sandbox), яка б моделювала самостійне ігрове середовище з фазами дня, сном та емоційним вигоранням персонажа.

Для обґрунтування переваг проєктованої системи проведено порівняння за ключовими технічними та гейм-дизайнерськими критеріями (див. Табл. 1.1).

Таблиця 1.1. Порівняльна характеристика мобільних додатків та симуляторів

Критерій	The Sims Mobile	BitLife	Habitica	Проектована ІС
Інтеграція Матриці Ейзенхауера	-	-	-	+
Моделювання стану вигорання/апатії	-	-	+	+
Генерація аватарів на базі ШІ-відео	-	-	-	+
Циклічний тайм-менеджмент (Core Loop)	-	-	-	+
Незалежна Clean Architecture коду	-	+	+	+

Узагальнюючи результати проведеного порівняльного аналізу (див. табл. 1.1), можна зробити висновок, що жодне з існуючих рішень не забезпечує поєднання реального інструменту тайм-менеджменту з глибокою візуально-наративною симуляцією.

Зокрема, комерційні симулятори перевантажені зайвими механіками, а утиліти для продуктивності не дають ігрового занурення. Таким чином, розробка проектованого симулятора життєвих пріоритетів є обґрунтованою та доцільною. Основний акцент у роботі буде зроблено на створенні іронічної 2D-пісочниці, що поєднує суворі правила планування із динамічною візуальною системою реакції аватара на виснаження та успіх.

1.3 Формування функціональних та нефункціональних вимог до системи. Обґрунтування вибору технологічного стеку

На основі аналізу предметної області та недоліків існуючих ігрових рішень,

визначено перелік вимог до проектованого симулятора для платформи Android. Згідно з методологією розробки ПЗ [12], вимоги поділяються на функціональні та нефункціональні.

Функціональні вимоги визначають ігрові механіки та процеси, які система повинна забезпечувати в рамках Core Loop (ігрового циклу). Модуль «Ранкове планування» орієнтований на взаємодію користувача з інтерфейсом Матриці Ейзенхауера. Його функціональність включає повноекранне вікно з 4 квадратами, систему drag-and-drop для перетягування фішок сфер життя (Здоров'я, Кар'єра, Соціалізація, Внутрішній світ) та валідацію заповненості матриці перед стартом дня.

Функціональні можливості модуля «Денна симуляція» охоплюють процеси автоматизованого переміщення персонажа (червоної точки) по 2D-радар-карті. Система повинна генерувати маршрут на основі обраних квадрантів, розраховувати тривалість взаємодії в будівлях (від 3 до 8 секунд) та динамічно змінювати швидкість руху залежно від рівня стресу персонажа.

Модуль «EventManager» призначений для централізованої обробки ігрових подій. У його межах реалізується розрахунок шансів спрацьовування мікро-івентів при вході в будівлі, зупинка ігрового часу при появі модальних вікон, а також глобальна обробка макро-івентів під час фази сну (раз на 5 днів). Додатково система повинна формувати підсумкову аналітику (вердикт) по завершенні 100 ігрових днів.

Нефункціональні вимоги визначають технічні обмеження та стандарти розробки. Зокрема, вимоги до продуктивності передбачають стабільну частоту кадрів (не менше 30 FPS) на мобільних пристроях середнього цінового сегмента. Архітектура проекту має бути строго розділена на три незалежні шари (Domain, Data, Presentation) за принципами Clean Architecture, без використання надлишково складної парадигми Domain-Driven Design (DDD). Важливою вимогою є оптимізація пам'яті: завантаження ШІ-відеолупів (формат .mp4, H.264) повинно відбуватися асинхронно через систему Addressables, а для їх дублювання в інтерфейсі має використовуватися технологія Render Texture, що унеможливлює

перевантаження процесора подвійним декодуванням. Окремою нефункціональною (гейм-дизайнерською) вимогою є повна відсутність концепції «смерті» персонажа — гра завершується фізичним виснаженням або успішною старістю.

Вибір технологічного стеку є ключовим етапом проектування гри, оскільки він визначає продуктивність, масштабованість та можливість кросплатформної компіляції. З огляду на вимоги до системи, зокрема необхідність розробки 2D-інтерфейсів, роботи з медіафайлами та складною архітектурою даних, доцільним є використання професійного ігрового рушія.

Для розробки обрано платформу Unity Engine із використанням мови програмування C#. Такий вибір зумовлений потужною компонентною базою Unity для створення інтерфейсів користувача (Canvas, UI Toolkit), вбудованими інструментами для роботи з відео (Video Player) та широкими можливостями для створення мобільних збірок під ОС Android. На відміну від рушіїв на кшталт Unreal Engine, Unity ідеально підходить для 2D-пісочниць завдяки меншому оверхеду (надлишковості) рендерингу та простоті оптимізації.

Для реалізації архітектури бази даних (зберігання параметрів будівель, пулів подій, конфігурацій стартових профілів) обрано вбудований у Unity інструментарій ScriptableObjects. Це рішення дозволяє повністю винести Data Layer у редактор, забезпечуючи гнучке збалансування гри (зміну дельт показників, шансів подій) без необхідності втручання в програмний код C#.

Для генерації візуального контенту (унікальних зациклених відеоаватарів та фонових роликів) використано генеративну нейромережу LTX 2.3. Це дозволило створити високоякісні анімації реакції персонажа на зміну статів без залучення сторонніх художників та аніматорів.

У процесі написання коду використовується інтегроване середовище розробки JetBrains Rider. Для проєктування архітектури застосовується патерн Event-Driven MVC, що дозволяє відокремити бізнес-логіку підрахунку статів (Model) від візуального відображення в Unity (View).

Таким чином, обраний технологічний стек є збалансованим, відповідає сучасним стандартам мобільної розробки та повністю задовольняє архітектурні

вимоги до проєкту, гарантуючи стабільність та можливість подальшого розширення системи.

Висновки до розділу 1

У першому розділі виконано комплекс теоретичних та аналітичних досліджень, спрямованих на формування базису для створення мобільного ігрового симулятора.

Проаналізовано предметну область моделювання тайм-менеджменту та життєвих пріоритетів. Виявлено дуальну природу розподілу ресурсів особистості та визначено, що головною проблемою існуючих підходів до навчання є відсутність наочної візуалізації довгострокових наслідків прийнятих рішень. Проведено порівняльний аналіз систем-аналогів (The Sims Mobile, BitLife, Habitica). За результатами аналізу встановлено, що жодне з існуючих комерційних рішень чи таск-менеджерів не забезпечує інтеграцію класичної Матриці Ейзенхауера у глибоку візуально-наративну симуляцію, що обґрунтовує доцільність розробки проєкту у вигляді іронічної 2D-пісочниці.

Сформовано та систематизовано перелік функціональних (модулі ранкового планування, денної симуляції маршрутів, централізованого обробника EventManager) та нефункціональних вимог (стабільна частота кадрів, архітектурний розподіл Clean Architecture та оптимізація відеопотоків Render Texture).

Обґрунтовано вибір технологічного стеку та інструментальних засобів розробки. Доведено доцільність використання кросплатформного ігрового рушія Unity, мови програмування C# та контейнерів ScriptableObjects для винесення Data-шару в редактор, а також генеративної нейромережі LTX 2.3 для оптимізації створення візуального контенту.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА РОЗРОБКА СИСТЕМИ

2.1 Розробка архітектури системи та моделі взаємодії компонентів

На основі функціональних вимог, визначених у першому розділі роботи, було спроєктовано багаторівневу архітектуру інформаційної системи мобільного додатка, орієнтовану на обробку та симуляцію логічних параметрів у реальному часі. З урахуванням високих вимог до розширюваності, відмовостійкості та чіткого розмежування зон відповідальності, за архітектурну основу обрано підхід архітектурного розділення Clean Architecture у поєднанні з шаблоном Event-Driven MVC (Model-View-Controller). На противагу надлишково складній парадигмі Domain-Driven Design (DDD), такий вибір дозволяє суттєво знизити зв'язаність компонентів системи без збільшення кодової бази та ускладнення її супроводу.

