Шифр «Мережева комп’ютерна гра»

СТУДЕНТСЬКА НАУКОВА РОБОТА

на тему:

«Мережева комп'ютерна гра в жанрі Multiplayer First-Person Shooter»

2018 рік

ЗМІСТ

ВСТУП 3
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 5
1.1. Аналіз предметної області........ 5
1.2. Аналіз існуючих програмних рішень на ринку 9
1.3. Постановка задачі........ 12
РОЗДІЛ 2 ПРОЕКТНІ РІШЕННЯ ТА ЇХ РЕАЛІЗАЦІЯ 13
2.1. Засоби розробки мережевої комп’ютерної гри 13
2.2. Проектування модулів мережевої комп’ютерної гри........ 15
2.3. Створення естетики комп’ютерної гри 21
2.4. Програмна реалізація функцій........ 23
ВИСНОВКИ........ 28
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 29
ДОДАТКИ........ 30

ВСТУП

Процес розробки комп'ютерних ігор є складним, що перш за все пов'язано із специфікою кінцевого продукту та суб’єктивізмом, який є найбільш значущим фактором у розробці комп’ютерних ігор і обумовлений позицією користувача (гравця). Суб’єктивне сприйняття ігор, динамічні зміни розвитку ігрової індустрії та, як наслідок, потреба в орієнтації на цільову аудиторію ускладнює процес розробки. Існує складність прогнозування та планування робіт при розробці ігор, це потребує одночасно експертного технічного досвіду, таланту тощо. Сукупність вищевідзначених факторів сприяла дослідженню особливостей розробки як комп’ютерних ігор в цілому, так і мережевих ігор в жанрі Multiplayer First-Person Shooter.

Актуальність розробки мережевої гри в жанрі Multiplayer First-Person Shooter обумовлена низкою чинників, а саме: незаповненості сегменту ігрової індустрії продуктами даного жанру з безкоштовною моделлю розповсюдження; малою кількістю безкоштовних ігор жанру Multiplayer First-Person Shooter без внутрішніх транзакцій; високими вимогами до апаратного забезпечення у існуючих програмних продуктів на ринку; потребою в якісних ігрових програмах українських розробників в цілому.

Проблемам розробки комп’ютерних ігор, визначення їх характеру, створенню прототипів, тестуванню, принципам ігрового дизайну присвячені праці Е. Аведона, Б. Саттон-Сміта, К. Суейна, С. Хоффмана, Т. Фуллертона, Дж. Шелла, С. Рабіна, П. Вордерер, П. Ван Пелт, разом з тим, незважаючи на велику різноманітність існуючих досліджень з цієї проблематики, усе більшої необхідності потребує вивчення принципів, методик, інструментів розробки комп’ютерних ігор.

Метою дослідження є створення системи механік для гри у жанрі First-Person Shooter з метою подальшої її реалізації у вигляді багатокористувацької командної мережевої гри.

Об’єктом дослідження є процес проектування та розробки ігор у жанрі

Multiplayer First-Person Shooter.

Предмет дослідження є гра у жанрі Multiplayer First-Person Shooter.

Відповідно до мети поставленні наступні завдання:

здійснити аналіз предметної області, що включає дослідження існуючих програмних рішень на ринку;

визначити проектні рішення та здійснити їх реалізацію.

Для досягнення поставленої мети та розв’язання визначених завдань використано наступні методи дослідження, а саме: аналітичний – для визначення основних процесів під час розробки комп’ютерних ігор, найбільш типових складнощів та прийнятних практик подолання криз; метод термінологічного аналізу, який дозволив формалізувати область дослідження і забезпечити розкриття сутності явищ, що досліджуються; метод наукової екстраполяції, що дозволив визначити можливості практичного застосування створеної гри та рекомендацій стосовно процесу розробки; метод експертних оцінок, що використовувався для оцінювання якості створеної гри.

Інформаційною базою дослідження при написанні наукової роботи слугували праці науковців, матеріали науково-практичних конференцій, семінарів, методичні посібники по розробці комп’ютерних ігор.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Аналіз предметної області

У сучасному ігровому світі існує велика кількість різноманітних ігрових продуктів, що мають певні відмінності та особливості ігроладу (процес гри та його відчуття), візуальне відображення історії тощо. Виходячи з цього виникає необхідність визначити, що є комп’ютерною грою, за якими критеріями комп’ютерні ігри поділяються на групи, визначити жанр гри з метою подальшої розробки.

Комп’ютерні ігри як вид мистецтва є комплексним поняттям і поєднують у собі елементи та напрацювання широкого спектру традиційних мистецтв (література, драматургія, графіка, музика, кінематограф). Таке поєднання ставить комп’ютерні ігри на новий рівень розвитку мистецтва і створює унікальну якість – інтерактивність [1].

Разом з тим, в індустрії розробки комп'ютерних ігор поняття «гра» не має чіткого визначення та сприймається як інтуїтивно зрозуміле і має недетермінований характер. Це підтверджується різними поглядами авторів та ігрових дизайнерів на визначення гри, а саме:

– це вправа двох добровільних систем управління, у якій існує змагання між силами, обмеженими правилами з метою створення незбалансованого результату [2]. В цьому визначенні виділяються особливості гри, однак, на нашу думку, бракує уточнення, що в ігри грають для задоволення і сам процес є активним з точки зору гравця;

– це інтерактивна структура ендогенного характеру, що потребує від гравців зусиль для досягнення мети [3]. Ми вважаємо, що це визначення додає важливі аспекти, які описують гру, а саме факт інтерактивності, наявність певного виклику гравцю і, найголовніше - здатність створювати власну внутрішню цінність;

– це закрита формальна система, яка захоплює гравців у структурований конфлікт, що вирішується нерівними результатами [4]. Це визначення розкриває два нових аспекти, властиві іграм. По-перше, ігри мають захоплювати гравців з метою їх емоційного і психологічного залучення, що дозволяє здолати ефект зневіри у нереальність подій, які відбуваються. По-друге, ігри є чітко описаними системами, які мають власні границі.

На наш погляд найбільш вдалим є визначення дизайнера Джессі Шелла [5], де гра – це спрямована на розв’язання проблеми діяльність, до якої підходять з грайливим відношенням. Зазначимо, що «розв’язання проблеми» в контексті комп’ютерних ігор – це розв’язання задачі, що ставить перед гравцем сама гра. Як правило така задача не несе практичного змісту з огляду на ряд психологічних факторів.

Комп’ютерні ігри, як вид програмного забезпечення (ПЗ), наділені спільними рисами, такими як: принципи розробки (при розробці комп’ютерних ігор використовуються сучасні методології, техніки та прийоми розробки ПЗ, використання яких адаптується до творчої специфіки створення комп’ютерних ігор); необхідність наявності персонального комп’ютера, ігрової консолі, мобільного телефона, спеціалізованого ігрового автомату; програмні та апаратні вимоги для роботи; наявність цільової аудиторії, процес розробки включає етапи маркетингової кампанії, реклами та реалізації.

Водночас, комп'ютерні ігри мають особливості і відрізняються від інших видів програмних продуктів, а саме: призначення (прикладне ПЗ в основному призначене для вирішення конкретних завдань, комп’ютерні ігри ж, як правило, спрямовані на культурні, естетичні, соціальні потреби людини, на задоволення потреб в дозвіллі та розвагах); комп’ютерні ігри формують критерії вибору цільової аудиторії не на потенційній користі від гри як продукту, а на основі вподобань до досвіду, який створюється грою; критерії вибору цільової аудиторії для комп’ютерних ігор є абсолютно суб’єктивними [6]; комп’ютерні ігри як мистецький процес не завжди придатний до формалізації та планування робіт. Також комп’ютерні ігри як багатокомпонентне мистецтво потребують

тривалих процесів розробки окремих модулів з залученням вузькопрофільних спеціалістів. Ці фактори обумовлюють процес розробки з використанням гнучких методологій розробки ПЗ [7]; управління ризиками, що пов'язано із суб’єктивним сприйняттям гравцем естетичної складової – вподобання кольорових схем, стилістики інтерфейсу, анімацій; завжди існує ризик, що кінцевий продукт не буде сприйнятий цільовою аудиторією.

На даний момент відсутня єдина класифікація комп’ютерних ігор, яка дозволяє чітко розподілити ігри по класам, що не мають перетину. З цієї причини, існує декілька класифікацій комп’ютерних ігор за їх технічними, естетичними чи ігроладними особливостями. Розглянемо загальноприйняті класифікації.

За кількістю гравців. Найбільш загальна класифікація розподіляє ігри за кількістю гравців, на які вони розраховані. *На одного користувача* – для гри не потрібне підключення до мережі Інтернет, користувач є єдиним гравцем-людиною у грі. *На двох та більше користувачів* – ігри, які потребують підключення до мережі Інтернет (або локальної мережі, LAN) і розраховані на взаємодію гравців-людей один з одним у рамках ігрового всесвіту.

За естетикою. Тривімірні ігри або 3D ігри (від англ. «three dimensional») – ігри, взаємодія з середовищем яких відбувається у горизонтальній площині в координатах X, Z, та у вертикальній площині по координаті Y. Двовимірні ігри або 2D ігри (від англ. «two dimensional») – ігри, взаємодія з середовищем яких відбувається лише в площинах X та Y. Як правило, такі ігри близькі до класичних настільних ігор. Попри те, що взаємодія з середовищем обмежена двома осями координат, це не означає, що сама естетика обмежена двовимірними графічними елементами.

За положенням камери. Попри те, що положення камери не є визначною характеристикою ігор, яка спадає на думку при формалізації, однак саме від нього залежать ігроладні особливості тієї чи іншої гри [8]. Від першої особи (англ. «first person») – погляд на візуальну складову гри подається з точки зору персонажа виглядом «з очей». Від третьої особи (англ. «third person») –

візуальна складова подається як «вигляд збоку», тобто гравець може бачити модель персонажа, за якого він грає та гравцю доступні інші кути огляду на світ в цілому. Вид згори (англ «top-down») – візуальна складова подається в такий спосіб, при якому спостерігач знаходиться на певній відстані по осі Y над середовищем гри, умовно даний тип подачі близький до подачі «з висоти пташиного польоту».

За жанром. Жанрова класифікація є найбільш загальноприйнятою та інформативною з точки зору ігроладу. В свою чергу, жанр гри визначається її ігроладними особливостями. З розвитком комп’ютерних ігор та послідовним запозиченням елементів одного напряму з іншого, чітке жанрове розмежування стало менш ефективним. Наразі виділяють такі основні категорії ігор: однокористувацькі ігри (розраховані на гру з одним користувачем); кооперативні ігри (до шести гравців об’єднуються в одну команду для гри проти супротивника, контрольованого комп’ютером, як з використанням мережі Інтернет, так і без); сесійні багатокористувацькі ігри (розраховані на користувачів, що поєднані через мережу Інтернет), сесійний характер вказує, що ігри є дискретними у часі і проводяться у рамках сесії, поза якою неможливо грати з вибраним набором гравців при заданих умовах; багатокористувацькі безперервні ігри (сесія гри умовно необмежена у часі і гравці приєднуються до сесії без обмежень) [9].

Нами запропонована розробка мережевої комп'ютерної гри в жанрі Multiplayer First-Person Shooter (англ. «багатокористувацька стрілецька гра від першої особи») – це гра, яка через мережу Інтернет залучає до процесу гри чотирьох та більше гравців (як правило, не менше десяти) та розподіляє їх у дві чи більше групи, які, завдяки зусиллям та кооперації гравців перетворюються на команди (як правило з чітким розподілом ролей) для досягнення спільної мети швидше за команду опонентів.

Термін «багатокористувацька» означає, що гра розрахована на взаємодію гравців, які є реальними людьми, один з одним, а не з системою, що контролюється комп’ютером. Термін «мережева» обумовлює певну архітектуру

гри, яка підтримується мережею Інтернет. Незалежно від конкретних деталей архітектури окремого проекту, ключовим аспектом є наявність підключення до мережі Інтернет з комп'ютера кожного окремого гравця. Остання складова «shooter», має найменший вплив на визначення в цілому, оскільки диктує жанр гри, який не зазнає істотних змін від усіх інших компонентів.

1.2. Аналіз існуючих програмних рішень на ринку

Вважаємо за доцільне розглянути існуючі найбільш успішні ігри в жанрі Multiplayer First-Person Shooter за характеристиками: фінансова модель дистрибуції; естетика, візуальне та аудіо оформлення; технічна реалізація з точки зору гравця. Для аналізу існуючих рішень обрано ігри: Team Fortress 2; Overwatch.

Гра Team Fortress 2 розроблена компанією Valve для персональних комп'ютерів, випущена у 2007 році.

Фінансова модель дистрибуції. На початку свого існування Team Fortress 2 розповсюджувалась як гра с фіксованою одноразовою платою, але з 2011 р. стала безкоштовною. Водночас, впроваджено магазин предметів гри, де реалізовано можливість придбання за реальні гроші косметичні предмети для персонажів, аби модифікувати їх зовнішній вигляд. Це стало основою зборів гри. В цілому, сучасна модель є прийнятною, нові гравці повною мірою мають можливість насолодитися грою безкоштовно. Система придбання предметів у магазині не впливає на фактичну ефективність гравця під час гри, а лише на візуальний стиль, тому даний підхід забезпечує справедливу гру для гравців незалежно від їх фінансових вкладень у продукт.

Естетика, візуальне та аудіо оформлення. Візуальне оформлення гри Team Fortress 2 виконане у мультиплікаційному стилі, який, водночас, підсилюється контрастом чорного гумору подій, що відбуваються у грі. Приклад візуального оформлення представлено на рис 1.1.

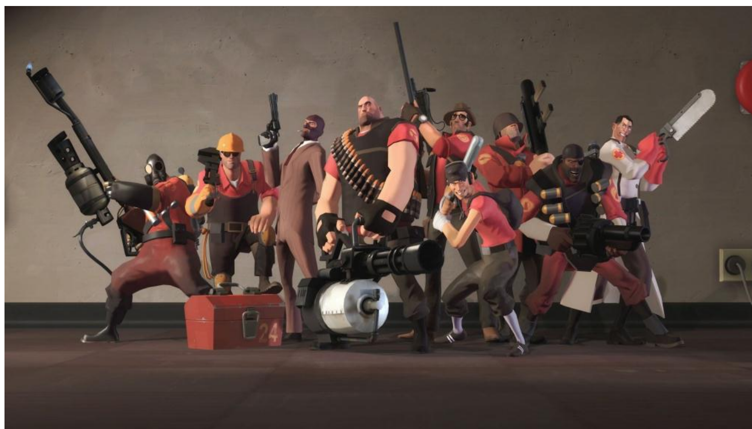

Рис. 1.1. Персонажі гри Team Fortress 2

Мультиплікаційний стиль головних героїв доповнено часовим відрізком, у якому відбувається гра – 1960-ті роки. Гра має елементи шпигунської тематики, що знаходить відображення в оформленні інтерфейсу гри та локацій. З моменту випуску гри в 2007 році була випущена велика кількість тематичних оновлень, які принесли у візуальну складову додаткові елементи. На наш погляд подібні зміни не досить позитивно вплинули на естетику та сприйняття гри в цілому. Музичне оформлення виконано для підтримки стилістики відповідного періоду, в якому відбувається гра. Виконано повне озвучення персонажів для точної передачі особливостей їх характерів та образів.

Технічна реалізація з точки зору гравця. На нашу думку, в плані технічної реалізації гра має певні недоліки через свій двигун. Існують опції використання серверів, що надаються видавцем гри, або є можливість власноруч виступити у ролі сервера для ігор.

