МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ЯВГУСІШИН БОГДАН АНАТОЛІЙОВИЧ

Допускається до захисту:
Завідувач кафедри
інформаційних технологій,
кандидат технічних наук, доцент
_____ Зелінська О.В.
«_ __ _» _____ 2024р.

РОЗРОБКА КАЗУАЛЬНОЇ ГРИ «CODE FROST: CYBERNETIC BATTLE» НА ОСНОВІ C# ТА UNITY

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Науковий керівник:
Антонов Ю.С., доцент кафедри
інформаційних технологій,
кандидат фіз.-мат. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____

Вінниця – 2024

АНОТАЦІЯ

Явгусішин Б.А. Розробка казуальної гри «Code Frost: Cybernetic Battle» на основі C# та Unity. Спеціальність 122 «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У дипломній роботі досліджено процес створення казуальної відеоігри від концепції до реалізації. Робота складається з трьох основних розділів. У першому розділі аналізується сучасний стан ігрової індустрії, зокрема види та жанри ігор, платформи для розробки та ігрові рушії. Другий розділ присвячено розробці дизайн-документу гри, де детально описуються ігрові об'єкти, їх логіка, ігрові механіки. Третій розділ включає технічні аспекти реалізації проекту та процес розробки.

Ключові слова: ігровий рушій Unity, казуальна гра, дизайн документ гри, Zeneject, MVC.

59 с., 29 рис., 12 фор., 60 джерел.

ABSTRACT

Yavhusishyn B.A. Development of the Casual Game «Code Frost: Cybernetic Battle» Based on C# and Unity. Specialty 122 "Computer Science". Vasyl Stus Donetsk National University, Vinnytsia, 2024.

The thesis researched the process of creating a casual video game from concept to implementation. The work consists of three main sections. The first chapter analyzes the current state of the game industry, in particular, game genres, development platforms, and game engines. The second section is devoted to the development of the game design document, which describes in detail the game objects, their logic, and game mechanics. The third section includes the technical aspects of project implementation and the development process.

Keywords: Unity game engine, casual game, game design document, Zeneject, MVC.

59 p., 29 pic., 12 formulas, 60 sources.

ЗМІСТ

ВСТУП

Актуальність роботи:

Сучасна ігрова індустрія залучає мільйони гравців з усього світу та отримує великі прибутки. Попри глобальну рецесію та зменшення ігрового ринку через введення обмежень у Китаї, ринок був в передбаченому падінні після двох років зростання в зв'язку з пандемію. Наразі ринок стабілізується, що є гарною можливістю для просунення власних ігрових додатків. На світовому ринку ігрової індустрії найуспішнішим та найприбутковішим сектором є ігри для мобільних пристроїв. Ігри казуального жанру займають особливе місце на ринку завдяки своїй доступності та широкій аудиторії. Вони приваблюють гравців різних вікових груп та рівнів підготовки, забезпечуючи легкість входження у гру та зрозумілий геймплей.

Розробка казуальної гри «Code Frost: Cybernetic Battle» є актуальною не лише з точки зору зростання ринку відеоігор та попиту на казуальні ігри, але й з огляду на можливості, які вона відкриває для навчання, професійного розвитку та впровадження новітніх технологій у процес створення ігрових додатків.

Мета дослідження: Створення ігрового додатку в казуальному жанрі для мобільних платформ на базі ігрового рушія Unity та мови програмування C#.

Завдання дослідження:

- огляд сучасних платформ для розробки ігор, ігрових рушіїв та існуючих казуальних ігор;
- створення дизайн документу гри;
- реалізація ігрового додатку.

Об'єкт дослідження: Процес розробки ігрового проєкту для мобільних платформ.

Предмет дослідження: Теоретичні та практичні підходи до процесу створення ігрового додатку з використанням ігрового рушія Unity, мови програмування C# та фреймворку Zenject.

Апробація результатів дослідження: результати досліджень апробовані на V Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих вчених «Прикладні інформаційні технології». (м.Вінниця 23.05.2024 р.)

Структура кваліфікаційної роботи: Бакалаврська робота складається із вступу, трьох розділів та висновків та списку використаних джерел.

Робота містить 59 сторінок, 29 рисунків, 12 формул та список літератури з 60 джерел.

РОЗДІЛ 1. СУЧАСНИЙ СТАН ІГРОВОЇ ІНДРУСТІЇ

1.1 Види та жанри ігор

Ігрова індустрія є важливою складовою сучасної культури, відображаючи не лише технологічний прогрес, але й соціокультурні та естетичні тенденції. За останні десятиліття вона перетворилась на багатомільярдну галузь, що об’єднує людей по всьому світу [1]. Ігри надають можливість незалежно від віку та статі побувати в віртуальних світах та відчути себе героями, дослідниками чи творцями. Ігри розподіляються на різноманітні жанри, що включає в себе, наприклад, платформери, візуальні новели, інтерактивне кіно, симулятори та багато інших. Кожен з жанрів має свої унікальні особливості та аудиторію.

Один з найпопулярніших жанрів є екшн (action). Він включає в себе швидкий геймплей, бойові сцени та основний акцент робить на рефлекторних здібностях гравця. Даний жанр представлений такими іграми як «Gog of War (2018)», «Devil May Cry 5» та «Assassin`s Creed IV: Black Flag».

Рисунок 1.1 – «Gog of War (2018)» [2]

Інший поширений жанр – стратегія (strategy), де гравець керує переважно обмеженими ресурсами та/або військами, щоб досягти перемоги треба приймати

рішення базуючись не тільки на короткочасних перемогах, а і на довгочасних. Представниками жанру є «Civilization VI», «StarCraft II», «Age of Empires II».

Рольові ігри (RPG) пропонують інтерактивне та наративне дослідження і взаємодію з ігровим світом, де гравець може відчути себе його частиною, приймаючи рішення відносно ігрових подій. Частим елементом даного жанру є можливість прокачки персонажа, що впливає на стиль проходження гри і надає варіативність проходження для кожного гравця. Популярними рольовими іграми є «Fallout: New Vegas», «Witcher 3: Wild Hunt» та «Final Fantasy XVI».

Рисунок 1.2 – «Witcher 3: Wild Hunt» [3]

Також ігри часто поєднують жанри, що створює нові, а іноді і унікальні, геймплейні (ігрові) можливості. Наприклад, гра «Divinity: Original Sin 2» поєднує в собі елементи рольової гри та стратегії, що створює глибоку та іммерсивну систему бойових вмінь та взаємодій з навколишнім ігровим всесвітом.

Велика кількість сучасних ігор не створюються на основі лише одного жанру, а й використовують особливості інших жанрів для збагачення геймплею. Наприклад, «The Legend of Zelda: Breath of the Wild» поєднує в собі елементи пригодницької відеогри та відкритого світу, пропонуючи гравцеві величезну територію для дослідження, а також складні головоломки та механіки

виживання, які представлені в форматі управління спорядженням, приготуванням їжі і тд.

Рисунок 1.3 –«The Legend of Zelda: Breath of the Wild» [4].

В історії ігрової індустрії були й випадки, коли відеогри ставали не лише популярними, але й впливали на розвиток жанрів або навіть створювали нові. Прикладами даних ігор є:

- серія ігор «Dark Souls». Перша частина випущена в 2011 році. Серія поєднує в собі елементи екшну та рольових ігор з нетипово високим рівнем складності. Успіх серії з великою кількість фанбази привело до прояви піджанру, відомого як «Soulslike», що характеризується високою складністю зі складними бойовими механіками.
- «PlayerUnknown's Battlegrounds». Випущена в 2017 році. Популяризувала жанр «battle royale» (королівська битва), де десятки гравців змагаються на територій в формі острову. З часом зона, де можна знаходитись без зниження показника здоров’я, зменшується, що призводить до неминучих сутичок. Матч завершується, коли залишається лише один гравець або одна команда.

- «Metroid» і «Castlevania» визначили новий піджанр, відомий як metroidvania (метроїдванія). Цей жанр поєднує в собі елементи ігрової механіки та ігрового процесу, подібні до вищезгаданих серій, і відкрив нові можливості для створення захоплюючих іммерсивних світів для гравців. Вони часто включають в себе нестандартні системи прокачки персонажа, де гравцеві потрібно знаходити або вдосконалювати різні здібності, щоб просуватися далі в грі. Також вони відомі «відкритими» світами, які гравець може досліджувати в нелінійному порядку, розв'язуючи головоломки та знаходячи секрети.

1.2 Сучасні платформи для розробки ігор

Сучасний ігровий ринок підтримує широке розмаїття платформ для яких створюються ігри. Найпопулярніші з низ включають в себе консольні системи, персональні комп’ютери, мобільні пристрої та як пристрої віртуальної реальності (VR), та і – доповненої реальності (AR).

Зазвичай спочатку створюють для однієї платформи, а для інших портують з часом, зважаючи на технічні особливості, але деякі видавці буває роблять одночасний реліз на декількох платформах для більшого охоплення аудиторії. Переважно вибір дистрибуції відеогри залежить від багатьох факторів, включаючи штат працівників, фінансові можливості компанії та обрану цільову аудиторію.

Наприклад, консольні системи, такі як PlayStation, Xbox та Nintendo Switch, мають свої унікальні особливості і екосистеми. PlayStation традиційно визнаний за його ексклюзивні ігри та високу якість графіки. Xbox забезпечує інтеграцію з сервісами Microsoft та акцентується на мережевій грі та хмарних послугах. Nintendo Switch пропонує ігри для всієї родини та ігри з унікальним контролером.

У світі персональних комп’ютерів (ПК) ігор головними платформами дистрибуції є Steam, Epic Games Store, Battle.net, GOG (Good Old Games) та інші. Steam, запущений у 2003 році, є найбільшим цифровим магазином ігор у світі з

великим асортиментом ігор та соціальними функціями. Epic Games Store, заснований у 2018 році, відомий своїми ексклюзивними угодами та акціями для гравців. Деякі акції проводяться з постійною періодичністю. Наприклад, безкоштовна роздача відеоігор різних років випуску та популярності. Battle.net від Blizzard, який запустився ще в 1996 році, є основним місцем для гри в ігри, розроблені компанією Blizzard, такі як "World of Warcraft", "Overwatch" і "Hearthstone". Ця платформа також надає можливість спілкування з іншими гравцями та придбання додаткового контенту для ігор. GOG, запущений у 2008 році, спеціалізується на старих іграх без DRM-захисту та створив свою власну спільноту гравців.

1.3 Сучасні ігрові рушії

Розробка ігор починається з вибору редактору для створення ігор, тобто ігрового рушія. Вони надають розробникам широкий спектр інструментів для втілення творчості у форматі ігрового контенту.

Деякі розробники/компанії використовують розроблені власноруч, деякі покупають ліцензії на використання розробок інших компаній. Зазвичай перші вже багато років використовують власний рушій та мають штат розробників, які його покращують та впроваджують новий функціонал. До даної категорії переважно відносяться розробники та компанії, що розробляють AA/AAA-ігри. Такі ігри відзначаються великими бюджетами, командами розробниками та головною ціллю є велика кількість проданих копій на більшості ринках.

Другим ж, в свою чергу, вигідніше не починати розробку ігор з самого початку, так як це великі витрати грошей та часу, тому краще взяти якийсь готовий продукт і вже його підлаштувати під себе. Головними представниками є indie-розробники, що працюють самостійно або в невеликій компанії та створюють ігри з відносно невеликим бюджетом, часто з фокусуванням на введення новизни в обраний жанр.

Ігрових рушіїв є велика кількість та з різними перевагами. Кожен зможе обрати те, що йому більше підходить. Популярними представниками даного програмного забезпечення є:

- Unity - один з найпоширеніших ігрових рушіїв у світі, який використовується для створення різноманітних ігор з можливістю дистрибуції на велику кількість різних платформ, включаючи мобільні пристрої, персональні комп'ютери, консолі та веб [5]. Широкий спектр підтримуваних платформ є однією з ключових переваг, що надає рушій в порівнянні з іншими, так як це надає можливість охопити більшу кількість кінцевих користувачів ігрового додатку. Він славиться гнучкістю в розробці, так як в рушії реалізовано компонентний архітектурний підхід, що дає можливість як для початківців, так і для досвідчених розробників, використовувати чужі або власні створенні компоненті частини, які можуть включати код, 3D-моделі, музику і тд. Наприклад, система генерації рельєфу місцевості або різні реалізації моделі персонажів. Unity має велику спільноту розробників, що сприяє обміну знаннями та розвитку навичок.

Рисунок 1.4 – Зображення з інтерфейсу рушія Unity [6]

- Unreal Engine 5 є потужним ігровим рушієм, відомим своєю вражаючою фотореалістичною 3D-графікою. Переважно використовується для розробки AAA-ігор та інших високобюджетних проєктів завдяки своїй високій продуктивності та можливостям. Unreal Engine 5 надає розробникам безкоштовний доступ до програмного коду (source code), що дозволяє налаштовувати рушій під конкретні потреби проєкту або привносити покращення в сам рушій. Однак, для новачків він може бути складним у освоєнні через великий обсяг функцій та складність деяких інструментів не в останню чергу через використання C++, як основної мови програмування. Крім того, Unreal Engine 5 має велику кількість підтримуваних платформ, але порівняно з Unity може бути потрібна деяка додаткова оптимізація додатку для деяких платформ, що може бути обмеженням для розробників, які прагнуть охопити широку аудиторію гравців [7].

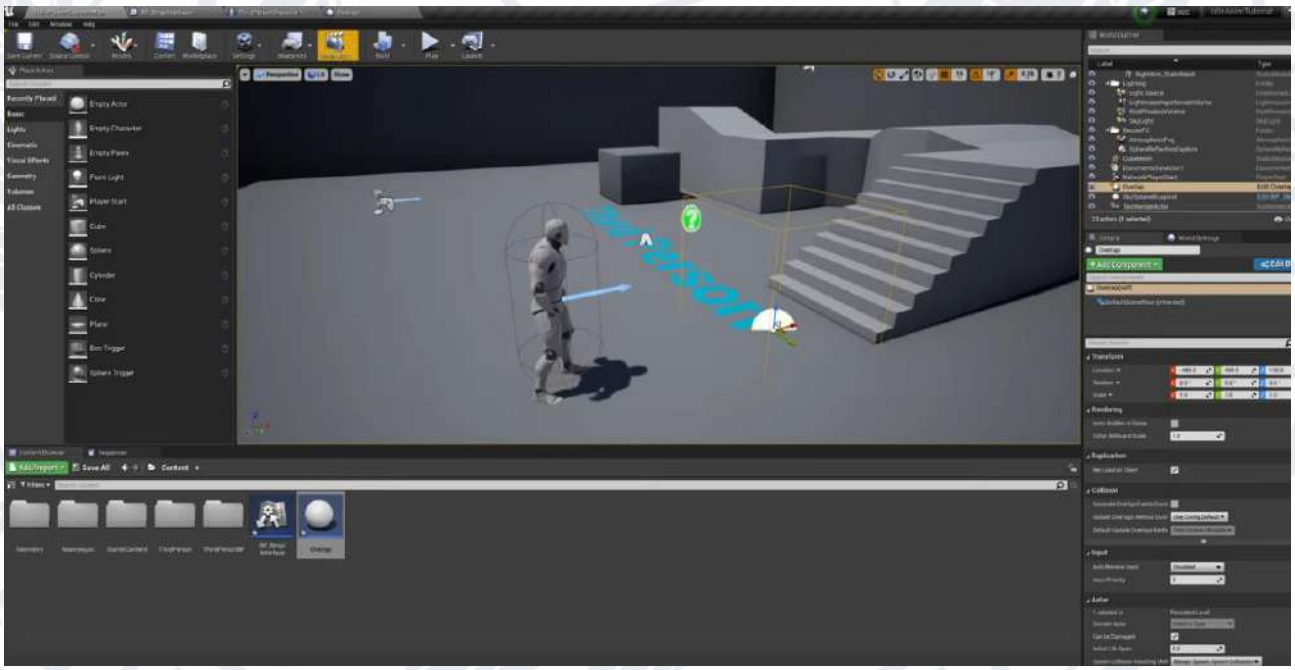

Рисунок 1.5 – Зображення з інтерфейсу рушія Unreal Engine 5 [8]

- Godot - це інший популярний рушій для розробки відеоігор, відомий своєю легкістю в освоєнні та використанні. Даний рушій як і Unreal Engine має відкритий доступ до програмного коду, але окрім C++ використовує власну мову програмування GDScript, яка схожа на Python. Окрім того на

відміну від більшості інших готових рушіїв розповсюджується на безкоштовній основі з ліцензією MIT. (MIT license, which means developers can use the engine for commercial projects without being required to disclose their project's code or any other resources provided alongside it.) Надає можливості для портування ігор на різні платформи, включаючи мобільні пристрої, ПК та віртуальна реальність, але не на консолі. Також рушій має доволі низькі вимого апаратного забезпечення для роботи редактору в порівнянні з іншими рушіями. Однак, порівняно з Unity та Unreal Engine, Godot має меншу кількість доступних ресурсів в форматі asset'ів та меншу спільноту, переважно indie- розробників [9].