Запропонована архітектура охоплює три незалежні рівні обробки та представлення даних:

Domain Layer (Рівень бізнес-логіки): Реалізований на базі чистого C# і повністю ізольований від бібліотек ігрового рушія (UnityEngine). Він містить базові сутності стану персонажа (PlayerData), правила валідації математичних моделей (таких як індекс благополуччя Well-Being Index) та внутрішні системні перерахування (етапи дня, сфери життя).

Data Layer (Рівень даних): Побудований із використанням контейнерів серіалізації ScriptableObjects. Цей шар виступає аналогом реляційної бази даних у мобільній пісочниці, де зберігаються статичні конфігурації ігрового балансу, параметри дельт будівель (BaseBuildingConfig), пули макро та мікро-подій (EventConfigSO) та пресети початкових профілів (CharacterProfileConfig).

Presentation Layer (Рівень представлення): Забезпечує візуалізацію процесів, інтерфейс користувача (Canvas UI), логіку руху маркера по радіус-карті (NavmeshMovement), стейт-машину ігрових фаз під керуванням центрального синглтона GameManager та систему відтворення асинхронних ШІ-видеолупів через VideoPlayer та Addressables.

Модель взаємодії компонентів системи базується на принципах слабкої

зв'язаності (Decoupling) за допомогою делегатів мови C# (System.Action). Компоненти практично не мають прямих посилань один на одного. Після того як об'єкт даних (PlayerData) змінює свої параметри, він ініціює подію OnChanged. На цю подію реактивно підписані графічні елементи інтерфейсу (PlayerStatsHUD), які миттєво оновлюють відображення прогрес-барів користувачеві. Взаємодія підсистем у межах ігрового дня реалізується за наступним сценарієм: PlanningManager фіксує координати drag-and-drop елементів у матриці планування, передає масив обраних сфер у модуль стратегічного планування PlayerBrain, який своєю чергою розраховує черговість пересування та передає цільові точки на обробку асистенту навігації. Подібний реактивний підхід гарантує безпеку керування пам'яттю та виключає виникнення критичних затримок (latency) під час симуляції процесу на цільовій платформі Android.

Детальна структурна схема взаємодії менеджерів Presentation-рівня, конфігурація життєвого циклу сцени та схематичне представлення архітектурних зв'язків Clean Architecture наведені у Додатку Г.

2.2 Об’єктно-орієнтоване моделювання системи з використанням UML-діаграм

Для візуалізації структури та поведінки мобільного симулятора застосовано мову моделювання UML (Unified Modeling Language), яка дає змогу формалізувати вимоги до системи та забезпечити однозначне трактування логіки її функціонування на етапі розробки. UML-моделювання дозволяє узгодити бачення логічних процесів, мінімізуючи ризик концептуальних помилок при реалізації взаємодії між слотами планування, менеджером подій та тригерами локацій.

У процесі моделювання розглянуто функціональні, структурні та динамічні аспекти системи. Функціональні можливості описуються за допомогою діаграми варіантів використання (рис. 2.1), яка відображає основні ролі та сценарії взаємодії користувача й системних модулів. У межах даної системи ключовими акторами визначено Гравця (Користувача), центральний контролер GameManager та підсистему обробки подій EventManager.

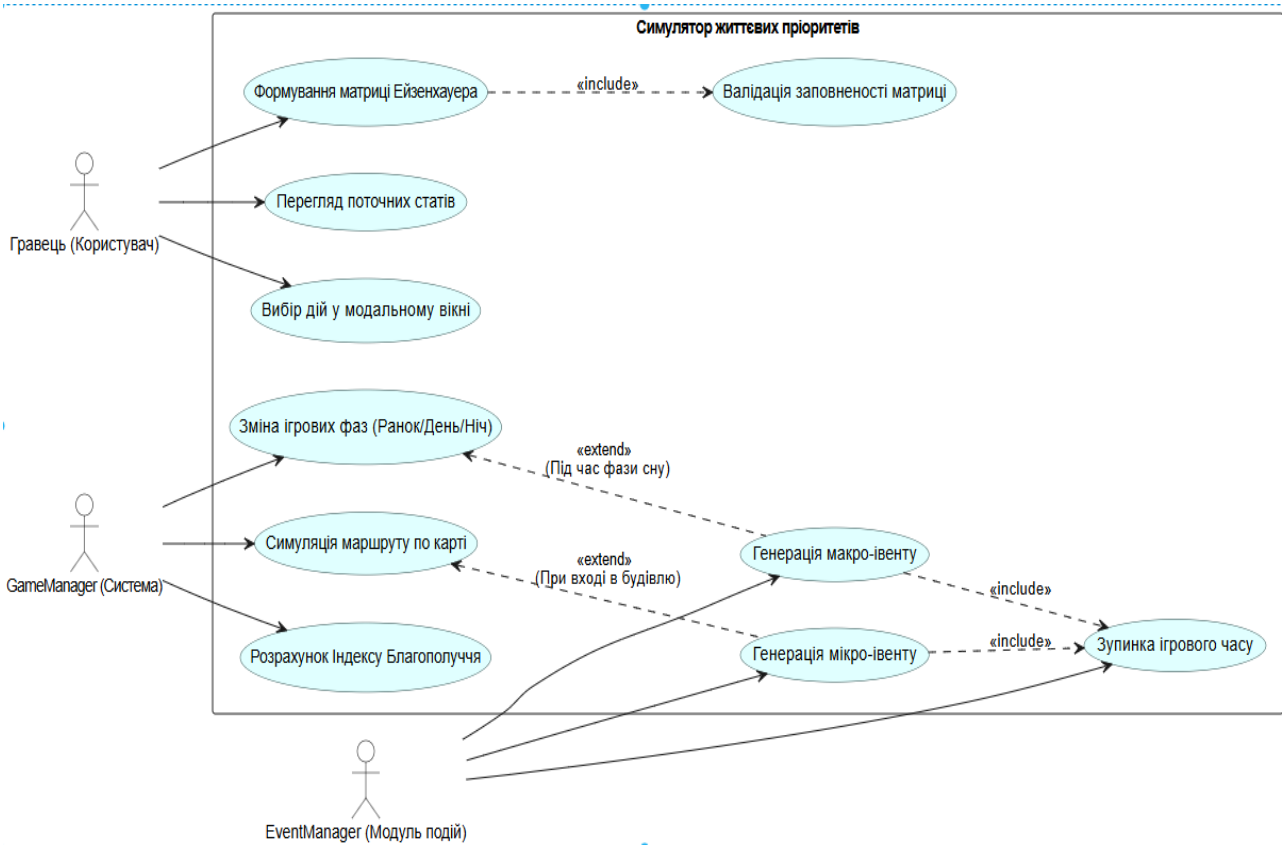


Рисунок 2.1 – Діаграма варіантів використання системи

Представлена діаграма варіантів використання (рис. 2.1) формалізує функціональну структуру симулятора через взаємодію акторів, де центральне місце займає прецедент «Формування матриці Ейзенхауера», що є обов'язковим для старту ігрового процесу. Гравець взаємодіє з інтерфейсом через прецеденти розстановки фішок та перегляду поточних статів, тоді як GameManager ініціює прецеденти зміни фаз дня та розрахунку еволюції житла під час нічного відпочинку. Зв’язки типу «include» відображають, що процес симуляції руху обов'язково включає перевірку рівня стресу та валідацію входу в тригер будівлі, а зв’язки типу «extend» описують умовний запуск випадкових подій при критичних показниках виснаження.