Отже, Team Fortress 2 має ряд переваг: вдало підібрана естетика, збалансований у більшості ситуацій класовий апарат, фінансова модель розповсюдження та підтримка після офіційного запуску. До недоліків гри слід віднести баланс ігрових предметів, що з'являються з оновленням, загальний регрес естетики з доданням магазину косметичних предметів та деякі проблеми ареною змагань.

Гра Overwatch від компанії Blizzard Entertainment, випущена у 2016 році для основних ігрових платформ.

Фінансова модель дистрибуції. Гра Overwatch розповсюджується на комерційній основі з одноразовим придбання, доступний магазин всередині гри для покупки за реальні гроші косметичних предметів та аксесуарів, що не мають впливу на іхролад.

Естетика, візуальне та аудіо оформлення. Естетика гри виконана в стилі мультиплікації, схожому до Team Fortress 2, це стало причиною конкуренції між двома іграми. Події відбуваються в майбутньому, і це вплинуло на представлення озброєння, спорядження та здібностей персонажів гри. Візуальне виконання у поєднанні з професійним озвученням, характер персонажів та музикальний супровід створили привабливу естетику для гравців. На наш погляд, дещо естетика постраждала від впровадження магазину косметичних предметів та аксесуарів, як це сталося з Team Fortress 2 за аналогічними причинами. Приклад візуального оформлення представлено на рис. 1.2.

Рис. 1.2. Обкладинка гри з однією з героїнь

Технічна реалізація з точки зору гравця. З точки зору кінцевого користувача, гра вдало реалізована та оптимізована. В плані організації матчів усі обов’язки покладені на команду видавця, гравці лише запускають пошук гри.

Отже, Overwatch є успішним продуктом на ринку за колоритом естетики, цікавим персонажам, історії та іхроладним механікам. Вважаємо, варіативність гри дещо обмежена її дизайном, а платна форма дистрибуції є перешкодою до появи нових гравців.

1.3. Постановка задачі

Наукова робота полягає у створенні командної багатокористувацьої мережевої гри у жанрі Multiplayer First-Person Shooter. Необхідно розробити самостійний програмний продукт, доступний користувачам з безкоштовною моделлю розповсюдження, який буде використано для організації дозвілля та соціальних взаємодій.

Платформою програмного продукту є персональні комп'ютери з операційними системами Windows, прийнятні системи починаючи з Windows 7. Комп'ютерна гра має бути максимально простою у встановленні, з мінімальним набором операцій.

Комп'ютерна гра повинна мати засоби для створення облікових записів кожного гравця, відслідковувати ігрову статистику користувачів, створення ігрових сесій. Необхідно реалізувати можливості соціальної взаємодії користувачів під час ігрової сесії.

Враховуючи специфіку комп'ютерних ігор як програмного забезпечення, гра повинна мати мінімальні вимоги до апаратної складової персонального комп'ютера, а саме необхідна наявність:

1) відео-карти з підтримкою технології DirectX версії 9 або вище, що потребує як мінімум 512 мегабайт оперативної пам'яті для карти;

2) вільна оперативна пам'ять в обсязі не менше 1 гігабайту;

3) процесор з сумарною тактовою частотою ядер не менше ніж 2 гігагерци;

4) звукова карта для підтримки звуку.

Процес розробки мережевої комп'ютерної гри та її етапи: проектування гри та її модулів (вибір мережевої архітектури; створення дизайну ігроладу; створення дизайну естетики гри); програмування ігроладу; створення естетики гри; інтеграція складових частин програмного забезпечення; тестування та виправлення дефектів.

РОЗДІЛ 2
ПРОЕКТНІ РІШЕННЯ ТА ЇХ РЕАЛІЗАЦІЯ

2.1. Засоби розробки мережевої комп’ютерної гри

Для вибору використання програмного інструментарію проведена декомпозиція завдань з виділенням особливостей з технічної точки зору:

1) створення системи, яка забезпечить симуляцію фізичної взаємодії гравців: рух в горизонтальній та вертикальній площинах; гравітація; вплив на інші об’єкти; колізія один з одним тощо;

2) відображення графічних об’єктів на екрані монітору фізичною системою або її доповненням у вигляді окремого модулю;

3) відтворення звуку системою;

4) інтеграція в мережеву систему фізичної системи та її графічне зображення, з метою транслювання змін, зроблених одним з гравців усім іншим;

5) візуальне представлення об’єктів гри;

6) механізм ведення обліку даних користувачів.

Задачі 1 – 4 є складовими одного над-завдання, а саме створення ігрового двигуна.

Ігровий двигун – програмний комплекс, який надає розробнику можливості симуляції умов реального світу у цифровому просторі. Даний тип програмного забезпечення дає високорівневі програмні рішення для втілення ігроладних задумів.

Розрізняють два найбільш загальних підходи до використання ігрових двигунів при розробці ігор: власна реалізація; використання готових ринкових рішень.

Перший підхід передбачає, що усі функції, необхідні дизайнеру гри для її створення, розробляються власноруч з використанням бажаної мови програмування. Даний підхід потребує значних вмінь та знань для реалізації. До переваг підходу відноситься можливість покращеної оптимізації, унікальних

особливостей, не властивих іншим системам. До недоліків відносять вкрай високу складність реалізації, яка тягне за собою значні фінансові та часові витрати. Також, через масштабність завдання, значно підвищується кількість потенційних дефектів, які з’являються як при використанні розробниками, так і під час експлуатації кінцевим користувачем.

Використання готових ринкових рішень є наслідком першого підходу, за винятком того факту, що розробники гри в свою чергу є користувачами іншого програмного забезпечення – ігрового двигуна в даному конкретному випадку. Прийнято рішення скористатися готовими ринковими продуктами. Вибір ігрового двигуна обмежений мовою програмування, яку він підтримує.

Вирішено обрати ігровий двигун Unity Engine з наступних причин:

1) використання мови C# в якості основної – цей підхід надає баланс між швидкодією кінцевих продуктів та швидкістю і зручністю розробки;

2) вигідні умови ліцензування – безкоштовна розробка та випуск ігор командами, які зібрали на розробку менше ніж 100 тисяч доларів США або ж річний дохід яких не перевищує 100 тисяч доларів США;

3) широкий вибір засобів мережевого проектування;

4) наявність магазину готових пакетів – Unity Asset Store.

Враховуючи усі вищезазначені пункти та переваги обрано Unity Engine в якості ігрового двигуна для вирішення завдань 1 – 3.

Для розв’язання задачі 4 вирішено використовувати засоби, які надаються саме Unity Engine.

Обрано Unity Networking (Unet) та його складову Unity Matchmaking, яка доповнена власною реалізацією.

Для вирішення задачі 5 необхідні засоби комп’ютерного моделювання. Оскільки традиційно ігри в жанрі shooter є тривимірними, цей проект також з подібною естетикою. Для створення естетики з наявними ресурсами розробки обрано комбінований підхід – частина моделей взята з Unity Asset Store, інша змодельована власноруч. Для створення комп’ютерної графіки використано безкоштовний програмний пакет Blender. Інші опції, як от Autodesk 3Ds Max,

Autodesk Maya тощо не були використані через їх ціну, умови ліцензування та відсутність вбудованої інтеграції з Unity Engine напряму. Для створення елементів графічного інтерфейсу використано безкоштовний растровий редактор Gimp.

Для виконання завдання 6 вирішено використати готовий модуль з Unity Asset Store – Database Control Free. Дане рішення надає базу даних MySQL, розміщену у хмарному сервісі, для ведення обліку користувачів системи та їх даних. Даний модуль обмежений у своїй безкоштовній ліцензії, однак його обмеження задовольняють вимогам. Інший доступний варіант – це використання власної реалізації, що тягне за собою завдання проектування бази даних, її розміщення та підтримки. Подібні завдання не є прийнятними з огляду на доступні ресурси розробки.

Отже, обрано наступний інструментарій: Unity Engine та мову програмування C#; Unity Networking; Database Control; Blender; Gimp.

2.2. Проектування модулів мережевої комп’ютерної гри

Зважаючи на ресурси, виділені на розробку, доцільно розпочати проектування саме з мережевої архітектури, адже саме пропускні можливості підключення та спроможність програмної реалізації забезпечити необхідну швидкість роботи з даними, отриманими з мережі визначать подальший вплив на ігролад та естетику.

Проектування мережевої архітектури. Для реалізації мережевої складової гри вирішено використовувати Unity Networking, або ж Unet. Unet пропонує набір високорівневих програмних інтерфейсів (HLAPI, від англ. «high level application programming interface» [10]) які призначені для створення мережевих взаємодій [11]. Всі ігри, побудовані за допомогою даної системи, включають:

1) сервер – копія гри, до якої під’єднуються усі інші гравці при бажанні зіграти разом. Сервер, як правило, здійснює контроль над різноманітними

аспектами гри, такими як збереження рахунку гравців, перевірка законності запитів від користувачів, відправка даних клієнтам;

2) клієнти – це копії гри, які підключаються до серверу з інших комп'ютерів. Клієнти можуть підключатись як локальною мережею (LAN, від англ. «local area network»), так і через мережу Інтернет. Особа, що керує клієнтом, може грати з іншими людьми, які підключились до даного серверу зі своїх клієнтів.

Сервер з точки зору Unet може бути виділеним (host-сервером):

1) виділений сервер – це копія гри, яка запущена для того, щоб діяти в якості серверу, з метою забезпечення транспортного вузла та контролю стану гри;

2) host-сервер – у випадку відсутності виділеного серверу, один з клієнтів, які грають, бере на себе цю роль і стає host-сервером. Такий сервер створює копію гри, яка називається host, що одночасно працює як сервер і як клієнт.

На рис. 2.1 представлено будову гри, коли один з клієнтів виступає як host.

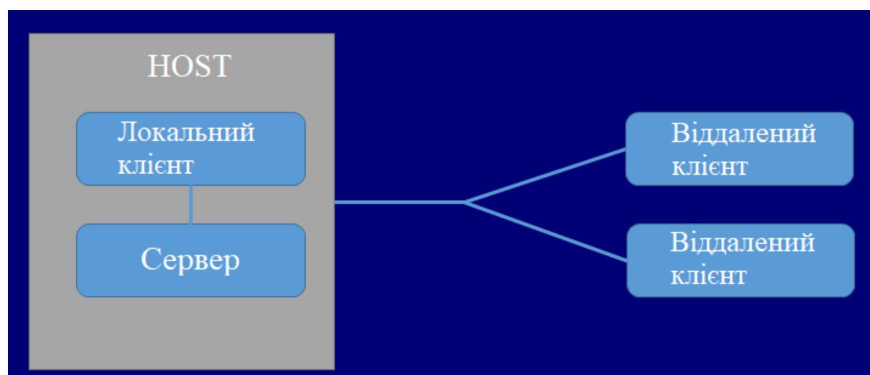

Рис. 2.1. Будова мережевої гри з host-сервером

Як видно з рис. 2.1, локальний клієнт підключається до серверу і відбувається запуск на одній машині. Віддалені клієнти запущені на інших пристроях їм необхідно підключитись до серверу.

В мережевих іграх існує два загальних підходи до ролі сервера:

1) авторитарний сервер – при такій моделі саме сервер, як уповноважена одиниця вирішує, чи міг гравець здійснити дію, про яку повідомив серверу, чи має права на знаходження у стані, про який повідомив. Сервер вирішує питання

компенсації часу відгуку по мережі, передбачення дій гравця для оптимізації швидкодії, аналізу стану гри, синхронізації гравців. Ця модель має переваги в ній реалізована можливість боротьби з недобросовісними гравцями та забезпечення більш прийнятного досвіду гри [12];

2) не-авторитарний сервер – даний тип серверу передає іншим гравцям дії певного конкретного клієнта, але не виконує їх самостійно і не перевіряє на достовірність. Отже, сервер довіряє клієнтам і не перевіряє їх дії і не коригує. Основними перевагами такого методу є збільшення швидкодії, за рахунок того, що сервер не здійснює обчислення перед відправкою даних іншим гравцям і затримка мережі є меншою. Подібний підхід є прийнятним для кооперативних ігор, при командній грі проти супротивників з контролем комп'ютера, де немає сенсу в недобросовісній грі, але для змагальних ігор подібна модель не доцільна [12].

З урахуванням ресурсів розробки та задач обрано архітектуру авторитарного host-серверу, інтегровану з сервісами Unity для роботи з підключенням гравців до серверу.

Створення ігроладу. Igролад комп'ютерної гри поділяється на дві складові – взаємодія гравця з грою поза ігровою сесією та під час ігрової сесії.

Взаємодія гравця поза ігровою сесією представлена на рис. 2.2.

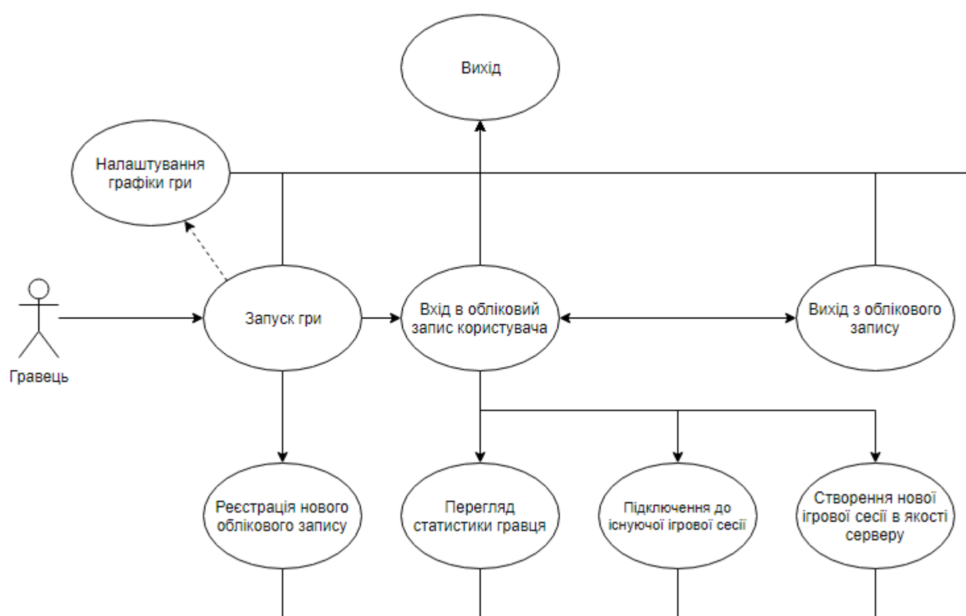

Рис. 2.2. Схематичне представлення взаємодії гравця поза ігровою сесією

Гравець має можливість проводити налаштування гри, переглядати статистику своїх досягнень, підключитись до існуючої ігрової сесії або створити нову, в якій даний конкретний гравець виступає в ролі host-серверу.

Взаємодія з грою на цьому етапі передбачена наступним чином:

1) користувач запускає гру, йому надається вікно налаштувань для обрання бажаного відображення графічних елементів, розмірів екрану монітору, прив'язки клавіш, режиму відображення гри у згорнутому чи розгорнутому вікні. На даному етапі користувач повинен мати можливість перервати налаштування, скасовуючи запуск гри, або підтвердити налаштування та запустити гру;

2) після підтвердження запуску перед користувачем відображаються логотипи Unity Engine та розробника гри;

3) відбувається вхід до головного меню, користувач повинен увійти в обліковий запис, якщо гравець його не має, він має можливість зареєструвати новий. Є можливість відразу вийти з програми;

4) після входу або реєстрації відбувається перехід до сторінки гравця, де гравцю представлено кількість зароблених очок, статистику гри, а також форми з переліком доступних ігрових сесій та для створення нової. На даному етапі гравець має можливість вийти з облікового запису або ж приєднатися до бажаної гри чи створити нову.