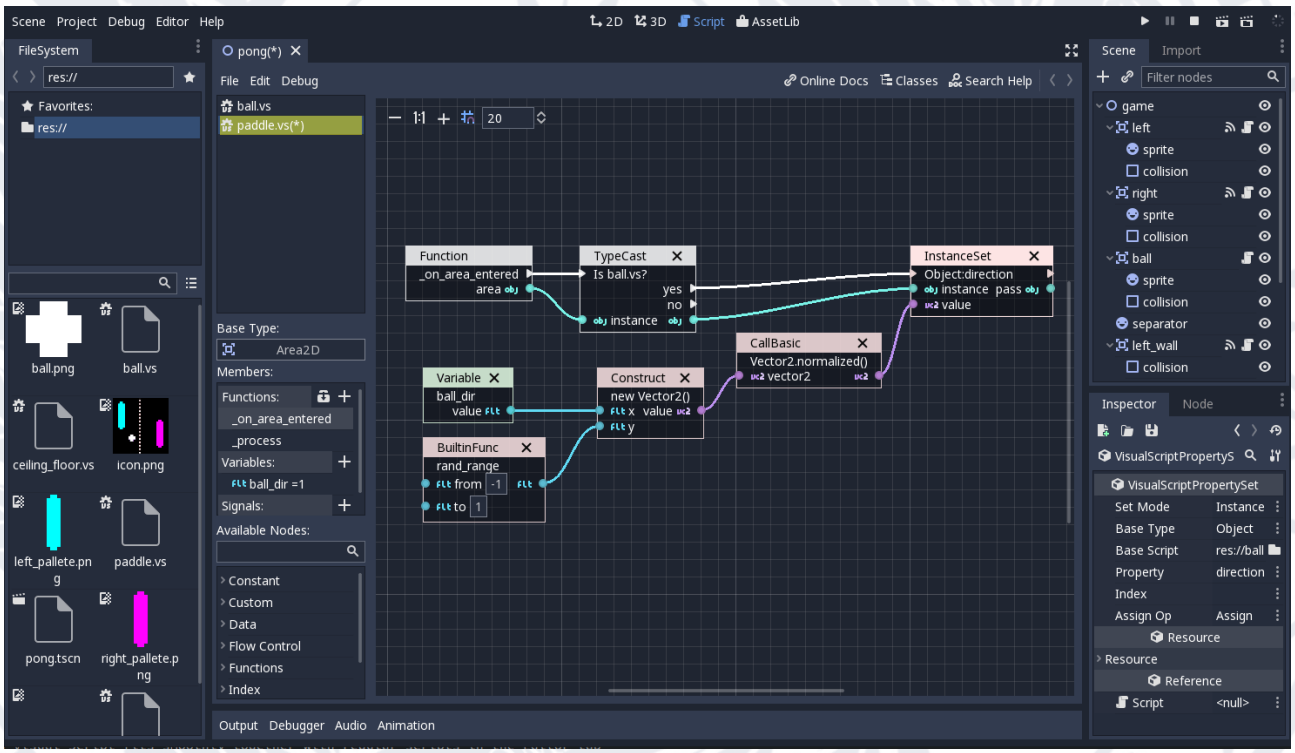

Рисунок 1.6 – Зображення з інтерфейсу рушія Godot [10]

- Source 2 - це ігровий рушій, розроблений компанією Valve Corporation. Цей рушій є наступником Source, який використовувався в розробці, таких відомих ігор, як Half-life, Counter-Strike: Global Offensive та є відомий за досягнення у фізиці, ШІ та графіці. Source 2 вперше був анонсований в 2014, а вже в 2015 Dota 2 була портована на нього. Однією з його ключових переваг є вбудована підтримка віртуальної реальності, що дозволяє створювати ігри з іммерсивним геймплеєм [11]. Чудовим прикладом є

Half-Life: Alyx, яка здобула високі оцінки від критиків та гравців не в останню чергу за вражаюче поєднання атмосфери та іммерсивності досягнутого за допомогою VR [12]. На противагу до цього рушій в порівнянні з іншими має відносно малу кількість ігор, які розробили сама компанія Valve, так як він ще розробляється та покращується перед випуском в маси. Хоча варто зазначити, що деякі розробники мають доступ до рушія.

Незалежно від вибору зробленого розробниками/компанією рушій є лише частиною успіху ігрових додатків, що може як пришвидшити розробку, так і уповільнити. Переважно це залежить від рівня знань команди розробників, кількість напрацьованого матеріалу та доцільності використання в реалізації ігрового додатку.

1.4 Сучасні мови програмування в ігровій індустрії

Мова програмування, що використовуватиметься в розробці ігрового додатку, головним чином залежить декількох аспектів, таких як підтримки в обраному ігровому рушії та досвіду роботи команди розробників.

В ігровій індустрії переважно використовуються:

- C++. Компільована, багатопарадигмова мова програмування, створена Б'ярном Страуструпом у 1983 році як розширення мови С. Ключовою особливістю є висока продуктивність та ефективність, що досягається завдяки прямому доступу до пам'яті та низькорівневим операціям. Як наслідок даної переваги є великий набір бібліотек та інструментів для розробки ігор, таких як DirectX, OpenGL та Vulkan, що свідчить про широку підтримку та використання в ігровій індустрії, особливо для ААА-ігор. Наприклад, використовується як основна мова програмування для рушія Unreal Engine, відомого за такі проєкти як: Little Nightmares, Sea of Thieves та Borderlands 3. На противагу до переваг є складність керування пам'яттю, так як через відсутність вбудованого garbage collector є можливість появи витоків пам'яті через ручне керування. Також мова має

крута криву навчання, особливо для початківців, через складний синтаксис та концепції [13].

- C#. Об'єктно-орієнтована мова програмування, яка була розроблена Microsoft у 2000 році. Як і C++ є кросплатформеним та має інтеграцію в Microsoft Visual Studio - однією з найпопулярніших інтегрованих середовищ розробки (IDE). C# є офіційною мовою програмування для Unity. В порівнянні з C++ має вбудований garbage collector для автоматичного керування пам’яттю. Також перевагами є зручний і простіший у вивченні синтаксис та доступ до великого спектру бібліотек і інструментів, що дозволяє пришвидшити розробку та оптимізацію додатків. C# також підтримує багатопотоковість і паралельне програмування, що має вирішальне значення для розробки ігор, так як дозволяє ефективно використовувати апаратні ресурси [14].
- Java. Об'єктно-орієнтована мова програмування, розроблена Sun Microsystems у 1995 році. Вона була створена з метою забезпечення портування між різними платформами за допомогою концепції "один раз написано, всюди запущено". Також з ключових переваг варто зазначити наявність garbage collector та багату екосистему бібліотек та інструментів, що включає в себе OpenGL, JOGL та ігровий рушій LibGDX. В порівнянні з C++ є менш продуктивною через наявність віртуальної машини Java (JVM) та автоматичне керування пам’яттю. В ігровій індустрії на даній мові були створенні Tom Clancy's Politika, Runescape, Powder Game, Star Wars Galaxies, Roboforge [15]. Java є легшою для вивчення, ніж C++, але вона все одно може бути складною для початківців. Однак він не такий гнучкий, як C++, і не так широко використовується в колах розробників ігор [16].
- Python. Інтерпретована, високорівнева мова програмування із зручним синтаксисом та акцентом на читабельність коду. Python менш широко використовується у світі розробки ігор, але це хороша відправна точка для нових розробників. Для розробки ігор використовується безкоштовна

бібліотека PyGame. Вивчення Python також полегшує використання GDScript, мови програмування для ігрового рушія Godot [17].

- GDScript. Скриптова мова високого рівня, розроблена спеціально для ігрового рушія Godot. Дану мову легко освоїти, особливо якщо ви знайомі з Python, оскільки його синтаксис і структура досить подібні. Була розроблена з метою надати зручний і легкий для освоєння інструмент для створення ігор. GDScript є потужним і універсальним, що дозволяє створювати складну ігрову логіку з мінімальними зусиллями. Основною перевагою використання є тісна інтеграція з механізмом і редактором Godot. GDScript призначений для швидкого прототипування та ітерації [18].
- Swift. Мова програмування, розроблена Apple для розробки додатків для iOS, iPadOS, macOS, watchOS та tvOS. Сучасний синтаксис, заснований на Objective-C, але з покращеннями, такими як безпека типів, автоматичне перевизначення посилань, багатопотоковість та лямбда-вирази. Висока продуктивність, близька до C, завдяки оптимізованому компілятору LLVM. Використовується для розробки ігор для платформ Apple за допомогою SpriteKit, SceneKit та інших фреймворків [19].
- Kotlin. Стаціонарна мова програмування, розроблена JetBrains як альтернатива Java для Android-розробки. Сумісна з Java і може бути використана в існуючих Java-проєктах. Забезпечує більш лаконічний та безпечний синтаксис порівняно з Java. Підтримується Google як офіційна мова для Android-розробки, включаючи розробку ігор з використанням Android SDK [20].
- JavaScript. Скриптова мова програмування, спочатку розроблена для забезпечення інтерактивності веб-сторінок. З появою платформ HTML5 та WebGL, JavaScript став використовуватися для розробки як веб-ігор, так і нативних додатків за допомогою технологій, таких як Electron та Node.js. Величезна екосистема бібліотек і фреймворків для розробки ігор, включаючи Pixi.js, Phaser, BabylonJS та Three.js. Можливість розробляти

крос-платформні ігри, які працюють у веб-браузерах та на настільних і мобільних пристроях. Популярності у використанні JavaScript для розробки ігор відбувся після закінчення ери Flash Player. Багато розробників застосували цю технологію для створення браузерних ігор. Він демонструє, як розробники ігор можуть використовувати JS і HTML5 для представлення візуальних елементів просто в Інтернеті. Ось список найпопулярніших онлайн-ігор, створених за допомогою JavaScript: 2048, Pseudo 3D Racer, Dinosaur Game, Tower Building та Mk.js [21].

- Rust. Системна мова програмування, що була офіційно анонсована у 2010 році. Вона була розроблена з акцентом на продуктивність, безпеку пам'яті та паралельне програмування. Як правило, Rust принаймні такий же швидкий, як C/C++, і в майбутньому він може стати ще швидшим через майбутні оновлення продуктивності мови. Rust також має великий потенціал для розробки ігор завдяки паралелізму. Паралелізм у Rust запобігає перегонам даних і забезпечує ефективне керування пам’яттю, що унеможливлює збій у роботі вашої програми. Розробка ігор, як направлення, все ще розвивається в екосистемі Rust, і багато існуючих ігрових рушіїв і бібліотек все ще перебувають у стадії активної розробки [22].

1.5 Опис жанру

Казуальні мобільні ігри - це ігри, які призначені для широкої аудиторії користувачів, так як мають невелику кількість механік. Вони зазвичай мають прості механіки, що не потребують глибокого вивчення для комфортної гри, легке управління та короткі ігрові сесії. Дані характеристики роблять їх ідеальними для гри в дорозі, на перервах або в будь-який час, коли користувач має вільний час [23].

Хоча спочатку казуальні ігри були популярні на комп’ютерах, історія казуальних мобільних ігор почалася разом із зростанням популярності мобільних пристроїв, таких як смартфони та планшети, і розвитком магазинів

додатків, таких як App Store та Google Play. Перші казуальні ігри були досить простими ігровими механіками, але з часом вони стали більш різноманітними та складнішими.

Одним із перших прикладів популярної казуальної мобільної гри є "Angry Birds", розроблена компанією Rovio Entertainment Ltd.. Вона була випущена в 2009 році і миттєво стала хітом завдяки своїй простій геймплейній механіці, яскравій графіці та аркадному характеру. Гравці контролюють групу пташок, які падають у різноманітні структури, щоб знищити своїх ворогів — свиней. Ця гра виявилася настільки популярною, що породила безліч продовжень та спін-оффів, а також перетворилася на медійну франшизу.

Інший приклад - "Candy Crush Saga", розроблена компанією King Digital, яка була випущена в 2012 році. Ця гра представляє собою головоломку типу "три в ряд", де гравцям потрібно збирати різнокольорові цукерки у лінії, щоб вони зникли з екрану. "Candy Crush Saga" швидко стала однією з найпопулярніших мобільних ігор у світі, завдяки своєму простому, але дуже привабливому геймплею та безлічі рівнів, які пропонують гравцям безкінечні виклики.

Розробники казуальних ігор для отримання прибутку покладаються на різноманітні та ефективні механізми монетизації, наприклад:

- Внутрішньо ігрові покупки (In-App Purchases): Багато казуальних ігор пропонують гравцям можливість придбати внутрішню валюту або предмети, що полегшують гру або додають до неї різноманітності. Наприклад, гравці можуть купувати нові рівні, спеціальні предмети або вигадані валютні одиниці. Часто гравці відчувають нетерпіння щодо свого прогресу в грі, тому вони змушені купувати додаткові речі, які прискорять їх прогрес у грі [24].
- Реклама: Безкоштовні казуальні ігри часто монетизуються за допомогою відображення реклами. Це може бути реклама у вигляді відеороликів, банерів або інтерактивних оголошень, які гравці бачать під час гри або на паузі між рівнями. Доходи від реклами в програмі швидко наздоганяють покупки в програмі за часткою доходу від казуальних ігор. Проблема, з

якою стикаються багато ігор цього жанру, пов’язана із перенасиченням додатку рекламою, що призводить до зменшення кількості кінцевих користувачів [24].

- Ускладнені рекламні моделі: Деякі ігри використовують ускладнені рекламні моделі, такі як "реклама за винагороду". Наприклад, гравець може отримати бонуси або внутрішню валюту, якщо він перегляне рекламний ролик або встановить інший додаток.

Дані формати монетизації зумовлені тим, що виробництво казуальних ігор є відносно недорогим, і вони зазвичай безкоштовні.

У категорії казуальних ігор можна знайти головоломки, аркади, гіпер-казуальні ігри, симулятори та AR/локаційні ігри. Деякі з найкращих казуальних ігор включають Township, Family Island, Temple Run, Fruit Ninja, Homescapes, Subway Surfers, Project Makeover і Crossy Roads [25].

1.6 Використання дизайн документів для розробки ігор.

Створення ігор починається з концептуальної ідеї, що була натхненна пережитими враженнями в іншій грі або від власного досвіду. Але дана ідея є гарно відомою лише тому ким була придумана і може бути доволі нечіткою у розумінні для інших, а може навіть відрізнятися від оригінальної версії. Для запобігання даних ситуацій створюється дизайн документ гри. Даний документ містить детальний опис концептуальної ідеї, тобто усіх аспектів гри, від геймплею з механіками до графіки та звукового супроводу. Він є одним з ключових інструментів для розробників гри, щоб узгодити та зберегти всі ідеї, концепції та вимоги до гри, що допомагає забезпечити чітке розуміння завдань для всіх учасників команди, таких як програмісти, дизайнери, художники та інші.

Дизайн документи не мають якоїсь загальної узгодженої структури, але вони можуть містити в собі такі розділи, як:

- Загальний огляд і концепція гри: Опис загальної концепції гри, жанру, цільових користувачів, загальний наратив сюжету, атмосфери, тематики і основної ідеї гри [26].

- Механіки гри: Розділ має включати опис основних ігрових можливостей, наприклад, таких як управління, бойова система , система прогресу тощо, тобто опис взаємодіє гравця з ігровим додатком та як вона буде реагувати на його дії. Також перевагою є наявність візуальна ілюстрація механік [27].
- Персонажі: Опис головних та другорядних персонажів, а саме опис їхньої сюжетної лінії, їх знання, навички та особистості кожного [28].
- Локації та рівні: Опис структури локацій, рівнів. Додаються їх ескізи, концепт-арти, що може допомогти візуалізувати дизайн. Виділяються ключові особливості та цілі, що будуть поставлені перед гравцем [26].
- Графіка і анімація: Опис арт-стилю, концепт-арту, моделей персонажів та оточення, анімації, спеціальні ефекти.
- Звуковий дизайн: Опис звукового середовища гри, музики, звукових ефектів, що , наприклад, будуть відтворюватися при взаємодії гравця з об’єктами.
- Інтерфейс користувача: Має містити опис інтерфейсу гри, включаючи детальні пояснення щодо взаємодії користувача з елементами меню, HUD або як користувачі можуть користуватися складними візуальними системами, такі як керування інвентарем [29].
- План розробки: Даний пункт може включати цілі та задачі поставлені на певні періоди, такі як дедлайн закінчення реалізації першого прототипа зі всіма механіками, дата публічного оголошення проєкта або проведення закритого бета-тестування для отримання відгуків.
- Бізнес-план: Містить оцінку витрат, шляхи отримання прибутків, опис маркетингової стратегії, аналіз конкурентів. Можуть бути наявні подробиці відносно ресурсів потрібних від команди для створення рекламної кампанії [27].