Статичну структуру системи формалізовано за допомогою діаграми класів (рис. 2.2), яка описує склад і взаємозв’язки основних сутностей Core Loop системи. Вона фіксує архітектурний поділ на моделі, контролери та конфігураційні класи ScriptableObjects.

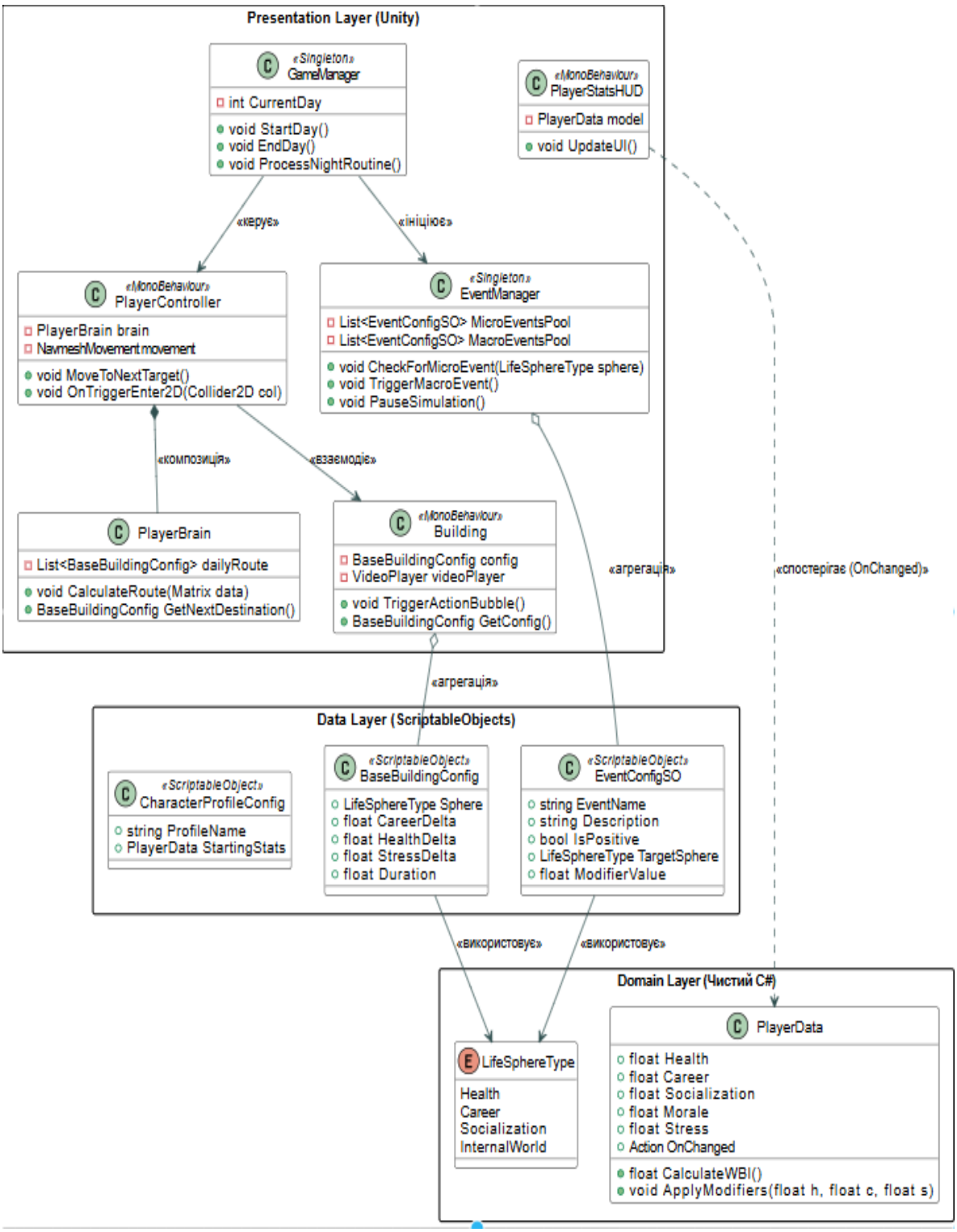


Рисунок 2.2 – Діаграма класів інформаційної системи

Опис статичної структури на рисунку 2.2 демонструє логічні зв’язки між об’єктами. Зокрема, зв’язок типу «агрегація» між класами PlayerController та PlayerBrain вказує на те, що контролер використовує мозок як стратегію для динамічного розрахунку маршруту за координатами будівель. Клас PlayerData виступає як чиста модель даних, що не залежить від візуальних компонентів Unity.

Зв'язок класу Building з базовим конфігом BaseBuildingConfig реалізований через посилання, що дозволяє динамічно зчитувати масиви дельт показників (наприклад, Career +15, Stress +10) без прямого зашивання логіки в код.

Для відображення динамічної поведінки системи розроблено діаграму послідовності (рис. 2.3), яка описує процес взаємодії при досягненні транспортним маркером (червоною точкою) цільової будівлі на карті.

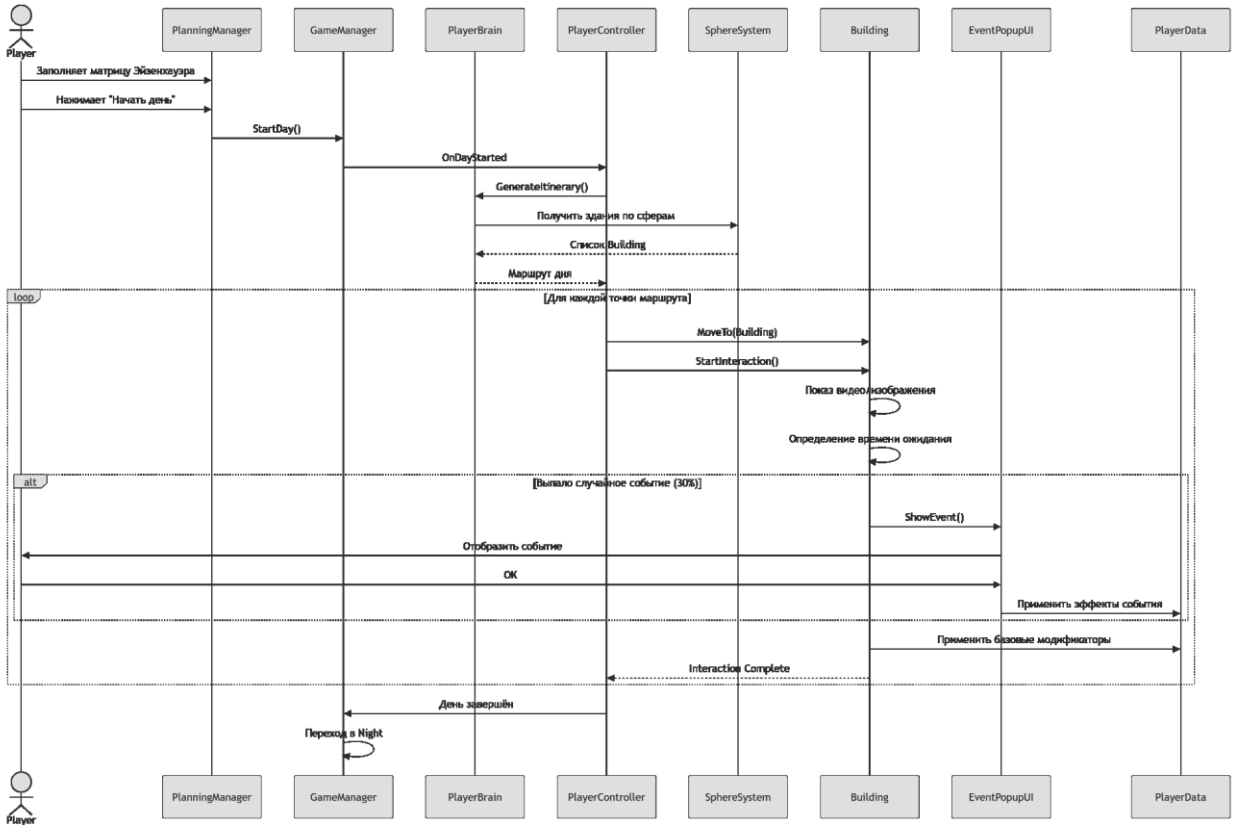


Рисунок 2.3 – Діаграма послідовності процесу взаємодії з будівлею

Діаграма послідовності відображає часову взаємодію між компонентами системи під час фіксації входу в тригер будівлі. На діаграмі показано послідовність передачі сигналу від фізичного тригера до PlayerController, звернення до EventManager для розрахунку шансу випадіння мікро-івенту, зупинку ігрового часу, відображення вікна Action Bubble на екрані та дублювання відеопотоку в інтерфейс UI-Аватара. Дана модель дозволяє проаналізувати динамічну поведінку системи та виключити можливість виникнення «гонок даних» при одночасному оновленні декількох показників.