Процес гри відбувається в реальному часі та більшість дій, що здійснює гравець, таких як рух, атака, повороти камери, відбуваються здебільшого одночасно, на нашу думку не є вдалим зображення за допомогою діаграм. Перейдемо до словесного опису іґроладу з точки зору ігрової сесії.

Перш за все, за традицією ігор жанру shooter, гра повинна бути тривімірною для забезпечення більшої тактичної свободи та динаміки. Необхідним є створення гри від першої особи з міркувань балансу. Попри те, що існують ігри в жанрі shooter від третьої особи (Fortnite, Playersunknown Battlegrounds, Mass Effect), подібна перспектива не сприяє реалізму, оскільки гравцям доступні кути огляду, які неможливі для людини. Подібний підхід

ускладнює контроль стрільби та візуальне сприйняття якості.

Враховуючи вищезазначене, запропоновано процес взаємодії гравця під час ігрової сесії:

1) після підключення до ігрової сесії перед гравцем з'являється огляд ігрової мапи в реальному часі. Відображається меню з вибором команд, доступних для приєднання та кількість гравців в кожній з них. Для продовження, гравець має обрати одну з доступних команд або ж він може відразу покинути ігрову сесію;

2) після вибору команди перед гравцем з'являється інше меню для вибору класу персонажа. Персонажі відрізняються зовнішнім виглядом, наборами доступного спорядження та своїми ролями під час гри. Вибір персонажа залежить від вподобань гравця. Є можливість повернення до меню вибору команди. Для продовження необхідно обрати конкретний клас персонажа;

3) після вибору класу здійснюється у вікні вибір доступного спорядження (гравець може залишити доступні значення за замовчанням та продовжити, підтвердивши вибір);

4) після підтвердження, модель гравця генерується у ігровій сцені в одній з точок генерації гравців, або стартових позицій, які під умовним контролем команди гравця. З цього моменту управління передано гравцю;

5) мета кожного матчу полягає у ліквідації необхідної кількості супротивників за відведений час з командними зусиллями. Якщо команда досягає необхідної кількості, вона оголошується переможцем і матч завершується. Якщо час матчу закінчується до моменту досягнення будь-якою з команд критичного числа ліквідованих супротивників, переможець визначається за поточною кількістю переможених опонентів. У випадку рівного рахунку оголошується нічия;

6) для досягнення мети матчу гравці повинні ліквідовувати гравців іншої команди та уникати ліквідацій з власної сторони;

7) кожен гравець озброєний двома одиницями вогнепальної зброї різних характеристик, набором метальних снарядів – гранат різної потужності, та особливим предметом, тип якого визначається класом гравця;

8) кожен гравець має набір базових характеристик, які напряму впливають на його ефективність у грі:

a) здоров’я – умовна величина, що визначає, яку кількість пошкоджень може отримати гравець до того, як буде вважатися ліквідованим та втратить боєздатність і призведе до потреби у відродженні.

b) енергія – умовна величина, що використовується для вертикального переміщення вгору (польотів) та пришвидшення горизонтального пересування (спринту). Якщо гравець не має енергії, спринт та польоти стають недоступними до відновлення енергія не відновиться з плином часу самостійно;

c) боєзапас зброї – кількість амуніції, що доступна для кожної конкретної зброї гравця в даний момент часу;

d) запас метальних предметів – кількість метальних предметів, доступних у даний момент часу, це характеристика аналогічна боєзапасу;

9) окрім базових характеристик, існують показники кількості балів гравця в поточній сесії, які нараховуються за: ліквідацію супротивників; надання підтримки боєприпасами союзникам; надання підтримки медичними пакетами союзникам;

10) після завершення матчу гравці спостерігають за графічними елементами, що зображують результат матчу, усіх його учасників та їх бали, відбувається закриття матчу, а всі гравці повертаються в меню вибору ігрових сесій.

Ігролад в цілому налаштований на те, щоб гравці якомога швидше мали можливість приєднатись до матчу, обрати улюблений клас, спорядження та перейти до динамічної гри.

2.3. Створення естетики комп’ютерної гри

Важливим аспектом ігор жанру shooter є відповідність естетики ігроладу. Оскільки прийнято рішення реалізувати швидкий темп гри зі значною динамікою подій, яка, однак, не відповідає реалізму, необхідно закріпити ефект, створюваний ігроладом за допомогою відповідної естетики. У якості стилю оформлення вибрано малополігональний мінімалістичний стиль, відомий як low poly. Такий стиль властивий мультиплікації, що додатково підкреслює нереалістичні аспекти ігрових механік у вигідному світлі. Проектування естетики складається з розв’язання завдань: створення зовнішнього вигляду ігрового оточення; створення прототипу моделей гравців; концептуальний дизайн спорядження; розробка інтерфейсу користувача; анімація гравців та спорядження; інтеграція звукових ефектів.

Ігрове оточення. Для створення зовнішнього вигляду ігрового оточення використано наступні інструменти: Blender; вбудовані засоби Unity Engine; Pro Builder – офіційне доповнення Unity Engine для створення низькополігональної тривимірної графіки; доступні безкоштовні моделі оточення, отримані з Unity Asset Store.

В результаті прийнято рішення сформувати ігрову мапу в декораціях майданчику портового складу. Даний підхід дозволяє виконати усі завдання, які ставить перед дизайнером мап жанр shooter, а саме:

1) наявність укриття – критично важливо уникати пошкоджень від інших гравців. Саме тому ігрова мапа повинна бути насиченою об’єктами та укриттями, аби гравці могли швидко переміститися та сховатись від вогню супротивника;

2) поєднання стрілецьких дистанцій – важливим аспектом гри є правильне поєднання та чергування різних стрілецьких дистанцій. Наприклад, закриті приміщення або особливості місцевості, які потребують одночасного контролю багатьох точок появи супротивника, вимагають від гравця швидкої реакції та вміння. Такі вимоги не задовольняють потреби іншої групи гравців,

наприклад снайперів, яким потрібна лінія видимості до супротивника та значна відстань до нього. Обрані декорації дозволяють вміло поєднувати обидва види стрілецьких дистанцій;

3) створення «зон активності» – як правило, для забезпечення динамізму, потрібно максимізувати присутність гравців різних команд у певній ділянці. Оточення складських приміщень дозволяє ефективно створювати зони активності різних стрілецьких дистанцій;

4) вертикальний рух – враховуючи ігроладні особливості, обрані декорації чудово підходять для створення руху не лише в горизонтальній, а і у вертикальній площині, що значно збільшує можливості пересування та динаміку гри.

В цілому, ігрове оточення проектується таким чином, щоб сприяти підтримці динаміки гри, надавати можливості руху, створювати різноманітні маршрути та зони активності.

Моделі гравців. Гравці поділяються на два класи – бійця-медика та бійця підтримки. Вирішено стилізувати моделі гравців під роботів, що левітують на певній висоті над землею. Моделі гравців різних команд відрізняють кольором, властивим команді. У додатку А представлено моделі гравців, що використовуються у грі. Моделі гравців відрізняються один від одного та надають певне усвідомлення іншим гравцям, чого очікувати від даного конкретного бійця. Зброя даними роботами тримається магнітним полем, закладеним в конструкцію.

Моделі спорядження. Моделі спорядження, а саме зброї та боєприпасів, будувались на основі реальних прототипів зброї та деяких фантастичних варіацій, щоб підтримати футуристичний вигляд моделей гравців. Готові моделі зброї та спорядження до застосування текстур представлено у додатку Б. В цілому, естетика зброї та спорядження націлена на синергію з естетикою оточення та зовнішнім виглядом гравців.

Інтерфейс. Головні завдання, що стояли при проектуванні інтерфейсу користувача – простота та зрозумілість. Дизайн виконано в мінімалістичному

стилі для підтримки загальної естетики. Макети дизайну наведено у додатку В. Інтерфейс є простим та інтуїтивним, для зручності використання. Також, елементи інтерфейсу масштабуються відповідно до розміру екрана гравця.

Анімації. Більшість анімацій стосується зброї та взаємодії з нею. Попри те, що процес створення анімацій є трудомістким та зазвичай виконується у сторонніх програмах, Unity Engine надає вбудовані засоби для подібних завдань. Вони використані для анімації частин зброї під час стрільби, руху та перезарядки. Всі інші візуальні рухи відбуваються за рахунок фізичного двигуна і не є анімаціями.

Звуковий супровід. В якості звукового супроводу використано звукові бібліотеки з ліцензією використання Royalty Free. Озвучення реплік персонажів здійснено з використанням звукових доріжок з подібною ліцензією та частково самостійно. У гравців є можливість з використанням елементів інтерфейсу надсилати попередньо записані голосові команди мережею до інших гравців.

2.4. Програмна реалізація функцій

Після проектування ігроладу та дизайну, здійснено реалізацію спроектованих функцій. Для реалізації використано принципи об’єктно орієнтованого програмування, прийоми рефлексії та абстракції. Програмування здійснено за допомогою мови C# та платформи .Net. Для деяких аспектів роботи з освітленням графіки використовувалась мова програмування HLSL – high level shader language.

Реалізація взаємодій поза ігровою сесією. Після вибору налаштувань та запуску гри гравець потрапляє до екрану авторизації. На відміну від прикладного ПЗ та інших ігрових двигунів, у Unity Engine використовується поняття «сцени» для позначення контейнеру для об’єктів. Тому в одній сцені може бути довільна кількість так званих екранів. В даному випадку, всі дії гравця до реєстрації чи авторизації відбувають в межах однієї сцени – сцени авторизації.

Користувач повинен має змогу зареєструватись. В обов’язки форми

реєстрації входить перевірка введених даних –допускаються лише латинські символи для уникнення пошкодження цілісності бази даних та коректного відображення на машинах інших гравців, які можуть не мати відповідного кодування. Також, найголовнішим критерієм є необхідність у перевірці імен вже існуючих користувачів для уникнення дублікатів. У разі допущення помилок користувачем виводиться повідомлення про помилку. У випадку, якщо введені дані імені користувача та паролю відповідають нормам, користувач реєструється та відразу має змогу увійти в обліковий запис.

При натисканні на кнопку виходу з програми відбувається закриття гри зі звільненням усіх ресурсів, які були виділені на її роботу.

Взаємодія з базою даних здійснюється за допомогою пакету Database Control Free. Згідно умов ліцензії, детальна структура бази даних не розкривається кінцевому користувачу. Методом зворотного інжинірингу з’ясовано, що база даних складається з трьох атрибутів текстових типів, які зберігають дані ім’я користувача, його пароль та збірні додаткові дані. Шифрування не надається.

Двигун бази даних – MySQL. Відомості про серверну складову кінцевому користувачу не надаються. В подальшому, при розвитку проекту, можлива власна реалізація або ж оновлення ліценції вищезгаданого пакету для отримання додаткових можливостей. Вихідні коди модулів наведено у додатку Г.

Після авторизації користувач потрапляє до іншої сцени, а саме до так званого «лоббі». Дана сцена поєднує у собі наступні функції: відображення статистики користувача; надання форми для створення нової ігрової сесії; надання інтерфейсу для підключення до ігрової сесії.

В цілому, взаємодії гравця з грою поза матчем є простими та зводяться до взаємодії з інтерфейсом гри.

Взаємодія гравця під час ігрової сесії. Щойно гравець під’єднується до конкретної ігрової сесії, обирається одна з двох доступних команд. Команди відрізняються візуально та набором аудіо супроводу в розпорядженні гравців. Можливість приєднатися до певної команди обмежена поточною кількістю

гравців на її боці – дане обмеження обране з міркувань балансу, щоб уникнути нерівних команд. На етапі вибору сторони гравцю надається огляд поля бою для оцінки становища, а також можливість одразу ж покинути ігрову сесію.

Після вибору однієї з команд, перед гравцем постає вибір, який саме клас обрати. Доступні два класи – боєць-підтримки та медик. Форма вибору класів надає опис їх ролей, також є можливість повернення до форми вибору команди на випадок, якщо гравець передумав, або змінилась кількість гравців у іншій команді і є можливість переходу.

Після вибору класу персонажу, відбувається вибір спорядження, з яким гравець з'явиться у грі. Можливе повернення до вибору класу персонажа або ж безпосереднє дислокація гравця у ігровій сцені.

Функції дислокації гравця у сцені реалізовані з використанням трьох окремих модулів. Перший – менеджер дислокації – відповідальний за взаємодію користувача з інтерфейсом та передачу обраних параметрів на рівень програми. Другий – відповідає за взаємодію з менеджером мережевих дій, а саме за відправку низькорівневих транспортних запитів до серверу та всім іншим гравцям з серверу. Третій є розподіленим по окремим модулям з використанням часткових класів мови C#, відповідає за генерацію інтерфейсу користувача, генерацію графіки та логіки зброї для локального клієнта та усіх в межах ігрової сесії, створення та налаштування об'єкта клієнта в ігровій сцені. Відразу після дислокації гравцю доступні усі можливості управління. Розглянемо їх більш детально.

Рух. Передбачено рух у горизонтальній та вертикальній площинах. Кожен тип руху не може подолати цілісні перепони зовнішнього оточення – стіни, контейнери, бочки, інші гравці. Окрім цього, горизонтальний рух не обмежений нічим, в той час як вертикальний рух обмежений запасом енергії гравця. Синхронізація руху мережею відбувається від клієнта до серверу, а від нього – іншим клієнтам. Це здійснюється завдяки використанню високорівневих інтерфейсів від Unity Networking, а саме Network Identity компонентів.

Стрільба. Стрільба у грі є основним способом ліквідації супротивників.

У грі передбачено три режими стрільби: *одиночний* – кожний постріл відбувається після натискання на клавішу пострілу; *відсічкою* – на кожне натискання на клавішу пострілу відбувається фіксована послідовність пострілів; *автоматичний* – стрільба виконується безперервно з урахуванням темпу стрільби конкретної зброї та кількості боєприпасів, що лишилась в магазині.

Гранати. Вибухівка виконана таким чином, щоб симулювати політ гранати як фізичного тіла. Гранати виступають зброєю масового ураження, яка, поміж іншим, дозволяє наносити пошкодження без прямої лінії видимості.

Боєприпаси. Амуніція гравця прив’язана до його зброї, однак її відновлення є критичним аспектом гри. В цілому, принцип реалізації схожий на такий як для гранат – гравець, що грає бійцем-підтримки, натискає відповідну кнопку, модель гравця кидає так званий пакет з боєприпасами.

Медичні пакети. В цілому, повна аналогія з ящиками з боєприпасами у принципі дії, однак в даному контексті відновлюється здоров’я до максимуму та збільшена кількість очок за лікування союзників.

Інтерфейс користувача. Інтерфейс користувача є, здебільшого, графічним відображенням стану характеристик гравця. Саме тому реалізовано окремий модуль, який здійснює моніторинг таких змінних, як здоров’я, енергія, кількість набоїв та гранат. Також відображається так зване перехрестя прицілу, що дає уявлення про область можливого влучання пострілу в конкретний момент часу.

Меню голосових команд. Дане меню створене для того, щоб відтворити голосову команду певної тематики з метою привернення уваги до подій, що відбуваються.

Стрічка ліквідацій. Даний елемент інтерфейсу покликаний підтримувати актуальну інформацію про останніх ліквідованих гравців. Це дозволяє приблизно оцінювати стан матчу, хто саме є головною загрозою та, потенційно, його місцезнаходження. Стрічка оновлюється щоразу при ліквідації гравця будь-якої з команд. Ідентифікація команд здійснюється кольором.