Дизайн документ гри використовується на різних етапах життєвого циклу проєкту, що дозволяє забезпечити чітку спрямованість розробки під час кожного етапу та зменшити можливість непорозумінь та конфліктів серед команди. Крім того, він може бути використаний як основа для оцінки прогресу та визначення

відповідності результатів очікуванням. Ось деякі з основних етапів, на яких він може бути використаний:

- Планування: На початковому етапі розробки проєкту дизайн-документ визначає загальну концепцію гри, визначає основні механіки та функціональність, що допомагає встановити параметри та обсяг проєкту.
- Розробка: Під час цього етапу дизайн-документ служить основним джерелом інформації для всіх членів команди розробників. Він допомагає програмістам розуміти, як повинні функціонувати різні елементи гри, художникам — як створювати графіку та анімацію, а тестерам — на що треба звернути увагу під час тестування.
- Контроль якості: Дизайн-документ використовується для порівняння реального продукту з задокументованими вимогами. Він служить важливим джерелом для тестувальників та QA-інженерів, щоб перевірити, чи відповідає гра задокументованим стандартам та очікуванням.
- Розширення та оновлення: Під час розвитку гри і випуску оновлень або додаткового контенту, дизайн-документ може використовуватися як основа для розробки нових функцій або вдосконалень.
- Документація: Наприкінці проєкту дизайн-документ стає важливим джерелом документації для подальшого обслуговування гри. Він може використовуватися для навчання нових членів команди, а також для виправлення помилок або вдосконалення гри в майбутньому.

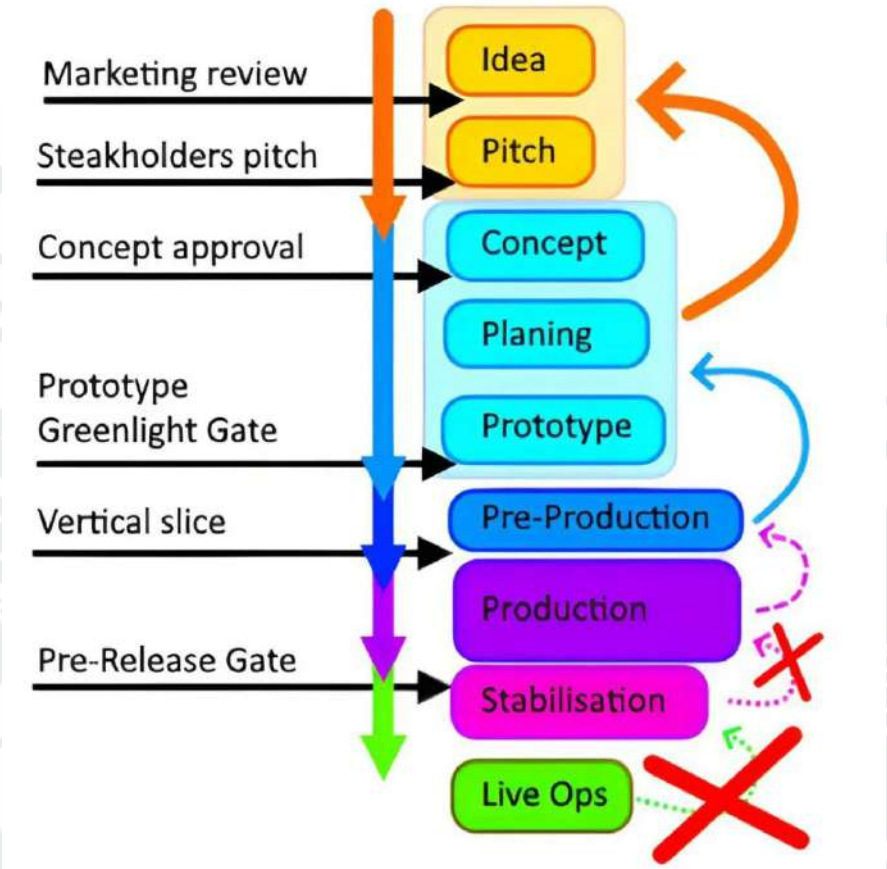

Рисунок 1.7 – Схематичне зображення використання дизайн документа гри в життєвому циклі проєкта [28]

Висновок до розділу 1

В першому розділі був зроблений загальний огляд стану сучасної ігрової індустрії. Були розглянуті різноманітні види та жанри ігор, сучасні платформи для дистрибуції ігрових додатків. Також було проаналізовані та розглянуті сучасні ігрові рушії з їх перевагами, які мови програмування використовуються для розробки ігор та використання дизайн документів для розробки ігор.

РОЗДІЛ 2. ДИЗАЙН ДОКУМЕНТ ГРИ

2.1 Опис гри

Гра "Code Frost: Cybernetic Battle" буде виконана в казуальному жанрі, який має велику популярність на мобільних платформах. В притаманному до даного жанру особливостей ігрові механіки будуть створені в легкому та зрозумілому форматі та в невеликій кількості. З основних можливостей це керування персонажем одним пальцем та автоматична атака цілей. Це робиться з метою швидкого освоєння гри кінцевими користувачами. В подальших пунктах буде більше детально розписано за ігрові можливості.

2.2 Структура ігрового додатку

Ігровий додаток буде розроблена з урахуванням на одного гравця. При заході користувача в гру буде завантажуватися сцена завантаження, що буде ініціалізовувати основні системи ігрового додатку. При завершені даного процесу буде завантажена головна ігрова сцена. Спочатку буде увімкненою UI головного меню, що буде мати чотири кнопки для взаємодії з грою. При натисканні відповідних кнопок користувач зможе відкрити такі UI вікна, як: налаштування, магазин покращення зброї, магазин покупки ігрової валюти. Четверта головна кнопка буде запускати ігрову сесію під час якої користувач зможе керувати ігровим персонажем. Під час ігрової сесії користувач зможе керувати ігровим персонажем. Даний персонаж буде вільно переміщатися на ігровій площині, що буде процедурно генеруватися. Одночасно будуть генеруватися такі ігрові об’єкти як зброя та об’єкти оточення. Після початку ігрової сесії навколо ігрового персонажу будуть з’являтися ворожі персонажі. При смерті ворогів гравець буде отримувати ігрову валюту. Основною ціллю ігрової сесії буде пережити як найбільше хвиль ворогів та знаходження порталу для успішного її завершення зі збереженням всієї набраної ігрової валюти. При поразці гравця, що буде являтися смерть ігрового персонажу, гравець буде отримувати лише частину набраної ігрової валюти.

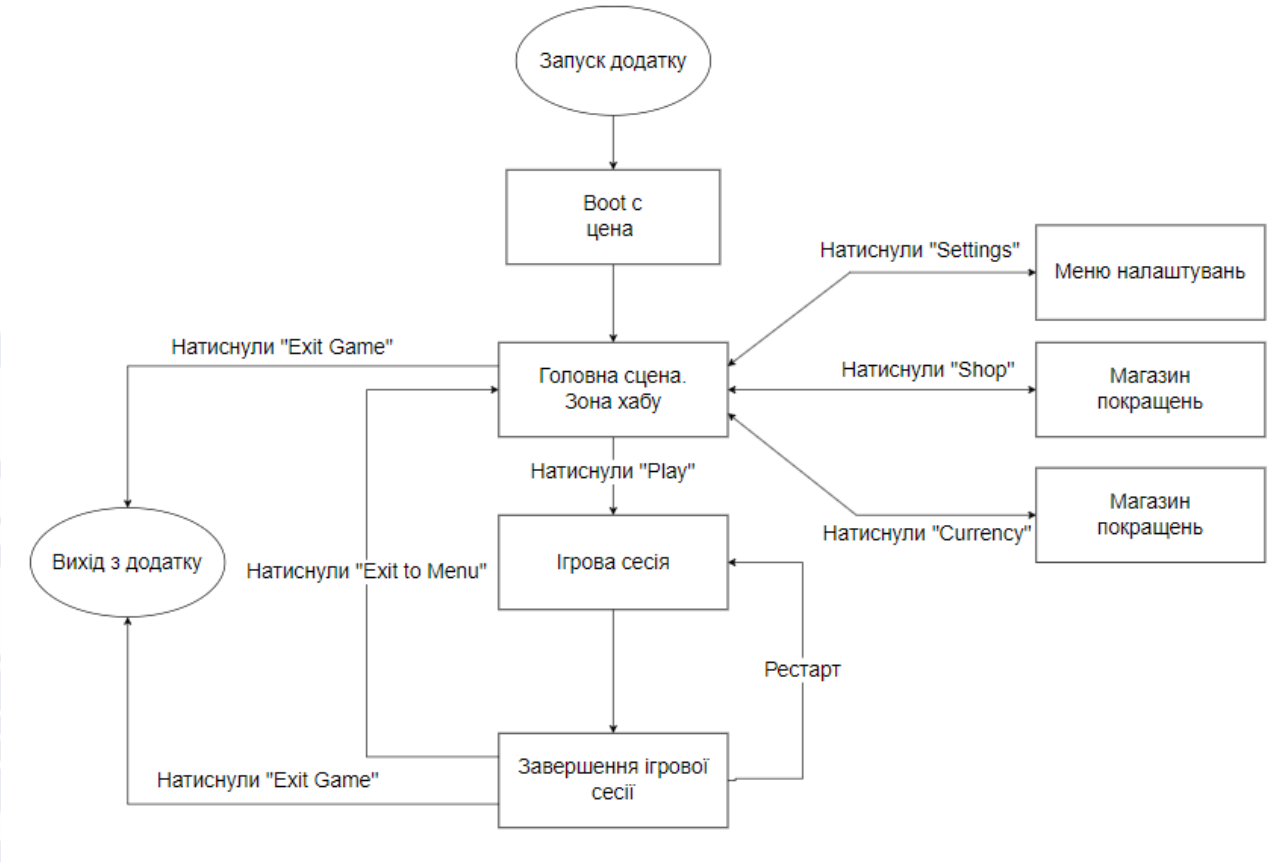

Рисунок 2.1 - Графічне зображення структури гри

2.3 Ігрові об’єкти та їх логіка

Кожна гра складається з різноманітних ігрових об'єктів, які створюють унікальний інтерактивний світ для гравців. Ці об'єкти включають у себе персонажів, предмети, локації, ефекти та багато іншого. Кожен об'єкт має власну логіку та взаємодію з іншими об'єктами у грі. У цьому розділі ми розглянемо основні типи ігрових об'єктів, їх функціональність та взаємодію в межах гри "Code Frost: Cybernetic Battle".

Першим об’єктом, що ми розглянемо, буде сам ігровий персонаж. Має базові середні значення характеристик здоров’я та швидкості. Є єдиним ігровим персонажем в додатку. Починає сесію гри з базовою зброєю, що має посередні значення. Переміщення здійснюватиметься за допомогою нової Input System [30-31]. Сам об’єкт персонажу складатиметься з декількох підоб’єктів, що будуть відповідати за різні скрипти компоненти. На основному буде знаходиться головний скрипт, що має посилання на інші компоненти, які є прикріпленими як

до головного об’єкта, так і до підоб’єктів. Один з двох підоб’єктів є об’єкт, що відповідатиме за модель персонажа, тобто за його відображення. Другий же в свою чергу відповідатиме за collider (колайдер) та скрипти пов’язані з ним. Компонент collider є одним з вбудованих у рушій Unity та в сферу його обов’язків входить визначення форми ігрових об’єктів для прорахунку фізичних зіткнень і виклик подій до відповідних колізій між об’єктами. Візуальним представленням під час розробки буде 3D об’єкт в формі капсули зеленого кольору з моделлю прямокутної форми на рівні очей, що буде використовуватись для розуміння, куди є повернутий ігровий персонаж.

Наступними об’єтами є вороги. Їх буде декілька видів, кожен з яких зможе мати різню зброю. Значення їх характеристик буле залежити від типу ворогу. Дані значення будуть корегуватися для балансування геймплею на основі отриманих відгуків від тестувальників та кінцевих користувачів. Переміщення кожного окремого ворога буде здійснюватися за допомогою фреймворка AI Navigation [32], що раніше називався NavMeshComponents. Під час розробки візуальне представлення буде таким самим як і в ігрового персонажу, але буде використаний відмінний колір від зеленого для кожного з тестових ворогів.

Ігровий об’єкт зброя являє собою предмет для підбору. Вони будуть з’являтися на ігровому полі в рандомізованих місцях. Дані предмети зможе підбирати тільки гравець, тим самим замінюючи поточну зброю, як наслідок може змінитися тип снаряду, сила атаки, швидкість стрільби та можливо ефекти від влучання зброї. На момент розробки доцільним відображенням буде 3D об’єкт в формі сфери. Кожний тип з тестової зброї буде мати різний колір для візуальної різниці під час тестування.

Портал – являтиме собою об’єкт, який при колізії з ігровим персонажем буде завершувати ігрову сесію гравця. Даний об’єкт буде з’являтися, так само в рандомізованих місцях, але з відносно низькими шансами, ніж інші об’єкти.

Різноманітні об’єкти оточення будуть об’єктами-перешкодами. Функціональних задач, окрім можливого створення колізії об’єкта з рухомими об’єктами, як снаряди, гравець, вороги, та можливого обмеження доступної

області руху для NavMesh Agent компонентів прикріплених на ворогів, не передбачено. Візуальним представленням даних об’єктів може бути дерева, каміння, і тд.

2.4 Ігрові механіки

У даному розділі ми розглянемо ключові ігрові механіки нашої гри "Code Frost: Cybernetic Battle", що визначають геймплей та ігровий досвід кінцевого користувача. Ігрові механіки або ігрові можливості - це основні способи взаємодії гравців з грою, які включають у себе різноманітні дії, системи та функції.

Кожна гра має одну або декілька ігрових механік, що є основою геймплею. Гра "Code Frost: Cybernetic Battle" не буде виключенням. Її базовими механіками буде: переміщення ігрового персонажу за допомогою одного пальця та автоматична атака ігрового персонажа по об’єктам. Дані механіки є прикладами елементів казуальних жанру, так як не потребують великої кількості часу для розуміння і є відносно простими в освоєнні. Їхня реалізація буде здійснена аналогічно до таких ігрових додатків, як «Magic Survival» [33] та «Archero» [34].

Переміщення ігрового персонажу здійснюватиметься, коли гравець натисне в нижній частині екрану та після появи на екрані мобільного пристрою візуального відображення стіку геймпаду проведе пальцем в одну із сторін. Для отримання сигналів від даного відображення буде використовуватися нова InputSystem, що є покращеною системою та є більш гнучкою в порівнянні з минулою.

Автоматична атака буде здійснюватися на основі зброї, що є вдягненою на ігрового персонажа, тобто на основі характеристик зброї, таких як радіус дії. Якщо об’єкт, який може атакувати гравець заходить в радіус дії зброї, то він автоматично буде атакувати його. Також варто зазначити, що ігровий персонаж буде атакувати найближчий до себе об’єкт.

Додатковими механіками більше представлятимуть неосновні можливості для гравця. Представниками таких механік є:

- Зона хабу. В дане місце можна попасти після завершення ігрової сесії, як після поразки, так і через портал повернення. В хабі буде надана можливість витрачати ігрову валюту в магазині;
- Зменшення здоров’я ігрового персонажу з плином часу. Дане обмеження буде стримувати гравця від можливості буду в нескінченній ігровій сесії. Дана сесія, якщо виникне, може призвести до появи змішаних або навіть більше до негативних емоцій, так в кінцевого користувача буде тільки три варіанта завершення сесії: закрити ігровий додаток, здатися або знайти портал повернення в зону хабу;

2.5 Інтерфейс

Для реалізації інтерфейсів буде використано реалізацію MVC патерну. Головною перевагою даного патерну буде можливість швидкої ітерації UI/UX дизайну та зменшення впливу візуальної частини на кодову.