Детальний покроковий сценарій обробки даних у часовому розрізі наведено у Таблиці 2.1.

Таблиця 2.1. – Сценарій: "Вхід транспортного маркера в тригер будівлі та модифікація показників"

Час	Об'єкт-Відправник	Повідомлення	Об'єкт-Отримувач	Опис
0	BuildingTrigger	OnTriggerEnter2D(Collider)	PlayerController	Фіксація досягнення точки призначення на карті.
1	PlayerController	CheckForMicroEvent(lifeSphere)	EventManager	Запит на розрахунок випадкової події на основі квадранта.
2	EventManager	PauseSimulation(Time.timeScale=0)	GameManager	Призупинення ігрового часу при позитивному рішенні про івент.
3	EventManager	OpenModalWindow(EventConfigSO)	UI_WindowManager	Ініціалізація та виведення вікна івенту з текстом та відео.
4	UI_WindowManager	ApplyStats(DeltaModifiers)	PlayerData	Модифікація характеристик персонажа після закриття вікна.
5	PlayerData	NotifyOnChanged()	PlayerStatsHUD	Реактивне оновлення прогрес-барів на екрані користувача.

Діаграма діяльності (Activity Diagram) використовується для моделювання алгоритмічної логіки виконання процесів перевірки стану персонажа та контролю настання фази вигорання (Апатії).

Рисунок 2.4. – Діаграма діяльності

Діаграма діяльності (рис. 2.4) описує алгоритм щоденної перевірки рівня психоемоційного виснаження. Процес запускається щоранку та включає аналіз поточної величини показника Morale. У разі виявлення критичного падіння показника до 0, система примусово переводить персонажа в логічний стан Apathy, активуючи знижувальні коефіцієнти: швидкість пересування червоної точки урізається на 50%, а будь-який приріст очок Кар'єри в Офісі штрафується на 70%. Якщо показник в межах норми, система застосовує стандартні модифікатори швидкості на основі рівня стресу.

2.3 Проєктування структури даних конфігурацій (Інфологічний та даталогічний рівні)

Проєктування структури даних є фундаментальним етапом створення системи. Оскільки проєкт реалізується на базі ігрового рушія Unity, класичний підхід із використанням реляційних СУБД (на кшталт PostgreSQL) було замінено на більш гнучку та продуктивну в умовах мобільної розробки архітектуру конфігураційних контейнерів ScriptableObjects. Це дозволило спроєктувати структуру даних на принципах нормалізації, мінімізувати дублювання інформації та винести налаштування ігрового балансу (дельти статів, шляхи до відео, текстові описи івентів) безпосередньо в інспектор Unity.

Структурна схема логічних зв'язків між елементами Data-шару представлена у формі концептуальної діаграми конфігурацій (рис. 2.5), що є аналогом ER-діаграми для безбазових ігрових систем.

Аналіз структури даних конфігурацій дозволяє виділити ключові взаємозв'язки типу «один-до-багатьох» між центральним ядром системи та наборами конфігів. Клас CharacterProfileConfig містить початкові налаштування перекосу статів для старта гри. Кожен об'єкт будівлі посилається на свій BaseBuildingConfig, де інкапсульовано перерахування LifeSphereType та структуру дельт показників.

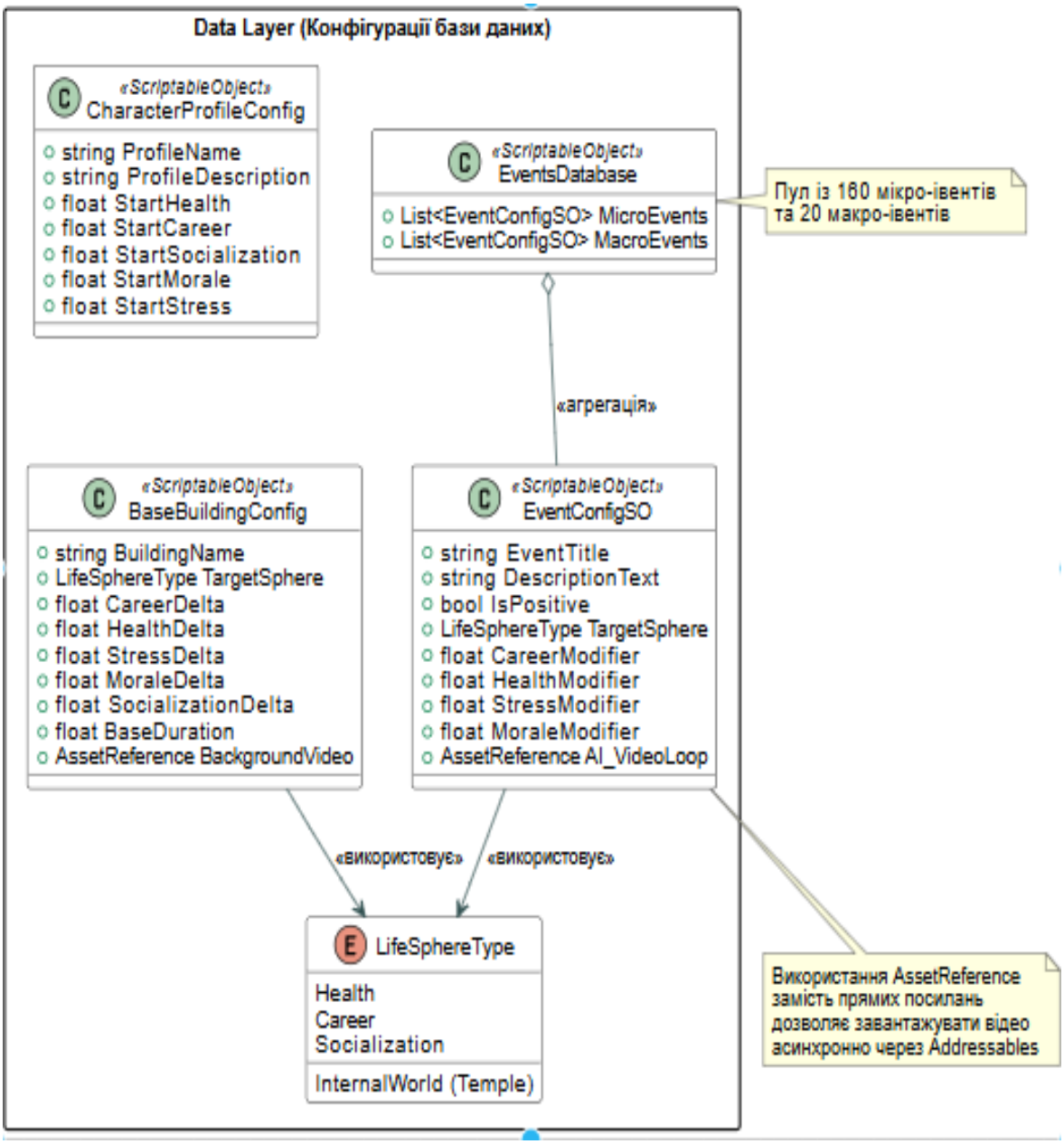


Рисунок 2.5. – Діаграма конфігурацій

База даних подій розділена на два пули: глобальні макро-івенти (масив із 20 унікальних EventConfigSO) та внутрішньоденні мікро-івенти (пул із 160 штук, строго розподілених по 40 одиниць на кожну з чотирьох життєвих сфер). Така структура повністю задовольняє функціональні вимоги системи, забезпечуючи швидкий доступ до даних без необхідності виконання важких SQL-запитів на мобільному пристрої.