Меню паузи. Дозволяє відрегулювати гучність звукових ефектів у грі, покинути кімнату або ж продовжити. Слід зазначити, що пауза не впливає на інших гравців, а призупиняє рух лише для поточного гравця.

Стрічка стану матчу. Відображає, скільки часу залишилось до кінця матчу, а також скільки ліквідацій має кожна з команд. Є критичним елементом для усвідомлення цілей матчу та мотивації гравця до дій.

Рейтингова таблиця. Надає поточний рейтинг усіх гравців для усвідомлення ефективності дій команд. Корисна для задання пріоритету цілей команди.

Завершення матчу. В момент, коли одна з команд досягає необхідної кількості ліквідацій або ж спливає час матчу, визначається команда переможець. Усім гравцям команди демонструється екран зі статусом матчу відповідно до їх команди, після цього відображається таблиця результатів. Далі, сервер обриває підключення до клієнтів повідомляючи, що матч завершено. Щойно останній клієнт покидає матч, сервер знищує матч та йде сам. Таким чином ігрова сесія не буде з'являтись у списку доступних, оскільки є завершеною.

Відродження. Після ліквідації, гравець втрачає контроль над персонажем та здатність діяти на певний період часу. На цей інтервал гравцю відображається меню вибору спорядження на випадок, якщо той побажає скористатись іншою зброєю чи змінити клас. Після завершення часу, що має минути до відродження, гравець може знову відродитися на одній зі стартових точок, що контролюються його командою.

Налаштування гри. Дана структура необхідна для комунікації з іншими гравцями та передачі їм базових даних про матч, стандартний час матчу, кольори команд, базова кількість ліквідацій для перемоги, назви команд, наявність чи відсутність пошкоджень по союзникам, їх коефіцієнт у разі наявності тощо. Ці налаштування недоступні для редагування гравцями і є закодованими у вихідний код гри.

ВИСНОВКИ

В результаті виконання наукової роботи здійснено проектування мережевої комп’ютерної гри у жанрі Multiplayer First-Person Shooter, її послідовну розробку та тестування.

Розробка мережевої комп'ютерної гри в жанрі Multiplayer First-Person Shooter має певні особливості і є унікальним процесом, що поєднує творчі здібності, технічний талант та вміння представити створений продукт. Комп’ютерна гра реалізована ітеративно, з використанням методології гнучкої розробки, цей підхід дозволив коригувати поставлені задачі у відповідності до умов. Під час розробки здійснено чітке планування завдань та розподіл їх пріоритету. Створення мережевої комп’ютерної гри на основі сучасних програмних інструментів дозволило значно спростити процес розробки та полегшити виконання низькорівневих завдань реалізації технічних деталей. Це забезпечило адаптивність розробки гри до нових вимог та ефективність використання ресурсів – як матеріальних, так і часових.

Серед варіантів розвитку мережевої комп’ютерної гри є наступні:

1) продовження фінансування, нарощування мережевої інфраструктури з Unity Engine та підготовка до комерційної дистрибуції у відповідних платформах;

2) припинення фінансових вкладень та безкоштовне розповсюдження проекту;

3) підготовка проекту до дистрибуції через Unity Asset Store з метою навчання бажаючих. Проект пропонує нетривіальні можливості з точки зору ігроладу та значний набір високоякісних моделей, які, у комплексі, можуть заохотити інших розробників отримати проект в освітніх цілях для використання у розробках.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Фурсова Н.А. Особливості розробки мережевої комп'ютерної гри в жанрі Multiplayer First-Person Shooter / Н.А. Фурсова, О.Є. Козак // Наука і виробництво: ДВНЗ «ПДТУ». –Маріуполь – 2019. – № 1 (20).

2. E. Avedon, B. Sutton-Smith The Ambiguity of Play: Harvard University Press, 2009. – 288 p.

3. G. Costikyan Uncertainty in Games: MIT Press, 2013 – 152 p.

4. Game design workshop : a playcentric approach to creating innovative games / T. Fullerton, Ch. Swain, S. Hoffman. –2nd ed. – 2008. – 491 p.

5. J. Schell The Art of Game Design: Carnegie Mellon University – 2008 – 518 p.

6. Hocking J. Unity in Action: Multiplatform Game Development in C# with Unity 5 1st Edition / Joe Hocking., 2015. – 352 с.

7. Isbister K. How Games Move Us: Emotion by Design (Playful Thinking) / Katherine Isbister. – Cambridge: The MIT Press, 2016. – 192 с.

8. Mayra F. An Introduction to Game Studies / Frans Mayra. – Thousand Oaks: SAGE Publications Ltd, 2008. – 208 с.

9. Фурсова Н.А. Розробка мережевої комп’ютерної гри з використанням Unity Engine // Н.А. Фурсова, О.Є. Козак /Тези 70-ої ювілейної наук. конф. проф., викл., наук. прац., аспір. та студ. університету. Том 2. (Полтава, 23 квітня – 18 травня 2018 р.) – Полтава: ПолтНТУ, 2018. – С. 244–245.

10. Networking HLAPI System Concepts [Електронний ресурс] // Unity Technologies. – 2017. – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/UNetConcepts.html.

11. UNet Overview [Електронний ресурс] // Unity Technologies. – 2017. – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/UNet.html.

12. Albahari J. C# 7.0 in a Nutshell: The Definitive Reference / J. Albahari, B. Albahari. – 1st ed. – Sebastopol: O'Reilly Media, 2017. – 1090 с.

ДОДАТКИ

Додаток А

Рис. А.1. – Модель гравця підтримки до застосування текстур, частина 1

Рис. А.2 – Модель гравця підтримки до застосування текстур, частина 2

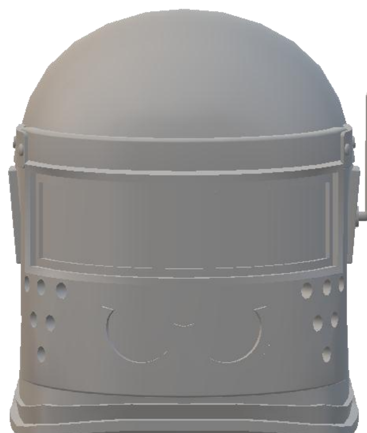

Рис. А.3 – Модель гравця підтримки до застосування текстур, частина 3

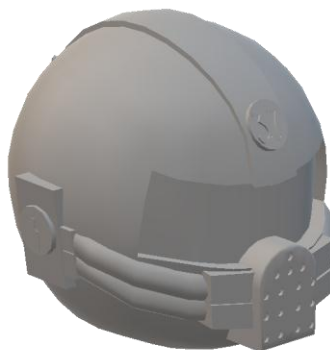

Рис. А.4 – Модель гравця медика до застосування текстур, частина 1

Рис. А.5 – Модель гравця підтримки до застосування текстур, частина 2

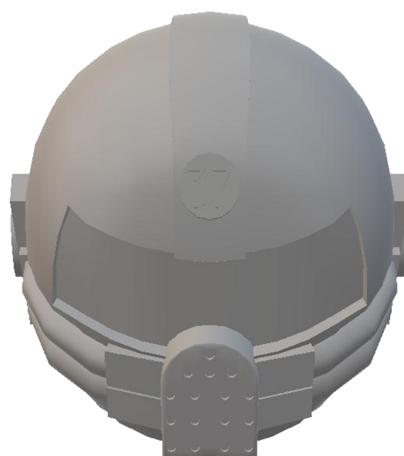

Рис. А.6 – Модель гравця медика до застосування текстур, частина 3

Додаток Б

Рис. Б.1 – Модель револьверу Raptor S4

Рис. Б.2 – Модель напівавтоматичного пістолету Raider MK2

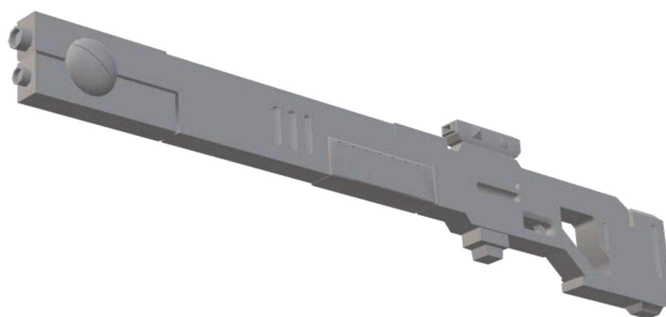

Рис. Б.3 – Модель гвинтівки FW45

Рис. Б.4 – Модель автоматичної гвинтівки LA-416

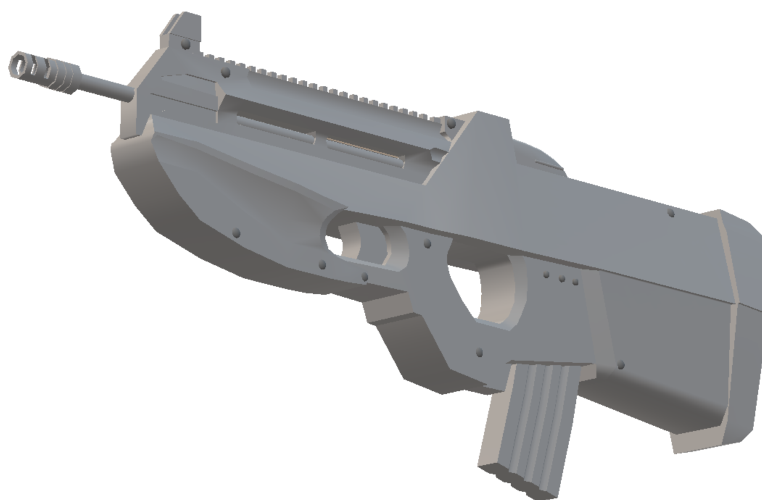

Рис. Б.5 – Модель автоматичної гвинтівки FAR-1000

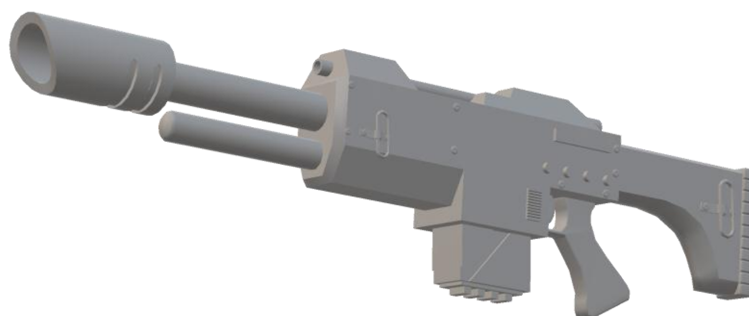

Рис. Б.6 – Модель дробовика Jungle Fighter

Рис. Б.7 – Модель гвинтівки GEWHER-7.62

Рис. Б.8 – Модель протипіхотної гранати M1

Рис. Б.9 – Модель лазерного пістолету Pigeon MK4

Рис. Б.10 – Модель пістолету Tazer X2

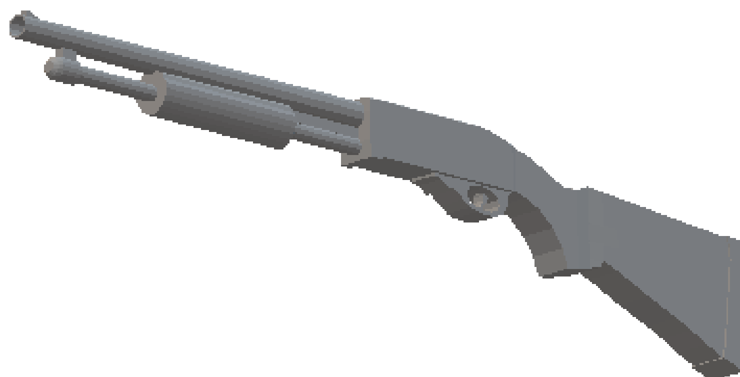

Рис. Б.11 – Модель помпового дробовика Rammington MCS Combat

Рис. Б.12 – Модель револьверу Colt «Old But Gold»