В казуальній грі «Code Frost: Cybernetic Battle» буде створено декілька екранів та вікон для відображення візуальних інтерфейсів. Першим екраном, що буде відображений для користувача, буде головне меню. На ньому буде знаходитись чотири кнопки, що будуть свою візуальною імплементацією показувати на такі дії, як почати ігрову сесію, відкрити налаштування ігрового додатку, відкрити магазин покращень. Вони будуть знаходитись в нижні частині екрану. Таке розташування буде в зручній досяжності для користувача. Також в лівому верхньому куті буде відображення поточної кількості ігрової валюти у користувача. При натисканні на дане відображення можна буде додатково купити ігрову валюту.

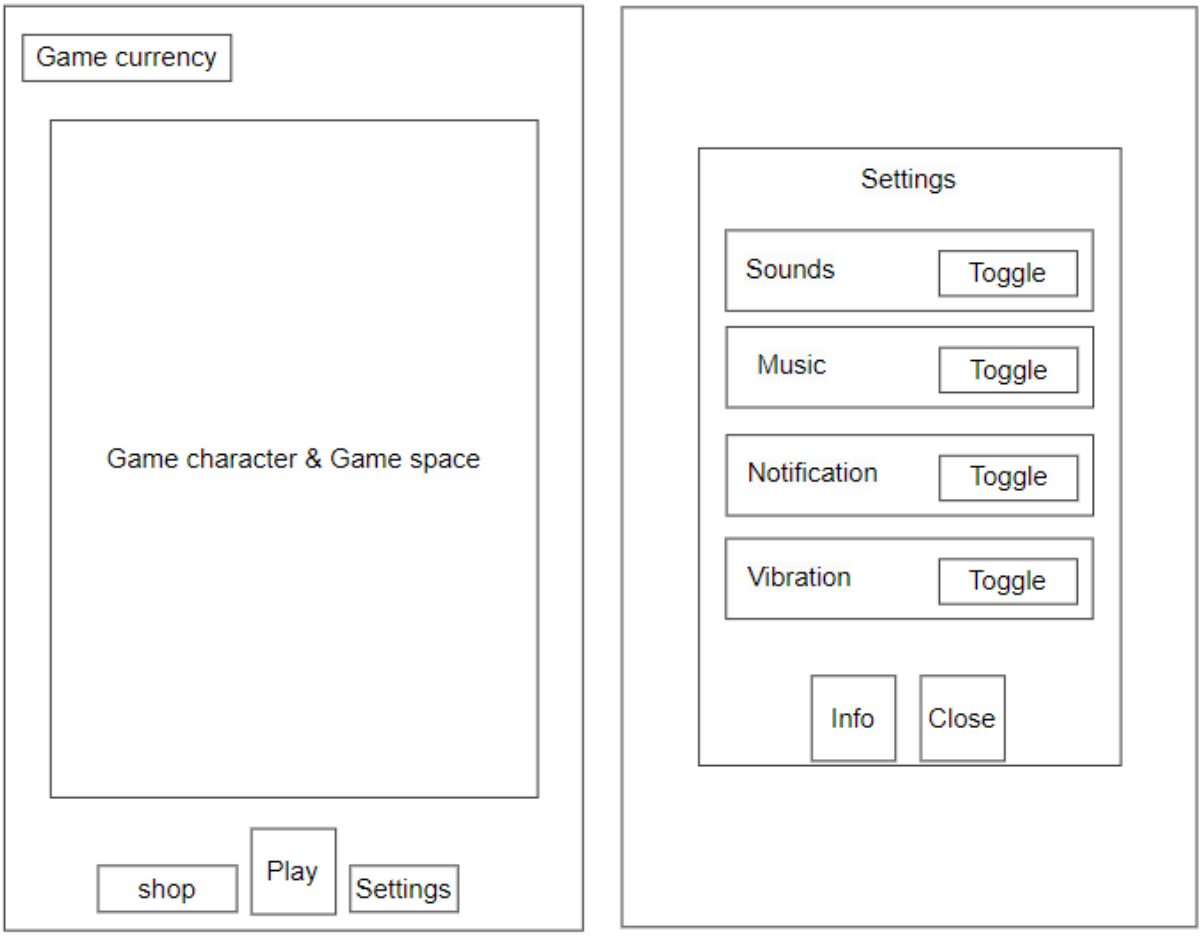

Рисунок 2.2 – Схематичні моделі відображення головного меню та меню налаштувань

Після натискання кнопки налаштувань на екрані головного меню буде відображено відповідне вікно. Воно буде містити в собі чотири перемикача для вмикання та вимикання таких налаштувань, як: програвання звуків, музики, показ повідомлень від ігрового додатку та вібрація. Окрім перемикачів будуть наявні дві кнопки. Одна буде закривати дане вікно і повертати на екран головного меню, а інша буде показувати додаткову інформацію відносно додатку. Наприклад інформацію щодо поточної версії гри, розробників додатку.

При натисканні кнопки магазину буде відображено відповідне вікно та поточна кількість ігрової валюти у користувача. Гравці зможуть витрачати ігрову валюту для придбання покращень характеристик зброї. Розширенням функціоналу даного вікна може бути можливості купівля та заміни моделі

ігрового персонажу. В нижній частині вікна буде наявна кнопка для закриття вікна та повернення на минулий екран UI.

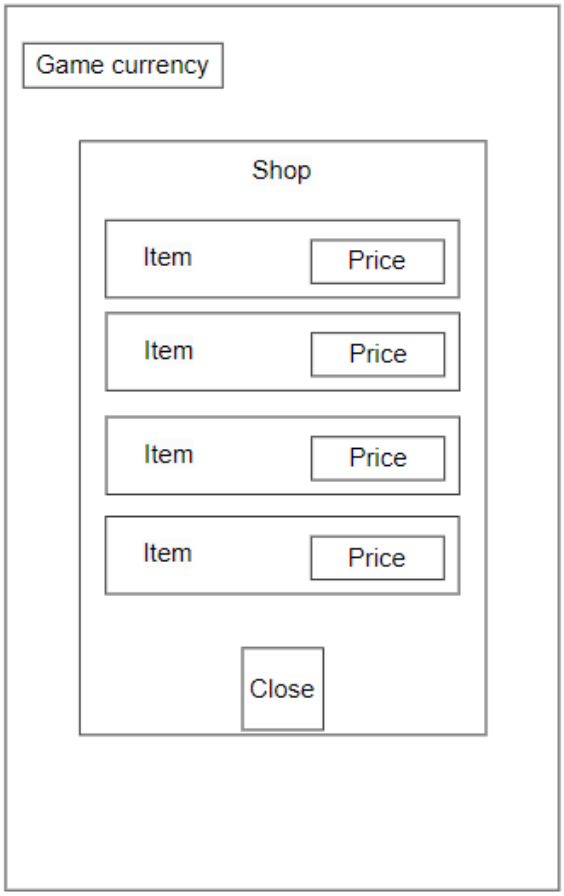

Рисунок 2.3 – Схематичне модель відображення зони хабу

Третій екран буде відображатися під час самої ігрової сесії. Він міститиме відображення кількості зібраної ігрової валюти та кнопку паузи, після натискання якої буде призупинено ігрову сесію. Під час цього екрану гравець зможе керувати ігровим персонажем за допомогою відображення на екрані геймпадного стіка, що буде з’являтися на місця натискання. Варто зазначити, що він має з’являтися в нижній половині екрану та немає свою основною частиною візуально виходити за межі екрану.

Четвертим екраном виступає екран після натискання паузи, де буде відображено три кнопки: «Продовжити», «Рестарт» та «Меню». При натисканні кожна кнопка буде виконувати відповідну їй дію:

- «Продовжити». Буде виключено екран паузи та буде продовжена ігрова сесія.

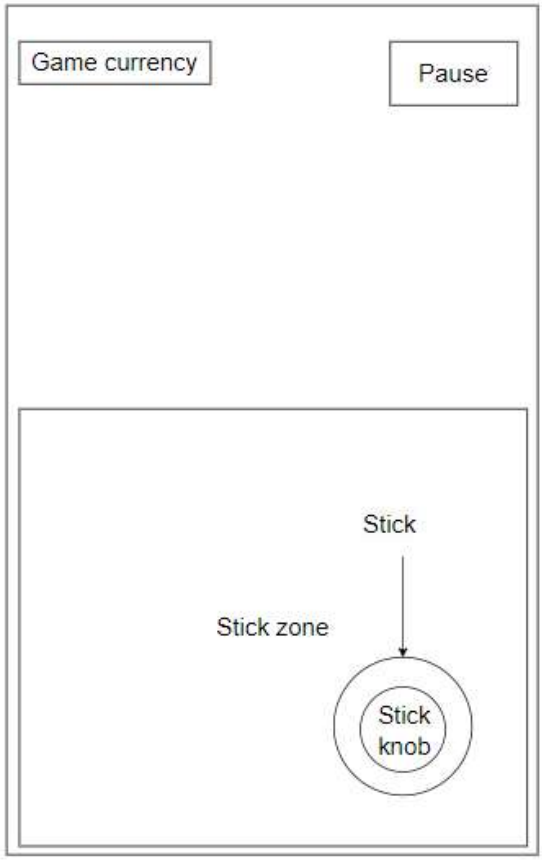

Рисунок 2.4 – Схематичне модель відображення екрану під час ігрової сесії

- «Рестарт». Буде запущена нова ігрова сесія та накопичена ігрова валюта за минулу сесію не буде зарахована, як і будь які накопичені можливі бонуси ігрової сесії.
- «Меню». Буде закінчена поточна ігрова сесія та накопичена ігрова валюта за дану сесію не буде зарахована. Буду показано екран головного меню.

П'ятим екраном та шостим екранами будуть екрани завершення ігрової сесії як при успішному, так і при програші. В верхній частині екрану буде відображатися текст з відповідним написом до стану завершення ігрової сесії. Посередині будуть два поля. Перший буде відображати загальну кількість набраної ігрової валюти за ігрову сесію, а другий – множник на який буде перемножено загальну кількість набраної валюти та результат даної операції буде збережений у користувача.

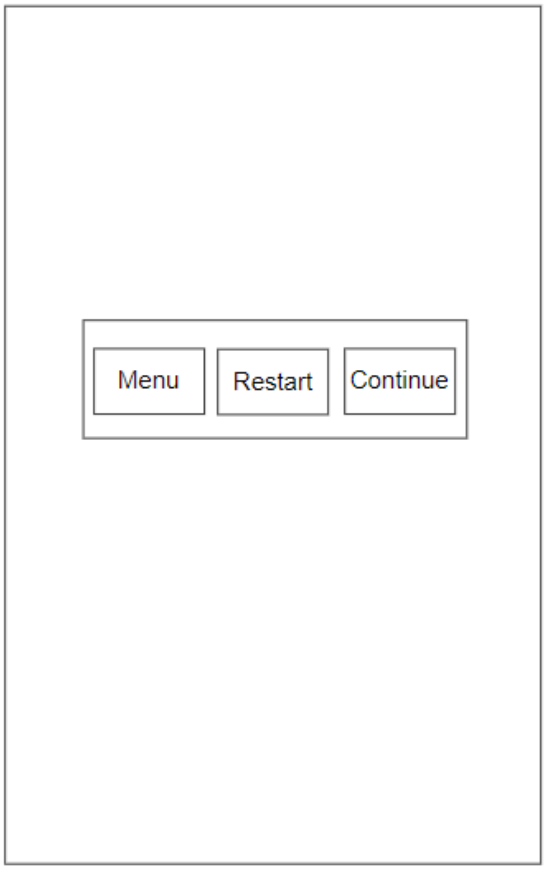

Рисунок 2.5 – Схематичне модель відображення меню паузи

Також дані екрани будуть містити три кнопки: «Меню», «Нова сесія» та «Магазин». При натисканні кожна кнопка буде виконувати відповідну їй дію:

- «Меню». Користувачу буде показано екран головного меню.
- «Нова сесія». Буде запущена нова ігрова сесія.
- «Магазин». Буде показано вікно магазину.

Окрім вище згаданих екранів буде реалізовано UI елемент, що відображатиме поточний стан показника здоров’я як ігрового персонажа, так і ворогів, що знаходяться в зоні екрану. Дані елементи будуть знаходитись біля відповідних їм ігрових об’єктах.

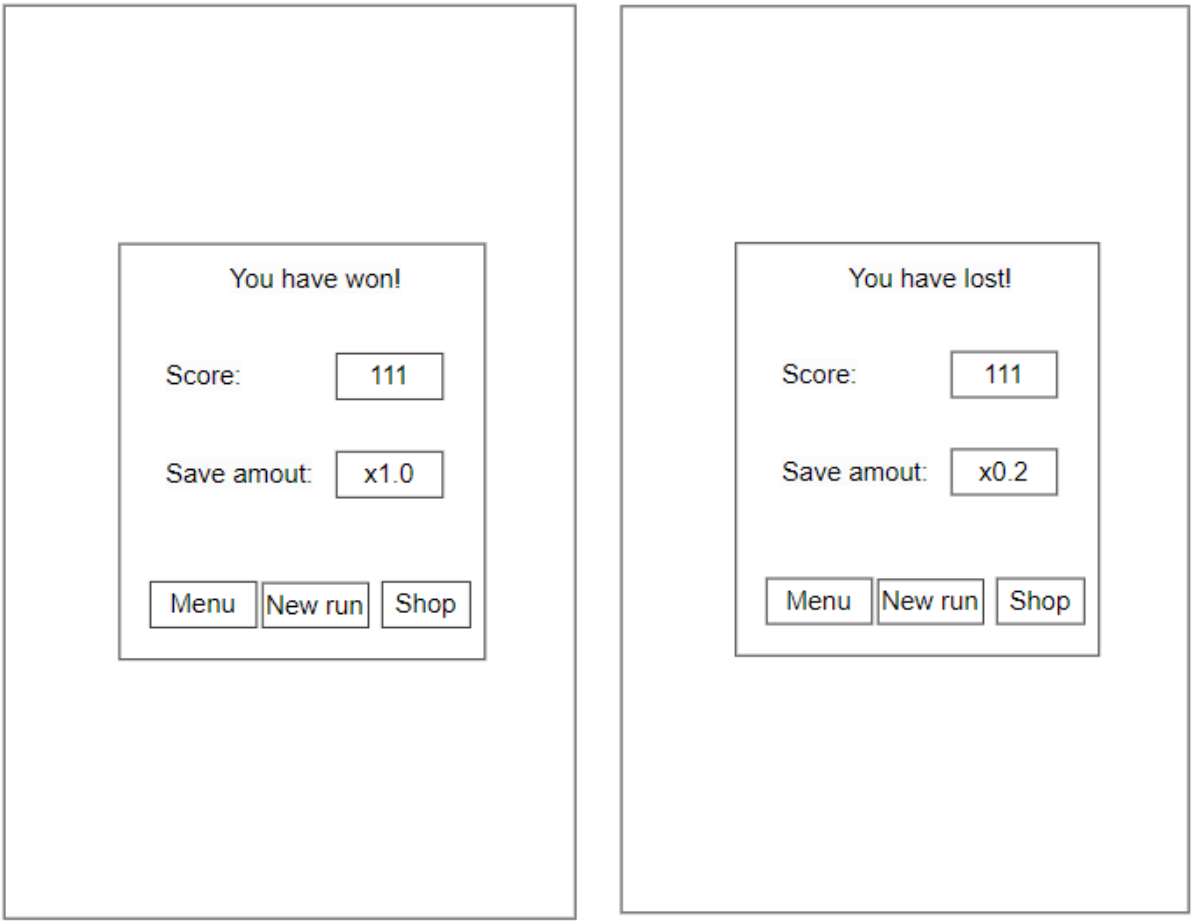

Рисунок 2.6 – Схематичні моделі відображення екранів успішного завершення ігрової сесії та меню поразки

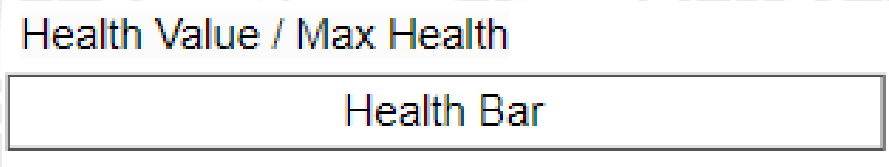

Рисунок 2.7 – Схематичне модель відображення поточного здоров’я персонажів

2.6 Технології для реалізації

Для реалізації казуальної гри «Code Frost: Cybernetic Battle» будуть використані фреймворки та інструменти. Дані інструменти будуть використанні як для реалізації та покращення по всьому проєкту, наприклад, оптимізацію очікування виконання функцій написаного коду, так і для реалізації окремих

механік, наприклад, переміщення ворогів по ігровій площині, що є процедурно згенерованою.