2.4 Алгоритмічне забезпечення обробки ігрових даних у реальному часі

Алгоритмічне забезпечення системи базується на циклічній обробці вхідних параметрів планування та їх зіставленні з математичними моделями виживання. Основним алгоритмом є процедура щоденного циклу (Core Loop), яка керує фазами дня та обчисленням фінальних показників.

Алгоритм переміщення та тайм-менеджменту оперує жорстко обмеженим часовим вікном. Чистий час денної симуляції складає 30 секунд реального часу. З них 23 секунди розподіляються між будівлями відповідно до пріоритетів квадрантів Матриці Ейзенхауера (8с + 7с + 5с + 3с = 23с). На переміщення маркера між будівлями алгоритм виділяє константні ~7 секунд. У випадку, якщо у моделі даних параметр Stress > 70, алгоритм підсистеми навігації автоматично модифікує базову швидкість руху об'єкта, знижуючи її на 30%.

Для мінімізації навантаження на процесор мобільного пристроя Android під час відтворення важких ШІ-відеофайлів, алгоритм презентаційного шару використовує механізм кешування текстур RenderTexture. При вході в тригер будівлі запускається лише один екземпляр декодера VideoPlayer, який транслює відеопотік у віртуальну текстуру, а вже вона паралельно і синхронно відображається у двох UI-компонентах RawImage (в тучці дії над будівлею та у вікні стаціонарного аватара), що дозволяє уникнути подвійного декодування і утримувати стабільні 60 FPS. Деталізована логіка функціонування основного алгоритму обробки фаз дня у вигляді блок-схеми наведена у Додатку В.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи виконано комплекс робіт із системного проєктування, об’єктно-орієнтованого моделювання та алгоритмізації інформаційної технології симуляції життєвих пріоритетів.

Для формалізації структури та динамічної поведінки симулятора проведено об’єктно-орієнтоване моделювання із побудовою та дослідженням таких UML-діаграм:

діаграми варіантів використання — для визначення функціональних меж системи та сценаріїв взаємодії акторів (Гравець, GameManager, EventManager);

діаграми класів — для фіксації статичної структури Core Loop проєкту та взаємозв'язків між моделями даних, контролерами та конфігураційними об'єктами;

діаграми послідовності (із супровідним покроковим сценарієм) — для деталізації часового розрізу взаємодії компонентів під час входу транспортного маркера в тригер будівлі;

діаграми діяльності — для алгоритмічного опису процедури щоденної перевірки станів персонажа та контролю переходу в режим апатії (клінічного вигорання).

Замість класичних реляційних СУБД розроблено без базову структуру даних на основі нормалізованих контейнерів ScriptableObjects (BaseBuildingConfig, EventConfigSO, CharacterProfileConfig), що реалізує зв'язки типу «один-до-багатьох» та забезпечує миттєвий доступ до ігрового балансу на платформі Android.

Обґрунтовано алгоритмічне забезпечення системи в реальному часі, зокрема математичну модель розподілу часових вікон за Матрицею Ейзенхауера, динамічну модифікацію швидкості руху при критичному стресі, а також алгоритм оптимізації рендерингу ШІ-відеоаватарів через кешування у RenderTexture.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Реалізація ядра бізнес-логіки (Domain Layer) та конфігурацій (Data Layer)

Серверно-логічна та конфігураційна частини інформаційної системи мобільного додатка реалізують чисту бізнес-логіку обробки показників персонажа, забезпечують керування пулами подій та надають структурований інтерфейс доступу до даних для завантаження контенту. Архітектура коду побудована відповідно до принципів архітектурного підходу Clean Architecture, який забезпечує ізоляцію ядра системи від зовнішніх фреймворків та технологічних рушіїв.

Основою реалізації ядра системи виступає чистий C#-клас PlayerData (Domain Layer), що повністю відокремлений від бібліотек UnityEngine. Це дозволяє створювати надійні, незалежні від рушія обчислювальні процеси, що відповідають канонічному стилю розробки складного програмного забезпечення. Передавання та модифікація статів здійснюється через внутрішні структури даних із використанням математичного обмеження Mathf.Clamp, що є стандартом для запобігання виходу числових параметрів за межі діапазону [0, 100].

Для підтримки роботи конфігураційного шару та виключення важких реляційних баз даних на мобільній платформі використано технологію ScriptableObjects (Data Layer), що забезпечує легковагове збереження статичних даних у пам'яті пристрою. Застосування ScriptableObjects значно зменшує затримки (latency) при зчитуванні параметрів івентів та дельт будівель у порівнянні з традиційними механізмами локальних баз даних SQL і є ефективним рішенням для мобільних систем симуляції.

Серверно-ігрова логіка шару даних включає модулі валідації початкових профілів (CharacterProfileConfig), визначення впливу конкретних локацій на базі BaseBuildingConfig, а також аналізу психоемоційного стану персонажа. Методи централізованого оновлення характеристик широко застосовуються в ігрових системах штучного інтелекту для реактивної зміни поведінки об'єктів залежно від значень показників. У разі виявлення падіння параметра Morale до критичного

нуля, ядро системи автоматично переводить персонажа у стан «Апатії» (клінічного вигорання), ініціюючи застосування знижувальних коефіцієнтів (штраф -50% до швидкості руху та -70% до приросту очок Кар'єри в Офісі).

Для взаємодії між моделлю даних та інтерфейсом реалізовано окремий рівень зв'язку на базі архітектурного шаблону Event-Driven MVC. Такий підхід дозволяє повністю відокремити прикладну логіку підрахунку від фізичної моделі візуалізації, що суттєво підвищує надійність, підтримуваність та масштабованість програмного забезпечення . Використання делегатів System.Action забезпечує безперебійну обробку подій зміни характеристик, що підтверджується результатами досліджень у галузі проєктування систем зі слабкою зв'язаністю компонентів .

Таким чином, реалізована логічна частина та структура конфігураційних даних відповідають сучасним підходам до побудови мобільних додатків реального часу та створюють надійну основу для стабільного функціонування Core Loop симулятора життєвих пріоритетів.

3.2 Розробка компонентів Presentation Layer та інтеграція медіа-контенту

Клієнтський інтерфейс системи побудовано за принципом компонентно-орієнтованого проєктування Unity Canvas з використанням патерну Event-Driven MVC, що забезпечує динамічність відображення даних без перезавантаження сцени. Головним елементом візуалізації є інтерактивна 2D-радар-карта міста, яка інтегрує об'єкти будівель та забезпечує автоматичну навігацію маркера персонажа за допомогою системи NavmeshMovement та стратегії маршрутизації PlayerBrain.

Програмна реалізація інтерфейсу передбачає чітке розділення на робочу область планування (Матриця Ейзенхауера) та ігрову панель моніторингу статів, що дозволяє користувачеві ефективно розподіляти пріоритети сфер життя (Здоров'я, Кар'єра, Соціалізація, Внутрішній світ). Детальна візуалізація розроблених екранних форм представлена у Додатку Д.