Рис. Б.13 – Модель дробовика Maretti 780

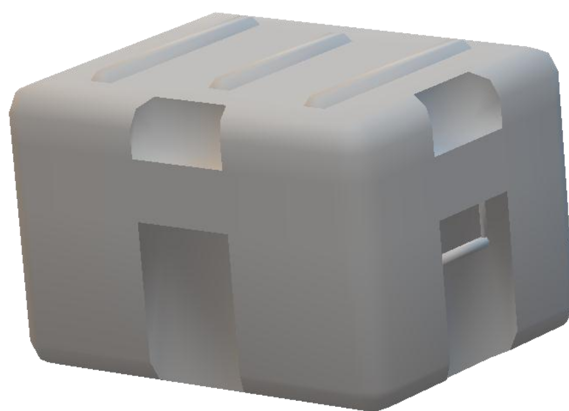

Рис. Б.14 – Модель пакету з боєприпасами

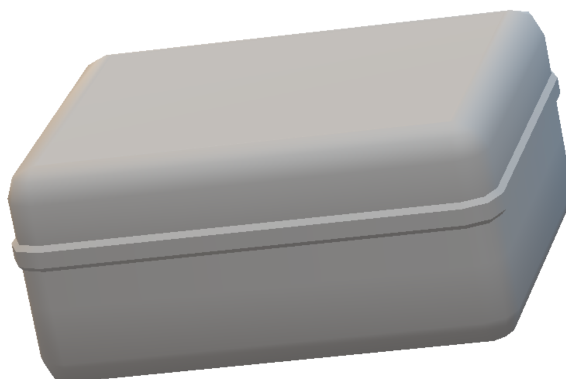

Рис. Б.15 – Модель медичного пакету

Додаток В

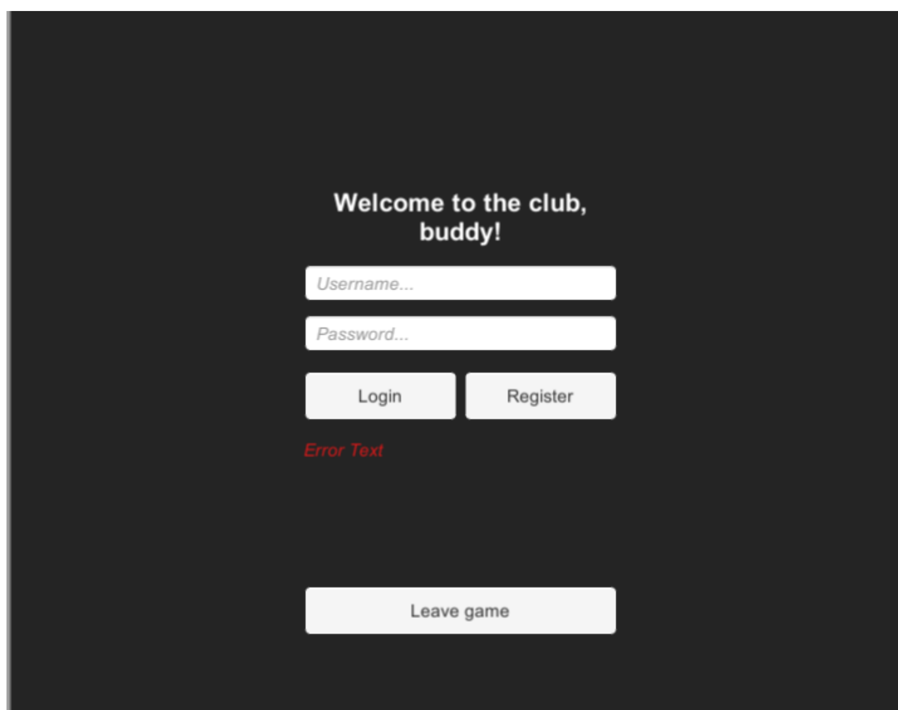

Рис. В.1 – Інтерфейс користувача після запуску гри

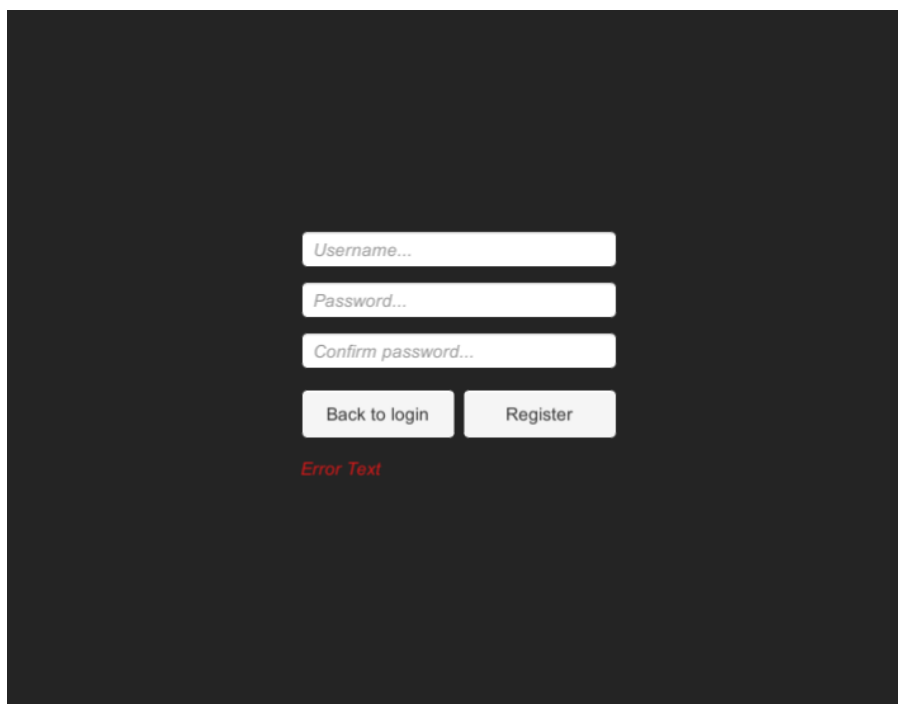

Рис. В.2 – Інтерфейс форми реєстрації нового облікового запису

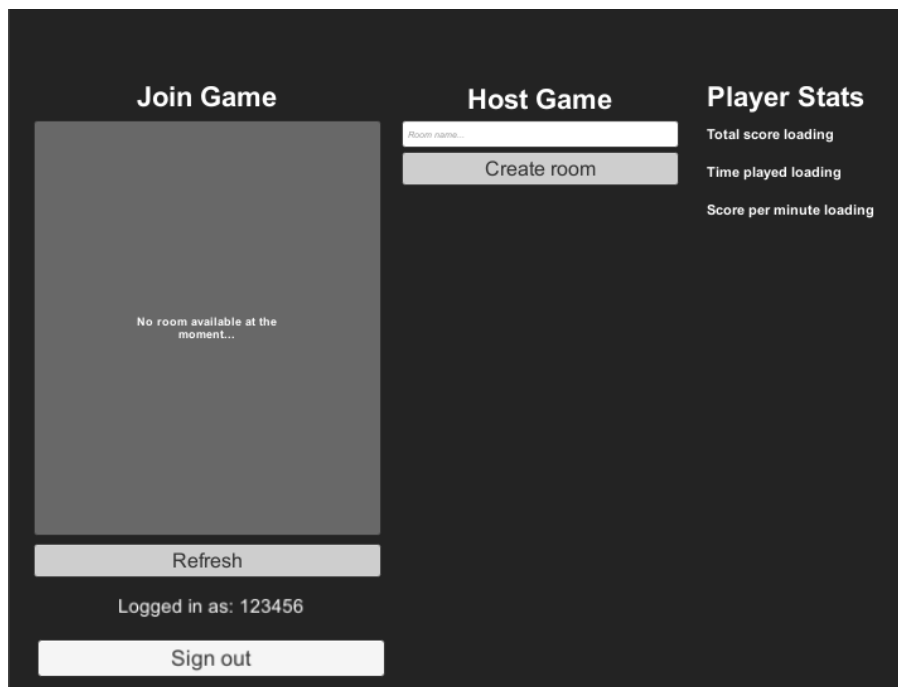

Рис. В.3 – Інтерфейс екрану пошуку та створення гри

Рис. В.4 – Інтерфейс вибору команди кімнати

Рис. В.5 – Інтерфейс вибору класу гравця

Рис. В.6 – Інтерфейс вибору спорядження гравця

Рис. В.7 – Інтерфейс таблиці очок учасників

Рис. В.8 – Інтерфейс меню паузи

Додаток Г

Вихідні коди програмного засобу

1. Контролер авторизації (LoginMenu.cs):

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using DatabaseControl;
public class LoginMenu : MonoBehaviour
{
    public GameObject loginParent;
    public GameObject registerParent;
    public GameObject loadingParent;
    public InputField Login_UsernameField;
    public InputField Login_PasswordField;
    public InputField Register_UsernameField;
    public InputField Register_PasswordField;
    public InputField Register_ConfirmPasswordField;
    public Text Login_ErrorText;
    public Text Register_ErrorText;
    void Awake(){
        ResetAllUIElements();
    }
    void ResetAllUIElements(){
        Login_UsernameField.text = "";
        Login_PasswordField.text = "";
        Register_UsernameField.text = "";
        Register_PasswordField.text = "";
        Register_ConfirmPasswordField.text = "";
        blankErrors();
    }
    void blankErrors(){
        Login_ErrorText.text = "";
        Register_ErrorText.text = "";
    }
    IEnumerator LoginUser(string username, string password){
        IEnumerator e = DCF.Login(username, password);
        while (e.MoveNext()){
            yield return e.Current;
        }
        string response = e.Current as string;
        if (response == "Success"){
            ResetAllUIElements();
            UserAccountManager.instance.LogIn(username, password);
        }
        else{
            loadingParent.gameObject.SetActive(false);
            loginParent.gameObject.SetActive(true);
            if (response == "UserError")
                Login_ErrorText.text = "Error: Username not Found";
```

```
            else{
                if (response == "PassError")
                    Login_ErrorText.text = "Error: Password Incorrect";
                else
                    Login_ErrorText.text = "Error: Unknown Error. Please try again later.";
            }
        }
    }
    IEnumerator RegisterUser(string username, string password, string data){
        IEnumerator e = DCF.RegisterUser(username, password, data);
        while (e.MoveNext()){
            yield return e.Current;
        }
        string response = e.Current as string;
        if (response == "Success"){
            ResetAllUIElements();
            UserAccountManager.instance.LogIn(username, password);
        }
        else{
            loadingParent.gameObject.SetActive(false);
            registerParent.gameObject.SetActive(true);
            if (response == "UserError")
                Register_ErrorText.text = "Error: Username Already Taken";
            else
                Login_ErrorText.text = "Error: Unknown Error. Please try again later.";
        }
    }
    public void Login_LoginButtonPressed(){
        string username = Login_UsernameField.text;
        string pass = Login_PasswordField.text;
        if (username.Length > 3){
            if (pass.Length > 5) {
                loginParent.gameObject.SetActive(false);
                loadingParent.gameObject.SetActive(true);
                StartCoroutine(LoginUser(username, pass));
            }
            else
                Login_ErrorText.text = "Error: Password format Incorrect (length must be > 5)";
        }
        else if (username.Length == 0 && pass.Length == 0)
            Login_ErrorText.text = "Error : Blank Field, Try again please.";
        else
            Login_ErrorText.text = "Error: Username format Incorrect (length must be > 3)";
    }
    public void Login_RegisterButtonPressed(){
        ResetAllUIElements();
        loginParent.gameObject.SetActive(false);
        registerParent.gameObject.SetActive(true);
    }
    public void Register_RegisterButtonPressed(){
        string username = Register_UsernameField.text;
        string password = Register_PasswordField.text;
        string confirmedPassword = Register_ConfirmPasswordField.text;
```

```
            if (username.Length > 3){
                if (password.Length > 5){
                    if (password == confirmedPassword){
                        registerParent.gameObject.SetActive(false);
                        loadingParent.gameObject.SetActive(true);
                        PlayerStatsObject initStats = new PlayerStatsObject(){
                            totalScore = 0,
                            scorePerMinute = 0,
                            timePlayed = 0.0f
                        };
                        StartCoroutine(RegisterUser(username, password, JsonUtility.ToJson(initStats)));
                    }
                    else
                        Register_ErrorText.text = "Error: Password's don't Match";
                }
                else
                    Register_ErrorText.text = "Error: Password too Short";
            }
            else
                Register_ErrorText.text = "Error: Username too Short";
        }
        public void Register_BackButtonPressed(){
            ResetAllUIElements();
            loginParent.gameObject.SetActive(true);
            registerParent.gameObject.SetActive(false);
        }
        public void LoggedIn_LogoutButtonPressed(){
            ResetAllUIElements();
            UserAccountManager.instance.LogOut();
            loginParent.gameObject.SetActive(true);
        }
}
```

2. Модуль обліку персональних даних (UserAccountManager.cs):

```
using System;
using System.Collections;
using UnityEngine;
using DatabaseControl;
using UnityEngine.SceneManagement;
public class UserAccountManager : MonoBehaviour
{
    public static UserAccountManager instance;
    public static string PlayerUsername { get; protected set; }
    private static string PlayerPassword = String.Empty;
    public delegate void OnDataReceivedCallback(string data);
    public static bool IsLoggedIn { get; protected set; }
    public string loggedInSceneName = "Lobby";
    public string loggedOutSceneName = "LoginMenu";
    void Awake(){
        if (instance != null){
            Destroy(gameObject);
            return;
        }
```

```
        instance = this;
        DontDestroyOnLoad(this);
    }
    public void LogOut(){
        PlayerUsername = String.Empty;
        PlayerPassword = String.Empty;
        IsLoggedIn = false;
        SceneManager.LoadScene(loggedOutSceneName);
    }
    public void LogIn(string _Username, string _Password){
        PlayerUsername = _Username;
        PlayerPassword = _Password;
        IsLoggedIn = true;
        SceneManager.LoadScene(loggedInSceneName);
    }
    public void SendData(string data){
        if (IsLoggedIn)
            StartCoroutine(SetData(data));
    }
    IEnumerator SetData(string data){
        IEnumerator e = DCF.SetUserData(PlayerUsername, PlayerPassword, data);
        while (e.MoveNext()){
            yield return e.Current;
        }
        string response = e.Current as string;
        if (response == "Success")
            Debug.Log("Succes sending data");
    }
    public void GetData(OnDataReceivedCallback onDataReceived) {
        if (IsLoggedIn)
            StartCoroutine(GetData_numerator(onDataReceived));
    }
    IEnumerator GetData_numerator(OnDataReceivedCallback onDataReceived){
        string data = "Error";
        IEnumerator e = DCF.GetUserData(PlayerUsername, PlayerPassword);
        while (e.MoveNext()){
            yield return e.Current;
        }
        string response = e.Current as string;
        if (response == "Error")
            Debug.Log("Error: Unknown Error. Please try again later. GetDataProblem");
        else
            data = response;
        if (onDataReceived != null)
            onDataReceived.Invoke(data);
    }
}
```

3. Модуль взаємодії з лоббі (UserAccount_Lobby.cs):

```
using UnityEngine.UI;
using UnityEngine;
public class UserAccount_Lobby : MonoBehaviour {
    public Text usernameText;
```

```
    void Start(){
        if (UserAccountManager.IsLoggedIn)
            usernameText.text = "Logged in as: " + UserAccountManager.PlayerUsername;
        else usernameText.text = "Not logged in.";
    }
    public void LogOut(){
        if (UserAccountManager.IsLoggedIn)
            UserAccountManager.instance.LogOut();
    }
}
```

4. Модуль елементу списку кімнат (RoomListItem.cs):

```
using UnityEngine;
using UnityEngine.Networking.Match;
using UnityEngine.UI;
using System;
public class RoomListItem : MonoBehaviour {
    public delegate void JoinRoomDelegate(MatchInfoSnapshot _match);
    private JoinRoomDelegate joinRoomCallback;
    [SerializeField] private Text roomNameText;
    [SerializeField] private Text roomSizeText;
    private MatchInfoSnapshot match;
    public void Setup(MatchInfoSnapshot _match, JoinRoomDelegate _joinRoomCallback){
        match = _match;
        joinRoomCallback = _joinRoomCallback;
        roomNameText.text = match.name;
        roomSizeText.text = String.Format("{0}/{1}", match.currentSize, match.maxSize);
    }
    public void JoinRoom(){
        joinRoomCallback.Invoke(match);
    }
}
```

5. Модуль для створення власного матчу (HostGame.cs):

```
using System;
using UnityEngine;
using UnityEngine.Networking;
public class HostGame : MonoBehaviour {
    [SerializeField] private uint roomSize = 20;
    private string roomName = "Default room name";
    private NetworkManager networkManager;
    private void Start(){
        networkManager = NetworkManager.singleton;
        if(networkManager.matchMaker == null)
            networkManager.StartMatchMaker();
    }
    public void SetRoomName (string _name){
        if (!String.IsNullOrEmpty(_name))
            roomName = _name;
    }
    public void CreateRoom(){
        if (roomName != String.Empty && roomName != null)
            networkManager.matchMaker.CreateMatch(roomName, roomSize, true,
```

```
                "", "", "", 0, 0, networkManager.OnMatchCreate);
    }
}
```

6. Модуль для підключення до існуючого матчу (JoinGame.cs):

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;
using UnityEngine.Networking.Match;
using System.Collections.Generic;
using System;
public class JoinGame : MonoBehaviour {
    [SerializeField] private Text status;
    [SerializeField] private GameObject roomListItemPrefab;
    [SerializeField] private Transform roomListParent;
    List<GameObject> roomList = new List<GameObject>();
    private NetworkManager networkManager;
    private void Start(){
        networkManager = NetworkManager.singleton;
        if(networkManager.matchMaker == null)
            networkManager.StartMatchMaker();
        RefreshRoomList();
    }
    public void RefreshRoomList(){
        ClearRoomList();
        networkManager.matchMaker.ListMatches(0, 20, "", false, 0, 0, OnMatchList);
        status.text = "Loading...";
    }
    public void OnMatchList(bool success, string extendedInfo, List<MatchInfoSnapshot> matchList){
        status.text = String.Empty;
        if(matchList == null){
            status.text = "Couldn't get room list.";
            return;
        }
        foreach (MatchInfoSnapshot match in matchList){
            if (NetworkManager_Custom.playerMatchStates.ContainsKey(match.networkId)){
                if (NetworkManager_Custom.playerMatchStates[match.networkId])
                    continue;
                else{
                    GameObject _roomListItemGameObject = Instantiate(roomListItemPrefab);
                    _roomListItemGameObject.transform.SetParent(roomListParent);
                    RoomListItem _roomListItem = _roomListItemGameObject.GetComponent<RoomListItem>();
                    if (_roomListItem != null)
                        _roomListItem.Setup(match, JoinRoom);
                    roomList.Add(_roomListItemGameObject);
                }
            }
            else{
                GameObject _roomListItemGameObject = Instantiate(roomListItemPrefab);
                _roomListItemGameObject.transform.SetParent(roomListParent);
                RoomListItem _roomListItem = _roomListItemGameObject.GetComponent<RoomListItem>();
                if (_roomListItem != null)
                    _roomListItem.Setup(match, JoinRoom);
```

```
                roomList.Add(_roomListItemGameObject);
            }
        }
        if (roomList.Count == 0)
            status.text = "No room available at the moment...";
    }
    void ClearRoomList(){
        foreach (GameObject roomItem in roomList){
            Destroy(roomItem);
        }
        roomList.Clear();
    }
    public void JoinRoom(MatchInfoSnapshot _match){
        if (!NetworkManager_Custom.playerMatchStates.ContainsKey(_match.networkId)){
            NetworkManager_Custom.playerMatchStates.Add(_match.networkId, false);
            NetworkManager_Custom.currentMatch = _match.networkId;
        }
        networkManager.matchMaker.JoinMatch(_match.networkId, "", "", "", 0, 0, networkManager.OnMatchJoined);
        ClearRoomList();
        status.text = "Joining...";
    }
}
```