UniTask – інструмент, що є реалізацією асинхронного програмування, спеціально розробленого для підтримки ігрового рушія Unity. Переважно виступає в ролі замінника для Coroutine, які є реалізацією вбудованою в рушій, але мають суттєвий недолік. Для їх використання має бути створений об’єкт з доданим компонентом класу MonoBehaviour, який має запустити їх виконання та бути активованим об’єктом на ігровій сцені. Також може заміняти функціонал наданий бібліотекою System.Threading.Tasks. Як зазначено на сторінці публічного репозиторія надає сумісну поведінку з Task/ValueTask/IValueTaskSource [35]. Головною перевагою в порівнянні зі схожими інструментами є ефективне виділення пам’яті на виконання UniTask методів. [36] З результатів дослідження проведеного з порівнянням Coroutine, Task та UniTask останній не генерує сміття в пам’яті, що має суттєвий вплив на ігрові додатки, так як вони потребують кожного доступного ресурсу для стабільного функціонування.

Рисунок 2.8 – Порівняльний рафік згенерованої GC(garbage collector) пам’яті [36]

Zenject – це фреймворк побудований для ігрового рушія Unity, який реалізовує шаблон впровадження залежностей (Dependency injection, DI). Другою назвою фреймворку є Extenject. Даний фреймворк надає відносно великий спектр можливостей. Основним з яких є сама реалізація DIContainer за допомогою якого в проєкті можна налаштовувати залежності, що як наслідок робить код менш зв'язним та надає розробникам простіше розуміння роботи коду з можливість швидкого впровадження нового функціоналу. Zenject розповсюджується за ліцензією MIT, що дає можливість вільно використовувати його як в персональних, так і комерційних проєктах. Завантажити фреймворк можна як через репозиторій на Github, так за допомогою Unity Asset Store [37-38].

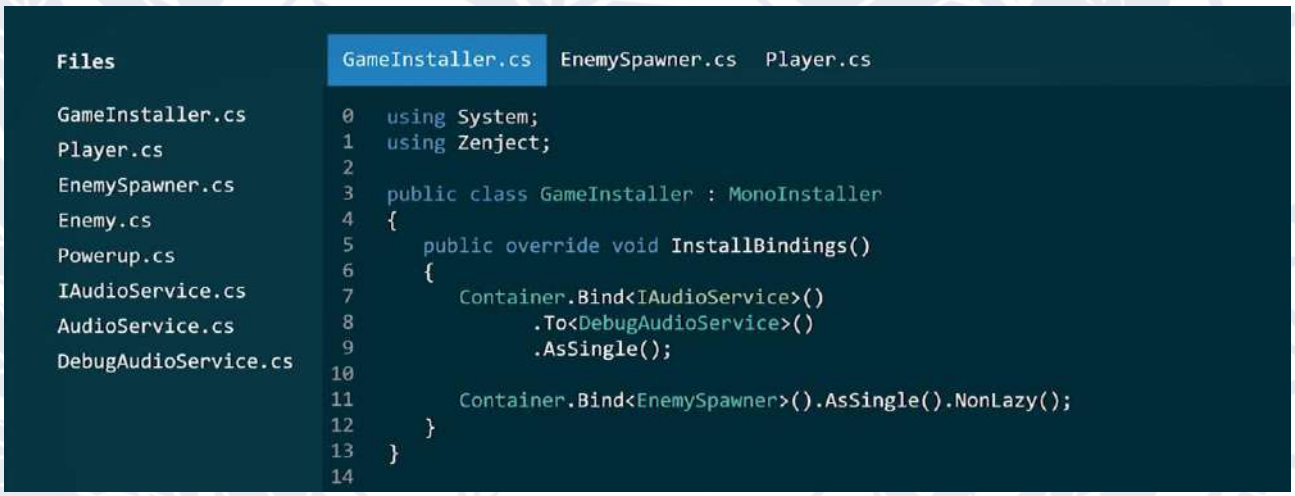

Рисунок 2.9 –Приклад додавання об'єктів в DIContainer за допомогою Zenject [39]

AI Navigation – фреймворк написаний розробниками компанії Unity Technologies. Спочатку розроблявся під назвою NavMeshComponents та розповсюджувався через відкритий репозиторій [32]. Ключовим функціоналом, що надає даний пакунок(package), являється навігаційна система. Вона надає функціонал для створення персонажів, що зможуть реалістично пересуватися по створеному ігровому світу [40]. Такі персонажі іменуються як NavMesh Agent та пересуваються вони по площині, яка називається NavMesh Surface. Дана площина є об'єктом, що зберігає в собі дані відносно полігонів, які не мають ніяких перешкод між їх точками. Переміщення же в свою чергу є комплексним

процесом, що досягається за допомогою декількох етапів. Спочатку визначається шлях від початкової точки до кінцевої. Далі йде слідування визначеному шляху під час якого шлях може бути скорегований відносно нових динамічних перешкод або руху інших агентів. Також система корегує швидкість та прискорення агентів для досягнення бажаної швидкості пересування агента [41]. Окрім даної функції AI Navigation надає можливість корегувати умовну вартість пересування по поверхностям, що дає реалізовувати площини з ускладненим пересування, наприклад, локація болота, що є розповсюдженою в іграх жанру «Soulslike».

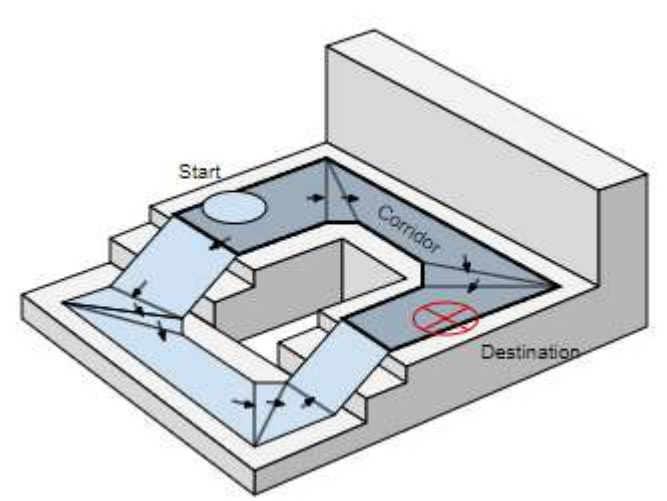

Рисунок 2.10 –Схематичне зображення NavMesh Surface з сайту документації [41]

Cinemachine, як і AI Navigation, є розробкою компанії Unity Technologies. Даний інструмент надає розробникам зручний функціонал для керування камерами на сцені, а саме виконує математичні розрахунки, логіку переслідування персонажа, зміну кадрів [42]. Як результат даний інструмент прискорює розробку ігрових додатків, скорочуючи час на розробку систем керування камерами [43-44]. Так як ігровий рушій Unity може бути використаний для створення екранізацій, наприклад, мільтфільмів, то Cinemachine є чудовим інструментом для процедурної прототипізації анімацій з можливістю отримання миттєвого відгуку та впровадження змін на проти вагу звичайному підходу з часозатратним процесом рендерингу. Unity Technologies в співпраці з Disney

Television Animation використали Cinemachine для створення короткометражки «Baymax Dreams» та отримали премію Emmy Awards у сфері технологій та інженерії [45].

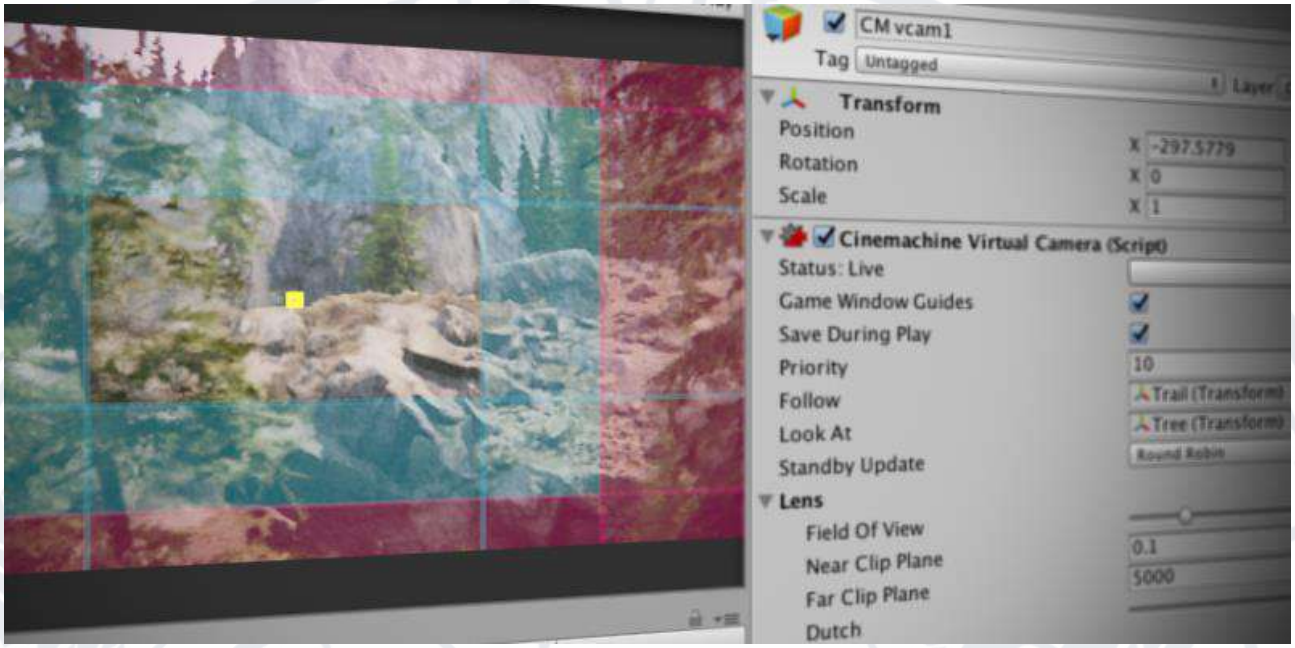

Рисунок 2.11 – Інтерфейс компоненту Cinemachine Virtual Camera[43]

Input System – фреймворк, що є оновленою версією системи вводу. Минула система, яка називається Input Manager, наразі залишається вбудованою базовою версію в ігровий рушій Unity [30-31]. Input System привносить суттєві зміни для налаштування вводу. Вона вводить такі поняття, як Binding, Action, Action Map. Binding – являє собою клавішу, кнопку після натискання якої буде викликано дію (Action). Кожен Action Map може мати деяку кількість дії з прив’язаними до них сигналами вводу. Action Map’и треба використовувати для створення різних дії з прив’язками до одних і тих же вводів, що робить можливим, наприклад, використання одного Action Map для переміщення персонажу, а інший для переключення між UI елементами, але при цьому використовувати однакові клавіші вводу. Також Input System краще підтримує кросплатформність в порівнянні з Input Manager [46].

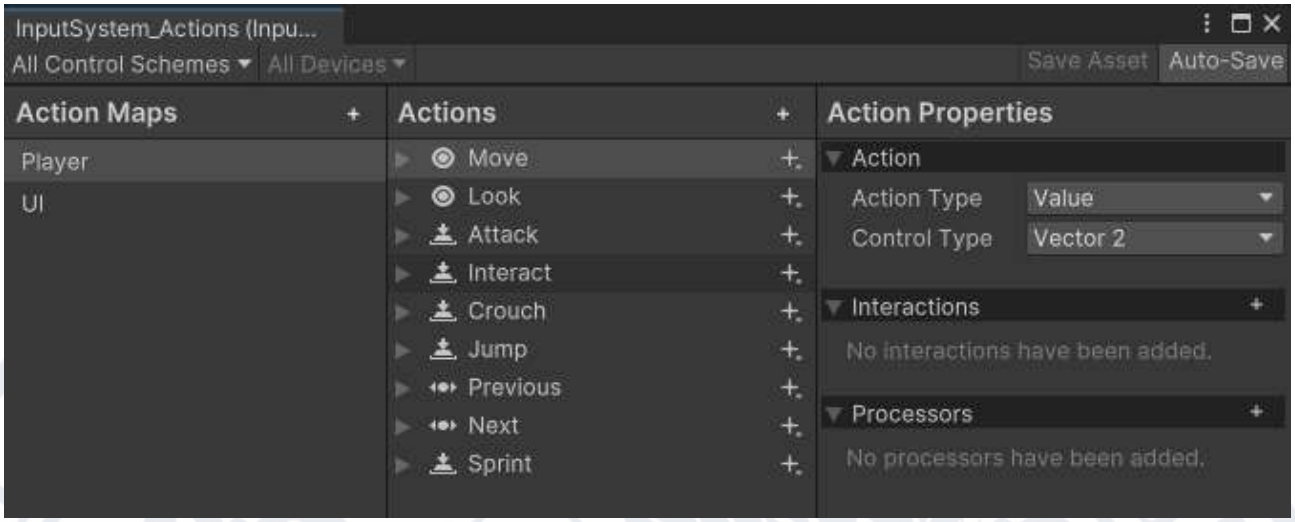

Рисунок 2.12 – Інтерфейс вікна налаштувань Action Map [30]

ObjectPool<T0> даний клас є частиною іменного простору UnityEngine.Pool [47]. Він реалізовує один з найпопулярніших ігрових патернів проєктування для оптимізації ігрового додатку, а саме «Object Pool». Основною ціллю є створення за потреби визначених об’єктів та повторне використання набору ініціалізованих після повернення їх до «Object Pool»., а не постійне створення нових об’єктів [48]. Як результат використання шаблону ігровий додаток є більш оптимізованим в використанні пам’яті цільових платформ, особливо, на мобільних пристроях [49].

Addressables – фреймворк, що є розробкою компанії Unity Technologies та надає інструменти для керування та організації пакування ресурсів ігрового додатку [50]. В ігровому рушії Unity є три традиційні підходи до отримання ресурсу проєкту за посиланням, а саме додавання їх на сцену, отримання з папки Resource або використання AssetBundles [51]. Addressables є покращеною версією цих підходів, так як має реалізацію асинхронного завантаження ресурсів. Окрім цього використання фреймворку Addressables надає наступні переваги:

- Гнучкість. Досягається за допомогою можливість регулювати місце розміщення ресурсів. На будь-якому етапі проєкту можна змінити місце доступу до певного ресурсу, не переписуючи код додатку.
- Керування залежностями. Фреймворк автоматично завантажує всі залежності будь-яких ресурсів.