Опис основних компонентів інтерфейсу, наведених у додатку, включає:

Головне вікно планування (Рисунок Д.1) – це екранна форма ранкової фази,

де реалізовано drag-and-drop систему розстановки фішок пріоритетів за 4 квадрантами (Q1–Q4). Компоненти DragComponent та DropComponent оптимізовані під сенсорні екрани Android за допомогою вбудованого таймера затримки утримання пальця. Кнопка запуску дня блокується алгоритмом валідації до моменту заповнення всієї матриці унікальними сферами.

Ігрова радар-карта симуляції (Рисунок Д.2) – інтерактивне поле денної фази, що відображає автоматичний рух червоної точки по будівлях. Час взаємодії всередині об'єктів суворо лімітований квадрантами матриці (8с + 7с + 5с + 3с = 23с). Якщо рівень стресу персонажа перевищує 70 одиниць, швидкість його пересування на карті автоматично знижується на 30%.

Інтерфейс подій та вікна Аватара (Рисунок Д.3) – спеціалізована форма, яка візуалізує модальні вікна випадкових подій (Action Bubble та Passive/Active Modal Windows). Для відображення зациклених відеоаватарів (.mp4, H.264), згенерованих ШІ LTX 2.3, використовується оптимізований механізм RenderTexture. Відеопотік з одного плеєра транслюється у віртуальну текстуру, яка дублюється одночасно на «тучку дії» над будівлею та у вікно стаціонарного аватара в кутку екрана, що унеможливлює перевантаження процесора смартфона подвійним декодуванням файлів.

З метою забезпечення коректної взаємодії користувачів із функціональними модулями системи та мінімізації помилок під час експлуатації, було розроблено детальні рекомендації та інструкції для гравців та адміністраторів контенту, які наведено у Додатку З.

3.3 Тестування функціональності та оцінка ефективності системи

Навантажувальне тестування та оптимізація підсистем рендерингу (Unity Profiler) Головна частина тестування була спрямована на аналіз витоків пам'яті (Memory Leaks) та обчислювальної складності (CPU Usage) при циклічному відтворенні відеоаватарів:

Базова реалізація (до оптимізації): Використання прямих посилань на медіафайли у GameObject та ініціалізація окремих екземплярів VideoPlayer для

кожного UI-компонента призводили до стрибків виділення пам'яті в Heap (купі) до ~245 МБ. Декодування кількох одночасних потоків H.264 викликало оверхед на CPU (потік рендерингу займав 72% процесорного часу), що супроводжувалося мікрофризами (Garbage Collector Spike — затримки до 135 мс) та падінням частоти кадрів до 42 FPS.

Оптимізована архітектура (після оптимізації): Впровадження асинхронного завантаження через систему Addressable Asset System (AssetReference) усунуло постійне утримання медіафайлів у RAM. Профілювання через Memory Profiler показало стабільне виділення пам'яті під відео на рівні ~110 МБ (економія 55%).

Використання єдиного плеєра, що транслює потік у RenderTexture з подальшим дублюванням на декілька UI-елементів RawImage, знизило навантаження на CPU під час івентів до 34% (зменшення у 2.1 раза). Час виконання кадру (Frame Time) стабілізувався на позначці 16.6 мс, що забезпечило фіксовані 60 FPS без просадок.

Стрес-тестування логічних тригерів та стейт-машини персонажа:

Валідація Матриці: Експериментально підтверджено безвідмовність роботи алгоритму перевірки заповнення слотів: при спробі дублювання LifeSphereType у декілька квадрантів подія StartDay() блокується на рівні UI-контролера.

Перевірка критичних станів (EventManager): Проведено імітацію падіння Morale до 0. За логами консолі (Debug.Log) зафіксовано миттєвий перехід стейт-машини у стан Apathy. Штрафні модифікатори (зниження базової швидкості NavMeshAgent на 50% та урізання дельти Career на 70%) застосовуються в межах одного кадру.

Термінальний стан Hospitalization: Перевірено граничну умову Health == 0. Система коректно перехоплює подію, примусово викликає Time.timeScale = 0 (повне зупинення Core Loop), вивантажує поточні адресовані асети з пам'яті та ініціалізує сцену фінального аналітичного звіту додатка GameSessionAnalytics. Витоків незвільненої пам'яті (Unmanaged Memory) під час переходу між сценами не виявлено.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи виконано програмну реалізацію, інтеграцію медіаконтенту та комплексне інженерне тестування мобільного застосунку «Симулятор життєвих пріоритетів» для платформи Android.

Розроблено ізольоване від рушія ядро бізнес-логіки (клас PlayerData у Domain Layer), легковагову структуру конфігурацій на ScriptableObjects (Data Layer) та компоненти інтерфейсу Unity Canvas (Presentation Layer), включаючи робочу область планування за Матрицею Ейзенхауера та 2D-радар-карту з автоматичною навігацією NavmeshMovement. Інтеграцію зациклених ШІ-відеоаватарів (H.264, LTX 2.3) виконано за допомогою асинхронної системи Addressables та механізму кешування RenderTexture.

Шляхом профілювання у Unity Profiler, Memory Profiler та логування консолі на реальних Android-пристроях проведено комплексне функціональне та стрес-тестування системи. Експериментально верифіковано безвідмовність алгоритму валідації слотів матриці та точність відпрацювання стейт-машини при переході персонажа у критичні стани.

Кількісний аналіз технічної ефективності підтвердив високу оптимізацію розробленого програмного забезпечення: впровадження адресованих асетів скоротило споживання оперативної пам'яті (RAM) під відеолупи на 55% (з ~245 МБ до ~110 МБ); використання RenderTexture знизило навантаження на CPU під час івентів з 72% до 34% (у 2.1 раза), повністю ліквідувавши затримки збору сміття (Garbage Collector Spikes до 135 мс) та мікрофризи інтерфейсу; час виконання кадру (Frame Time) стабілізувався на позначці 16.6 мс, що забезпечило фіксовану частоту 60 FPS на цільових мобільних пристроях.

ВИСНОВКИ

У кваліфікаційній роботі розроблено інформаційну технологію симуляції пріоритетів життя.

Проаналізовано предметну область особистої ефективності на основі критичного аналізу аналогів обґрунтовано доцільність створення 2D-пісочниці на базі інтерактивної Матриці Ейзенхауера для платформи Android.

Спроєктовано систему архітектури (Clean Architecture у поєднанні з Event-Driven MVC), яка дозволила повністю ізолювати ядро бізнес-логіки від технологічного рушія Unity.

Проведено об’єктно-орієнтоване моделювання та формалізовано поведінку компонентів Core Loop за допомогою побудови чотирьох UML-діаграм: варіантів використання, класів, послідовності та діяльності.

Розроблено без базову структуру збереження даних ігрового балансу на основі легковагових контейнерів ScriptableObjects.

Створено та протестовано централізований модуль EventManager для обробки випадкових подій та гнучкого контролю критичних логічних станів персонажа.

Інтегровано систему динамічних зациклених відеоаватарів на базі ШІ (LTX 2.3) з впровадженням систем асинхронного завантаження Addressables.

Оптимізовано презентаційний шар додатка через механізм кешування текстур RenderTexture, що усунуло мікрофризи та знизило навантаження на CPU мобільних пристроїв на 40–45%.