7. Модуль управління мережею у матчі (NetworkManager_Custom.cs):

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.Networking.Match;
using UnityEngine.Networking.NetworkSystem;
public class NetworkManager_Custom : NetworkManager {
    [Space(10)]
    [Header("Deployment configuration")]
    [SerializeField] private GameObject[] playerPrefabs;
    internal string characterToDeploy;
    internal string teamToDeployOnto;
    internal static Dictionary<string, Player> playersOnServer = new Dictionary<string, Player>();
    internal static UnityEngine.Networking.Types.NetworkID currentMatch = 0;
    internal static Dictionary<UnityEngine.Networking.Types.NetworkID, bool> playerMatchStates = new Dictionary<UnityEngine.Networking.Types.NetworkID, bool>();
    public override void OnStartServer(){
        for (int i = 0; i < playerPrefabs.Length; i++){
            this.spawnPrefabs.Add(playerPrefabs[i]);
        }
        base.OnStartServer();
    }
    public override void OnClientConnect(NetworkConnection conn){
        RegisterConnection(conn.connectionId, conn);
        for (int i = 0; i < playerPrefabs.Length; i++){
            ClientScene.RegisterPrefab(playerPrefabs[i]);
        }
    }
```

```
public override void OnServerAddPlayer(NetworkConnection conn, short playerControllerId, NetworkReader extraMessageReader)
{
    string prefabName = String.Empty;
    string team = String.Empty;
    if (extraMessageReader != null){
        StringMessage message = extraMessageReader.ReadMessage<StringMessage>();
        string [] data = message.value.Split('|');
        prefabName = data[0];
        team = data[1];
    }
    GameObject selectedPlayerPrefab = null;
    for (int i = 0; i < playerPrefabs.Length; i++){
        if (prefabName == playerPrefabs[i].name)
            selectedPlayerPrefab = playerPrefabs[i];
    }
    if (selectedPlayerPrefab == null)
        selectedPlayerPrefab = playerPrefab;
    GameObject characterModel;
    Transform transform = GetStartPosition();
    characterModel = (GameObject)Instantiate(selectedPlayerPrefab, transform.position, transform.rotation);
    NetworkServer.AddPlayerForConnection(conn, characterModel, playerControllerId);
    foreach (KeyValuePair<string,Player> pair in GameManager.players)
    {
        if (pair.Value.GetComponent<WeaponManager>().getCurrentWeapon() != null){
pair.Value.GetComponent<WeaponManager>().EquipWeapon(pair.Value.GetComponent<WeaponManager>().getCurrentWeapon().weap
onInfo.weaponName);
        }
        if (conn.connectionId == pair.Value.GetComponent<NetworkIdentity>().connectionToClient.connectionId){
            pair.Value.GetComponent<PlayerSetup>().CmdSetTeam(pair.Key, team);
            break;
        }
pair.Value.GetComponent<WeaponManager>().EquipWeapon(pair.Value.GetComponent<WeaponManager>().getCurrentWeapon().weap
onInfo.weaponName);
    }
}
public override void OnStartClient(NetworkClient client){
    base.OnStartClient(client);
}
public override void OnClientDisconnect(NetworkConnection conn){
    UnregisterConnection(conn.connectionId);
}
public override void OnMatchCreate(bool success, string extendedInfo, MatchInfo matchInfo){
    if (success){
        currentMatch = matchInfo.networkId;
        playerMatchStates.Add(matchInfo.networkId, false);
        currentMatch = matchInfo.networkId;
    }
    base.OnMatchCreate(success, extendedInfo, matchInfo);
}
public static void RegisterConnection(int id, NetworkConnection conn){
    GameManager.clientConnections.Add(id, conn);
}
public void RegisterPlayerOnServer(string id, Player player) {
```

```
            playersOnServer.Add(id, player);
        }
        private void UnregisterConnection(int id){
            GameManager.clientConnections.Remove(id);
        }
        internal NetworkConnection GetConnection(int id){
            return GameManager.clientConnections[id];
        }
}
```

8. Модуль спостереження за станом матчу (MatchMonitor.cs):

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.Networking.Match;
public class MatchMonitor : NetworkBehaviour {
        public static MatchMonitor instance;
        [SerializeField] private float endgamePauseBeforeDrop = 14f;
        [SyncVar] private float matchTimer;
        [SyncVar] private int firstTeamKillCount = 0;
        [SyncVar] private int secondTeamKillCount = 0;
        internal string winnerTeamName = String.Empty;
        private int killCount;
        public bool isPlaying = true;
        public delegate void OnTeamScoreChangedCallback();
        public OnTeamScoreChangedCallback onTeamScoreChangedCallback;
        public delegate void OnMatchEndCallback();
        public OnMatchEndCallback onMatchEndCallback;
        void Start() {
            matchTimer = GameManager.instance.matchSettings.standardMatchTime;
            killCount = GameManager.instance.matchSettings.standardKillAmount;
        }
        private void Awake(){
            if (instance != null)
                Debug.LogError("More than one manager in the scene!");
            else
                instance = this;
        }
        void Update() {
            if (matchTimer > 0f)
                matchTimer -= Time.deltaTime;
            if ((matchTimer <= 0f && isPlaying) || (secondTeamKillCount == killCount || firstTeamKillCount == killCount)){
                isPlaying = false;
                CalculateWinnerTeam();
                CmdEndMatch();
            }
        }
        [Command]
        public void CmdEndMatch(){
            RpcEndMatchOnClients();
```

```
            EndMatchOnServer();
        }
        [ClientRpc]
        internal void RpcEndMatchOnClients(){
            if (isServer)
                return;
            StartCoroutine(EndMatch());
        }
        internal void EndMatchOnServer(){
            StartCoroutine(EndMatch());
            NetworkManager_Custom.singleton.matchMaker.DestroyMatch(NetworkManager_Custom.currentMatch, 0, OnMatchDestroy);
        }
        public void OnMatchDestroy(bool success, string extendedInfo) {
            if (success)
                Debug.Log("Match destroyed, info is: " + extendedInfo);
        }
        internal IEnumerator EndMatch(){
            onMatchEndCallback.Invoke();
            yield return new WaitForSeconds(endgamePauseBeforeDrop);
            MatchInfo matchInfo = NetworkManager_Custom.singleton.matchInfo;
            NetworkManager_Custom.playerMatchStates[NetworkManager_Custom.currentMatch] = true;
            NetworkManager_Custom.currentMatch = 0;
            NetworkManager_Custom.singleton.matchMaker.DropConnection(matchInfo.networkId, matchInfo.nodeId, 0,
NetworkManager_Custom.singleton.OnDropConnection);
            NetworkManager_Custom.singleton.StopClient();
        }
        private void CalculateWinnerTeam(){
            if (firstTeamKillCount > secondTeamKillCount)
                winnerTeamName = GameManager.instance.matchSettings.firstTeamName;
            else if (firstTeamKillCount < secondTeamKillCount)
                winnerTeamName = GameManager.instance.matchSettings.secondTeamName;
            else winnerTeamName = String.Empty;
        }
        public float GetMatchTime(){
            return matchTimer;
        }
        private void SetMatchTime(float _time) {
            matchTimer = _time;
        }
        internal void SetFirstTeamKill(){
            firstTeamKillCount++;
        }
        internal void SetSecondTeamKill(){
            secondTeamKillCount++;
        }
        public int GetFirstTeamKills(){
            return firstTeamKillCount;
        }
        public int GetSecondTeamKills(){
            return secondTeamKillCount;
        }
        public int GetStandardKillCount(){
            return killCount;
```

```
    }
}
```

9. Модуль управління грою на стороні гравця (GameManager.cs):

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using System.Linq;
public class GameManager : MonoBehaviour {
    [SerializeField] internal GameObject sceneCamera;
    public static GameManager instance;
    public MatchSettings matchSettings;
    public delegate void OnPlayerKilledCallback(string _killer, string _killerTeam, string _source, string _victim,
        string _victimTeam, bool isObserver);
    public OnPlayerKilledCallback onPlayerKilledCallback;
    [Space(10)]
    [Header("Spawnable weapon info")]
    [SerializeField] internal GameObject[] medicFirstSlot;
    [SerializeField] internal GameObject[] medicSecondSlot;
    [SerializeField] private GameObject[] medicThirdSlot;
    [SerializeField] internal GameObject[] supportFirstSlot;
    [SerializeField] internal GameObject[] supportSecondSlot;
    [SerializeField] private GameObject[] supportThirdSlot;
    internal Dictionary<string, GameObject> availableMedicWeapons = new Dictionary<string, GameObject>();
    void Awake(){
        if (instance != null) {
            Debug.LogError("More than one match monitors in the scene!");
        }
        else{
            RegisterMedicWeapons();
            instance = this;
        }
    }
    public void SetSceneCameraActive(bool isActive) {
        if (sceneCamera == null)
            return;
        sceneCamera.SetActive(isActive);
    }
    private const string PLAYER_ID_PREFIX = "Player ";
    internal static Dictionary<string, Player> players = new Dictionary<string, Player>();
    internal static Dictionary<int, NetworkConnection> clientConnections = new Dictionary<int, NetworkConnection>();
    public static void RegisterPlayer(string _netID, Player _player){
        string _playerID = PLAYER_ID_PREFIX + _netID;
        players.Add(_playerID, _player);
        _player.transform.name = _playerID;
    }
    public static void UnregisterPlayer(string _playerID) {
        players.Remove(_playerID);
    }
    public static Player GetPlayer(string _playerID){
        return players[_playerID];
    }
```

```
        public static Player[] GetPlayers(){
            return players.Values.ToArray();
        }
        private void RegisterMedicWeapons(){
            for (int i = 0; i < medicFirstSlot.Length; i++){
                availableMedicWeapons.Add(medicFirstSlot[i].GetComponent<PlayerWeapon>().weaponInfo.weaponName,
medicFirstSlot[i]);
            }
            for (int i = 0; i < medicSecondSlot.Length; i++){
                availableMedicWeapons.Add(medicSecondSlot[i].GetComponent<PlayerWeapon>().weaponInfo.weaponName,
medicSecondSlot[i]);
            }
        }
    }
```

10. Основні характеристики гравця (Player.cs):

```
using System.Collections;
using UnityEngine.Networking;
using UnityEngine;
using System;
[RequireComponent(typeof(PlayerSetup))]
[RequireComponent(typeof(DeploymentManager))]
public class Player : NetworkBehaviour
{
    [SyncVar] private bool _isDead = false;
    public bool isDead{
        get { return _isDead; }
        protected set { _isDead = value; }
    }
    [SerializeField] private GameObject playerGraphics;
    [SerializeField] private float maxHealth = 100f;
    [SyncVar] private float currentHealth;
    [SyncVar] public string username;
    [SyncVar] public string team;
    [SyncVar] public float score;
    public float scorePerMinute;
    public float timePlayed;
    [SerializeField] private int basePointsForKill = 20;
    [SerializeField] private int additionalPointsForKill = 0;
    [SerializeField] private float pointModifier = 1.0f;
    [SerializeField] private Behaviour[] disableOnDeath;
    [SerializeField] private GameObject[] disableGameObjectsOnDeath;
    private bool[] wasEnabled;
    private bool firstSetup = true;
    private void Start(){
        if (isLocalPlayer)
            playerGraphics.GetComponent<MeshRenderer>().enabled = false;
        MatchMonitor.instance.onMatchEndCallback += OnMatchEnd;
    }
    public void SetDefaults(){
        isDead = false;
        currentHealth = maxHealth;
        for (int i = 0; i < disableOnDeath.Length; i++){
```

```
            disableOnDeath[i].enabled = wasEnabled[i];
        }
        for (int i = 0; i < disableGameObjectsOnDeath.Length; i++){
            disableGameObjectsOnDeath[i].SetActive(true);
        }
        Collider _col = GetComponent<Collider>();
        if (_col != null)
            _col.enabled = true;
    }
    [ClientRpc]
    public void RpcTakeDamage(float _damage, string _sourceID, string _weaponName){
        if (isDead)
            return;
        Player damager = GameManager.GetPlayer(_sourceID);
        if (damager.team == this.team && damager.username != this.username) {
            if (GameManager.instance.matchSettings.isFriendlyFire) {
                float teamDamage = _damage * GameManager.instance.matchSettings.friendlyFireModifier;
                if (teamDamage > 0.0001f)
                    currentHealth -= teamDamage;
            }
        }
        else
            currentHealth -= _damage;
        if (currentHealth <= 0)
            Die(_sourceID, _weaponName);
        if (damager != null && damager.username != this.username && damager.team != this.team){
            if (currentHealth >= _damage)
                damager.score = damager.score + (_damage) * damager.pointModifier;
            else damager.score = damager.score + (currentHealth * this.pointModifier);
        }
    }
    [ClientRpc]
    internal void RpcHeal(float _amount){
        currentHealth += _amount;
    }
    IEnumerator Respawn(){
        Transform _spawnPoint = NetworkManager.singleton.GetStartPosition();
        transform.position = _spawnPoint.position;
        transform.rotation = _spawnPoint.rotation;
        yield return new WaitForSeconds(GameManager.instance.matchSettings.respawnTime);
        yield return new WaitForSeconds(0.2f);
        SetupPlayer();
    }
    private void Die(string _sourceID, string _weaponName){
        isDead = true;
        PlayerVoiceController _voiceController = GetComponent<PlayerController>().voiceController;
        _voiceController.PlaySound(VoiceCallType.Death);
        WeaponManager wm = GetComponent<WeaponManager>();
        wm.ResetOnDeath();
        for (int i = 0; i < disableOnDeath.Length; i++){
            disableOnDeath[i].enabled = false;
        }
        for (int i = 0; i < disableGameObjectsOnDeath.Length; i++){
```

```
            disableGameObjectsOnDeath[i].SetActive(false);
        }
        Collider _col = GetComponent<Collider>();
        if (_col != null)
            _col.enabled = false;
        if (isLocalPlayer){
            GameManager.instance.SetSceneCameraActive(true);
            GetComponent<PlayerSetup>().playerUIInstance.SetActive(false);
        }
        StartCoroutine(Respawn());
        Player killer = GameManager.GetPlayer(_sourceID);
        if (killer != null) {
            GameManager.instance.onPlayerKilledCallback.Invoke(killer.username, killer.team, _weaponName, this.username,
this.team, isLocalPlayer);
            if (killer.username != this.username && killer.team != this.team){
                killer.score = killer.score + (this.basePointsForKill + this.additionalPointsForKill) * killer.pointModifier;
                AddKillToTeam(killer);
            }
        }
    }
    private void AddKillToTeam(Player killer){
        if (killer.team == GameManager.instance.matchSettings.firstTeamName){
            MatchMonitor.instance.SetFirstTeamKill();
        }
        else if (killer.team == GameManager.instance.matchSettings.secondTeamName){
            MatchMonitor.instance.SetSecondTeamKill();
        }
        MatchMonitor.instance.onTeamScoreChangedCallback.Invoke();
    }
    public float GetHealth(){
        return currentHealth;
    }
    public float GetMaxHealth(){
        return maxHealth;
    }
    public void SetupPlayer(){
        if (isLocalPlayer {
            GameManager.instance.SetSceneCameraActive(false);
            GetComponent<PlayerSetup>().playerUIInstance.SetActive(true);
        }
        CmdBroadCastNewPlayerSetup();
    }
    private void OnMatchEnd(){
        PlayerController playerController = GetComponent<PlayerController>();
        playerController.PauseCurrentMovement();
        for (int i = 0; i < disableOnDeath.Length; i++){
            disableOnDeath[i].enabled = false;
        }
    }
    [Command]
    private void CmdBroadCastNewPlayerSetup(){
        RpcSetupPlayerOnAllClients();
    }
```