- Управління пам’яттю. Фреймворк автоматично вивантажує ресурси, що не використовуються.
- Упаковка вмісту. Фреймворк ефективно пакує AssetBundle, навіть коли ресурси переміщуються або перейменовуються. AssetBundle можна використати для зменшення початкового розміру додатку при першому завантажені

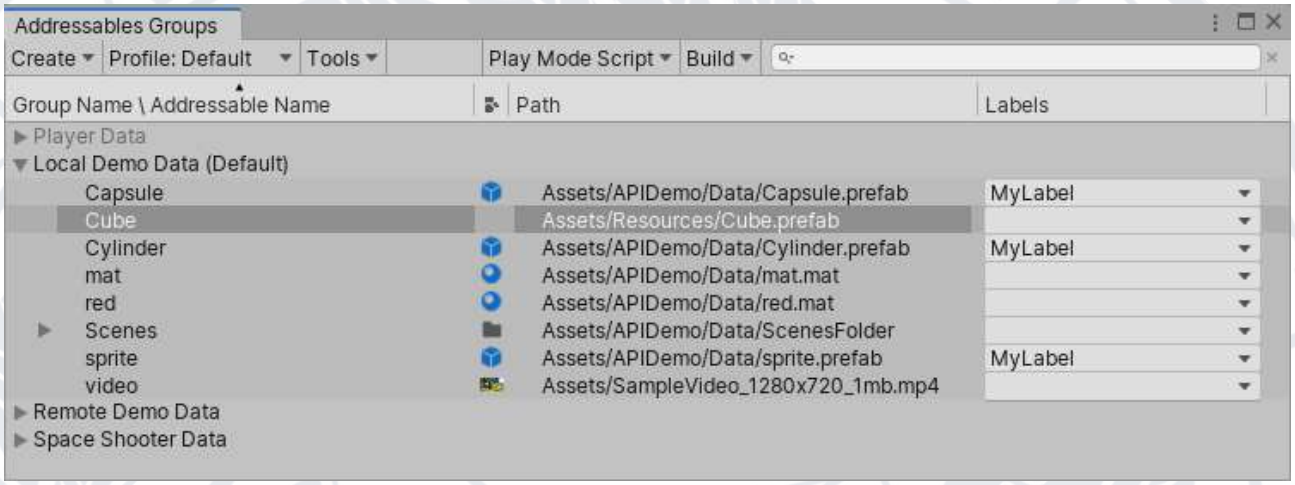

Рисунок 2.13 – Інтерфейс вікна налаштувань Addressables [52]

Висновок до розділу 2

В другому розділі був зроблений огляд дизайн документу гри «Code Frost: Cybernetic Battle». Було розглянуто основний жанр гри, ігрові об’єкти та їх логіку поведінки, геймплейні особливості, візуальний інтерфейс з яким буде взаємодіяти кінцевий користувач та технології для реалізації гри.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ГРИ «CODE FROST: CYBERNETIC BATTLE»

3.1 Використані патерни проєктування

Важливою частиною реалізації гри є використання патернів проєктування. Головною перевагою є забезпечення легкої підтримки та супроводу створеного програмного продукту. Також їх використання пришвидшує тестування та виявлення багів. Основними використаними патернами були:

- Патерн dependency injection. Дозволяє розділити класи від їх залежностей, переклавши відповідальність за вирішення даних залежностей на DIContainer. Було використано фреймворк Zenject, яка реалізовує даний патерн.
- State патерн. Дозволяє об’єктові змінювати свою поведінку, коли змінюється його внутрішній стан. В ігровому додатку було реалізовано state machine, що контролює основний процес гри. Наприклад, запускає появу ворогів на ігровому полі.
- Model-View-Controller (MVC). Розділяє UI елементи на три компоненти, що дає більш гнучку систему, де зміни в відображені не мають впливу на дані, а зміни даних могли здійснюватися без змін інтерфейсу. Використовується в представлені інтерфейсів та вікон користувачу.
- Observer. Поведінковий патерн, який дає можливість об’єктам стежити й реагувати на події, які відбуваються в інших об’єктах. Реалізовано в деяких класах для кращої незв’язності коду.
- Object pool. Патерн, який використовує набір ініціалізованих об’єктів, а не створює нові об’єкти, що як результат зменшує кількість очищення пам’яті, що в свою чергу зменшує навантаження на мобільний пристрій [48]. Використовується для появи нових ворогів на ігровій площині.
- Фабричний метод. Породжувальний патерн, який дозволяє створювати класи не прив’язуючись до конкретної реалізації.

- Медіатор. Поведінковий патерн, який зменшує зв’язність коду серед великої кількості класів зводячи ці зв’язки в реалізації одного класа-посередника.
- Event Bus. Патерн проєктування, що дозволяє при наявності великої кількості компонентів зменшити зв’язність коду за допомогою класу, що надає можливість викликати іншим класам події та реагувати на них.

3.2 Використання Zenject. Dependency Injection

В реалізації гри Zenject використовується за основним призначенням, а саме для прив’язки об’єктів в DIContainer. Переважно прив’язки робились до інтерфейсів, що дає можливість замінити ту чи іншу реалізацію класу без зміни коду в всіх залежних скриптах. Залежності прийнято прописувати в конструктор класу, але за деякими винятками вони може бути прописані або як поле з атрибутом «Inject», або як параметр в окремий метод так само з атрибутом «Inject». Останні два варіанта переважно використовувались, якщо залежності прокидуються в класи, які унаслідувалися від класу Monobehaviour. Це пов’язано з тим, що в розробників немає можливості викликати або змінювати конструктор класу Monobehaviour.

```
private void BindDataService()
{

Container.Bind<IDataService>().To<DataService>().AsSingle().NonLaz
y();
}
```

В даному коді реалізовано прив’язку класу DataService до інтерфейсу IDataService. Варто зазначити, що це не єдиний варіант реалізації прив’язки класу в DIContainer. Було використано також метод BindToInterfaces, що дозволяє прив’язати один клас до декількох інтерфейсів.

Також було створено класи, що реалізовують шаблон проєктування фабричний метод, що вирішують одну з особливостей Zenject. Дана особливість полягає в тому, що всі залежності вирішуються під час ініціалізації додатку, сцен і тд, що в свою чергу унеможливлює додавання залежностей у класи створені під час вже ініціалізованого додатку.

```
public class EnemyFactory : IEnemyFactory
{
    private readonly DiContainer m_diContainer;

    public EnemyFactory(DiContainer diContainer)
    {
        m_diContainer = diContainer;
    }

    public Enemy Create(Enemy enemy, Transform transform)
    {
        Enemy enemyInstance =
m_diContainer.InstantiatePrefabForComponent<Enemy>(enemy,
transform);
        return enemyInstance;
    }
}
```

В вище наведеному коді показана одна з реалізацій даного шаблону, а саме фабрика для створення ворогів. Даний клас в конструкторі приймає DIContainer за допомогою якого ми можемо створювати нові класи з всіма потрібними можливостями. Також класи шаблону фабричного методу відділяють логіку використання DIContainer від інших класів, що є гарною практикою серед Unity-розробників.

3.3 Реалізація State патерну. GameStateManager

В ході реалізації гри було використано шаблон проєктування State, який є одним з розповсюджених в розробці ігрових додатків. Даний шаблон було використано для реалізації менеджера станів гри. Дані стани обумовлюють в собі різні частини життя гри та імплементують структуру гри. Наприклад, був реалізований стан MenuGameState, що відкриває користувачу екран головного

меню, налаштовує камеру для відображення гри та ініціалізує системи додатку, такі як генерація ігрової площини, появу ігрових об’єктів на даній площині, а саме ворогів, зброї та об’єктів оточення.

За перемикання ігрових станів відповідає клас GameStateManager. Даний клас залежить тільки від класу GameStateFactory, що реалізовує патерн проєктування фабричний метод для створення всіх ігрових станів. Кожен ігровий стан додається до поля типу Dictionary<>, де ключем виступають значення структури даних enum, а значеннями самі ігрові стани.

```
private void ChangeState(GameStatus gameStatus)
{
    if (m_currentGameState?.CheckTransition(gameStatus) == false)
{ return; }
    bool containsKey = m_gameStates.TryGetValue(gameStatus, out
IGameState gameState);
    if (containsKey == false)
    {
        UnityEngine.Debug.LogFormat("No Status in Dictionary:
{0}", gameStatus.ToString());
        return;
    }

    m_currentGameState?.ExitState();
    m_previousGameStatus = m_currentGameStatus;
    m_currentGameStatus = gameStatus;
    m_currentGameState = gameState;
    m_currentGameState.EnterState();
}
```

В вище наведеному коді показано логіку, що виконується при зміні ігрового стану на інший. Кожен ігровий стан реалізований від абстрактного класу GameState, що забезпечує базовою реалізацією потрібних методів. Наприклад, перевірка чи можливий перехід з поточного ігрового стану в інший. Для керування системами ігрового додатку в кожному ігрову стані додаються потрібні залежності за допомогою фреймворка Zenject.

3.4 Реалізація руху ігровим персонажем

Для руху персонажу по ігровому світу в ігровому рушії Unity є два основних варіанта реалізації. Перший варіант реалізації за допомогою скрипта CharacterController. Був спеціально розроблений для керування персонажами та має вбудовані методі для переміщення, але потребує додаткових налаштувань для симулювання фізичних властивостей, таких як вплив гравітації на персонажа. Другим варіантом є використання компоненту Rigidbody, який на противагу CharacterController надає підтримку фізичних взаємодій з іншими об’єктами. Для коректної реалізації руху за його допомогою треба написати власний скрипт для керування, що буде базуватися на фізиці від рушія Unity.

В ході реалізації гри «Code Frost: Cybernetic Battle» було вирішено використати саме компонент Rigidbody. Основні проблеми, які мають бути вирішені при використанні даного підходу, це опрацювання нерівностей на площині для переміщення і підняття та спуск персонажу як по похилій поверхні, так і по сходах. Для їх вирішення було використано підхід реалізований розробниками компанії «Toyful Games» [53]. Для персонажів використали 3D-фігуру форми капсули, яка є піднятою на деяку висоту від поверхні. Дане підняття досягається за допомогою прикладання сили до компонента Rigidbody, що є доданим до персонажа. Підхід основується на використанні математичних формул з використанням вбудованих методів в ігровий рушій. Одним з основних методів рушія, що було використано, є Raycast. Він випускає уявний промінь з точки на певну відстань та повертає bool значення, що вказує чи зіткнувся промінь з якимось об’єктом, і повертає через параметр out структуру RaycastHit. Дана структура містить координати точки зіткнення, що дозволяє вирахувати відстань від початку променя до об’єкту зіткнення, яка потрібна для обрахування за формулами (3.1-3.5). В залежності від вихідного значення формули (3.5) застосовується відповідна сила для підняття або зниження висоти персонажу.

$$velocity_{ray} = \overrightarrow{V_{ray}} * \overrightarrow{V_{rb}} \quad (3.1)$$

де $\overrightarrow{V_{ray}}$ – вектор направленого променя;

$\overrightarrow{V_{rb}}$ – направлений вектор швидкості від Rigidbody компонента;

$velocity_{ray}$ – різниця між векторами променя та направлення персонажа.

$$velocity_{other} = \overrightarrow{V_{ray}} * \overrightarrow{V_{other}} \tag{3.2}$$

де $\overrightarrow{V_{ray}}$ – вектор направленого променя;

$\overrightarrow{V_{other}}$ – направлений вектор швидкості іншого об’єкта, що знаходиться під персонажем;

$velocity_{other}$ – різниця між векторами променя та направлення іншого об’єкта.

$$velocity = velocity_{ray} - velocity_{other} \tag{3.3}$$

де $velocity_{ray}$ – цільовий вектор направлення загального руху;

$velocity_{other}$ – вектор швидкості від Rigidbody компонента;

$velocity$ – різниця між нинішнім напрямком руху та бажаним.

$$height = S_{hit} - height_{desired} \tag{3.4}$$

де S_{hit} – відстань до точки перетину променя з об’єктом;

$height_{desired}$ – цільова висота персонажа від об’єктів під ним;

$height$ – відстань, на яку треба підняти/знизити висоту персонажа.

$$F_{spring} = \left(height * Str_{sping}\right) - \left(velocity * Damper_{sping}\right) \tag{3.5}$$

де $height$ – цільовий вектор направлення загального руху;

Str_{sping} – вектор швидкості від Rigidbody компонента;

$velocity$ – вектор швидкості від Rigidbody компонента;

$Damper_{sping}$ – вектор швидкості від Rigidbody компонента;

F_{spring} – різниця між нинішнім напрямком руху та бажаним.

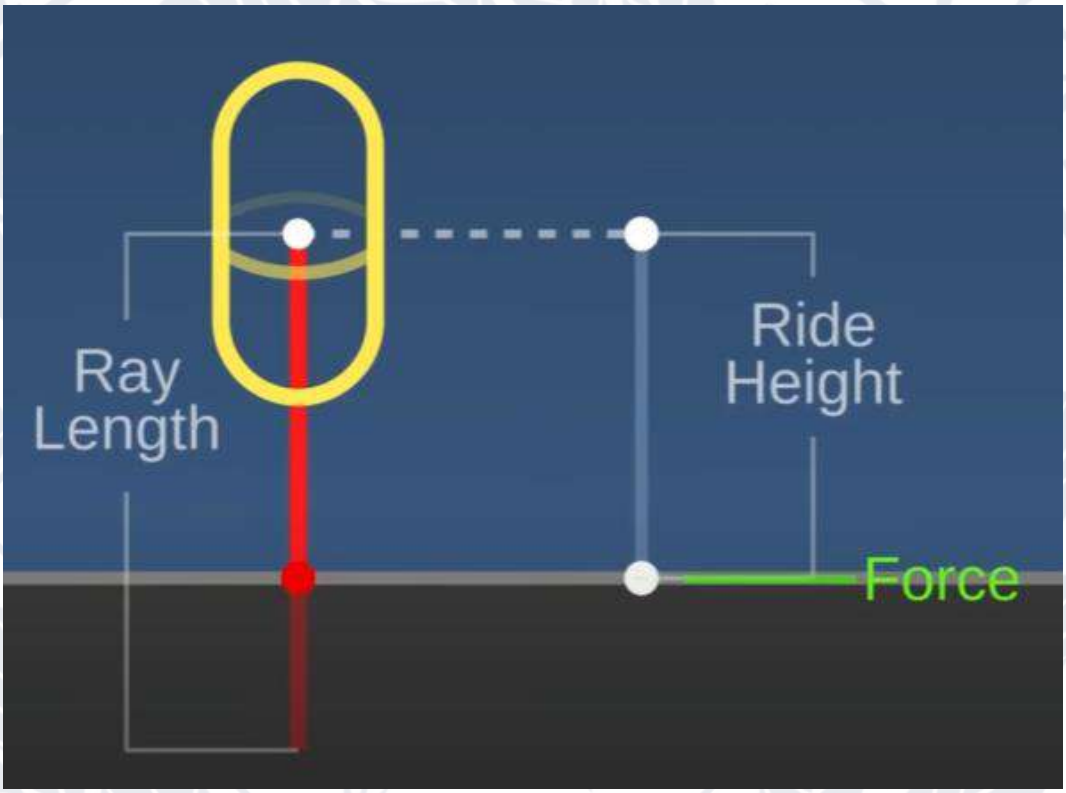

Рисунок 3.1 – Схематичне представлення роботи підходу [53]

Окрім реалізації тримання центру персонажа нал поверхнею, було реалізовано саме пересування персонажем з чутливим керуванням, що також основується на використанні фізики від ігрового рушія. Пересування реалізовано на основі декількох параметрів, що визначають швидкість пересування персонажа за формулами (3.6-3.11).

$$velocity = \overrightarrow{V_{dir}} * \overrightarrow{V_{rb}} \tag{3.6}$$

де $\overrightarrow{V_{dir}}$ – цільовий вектор направлення загального руху;

$\overrightarrow{V_{rb}}$ – вектор швидкості від Rigidbody компонента;

$velocity$ – різниця між нинішнім напрямком руху та бажаним.

$$a_{res} = a_{base} * AnimationCurve.Evaluate(velocity)$$

де a_{base} – базове значення прискорення;

$AnimationCurve.Evaluate$ – метод, що на основі вхідного значення повертає відповідне значення з заданої кривої;

a_{res} – результуюче значення прискорення.

$$\overrightarrow{V_{ideal}} = \overrightarrow{V_{dir}} * maxSpeed \tag{3.7}$$

де $\overrightarrow{V_{dir}}$ – цільовий вектор направлення загального руху;

$maxSpeed$ – значення максимальної швидкості руху;

$\overrightarrow{V_{ideal}}$ – вектор.

$$\overrightarrow{V_{goal}} = \begin{cases} \overrightarrow{V_{ideal}} & \left\|\overrightarrow{V_{ideal}} - \overrightarrow{V_{goal}}\right\| \leq \Delta T \\ \overrightarrow{V_{goal}} + \left(\frac{\overrightarrow{V_{ideal}} - \overrightarrow{V_{goal}}}{\left\|\overrightarrow{V_{ideal}} - \overrightarrow{V_{goal}}\right\|}\right) * a_{res} * \Delta T & \left\|\overrightarrow{V_{ideal}} - \overrightarrow{V_{goal}}\right\| > \Delta T \end{cases} \tag{3.8}$$

де $\overrightarrow{V_{goal}}$ – базове значення прискорення;

$\overrightarrow{V_{ideal}}$ – метод, що на основі вхідного значення повертає відповідне значення з заданої кривої;

a_{res} – результуюче значення прискорення;

ΔT – .

$$\overrightarrow{V_{accel}} = \frac{\left(\overrightarrow{V_{goal}} - \overrightarrow{V_{rb}}\right)}{\Delta T} \tag{3.9}$$

де $\overrightarrow{V_{goal}}$ – базове значення прискорення;

$\overrightarrow{V_{rb}}$ – метод, що на основі вхідного значення повертає відповідне значення з заданої кривої;

ΔT – результуюче значення прискорення;

$\overrightarrow{V_{accel}}$ – .

$$\overrightarrow{V_{accel}} = \begin{cases} \overrightarrow{V_{accel}} & \left\|\overrightarrow{V_{accel}}\right\| \leq maxAccel \\ \overrightarrow{V_{accel}} * \left(\frac{maxAccel}{\left\|\overrightarrow{V_{accel}}\right\|}\right) & \left\|\overrightarrow{V_{accel}}\right\| > maxAccel \end{cases} \quad (3.10)$$

де $\overrightarrow{V_{accel}}$ – базове значення прискорення;

$maxAccel$ – метод, що на основі вхідного значення повертає відповідне значення з заданої кривої.

$$Force = mass * \left(\overrightarrow{V_{accel}} \odot \overrightarrow{F_{scale}}\right) \quad (3.11)$$

де $mass$ – значення маси написаного в компоненті Rigidbody;

$\overrightarrow{V_{accel}}$ – метод, що на основі вхідного значення повертає відповідне значення з заданої кривої;

$\overrightarrow{F_{scale}}$ – метод, що на основі вхідного значення повертає відповідне значення з заданої кривої;

$Force$ – результуюче значення прискорення.