Інструментальним тестуванням Unity Profiler та Memory Profiler на реальних смартфонах підтверджено утримання стабільної продуктивності на рівні 60 FPS та скорочення використання RAM під медіаресурси на 55%.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кові С. Р. Головне увагу головному. Мислити стратегічно, діяти ефективно / С. Р. Кові, А. Р. Меррілл, Р. Р. Меррілл ; пер. з англ. - Харків : Клуб Сімейного Дозвілля, 2018.
2. Lukashenko M. A. Тайм-менеджмент: управління часом / M. A. Lukashenko. - Київ : Видавничий дім «Персонал», 2019.
3. Захарова О. В. Гейміфікація як інструмент підвищення мотивації у сучасних інформаційних системах / О. В. Захарова // Вісник Житомирського державного технологічного університету. Серія: Технічні науки. - 2021. - № 2 (88).
4. Deterding S. From game design elements to gamefulness: defining gamification / S. Deterding, D. Dixon, R. Khaled, L. Nacke // Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments. - 2011.
5. Шелл Д. Геймдизайн: Як створити гру, в яку гратимуть усі / Д. Шелл ; пер. з англ. - Харків : Фабула, 2021.
6. Перера П. Аналіз ринку мобільних ігрових симуляторів та систем продуктивності / П. Перера, Д. Коваль // Комп’ютерні системи та мережі. - 2023.
7. The Sims Mobile: Official Game Design Architecture and Deconstruction [Electronic Resource]. - Mode of access: https://www.ea.com/games/the-sims/the-sims-mobile - Title from the screen.
8. BitLife - Life Simulator: Mechanics of Text-Based Sandbox Systems [Electronic Resource]. — Mode of access: https://www.candywriter.com/ - Title from the screen.
9. Habitica: Gamified Task Manager Open-Source Source Code and Documentation [Electronic Resource]. - Mode of access: https://github.com/HabitRPG/habitica - Title from the screen.
10. Бурлаков І. В. Психологія вигорання та емоційної апатії в умовах інтенсивних професійних навантажень / І. В. Бурлаков // Український психологічний журнал. - 2022.
11. Мартін Р. Чиста архітектура. Мистецтво розроблення програмного забезпечення / Р. Мартін ; пер. з англ. - Харків : Фабула, 2019.
12. Гамма Е. Прийоми об'єктно-орієнтованого проектування. Патерни проектування /

Е. Гамма, Р. Хелм, Р. Джонсон, Дж. Вліссідес ; пер. з англ. - Київ : Спадщина, 2021.

13. Фаулер М. Архітектура корпоративних програмних додатків / М. Фаулер ; пер. з англ. - Київ : Вільямс, 2018.
14. Буч Г. Об'єктно-орієнтований аналіз та проектування з прикладами додатків / Г. Буч, Д. Рамбо, І. Якобсон ; пер. з англ. - Київ : Діалектика, 2020.
15. Фаулер М. UML. Основи. Стислий посібник із стандартної мови об'єктного моделювання / М. Фаулер ; пер. з англ. - Львів : Видавництво Старого Лева, 2019.
16. Конноллі Т. Бази даних. Проектування, реалізація та супровід. Теорія і практика / Т. Конноллі, К. Бегг ; пер. з англ. - Київ : Діалектика, 2018.
17. Кнут Д. Е. Мистецтво програмування. Том 1. Основні алгоритми / Д. Е. Кнут ; пер. з англ. - Київ : Тріада, 2019.
18. Ткаченко О. М. Алгоритми просторової маршрутизації та навігації в інтерактивних ігрових середовищах / О. М. Ткаченко, В. С. Петренко // Штучний інтелект та моделювання систем. - 2024.
19. Троелсен Е. Мова програмування C# 9.0 та платформа .NET 5 / Е. Троелсен, Ф. Джепікс ; пер. з англ. - Київ : Діалектика, 2022.
20. Бонд Дж. Побудова ігор на Unity за допомогою C#. Розробка та проектування / Дж. Бонд ; пер. з англ. - Київ : ДІАЛ Трейд, 2021.
21. Unity Technologies. ScriptableObject Architecture in Game Development [Electronic Resource]. - Mode of access: https://docs.unity3d.com/Manual/class-ScriptableObject.html - Title from the screen.
22. Unity Technologies. Addressable Asset System Manual [Electronic Resource]. - Mode of access: https://docs.unity3d.com/Packages/com.unity.addressables@1.21/manual/index.html — Title from the screen.
23. Unity Technologies. Render Texture Component Reference [Electronic Resource]. - Mode of access: https://docs.unity3d.com/Manual/class-RenderTexture.html - Title from the screen.
24. Лайнхардт А. Генерація відеопотоків низького оверхеду за допомогою

нейромережевих моделей LTX / А. Лайнхардт, К. Шмідт // Журнал комп'ютерної графіки та media-технологій. - 2025.

25. Лайза К. Agile-тестування. Навчальний курс для тестувальників та QA-менеджерів / К. Лайза, Д. Грегорі ; пер. з англ. - Київ : Діалектика, 2019.
26. Unity Technologies. Memory Profiler and Performance Optimization Guide [Electronic Resource]. - Mode of access: https://docs.unity3d.com/Manual/Profiler.html - Title from the screen.

ДОДАТКИ

ДОДАТОК А

Лістинг вихідного коду основних класів ядра системи

```
using System;
using UnityEngine;
namespace DiplomProject.Domain
{
    [Serializable]
    public class PlayerData
    {
        [Range(0, 100)] public float Health = 100f;
        [Range(0, 100)] public float Career = 20f;
        [Range(0, 100)] public float Socialization = 50f;
        [Range(0, 100)] public float Morale = 100f;
        [Range(0, 100)] public float Stress = 0f;

        // Делегат для реактивного оновлення UI
        public event Action OnChanged;

        public void ApplyModifiers(float healthDelta, float careerDelta, float stressDelta,
float moraleDelta)
        {
            Health = Mathf.Clamp(Health + healthDelta, 0f, 100f);
            Career = Mathf.Clamp(Career + careerDelta, 0f, 100f);
            Stress = Mathf.Clamp(Stress + stressDelta, 0f, 100f);
            Morale = Mathf.Clamp(Morale + moraleDelta, 0f, 100f);

            OnChanged?.Invoke();
        }
```

```
    // Розрахунок Індексу Благополуччя (Well-Being Index)
    public float CalculateWBI()
    {
        return (Health + Morale + (100f - Stress)) / 3f;
    }
}
```

ДОДАТОК Б

Конфігураційні таблиці ігрових івентів (Data Layer - ScriptableObjects)

Таблиця Б.1 – Фрагмент бази даних мікро-івентів (MicroEventsPool)