```
    [ClientRpc]
    private void RpcSetupPlayerOnAllClients(){
        if (firstSetup){
            wasEnabled = new bool[disableOnDeath.Length];
            for (int i = 0; i < wasEnabled.Length; i++)
            {
                wasEnabled[i] = disableOnDeath[i].enabled;
            }
            firstSetup = false;
        }
        SetDefaults();
    }
}
```

11. Модуль управління гравцем (PlayerController.cs):

```
using UnityEngine;
[RequireComponent(typeof(PlayerMotor))]
[RequireComponent(typeof(PlayerVoiceController))]
[RequireComponent(typeof(ConfigurableJoint))]
public class PlayerController : MonoBehaviour {
    [SerializeField] private float speed = 5f;
    [SerializeField] private float lookSensitivity = 5f;
    [SerializeField] private float thrusterForce = 1500f;
    [SerializeField] private float thrusterFuelBurnSpeed = 1f;
    [SerializeField] private float thrusterFuelRegenSpeed = 0.3f;
    [SerializeField] private float thrusterFuelAmount = 1f;
    public float getThrusterFuelAmount(){
        return thrusterFuelAmount;
    }
    private PlayerMotor motor;
    private ConfigurableJoint joint;
    internal PlayerVoiceController voiceController;
    [SerializeField] private LayerMask environmentMask;
    [Header("Spring settings:")]
    [SerializeField] private float jointSpring = 20f;
    [SerializeField] private float jointMaxForce = 40f;
    private void Start(){
        motor = GetComponent<PlayerMotor>();
        joint = GetComponent<ConfigurableJoint>();
        voiceController = GetComponent<PlayerVoiceController>();
        SetJointSettings(jointSpring);
    }
    private void Update(){
        if (PauseMenu.IsOn || GameManager.instance.sceneCamera.activeSelf){
            PauseCurrentMovement();
            if (Cursor.lockState != CursorLockMode.None)
                Cursor.lockState = CursorLockMode.None;
            return;
        }
        if (Cursor.lockState != CursorLockMode.Locked && !PlayerUI.isCommandOn && !PauseMenu.IsOn)
        {
            Debug.Log("Locking cursor");
            Cursor.lockState = CursorLockMode.Locked;
```

```
        }
        RaycastHit _springHit;
        if (Physics.Raycast (transform.position, Vector3.down, out _springHit, 100f, environmentMask)){
            joint.targetPosition = new Vector3(0f, -_springHit.point.y, 0f);
        }
        else
            joint.targetPosition = new Vector3(0f, -_springHit.point.y, 0f);
        float _xMov = Input.GetAxisRaw("Horizontal");
        float _zMov = Input.GetAxisRaw("Vertical");
        Vector3 _movHorizontal = transform.right * _xMov;
        Vector3 _movVertical = transform.forward * _zMov;
        Vector3 _velocity = (_movHorizontal + _movVertical).normalized * speed;
        motor.Move(_velocity);
        if (!PlayerUI.isCommandOn){
            float _yRot = Input.GetAxisRaw("Mouse X");
            Vector3 _rotation = new Vector3(0f, _yRot, 0f) * lookSensitivity;
            motor.Rotate(_rotation);
            float _xRot = Input.GetAxisRaw("Mouse Y");
            float _cameraRotationX = _xRot * lookSensitivity;
            motor.RotateCamera(_cameraRotationX);
        }
        else{
            motor.Rotate(Vector3.zero);
            motor.RotateCamera(0f);
        }
        PerformThrusterMovement();
    }
    private void PerformThrusterMovement(){
        Vector3 _thrusterForce = Vector3.zero;
        if (Input.GetButton("Jump") && thrusterFuelAmount > 0){
            thrusterFuelAmount -= thrusterFuelBurnSpeed * Time.deltaTime;
            if (thrusterFuelAmount > 0.01f){
                _thrusterForce = Vector3.up * thrusterForce;
                SetJointSettings(0f);
            }
        }
        else{
            SetJointSettings(jointSpring);
            thrusterFuelAmount += thrusterFuelRegenSpeed * Time.deltaTime;
        }
        thrusterFuelAmount = Mathf.Clamp(thrusterFuelAmount, 0f, 1f);
        motor.ApplyThruster(_thrusterForce);
    }
    internal void PauseCurrentMovement(){
        motor.Move(Vector3.zero);
        motor.Rotate(Vector3.zero);
        motor.RotateCamera(0f);
        SetJointSettings(jointSpring);
        thrusterFuelAmount += thrusterFuelRegenSpeed * Time.deltaTime;
        thrusterFuelAmount = Mathf.Clamp(thrusterFuelAmount, 0f, 1f);
        motor.ApplyThruster(Vector3.zero);
    }
    private void SetJointSettings(float _jointSpring)
```

```
    {
        joint.yDrive = new JointDrive{
            positionSpring = _jointSpring,
            maximumForce = jointMaxForce
        };
    }
}
```

12. Модуль управління рухом гравця (PlayerMotor.cs):

```
using System;
using UnityEngine;
[RequireComponent(typeof(Rigidbody))]
public class PlayerMotor : MonoBehaviour
{
    [SerializeField] private Camera cam;
    [SerializeField] private float cameraRotationLimit = 80f;
    [SerializeField] private GameObject graphicsObject;
    [SerializeField] private float graphicsRotationLimit = 30f;
    private Vector3 velocity = Vector3.zero;
    private Vector3 rotation = Vector3.zero;
    private float cameraRotationX = 0f;
    private float currentCameraRotationX = 0f;
    private float graphicsRotationX = 0f;
    private float currentGraphicsRotationX = 0f;
    private Vector3 thrusterForce = Vector3.zero;
    private Rigidbody rb;
    private void Start(){
        rb = GetComponent<Rigidbody>();
    }
    public void FixedUpdate(){
        PerformMovement();
        PerformRotation();
    }
    void PerformMovement(){
        if (velocity != Vector3.zero)
            rb.MovePosition(rb.position + velocity * Time.fixedDeltaTime);
        if (thrusterForce != Vector3.zero)
            rb.AddForce(thrusterForce * Time.fixedDeltaTime, ForceMode.Acceleration);
    }
    public void Rotate(Vector3 _rotation){
        rotation = _rotation;
    }
    public void RotateCamera(float _cameraRotationX){
        cameraRotationX = _cameraRotationX;
        graphicsRotationX = _cameraRotationX;
    }
    void PerformRotation(){
        rb.MoveRotation(rb.rotation * Quaternion.Euler(rotation));
        if (cam != null){
            currentCameraRotationX -= cameraRotationX;
            currentCameraRotationX = Mathf.Clamp(currentCameraRotationX, -cameraRotationLimit, cameraRotationLimit);
            currentGraphicsRotationX -= graphicsRotationX;
            currentGraphicsRotationX = Mathf.Clamp(currentGraphicsRotationX, -graphicsRotationLimit, graphicsRotationLimit);
```

```
                cam.transform.localEulerAngles = new Vector3(currentCameraRotationX, 0f, 0f);
                graphicsObject.transform.localEulerAngles = new Vector3(currentGraphicsRotationX / 2, 0f, 0f);
            }
        }
        public void Move(Vector3 _velocity){
            velocity = _velocity;
        }
        public void ApplyThruster(Vector3 _thrusterForce){
            thrusterForce = _thrusterForce;
        }
        public Vector3 GetCurrentVelocity(){
            return velocity;
        }
        public Vector3 GetCurrentRotation(){
            return rotation;
        }
        public Vector3 GetThrusterForce(){
            return thrusterForce;
        }
}
```

13. Модуль стрільби (PlayerShoot.cs):

```
using UnityEngine;
using UnityEngine.Networking;
using System.Collections;
[RequireComponent(typeof(WeaponManager))]
public class PlayerShoot : NetworkBehaviour {
        public float throwForce = 15f;
        private const string PLAYER_TAG = "Player";
        private const string DESTRUCTIBLE_TAG = "Destructible";
        private float fireTime;
        public float maxSpreadAngle = 0.5f;
        [SerializeField] internal Camera cam;
        [SerializeField] private LayerMask mask;
        internal PlayerWeapon currentWeapon;
        private Grenade currentThrowableItem;
        internal WeaponManager weaponManager;
        private PlayerMotor playerMotor;
        float currentGunZOffset = 0f;
        float currentGunXRotation = 0f;
        private void Start(){
            if (cam == null)
                this.enabled = false;
            weaponManager = GetComponent<WeaponManager>();
            weaponManager.onWeaponChangedCallback += OnWeaponChangedCallback;
            playerMotor = gameObject.GetComponent<PlayerMotor>();
        }
        private void Update(){
            if (!isLocalPlayer)
                return;
            currentWeapon = weaponManager.getCurrentWeapon();
            currentThrowableItem = weaponManager.getCurrentThrowableItem();
            if ((currentWeapon.weaponInfo.firingMode == WeaponFiringMode.Auto && !Input.GetButton("Fire1"))
```

```
|| (currentWeapon.weaponInfo.firingMode != WeaponFiringMode.Auto)){
    if (fireTime > 0f) fireTime -= Time.deltaTime;
    if (currentGunXRotation > 0f)
        currentGunXRotation -= Mathf.Lerp(currentGunXRotation, 0f, 0.5f);
    if (currentGunZOffset > 0f)
        currentGunZOffset -= Time.deltaTime;
}
if (currentWeapon.weaponInfo.firingMode == WeaponFiringMode.Auto)
{
    if (!Input.GetButton("Fire1"))
        CancelInvoke("Shoot");
}
if (PauseMenu.IsOn)
    return;
if (ManageReload())
    return;
if (currentWeapon.ammoInCurrentClip < currentWeapon.weaponInfo.maxBulletsInClip && currentWeapon.ammoLeft > 0)
{
    if (Input.GetButtonDown("Reload")){
        weaponManager.Reload();
        return;
    }
}
if (Input.GetKeyDown(KeyCode.G))
    ThrowExplosive();
if (currentWeapon.ammoInCurrentClip > 0 && !weaponManager.isReloading){
    if (currentWeapon.weaponInfo.firingMode == WeaponFiringMode.Single
        || currentWeapon.weaponInfo.firingMode == WeaponFiringMode.Shotgun)
    {
        if (Input.GetButtonDown("Fire1"))
            Shoot();
    }
    else if (currentWeapon.weaponInfo.firingMode == WeaponFiringMode.Auto){
        if (Input.GetButtonDown("Fire1") && !weaponManager.isFiring){
            InvokeRepeating("Shoot", 0f, 1 / currentWeapon.weaponInfo.fireRate);
        }
        else if (Input.GetButtonUp("Fire1")){
            CancelInvoke("Shoot");
        }
    }
}
weaponManager.currentWeaponModel.transform.localPosition = new
Vector3(weaponManager.currentWeaponModel.transform.localPosition.x,
    weaponManager.currentWeaponModel.transform.localPosition.y,
    -currentGunZOffset);
weaponManager.currentWeaponModel.transform.localEulerAngles = new Vector3(-currentGunXRotation,
    weaponManager.currentWeaponModel.transform.localEulerAngles.y,
    weaponManager.currentWeaponModel.transform.localEulerAngles.z);
}
private bool ManageReload(){
    if (currentWeapon.ammoInCurrentClip <= 0 && currentWeapon.ammoLeft > 0 &&
        (!weaponManager.isFiring || !weaponManager.isDrawing || !weaponManager.isReloading)){
        weaponManager.Reload();
```

```
                fireTime = 0f;
                return true;
            }
            else {
                return false;
            }
        }
        [ClientRpc]
        void RpcDisplayShootEffects(){
            weaponManager.getWeaponGraphics().muzzleFlash.Play();
        }
        [Command]
        void CmdOnShoot(){
            RpcDisplayShootEffects();
        }
        [ClientRpc]
        void RpcDisplayHitEffects(Vector3 _pos, Vector3 _normal){
            GameObject _hitEffect = (GameObject)Instantiate(weaponManager.getWeaponGraphics().hitEffectPrefab, _pos,
Quaternion.LookRotation(_normal));
            Destroy(_hitEffect, 2f);
        }
        [Client]
        void Shoot(){
            if (!isLocalPlayer || weaponManager.isReloading || weaponManager.isDrawing || weaponManager.isFiring)
                return;
            if (currentWeapon.ammoInCurrentClip > 0) {
                if (!weaponManager.isFiring) {
                    fireTime += currentWeapon.weaponInfo.firingMode == WeaponFiringMode.Auto ? Time.deltaTime :
currentWeapon.weaponInfo.baseAccuracy;
                    currentWeapon.ammoInCurrentClip--;
                    CmdOnShoot();
                    weaponManager.OnFire();
                    if (currentWeapon.weaponInfo.firingMode == WeaponFiringMode.Shotgun)
                    {
                        for (int i = 0; i < currentWeapon.weaponInfo.bulletsPerBurst; i++)
                        {
                            RaycastShot();
                        }
                    }
                    else
                        RaycastShot();
                }
            }
            else CancelInvoke("Shoot");
        }
        private void RaycastShot(){
            RaycastHit _hit;
            Quaternion _fireRotation = Quaternion.LookRotation(cam.transform.forward);
            Quaternion _randomRotation = Random.rotation;
            float maxCurrentSpread = CalculateCurrentSpreadAngle();
            _fireRotation = Quaternion.RotateTowards(_fireRotation, _randomRotation, Random.Range(0.0f, maxCurrentSpread));
            if (Physics.Raycast(cam.transform.position, _fireRotation * Vector3.forward, out _hit, currentWeapon.weaponInfo.range, mask))
            {
```