3.5 Реалізація елементу керування персонажем

Основуючись на дизайн документі було реалізовано елемент керування персонажем, що візуалізований як стік. В ігровий рушій Unity не є вбудованими реалізації контролерів по типу геймпадного стіка, але з фреймворком Input System додається клас абстрактний клас OnScreenControl [54]. Він є базовою імплементацією екранних контролерів основаних на Input System. Даний клас

був унаслідуваний в ArealOnScreenStick, що імплементує функціонал описаний в дизайн документі, а саме:

- з’являється при натисканні в відповідній зоні;
- не виходить за межі екрану;
- передає в Input System вектор направлення.

Для відслідковування натискання ArealOnScreenStick використовує API InputSystem.EnhancedTouch. що дає можливість використовувати покращені засоби зчитування дотику, такі як Touch, які не є ввімкнено за замовчуванням [55]. Ключовою метою використання є підписка на івенти реалізовані в даній API, а саме ті, що спрацьовують при натисканні, пересування та підняття пальця з екрану девайса.

Основний механізм дії спрацьовує під час натискання користувачем по екрану. Зчитується дотик користувача до екрану, потім перевіряється чи координати точки дотику є в межах екрану. Далі залежно від обраного типу обмеження знаходження основної візуальної частини обраховуються координати лівого-нижнього та правого-верхнього кутів. Дана логіка виконується в функції GetCorners.

```
private Vector3 ClampStartPosition(Vector3 startPosition)
{
    if (m_restrictionType == RestrictionType.None)
    {
        return startPosition;
    }

    float xJoystickHalf = m_floatingJoystick.Size.x * 0.5f *
m_scaleFactor;
    float yJoystickHalf = m_floatingJoystick.Size.y * 0.5f *
m_scaleFactor;
    float xPosition = startPosition.x;
    float yPosition = startPosition.y;

    if (xPosition < m_minCorner.x + xJoystickHalf)
    {
        xPosition = m_minCorner.x + xJoystickHalf;
    }

    if (xPosition > m_maxCorner.x - xJoystickHalf)
    {
```

```
            xPosition = m_maxCorner.x - xJoystickHalf;
        }

        if (yPosition < m_minCorner.y + yJoystickHalf)
        {
            yPosition = m_minCorner.y + yJoystickHalf;
        }

        if (yPosition > m_maxCorner.y - yJoystickHalf)
        {
            yPosition = m_maxCorner.y - yJoystickHalf;
        }

        var result = new Vector3(xPosition, yPosition,
startPosition.z);
        return result;
}
```

В вище наведеному коді представлена функція, що виконується для обрахунку позиції в якій буде відображено візуальний елемент стіку після дотику до екрану девайса. Після зміщення елементу керування зчитується пересування пальця, що викликає відповідний івент. В його логіку входить зміна положення «верхньої частини» стіка. Для обмеження можливих позицій знаходження даної «вершини» під час пересування перевіряється коректність позиції, що обраховується за формулою (3.12), яка основана на формулі еліпса:

$$value = \left(\frac{(x-x_c)^2}{r_x^2}\right) + \left(\frac{(y-y_c)^2}{r_y^2}\right) \quad (3.12)$$

де (x, y) – координати точки;

(x_c, y_c) – координати центру еліпса;

r_x – радіус еліпса по осі X;

r_y – радіус еліпса по осі Y.

Якщо результуюче значення менше або дорівнює одиниці, то позиція переміщеного дотика знаходиться в межах основної візуальної частини стіка. Також вектор направлений з центру стіка до позиція «вершини» стіка

передається в Input System, що зчитується вже для переміщення ігрового персонажа.

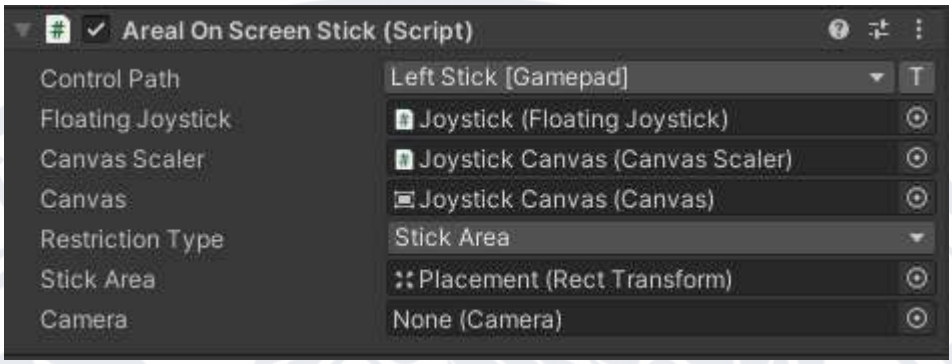

Рисунок 3.2 – Налаштування елемента керування

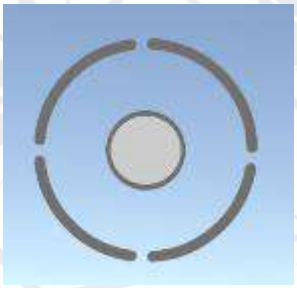

Рисунок 3.3 – Реалізація візуального елемента керування

3.6 Реалізація механіки автоматичної атаки

В ході реалізації такої ігрової механіки як автоматична атака ігрового персонажа по ворожим було реалізовано клас AttackService, що використовуються як ігровим персонажем, так і ворогами для завдання по супротивникам або підходящим цілям. В його обов'язки входить:

- менеджмент над вже створеними об'єктами класу Projectile з використанням клас ObjectPool<T0>, що забезпечує оптимізоване використання ресурсів пристрою користувача;
- реагування на паузу ігрового додатку, а саме зупинення всіх активних на момент паузи Projectile об'єктів та продовження їх роботи при відміни стану паузи;
- надання публічного методу, який на основі типу зброї створює Projectile об'єкт та виконує логіку прописану в відповідному класі

поведінці, що унаслідувався від абстрактного класу AttackBehaviour. В параметри методу передається клас WeaponAttackInfo, що включає в себе дані, що потрібні для реалізації атаки.

Для отримання даних пов'язаних з ціллю атаки, а саме її позиції на ігровій площині було створено такі класи як EnemyPositionService та PriorityTargetPositionService. Перший використовується саме ігровим персонажем для перевірки чи знаходиться хоча б один ворог в радіусі дії одягнутої зброї та отримання його позиції, а другий - ворожими персонажами. Обидва класи наслідувалися від абстрактного класу PositionService<T>, де T є класом, що реалізовує інтерфейс ITarget. В його реалізації були створені методи для отримання різних вибірок об'єктів наприклад, як в якомусь радіусі від якоїсь позиції, так і чиє даний об'єкт видимим з тої позиції. Також класи наслідники використовуються в поведінкових логіках AttackBehaviour , наприклад, для реалізації нанесення шкоди всім ворогам в деякому радіусі від місця влучання зброї.

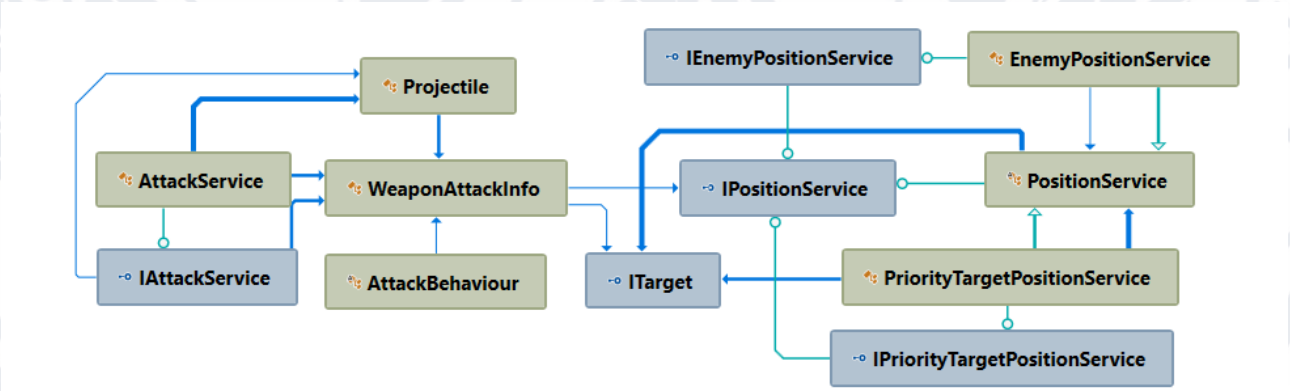

Рисунок 3.4 – Зображення залежностей класів в реалізації механіки автоматичної атаки

3.7 Реалізація ігрової площини

В ході реалізації ігрового додатку було розроблено процедурну генерацію ігрової площини. Основною метою, якої є надання користувачу можливості рухатися в будь-якому напрямку без обмежень, але при цьому мати різноманіття

об’єктів оточення, що зустрічаються, та надання унікальності кожній ігровій сесії. Для реалізації було створено клас TerrainManager. Даний клас відповідає за:

- реагування на виклик події про досягнення поточних границь ігрової площини, а саме створення нових частин ігрової площини, що є об’єктами класу TerrainBlock, з наданого переліку можливих. Нова частина обирається за допомогою алгоритму випадкового вибору на основі ваги елементу серед інших в вибірці. Елементами виступають саме унітарні одиниці ігрової площини, а їхня вага визначена довільним числом заданим в файлі налаштувань;
- менеджмент над вже створеними об’єктами класу TerrainBlock з використанням клас ObjectPool<T0>, що забезпечує оптимізоване використання ресурсів пристрою користувача;
- використання фреймворку AI Navigation для оновлення даних в NavMesh Surface при появі нової частини ігрової площини.

Головним недоліками такого підходу є ризик використання великої кількості ресурсів пристрою, що пов’язаний з можливістю користувача йти у будь-якому напрямку, а саме неконтрольованому розмірі ігрової площини та неточності в обрахуванні фізичних явищ обумовлені ігровим рушієм Unity при знаходженні об’єктів на великій відстані від початку координат. Для вирішення першої проблеми було розроблено логіку, що при досягненні ігрового персонажу границі ігрової площини TerrainManager вивантажує TerrainBlock’и разом з об’єктами оточення, які є нерелевантними для ігрової сесії та зберігає всі потрібні дані для можливості повернення користувача та повторного їх створення. Друга проблема була вирішена за допомогою впровадження події зсуву всіх ігрових об’єктів на якусь відстань до центру координат сцени. Дана подія спрацьовує за умови, що користувач досяг границі ігрової площини та наступна частина площини буде знаходитись на відстані більшій, ніж виставлена в файлі налаштувань TerrainManager.

При появі ігрового об’єкту TerrainBlock на сцені TerrainManager викликає подію у класі, що реалізовує патерн EventBus. На дану подію реагують класи, що відповідають за появу ігрових об’єктів оточення та за появу зброї на ігровій площині.

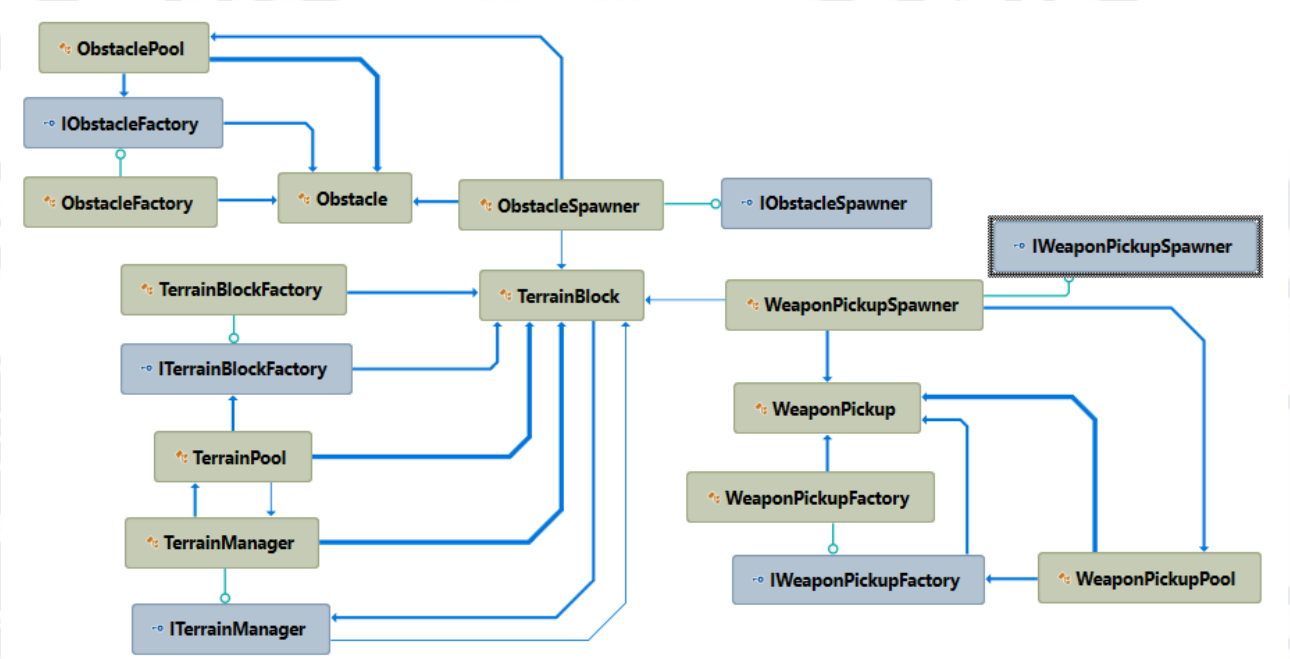

Рисунок 3.5 – Зображення залежностей класів в реалізації ігрової площини

3.8 Реалізація збережень даних

Збереження даних є важливою складовою в реалізації будь-якої гри. Для ігрового додатку «Code Frost: Cybernetic Battle» було створено клас DataService, що наслідується від інтерфейсу IDataService. Він свою чергу використовується в як залежність в потрібних класах та прокидується до них за допомогою фреймворка Zenject.

```
public DataService()
{
    string dataPath = Application.persistentDataPath;
    _saveDirectoryPath = dataPath + "/Saves/";
    if (!Directory.Exists(_saveDirectoryPath))
    {
        Directory.CreateDirectory(_saveDirectoryPath);
    }
```

```
    _serviceDataHandler = new ServiceDataHandler(this);
    _addressableJsonDataHandler = new
AddressableJsonDataHandler();
    _keyValuehandler = new KeyValuePairDataHandler(this);
}
```

Вище приведено конструктор класу DataService, що, окрім визначення шляху до директорії збережень, створює класи «хендлери» (handler) до яких інші класи можуть звернутися маючи в залежностях інтерфейс IDataService. В свою чергу «хендлери» реалізовують методи унаслідуванні від їхнього унікального інтерфейса.

```
public KeyValuePairDataHandler(DataService dataService)
{
    _dataService = dataService;
    var saveDirectoryPath = _dataService.SaveDirectoryPath;
    var serviceDataHandler = _dataService.ServiceData;

    _intProvider = new
KeyValuePairIntegerFileJsonProvider(saveDirectoryPath,
serviceDataHandler);
    _stringProvider = new
KeyValuePairStringFileJsonProvider(saveDirectoryPath,
serviceDataHandler);
}
```

В наведеному коді представлено приклад конструктора одного з «хендлерів», саме класу KeyValuePairDataHandler, що відповідає за збереження даних парами ключ типу string та значення, де значення може бути типу string або int. Він отримує DataService з якого забираються потрібні дані та класи. Наприклад інший «хендлер» ServiceData та шлях до директорії збереження. Також «хендлери» в конструкторах створюють класи «провайдери» (provider), що мають прямий доступ до даних. «Провайдери» реалізовують всі потрібні методи для роботи з відповідними їм збереженнями. Наприклад, сортування або забір даних за відповідними параметрами. В ході реалізації такого підходу було створено абстрактний клас BaseFileJsonDataProvider, що використовую таке поняття як Generic. Його використання в класі дозволяє не бути прив’язаннм до якогось конкретного типу даних. В випадку зі збереженнями це є зручним

інструментом для написання функцій, що будуть використані у всіх класах унаслідуваних від BaseFileJsonDataProvider. Наприклад, це функції збереження, зчитування, очищення даних. Даний абстрактний клас для збереження та зчитування використовує дані формату JSON. Основною перевагою якого є невеликий розмір збережених даних. Для серіалізації та десеріалізації використовується Newtonsoft, так як в порівнянні з вбудованим рішенням в рушій, є більш гнучким до збереження, наприклад, таких типів даних як Dictionaty<>.