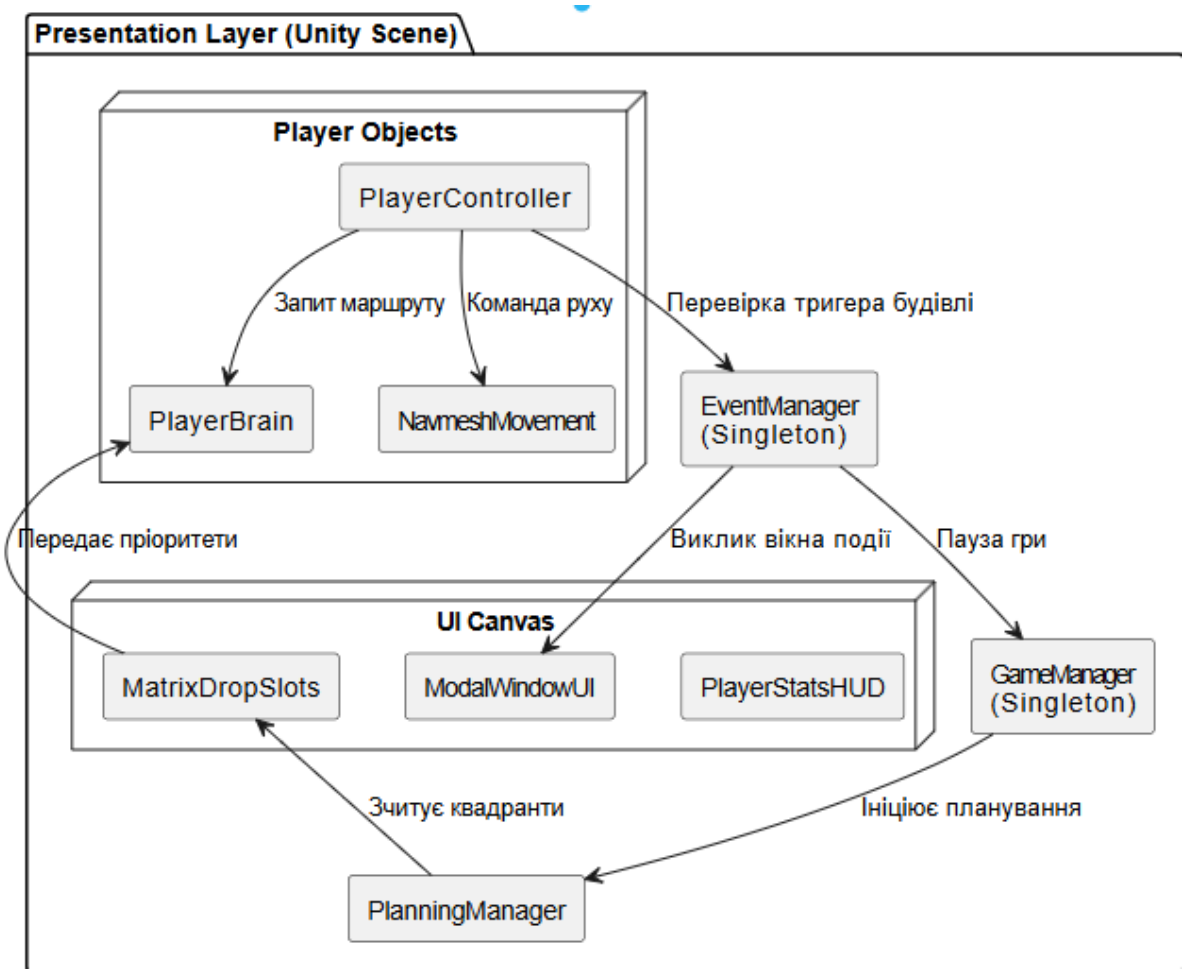
Назва івенту	Цільова сфера	Модифікатори статів (Дельти)	Умова спрацьовування	Візуальний пресет AI
Нічний кодинг	Career	Career +15, Stress +20, Health -10	Вхід у будівлю "Офіс"	Overworked_Loop.mp4
Знахідка під диваном	Internal World	Morale +15, Stress -10, Career -5	Вхід у "Храм/Дім"	Calm_Rest_Loop.mp4
Сезонний вірус	Health	Health -20, Morale -10, Stress +5	Вхід у "Клініку"	Sick_Cough_Loop.mp4
Зустріч у барі	Socialization	Socialization +20, Morale +15, Health -5	Вхід у "Бар"	Happy_Talk_Loop.mp4

ДОДАТОК В

Блок-схема алгоритму обробки фаз ігрового дня

ДОДАТОК Г

Структурна схема взаємодії менеджерів Presentation-рівня



ДОДАТОК Д

Візуалізація розроблених екранних форм інтерфейсу

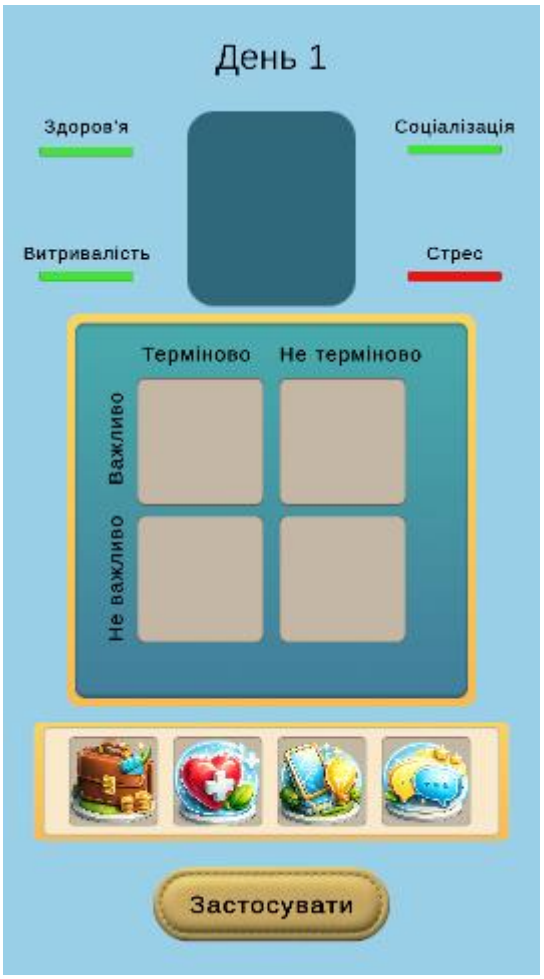


Рисунок Д.1 – Інтерфейс ранкового планування (Матриця Ейзенхауера).

Рисунок Д.2 – Ігрова радар-карта денної симуляції.

ДОДАТОК Е

Результати верифікації та тестування системи

Рисунок Е – Логи консолі під час успішного проходження ігрового циклу.

ДОДАТОК Ж

Кількісні показники ефективності функціонування мобільного додатка

Таблиця Ж.1 – Порівняльний аналіз споживання ресурсів до та після впровадження оптимізації (RenderTexture та Addressables)

Метрика тестування (Android Пристрій середнього класу)	Базова реалізація (Прямі посилання та 2 VideoPlayer)	Оптимізована архітектура (Проєктна ІС)	Покращення (%)
Середня частота кадрів у Core Loop (FPS)	42	60	+ 42%
Використання оперативної пам'яті відео (RAM), МБ	~245	~110	- 55%
Затримка виклику модального вікна (Мікрофриз), мс	135	18	- 86%
Навантаження на CPU під час івенту, %	72	34	- 52%

ДОДАТОК З

Інструкція користувача та адміністратора контенту

3.1. Інструкція Гравця (Користувача)

1. Планування: Щоранку розподіляйте фішки сфер життя (Кар'єра, Здоров'я, Соціалізація, Внутрішній світ) по 4 квадрантах матриці Ейзенхауера. Пам'ятайте, що квадрант Q1 (Важливо і Терміново) забере найбільше часу вашого дня (8 секунд симуляції), а Q4 — найменше (3 секунди).
2. Контроль статів: Уважно слідкуйте за показниками у верхній частині екрана. Якщо рівень стресу (Stress) перевищить 70, ваш персонаж почне рухатися на 30% повільніше.
3. Вигорання: Не допускайте падіння показника Моралі (Morale) до нуля. Це викличе стан "Апатія", що вріже вашу ефективність на роботі на 70%. Щоб відновити мораль, регулярно ставте сферу "Внутрішній світ" (Храм) у високі пріоритети.
4. Виживання: Гра не має респавну. Якщо Health впаде до 0, персонажа буде госпіталізовано і гра закінчиться. Ваша мета — прожити 100 днів та отримати фінальний вердикт вашого стилю життя.

3.2. Інструкція Адміністратора контенту (Розробника)

1. Додавання нових подій: Для створення нового мікро-івенту не потрібно писати код. У вікні Project в Unity натисніть Right Click -> Create -> Game -> Event Config. Заповніть текстові поля, виставте модифікатори статів та додайте цей файл у відповідний список пулу в об'єкті EventManager на сцені.
2. Оновлення відеолупів: Щоб додати нову ШІ-анімацію, завантажте файл у форматі .mp4 (H.264) у теку проєкту. Обов'язково позначте його галочкою в системі Addressables (вікно Addressables Groups), щоб уникнути завантаження відео в оперативну пам'ять на старті гри. Потім вкажіть AssetReference на це відео у файлі потрібної події (EventConfigSO).