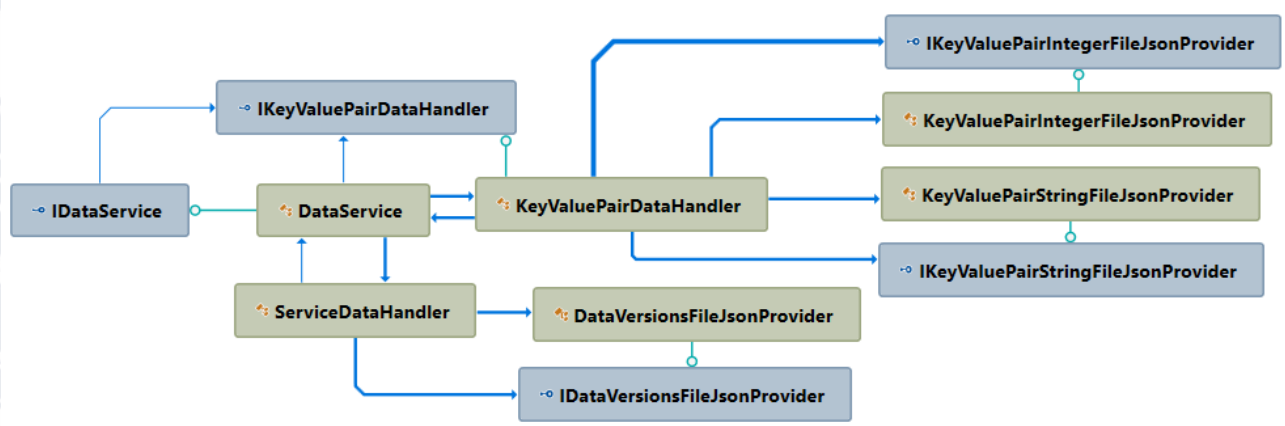

Рисунок 3.6 – Зображення залежностей класів в реалізації DataService

3.9 Реалізація MVC патерну. UIManager

В ході реалізації ігрового додатку було використано реалізацію патерну проєктування MVC. Було створено три класи абстрактних класи BasePopupView, BaseMediatedController та BasePopupMediator. BasePopupView є класом, що унаслідувався від MonoBehaviour. Він використовується для створених UI вікон/екранів та роботи з ними. Наприклад, зміна стану, позиції, тексту або реагування на натискання кнопок і тд. Також BasePopupView реалізовує методи показу та приховування UI під час якого виконуються анімації. Клас BaseMediatedController для передачі даних до BasePopupView та реалізації методів, що будуть виконуватися після натискання на кнопки та інші елементи. Його головною роллю є відокремлення логіки від відображення, наприклад, отримання поточного перекладу текстів UI вікна/екрану та передачі їх до

BasePopupView. Клас BasePopupMediator реалізує патерн медіатор та методи для відкриття та закриття UI вікон/екранів. При виконані відкриття звертається до класу UIManager, що відповідає за виставлення коректного шару для сортування відображення та виставляє камеру на яку має бути воно показано. Також UIManager має функціонал для закриття всіх поточних вікон та екранів.

Для візуальної частини відображення були взяти рисунки з сайту itch.io [56], а саме «Free Space Shooter Game User Interface» [57] та «Virtual Joystick Pack -Free-» [58]. Дані рисунки були спочатку використані в програмі Adobe Photoshop, де було зібрано UI та обрані рисунки для його реалізації на базі рушія Unity. Також на веб-ресурсі Google Fonts [59] було обрано шрифт Tektur [60], що використовується для відображення тексту на UI.

Рисунок 3.7 – Реалізації вікна налаштувань та екрану, що викликається при успішному завершені ігрової сесії

3.10 Результат реалізації гри

В форматі рисунків (3.8–3.9) наведені результати реалізації ігрового додатку «Code Frost: Cybernetic Battle».

Рисунок 3.8 – Скріншоти з ігрового додатка

Рисунок 3.9 – Скріншоти з ігрового додатка

Висновки до розділу 3

В третьому розділі був зроблений огляд реалізації гри «Code Frost: Cybernetic Battle» на основі дизайн документу. Були реалізовано ігрові об’єкти разом з їх властивостями, логікою взаємодії з іншими об’єктами. Також основуючись на дизайн документі було реалізовано геймплейні особливості гри. візуальний інтерфейс для взаємодії з кінцевим користувачем.

ВИСНОВКИ

У роботі було зроблено загальний огляд стану сучасної ігрової індустрії. Були розглянуті різноманітні види та жанри ігор, сучасні платформи для дистрибуції ігрових додатків. Також було проаналізовані та розглянуті сучасні ігрові рушії з їх перевагами, які мови програмування використовуються для розробки ігор та використання дизайн документів для розробки ігор.

Зроблено огляд дизайн документу гри «Code Frost: Cybernetic Battle». Було розглянуто основний жанр гри, ігрові об’єкти та їх логіку поведінки, геймплейні особливості, візуальний інтерфейс з яким буде взаємодіяти кінцевий користувач та технології для реалізації гри.

Зроблено огляд реалізації гри «Code Frost: Cybernetic Battle» на основі дизайн документу. В ігровому додатку було реалізовано ігрові об’єкти разом з їх властивостями, логікою взаємодії з іншими об’єктами. Також основуючись на дизайн документі було реалізовано геймплейні особливості гри. візуальний інтерфейс для взаємодії з кінцевим користувачем.

СПИСОК ЛІТЕРАТУРИ

1. Daley S. GAMING INDUSTRY OVERVIEW. URL: https://builtin.com/gaming (дата звернення: 05.03.2024)
2. Romano S. God of War ‘Trolls, Exploration, and More’ gameplay. URL: https://www.gematsu.com/2018/04/god-of-war-trolls-exploration-and-more-gameplay (дата звернення: 17.05.2024)
3. Paras A. The Witcher 3: Wild Hunt More Gameplay Screenshots Show Bar Brawl, Exploration and a Werewolf. URL: https://wccftech.com/witcher-3-wild-hunt-gameplay-screenshots-show-bar-brawl-exploration-werewolf/ (дата звернення: 17.05.2024)
4. Bowling S. Zelda: Breath Of The Wild: Best Recipes And How To Cook Food. URL: https://www.nintendolife.com/guides/zelda-breath-of-the-wild-best-recipes-and-how-to-cook-food (дата звернення: 17.05.2024)
5. SlashData Team. Did you know that 60% of game developers use game engines?. URL: https://www.slashdata.co/post/did-you-know-that-60-of-game-developers-use-game-engines (дата звернення: 25.04.2024)
6. Unity 2019.3 release. A modernized, refined Editor UI. URL: https://unity.com/releases/2019-3/editor-tools (дата звернення: 19.05.2024)
7. Game-Ace. Unity vs Unreal: Two Main Game Engines. URL: https://game-ace.com/blog/unity-vs-unreal/ (дата звернення: 25.04.2024)
8. Blueprint Interfaces In Unreal Engine 5. URL: https://cghero.com/tutorials/blueprint-interfaces-unreal-engine-5 (дата звернення: 17.05.2024)
9. Porokh A. Godot vs Unity: All You Need to Know. URL: https://kevurugames.com/blog/godot-vs-unity-which-one-suits-you-best/ (дата звернення: 26.04.2024)
10. Godot Docs. Getting started with Visual Scripting. URL: https://docs.godotengine.org/en/3.1/getting_started/scripting/visual_script/getting_started.html (дата звернення: 17.05.2024)

11. Source 2 wiki. URL: https://developer.valvesoftware.com/wiki/Source_2 (дата звернення: 26.04.2024)
12. Metacritic. Half-Life: Alyx Review. URL: https://www.metacritic.com/game/half-life-alyx/ (дата звернення: 26.04.2024)
13. Qazal. Best programming languages for game development. URL: https://winatalent.com/blog/best-programming-languages-for-game-development/ (дата звернення: 27.04.2024)
14. Johns R., Semah B. 7 Best Programming Languages for Game Development in 2024. URL: https://hackr.io/blog/best-programming-language-for-games (дата звернення: 27.04.2024)
15. CBNITS. Why Java is the Language of Choice for Game Programming. URL: https://connectbud-itsolutions.medium.com/why-java-is-the-language-of-choice-for-game-programming-a55e0ab4f2a9 (дата звернення: 27.04.2024)
16. App Academy. 5 Best Programming Languages for Game Development. URL: https://www.appacademy.io/blog/best-programming-languages-for-game-development (дата звернення: 27.04.2024)
17. MOOC BLOG TEAM. Best Programming Languages for Game Development. URL: https://www.mooc.org/blog/best-programming-languages-for-game-development (дата звернення: 28.04.2024)
18. Septian Ganendra S. K. GDScript Cheatsheet. URL: https://godot.community/topic/78/gdscript-cheatsheet (дата звернення: 28.04.2024)
19. Engr Sajid Khalil. Build games in Swift. URL: https://www.linkedin.com/pulse/build-games-swift-grayhatpk/ (дата звернення: 28.04.2024)
20. Hajilou Y. 2D cross-platform game with Kotlin on Android Studio. URL: https://yasinhajilou.medium.com/2d-cross-platform-game-with-kotlin-in-android-studio-fa1a0eabf6f0 (дата звернення: 29.04.2024)

21. ELITEX Team. JavaScript Game Development: Trends, Pros, and Cons. URL: https://elitex.systems/blog/javascript-game-development-trends/ (дата звернення: 29.04.2024)
22. Jonah V. 5 Rust game engines to consider for your next project. URL: https://blog.logrocket.com/5-rust-game-engines-consider-next-project/ (дата звернення: 29.04.2024)
23. 99Games. World of Gaming — Game Categories, Genres, and Sub-Genres. URL: https://medium.com/@99Games/world-of-gaming-game-categories-genres-and-sub-genres-c52f281a9fea (дата звернення: 01.05.2024)
24. Idil Ozkan. What Are Casual Games?. URL: https://adjoe.io/glossary/casual-games-and-casual-gamers/ (дата звернення: 29.04.2024)
25. Exploring the World of Mobile Game Genres. URL: https://denebgames.com/blog/exploring-the-world-of-mobile-game-genres (дата звернення: 29.04.2024)
26. Prots P. GAME DESIGN DOCUMENTATION: A ROADMAP TO COMPETITIVE GAME DEVELOPMENT. URL: https://stepico.com/blog/game-design-documentation/ (дата звернення: 07.05.2024)
27. Bokone A. Tips for Building an Effective Game Design Document. URL: https://get.assembla.com/blog/making-game-design-document/#what-goes-in-a-game-design-document (дата звернення: 07.05.2024)
28. Kliuch D. HOW TO WRITE GAME DESIGN DOCUMENT (WITH EXAMPLES). URL: https://whimsygames.co/blog/game-design-instructions-examples/ (дата звернення: 07.05.2024)
29. Jason W. Bay. How to Write a Game Design Document (Examples and Template). URL: https://www.gameindustrycareerguide.com/how-to-write-a-game-design-document/ (дата звернення: 07.05.2024)
30. Unity Technologies. Input System. URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8/manual/index.html (дата звернення: 10.05.2024)

31. Unity Game Dev Field Guide. 2020. С. 54-55.
32. Unity Technologies. NavMeshComponents. URL: https://github.com/Unity-Technologies/NavMeshComponents (дата звернення: 09.05.2024)
33. Leme. Magic Survival. URL: https://play.google.com/store/apps/details?id=com.vkslrzm.Zombie&pcampaignid=web_share (дата звернення: 27.05.2024)
34. Habby. Archero. URL: https://play.google.com/store/apps/details?id=com.habby.archero&pcampaignid=web_share (дата звернення: 27.05.2024)
35. Cysharp. UniTask. URL: https://github.com/Cysharp/UniTask?tab=readme-ov-file (дата звернення: 08.05.2024)
36. João Borks. Unity Async vs Coroutine. URL: https://www.linkedin.com/pulse/unity-async-vs-coroutine-jo%C3%A3o-borks/ (дата звернення: 08.05.2024)
37. Bakker M. Extenject. URL: https://github.com/Mathijs-Bakker/Extenject (дата звернення: 09.05.2024)
38. Bakker M. Extenject Dependency Injection IOC. URL: https://assetstore.unity.com/packages/tools/utilities/extenject-dependency-injection-ioc-157735 (дата звернення: 09.05.2024)
39. Getting Started with Zenject. URL: https://youtu.be/IS2YUIb_w_M?si=BzSNsa9L5BG1Gsi3 (дата звернення: 09.05.2024)
40. Unity Technologies. AI Navigation. URL: https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/index.html (дата звернення: 09.05.2024)
41. Unity Technologies. Inner Workings of the Navigation System. URL: https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/NavInnerWorkings.html (дата звернення: 09.05.2024)
42. Oriz E., Kikuchi A., Continisio C., Nobbs C.. The Unity Game Designer Playbook. 2021. 103 с.

43. Unity Technologies. Cinemachine package. URL: https://docs.unity3d.com/Packages/com.unity.cinemachine@3.1/manual/index.html (дата звернення: 10.05.2024)
44. Cinemachine. Powering cameras for films and games. URL: https://unity.com/unity/features/editor/art-and-design/cinemachine (дата звернення: 10.05.2024)
45. Riva I. Unity wins its first Technology and Engineering Emmy® Award. URL: https://blog.unity.com/technology/unity-wins-its-first-technology-and-engineering-emmy-award (дата звернення: 10.05.2024)
46. Luis Ramirez Jr. Unity's New Input System (+ How To Use It!). URL: https://zerotomastery.io/blog/unity-new-input-system/#Top-4-Reasons-to-use-the-new-Input-System-vs-the-Input-Manager (дата звернення: 10.05.2024)
47. Unity Technologies. Unity Documantation. ObjectPool<T0>. URL: https://docs.unity3d.com/ScriptReference/Pool.ObjectPool_1.html (дата звернення: 12.05.2024)
48. Lin W., Krogh-Jacobsen T., Andreasen P., Bilas S. Level up your programming with game programming patterns. 2022. 99 с.
49. Nystrom B. Game Programming Patterns. 2014. 354 с.
50. Unity Technologies. Addressables package. URL: https://docs.unity3d.com/Packages/com.unity.addressables@2.1/manual/index.html (дата звернення: 15.05.2024)
51. Unity Technologies. Upgrading to the Addressables system. URL: https://docs.unity3d.com/Packages/com.unity.addressables@1.11/manual/AddressableAssetsMigrationGuide.html (дата звернення: 15.05.2024)
52. Unity Technologies. Addressables. Getting started. URL: https://docs.unity3d.com/Packages/com.unity.addressables@1.11/manual/AddressableAssetsGettingStarted.html (дата звернення: 15.05.2024)
53. Toyful Games. Making A Physics Based Character Controller In Unity (for Very Very Valet). URL: https://youtu.be/qdskE8PJy6Q?si=SX3BSt_iUiU3S1xz (дата звернення: 19.05.2024)

54. Unity Technologies. On-screen Controls. URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8/manual/OnScreen.html (дата звернення: 20.05.2024)
55. Unity Technologies. Scripting API. Class EnhancedTouchSupport. URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8/api/UnityEngine.InputSystem.EnhancedTouch.EnhancedTouchSupport.html (дата звернення: 20.05.2024)
56. Itch.io. URL: https://itch.io/ (дата звернення: 25.05.2024)
57. Free Space Shooter Game User Interface. URL: https://free-game-assets.itch.io/free-space-shooter-game-user-interface (дата звернення: 25.05.2024)
58. Virtual Joystick Pack -Free-. URL: https://hannemann.itch.io/virtual-joystick-pack-free (дата звернення: 25.05.2024)
59. Google Fonts. URL: https://fonts.google.com/ (дата звернення: 25.05.2024)
60. Jagosz A. Шрифт Tektur. URL: https://fonts.google.com/specimen/Tektur?preview.layout=grid&classification=Display&subset=cyrillic&noto.script=Latn (дата звернення: 25.05.2024)

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)