```
                float actualDamage = currentWeapon.weaponInfo.damageDropOff.Evaluate(_hit.distance /
currentWeapon.weaponInfo.range) * currentWeapon.weaponInfo.damage;
                if (_hit.transform.tag == PLAYER_TAG)
                    CmdPlayerShot(_hit.collider.name, actualDamage, transform.name, currentWeapon.weaponInfo.weaponName);
                if (_hit.transform.tag == DESTRUCTIBLE_TAG)
                    CmdDestructibleShot(_hit.collider.gameObject, actualDamage);
                CmdOnHit(_hit.point, _hit.normal);
            }
            if (currentWeapon.weaponInfo.firingMode == WeaponFiringMode.Auto || currentWeapon.weaponInfo.firingMode ==
WeaponFiringMode.Burst){
                currentGunZOffset = Mathf.Lerp(currentGunZOffset, fireTime / 8, 1f);
                currentGunXRotation = Mathf.Lerp(currentGunXRotation, maxCurrentSpread, 1f);
            }
        }
        public float CalculateCurrentSpreadAngle(){
            float moveModifier = 0f;
            if (playerMotor.GetCurrentVelocity() != Vector3.zero)
                moveModifier = currentWeapon.weaponInfo.moveSpreadModifier;
            if (playerMotor.GetThrusterForce() != Vector3.zero)
                moveModifier = moveModifier + (currentWeapon.weaponInfo.moveSpreadModifier * 2);
            return currentWeapon.weaponInfo.spreadPattern.Evaluate((fireTime + moveModifier) /
currentWeapon.weaponInfo.timeToMaxSpread) * (currentWeapon.weaponInfo.maxBulletSpreadAngle);
        }
        public float GetFireTime(){
            return fireTime;
        }
        private void OnWeaponChangedCallback(){
            fireTime = 0f;
        }
        [Command]
        void CmdPlayerShot(string _playerID, float _damage, string _sourceID, string _weaponName){
            Player _player = GameManager.GetPlayer(_playerID);
            _player.RpcTakeDamage(_damage, _sourceID, _weaponName);
        }
        [Command]
        void CmdDestructibleShot(GameObject _item, float _damage){
            Debug.Log("Item name is " + _item.name);
            Destructible ds = _item.GetComponentInParent<Destructible>();
            if (ds != null)
                ds.RpcTakeDamage(_damage);
            else Debug.Log("Object tagged as destructible, but has no proper component on it.");
        }
        [Command]
        void CmdOnHit(Vector3 _pos, Vector3 _normal){
            RpcDisplayHitEffects(_pos, _normal);
        }
        void ThrowExplosive(){
            if ( isLocalPlayer == false)
                return;
            if (weaponManager.currentThrowableAmount > 0)
                CmdSpawnGrenade();
        }
        [Command]
```

```
        void CmdSpawnGrenade(){
            if (weaponManager.currentThrowableAmount < 1)
                return;
            GameObject grenade = Instantiate(currentThrowableItem.gameObject, weaponManager.weaponHolder.transform.position, weaponManager.weaponHolder.transform.rotation);
            Rigidbody rb = grenade.GetComponent<Rigidbody>();
            rb.velocity = cam.transform.forward * throwForce;
            NetworkServer.Spawn(grenade);
            weaponManager.currentThrowableAmount--;
        }
    }
```

14. Модуль управління зброєю та спорядженням гравця (WeaponManager.cs):

```
    using UnityEngine;
    using UnityEngine.Networking;
    using System.Collections;
    using System.Collections.Generic;
    using System;
    [RequireComponent(typeof(Player))]
    class WeaponManager : NetworkBehaviour {
        [SerializeField]    private string weaponLayerName = "Weapon";
        [SerializeField]    private GameObject firstSlot;
        [SerializeField]    private GameObject secondSlot;
        [SerializeField]    private GameObject thirdSlot;
        [SerializeField]    private Grenade throwableItem;
        [SerializeField]    internal Transform weaponHolder;
        internal Grenade currentThrowableItem;
        internal PlayerWeapon currentWeapon;
        private WeaponGraphics currentGraphics;
        internal GameObject currentWeaponModel = null;
        private Player player;
        private Queue<GameObject> activeItems;
        [HideInInspector]    public bool isReloading = false;
        [HideInInspector]    public bool isFiring = false;
        [HideInInspector]    public bool isDrawing = false;
        [SyncVar]    internal float medkitTimer = 0f;
        [SyncVar]    internal float healthRefillTimer = 0f;
        [SyncVar]    internal float ammoRefillTimer = 0f;
        [SyncVar]    internal float grenadeRefillTimer = 0f;
        private float itemDropTimer = 0f;
        internal int maximumThrowableAmount = 0;
        [SyncVar]    internal int currentThrowableAmount = 0;
        internal Dictionary<string, GameObject> activatedWeapons = new Dictionary<string, GameObject>(5);
        [SyncVar]    internal string firstSlotWeapon = String.Empty;
        [SyncVar]    internal string secondSlotWeapon = String.Empty;
        [SyncVar]    internal string thirdSlotWeapon = String.Empty;
        [Space(10)]
        [Header("Spawnable weapon prefabs")]
        [SerializeField]    private GameObject[] spawnableWeapons;
        public delegate void OnWeaponChangedCallback();
        public OnWeaponChangedCallback onWeaponChangedCallback;
```

```
private void Start()
{
    player = GetComponent<Player>();
    InitSetup();
    currentThrowableItem = throwableItem;
    currentThrowableItem.owner = player;
    maximumThrowableAmount = currentThrowableItem.maximumCarried;
    currentThrowableAmount = maximumThrowableAmount;
    if (thirdSlot.GetComponent<ThrowableActionable>() != null)
        activeItems = new Queue<GameObject>(thirdSlot.GetComponent<ThrowableActionable>().maxInstances);
}
private void Update()
{
    if (ammoRefillTimer > 0f){
        ammoRefillTimer -= Time.deltaTime;
    }
    if (healthRefillTimer > 0f){
        healthRefillTimer -= Time.deltaTime;
    }
    if (medkitTimer > 0f){
        medkitTimer -= Time.deltaTime;
    }
    if (itemDropTimer > 0f){
        itemDropTimer -= Time.deltaTime;
    }
    if (grenadeRefillTimer > 0f){
        grenadeRefillTimer -= Time.deltaTime;
    }
    if (PauseMenu.IsOn || PlayerUI.isCommandOn){
        return;
    }
    if (Input.GetKeyDown(KeyCode.Alpha1)){
        EquipWeapon(firstSlotWeapon);
    }
    if (Input.GetKeyDown(KeyCode.Alpha2)){
        EquipWeapon(secondSlotWeapon);
    }
    if (Input.GetKeyDown(KeyCode.Alpha3)){
        if (thirdSlot.GetComponent<PlayerWeapon>() != null)
            EquipWeapon(thirdSlot.GetComponent<PlayerWeapon>().weaponInfo.weaponName);
        else
ThrowItem(player.GetComponent<PlayerShoot>().cam.transform.forward, player.GetComponent<PlayerShoot>().throwForce * 0.4f);
    }
}
void InitSetup()
{
    for (int i = 0; i < spawnableWeapons.Length; i++)
    {
        SpawnWeaponDirty(spawnableWeapons[i].GetComponent<PlayerWeapon>(), false);
    }
    firstSlotWeapon = DeploymentManager.firstSlotName;
    secondSlotWeapon = DeploymentManager.secondSlotName;
    thirdSlotWeapon = DeploymentManager.thirdSlotName;
```

```
        CmdSetWeaponName(1, DeploymentManager.firstSlotName);
        CmdSetWeaponName(2, DeploymentManager.secondSlotName);
        firstSlot = activatedWeapons[firstSlotWeapon];
        secondSlot = activatedWeapons[secondSlotWeapon];
        EquipWeapon(firstSlot.GetComponent<PlayerWeapon>().weaponInfo.weaponName);
        StartCoroutine(WeaponDrawDelay());
    }
    internal void EquipWeapon(string _weaponName)
    {
        if (!isLocalPlayer)
            return;
        if (isReloading)
            return;
        if (currentWeapon != null){
            if (currentWeapon.weaponInfo.weaponName == _weaponName){
                ToggleWeapon(true, currentWeapon.weaponInfo.weaponName);
                return;
            }
        }
        if (activatedWeapons.ContainsKey(_weaponName)){
            ToggleWeapon(false, currentWeapon.weaponInfo.weaponName);
            currentWeapon = activatedWeapons[_weaponName].gameObject.GetComponent<PlayerWeapon>();
            StartCoroutine(WeaponDrawDelay());
            currentWeaponModel = activatedWeapons[_weaponName];
            ToggleWeapon(true, currentWeapon.weaponInfo.weaponName);
            onWeaponChangedCallback.Invoke();
            return;
        }
    }
    void SpawnWeaponDirty(PlayerWeapon _weapon, bool state){
        GameObject _weaponIns = Instantiate(_weapon.gameObject, weaponHolder.position, weaponHolder.rotation);
        _weaponIns.transform.SetParent(weaponHolder);
        currentGraphics = _weaponIns.GetComponentInChildren<WeaponGraphics>();
        if (currentGraphics == null)
            Debug.LogError("No WeaponGraphics component attached to object: " + _weaponIns.name);
        if (isLocalPlayer){
            Util.SetLayerRecursively(_weaponIns, LayerMask.NameToLayer(weaponLayerName));
        }
        _weaponIns.SetActive(state);
        currentWeaponModel = _weaponIns;
        activatedWeapons.Add(_weapon.weaponInfo.weaponName, _weaponIns);
        currentWeapon = activatedWeapons[_weapon.weaponInfo.weaponName].gameObject.GetComponent<PlayerWeapon>();
        currentWeapon.SetDefaults();
    }
    public PlayerWeapon getCurrentWeapon(){
        return currentWeapon;
    }
    public Grenade getCurrentThrowableItem(){
        return currentThrowableItem;
    }
    public WeaponGraphics getWeaponGraphics(){
       return currentGraphics;
    }
```

```
public void Reload(){
    if (isReloading || !isLocalPlayer)
        return;
    StartCoroutine(ReloadDelayCoroutine());
}
public void OnFire(){
    if (isFiring || !isLocalPlayer)
        return;
    StartCoroutine(WeaponFireDelay());
}
private void ToggleWeapon(bool state, string weaponName){
    if (!isLocalPlayer)
        return;
    CmdToggleWeapon(state, weaponName);
}
private IEnumerator WeaponDrawDelay(){
    isDrawing = true;
    yield return new WaitForSeconds(currentWeapon.weaponInfo.drawTime);
    isDrawing = false;
}
private IEnumerator ReloadDelayCoroutine(){
    isReloading = true;
    if (currentWeapon.ammoInCurrentClip > 0){
        CmdOnReload(false);
        yield return new WaitForSeconds(currentWeapon.weaponInfo.reloadTimeQuick);
    }
    else{
        CmdOnReload(true);
        yield return new WaitForSeconds(currentWeapon.weaponInfo.reloadTimeLong);
    }
    if (currentWeapon.ammoLeft >= (currentWeapon.weaponInfo.maxBulletsInClip - currentWeapon.ammoInCurrentClip)) {
        currentWeapon.ammoLeft = currentWeapon.ammoLeft - (currentWeapon.weaponInfo.maxBulletsInClip - currentWeapon.ammoInCurrentClip);
        currentWeapon.ammoInCurrentClip = currentWeapon.weaponInfo.maxBulletsInClip;
    }
    else{
        currentWeapon.ammoInCurrentClip += currentWeapon.ammoLeft;
        currentWeapon.ammoLeft -= currentWeapon.ammoLeft;
    }
    isReloading = false;
    CmdSetAmmoAmount(currentWeapon.ammoLeft, currentWeapon.weaponInfo.weaponName);
}
private IEnumerator WeaponFireDelay(){
    isFiring = true;
    CmdOnFire();
    yield return new WaitForSeconds(currentWeapon.weaponInfo.minShootTime);
    isFiring = false;
}
[Command]
private void CmdToggleWeapon(bool state, string weaponName){
    activatedWeapons[weaponName].SetActive(state);
    if (state == true)
        currentGraphics = activatedWeapons[weaponName].GetComponentInChildren<WeaponGraphics>();
```

```
        RpcToggleWeapon(state, weaponName);
}
[Command]
private void CmdSetAmmoAmount(int ammo, string weaponName){
        activatedWeapons[weaponName].GetComponent<PlayerWeapon>().ammoLeft = ammo;
}
[ClientRpc]
private void RpcToggleWeapon(bool state, string weaponName){
        activatedWeapons[weaponName].SetActive(state);
        if (state == true){
                currentGraphics = activatedWeapons[weaponName].GetComponentInChildren<WeaponGraphics>();
        }
}
[Command]
void CmdOnReload(bool longReload){
        RpcOnReload(longReload);
}
[ClientRpc]
void RpcOnReload(bool longReload){
        Animator anim = currentGraphics.GetComponent<Animator>();
        anim.ResetTrigger("Fire");
        if (anim == null)
                return;
        else{
                if (longReload)
                        anim.SetTrigger("LongReload");
                else anim.SetTrigger("Reload");
        }
}
[Command]
private void CmdOnFire(){
        RpcOnFire();
}
[ClientRpc]
private void RpcOnFire(){
        Animator anim = currentGraphics.GetComponent<Animator>();
        if (anim == null)
                return;
        else
                anim.SetTrigger("Fire");
}
[Command]
public void CmdSetWeaponName(int _slotNumber, string _weaponName){
        RpcSetWeaponName(_slotNumber, _weaponName);
}
[ClientRpc]
private void RpcSetWeaponName(int _slotNumber, string _weaponName) {
        switch (_slotNumber)
        {
                case 1:
                        firstSlotWeapon = _weaponName;
                        break;
        }
```

```
}
public void ResetOnDeath(){
    firstSlot.GetComponent<PlayerWeapon>().SetDefaults();
    secondSlot.GetComponent<PlayerWeapon>().SetDefaults();
}
void ThrowItem(Vector3 direction, float throwForce){
    if (isLocalPlayer == false)
        return;
    if (itemDropTimer <= 0.001f){
        ThrowableActionable throwable = thirdSlot.GetComponent<ThrowableActionable>();
        if (throwable == null)
            return;
        if (activeItems.Count >= throwable.maxInstances)
            CmdDestroyFirstThrowedItem();
        CmdThrowItem(direction, throwForce);
        if (activeItems.Count < throwable.maxInstances)
            itemDropTimer = throwable.minimumSpawnDelay;
        if (activeItems.Count >= throwable.maxInstances)
            itemDropTimer = throwable.spawnDelay;
    }
}
private void DestroyFirstThrowedItem(){
    NetworkServer.UnSpawn(activeItems.Peek());
    Destroy(activeItems.Peek());
    activeItems.Dequeue();
}
[Command]
private void CmdDestroyFirstThrowedItem(){
    NetworkServer.UnSpawn(activeItems.Peek());
    Destroy(activeItems.Peek());
    activeItems.Dequeue();
    TargetUnregisterActiveItem(connectionToClient);
}
[Command]
void CmdThrowItem(Vector3 direction, float throwForce) {
    GameObject throwable = Instantiate(thirdSlot, weaponHolder.transform.position, weaponHolder.transform.rotation);
    ThrowableActionable ta = throwable.GetComponent<ThrowableActionable>();
    ta.ownerName = this.player.name;
    Rigidbody rb = throwable.GetComponent<Rigidbody>();
    rb.velocity = direction * throwForce;
    NetworkServer.Spawn(throwable);
    activeItems.Enqueue(throwable);
    TargetRegisterActiveItem(connectionToClient, throwable);
}
[TargetRpc]
public void TargetRegisterActiveItem(NetworkConnection connection, GameObject activeItem){
    if (!isServer)
        activeItems.Enqueue(activeItem);
}
[TargetRpc]
private void TargetUnregisterActiveItem(NetworkConnection connection){
    if (!isServer)
        activeItems.Dequeue();
```

```
        }
}
```

15. Основний модуль інтерфейсу користувача (PlayerUI.cs):

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class PlayerUI : MonoBehaviour {
    [SerializeField] RectTransform thrusterFuelFill;
    [SerializeField] Text healthText;
    [SerializeField] Text ammoText;
    [SerializeField] Text throwableText;
    [SerializeField] GameObject pauseMenu;
    [SerializeField] GameObject scoreboard;
    [SerializeField] GameObject commandMenu;
    [SerializeField] GameObject crosshair;
    [SerializeField] GameObject matchMonitorUI;
    [SerializeField] Text matchTimer;
    [SerializeField] RectTransform firstTeamProgressBar;
    [SerializeField] RectTransform secondTeamProgressBar;
    [SerializeField] GameObject InGameUI;
    [SerializeField] GameObject EndGameUI;
    [SerializeField] Text gameEndStatus;
    [SerializeField] CanvasScaler rootScaler;
    [SerializeField] Transform crosshairContainer;
    private Transform crossTop;
    private Transform crossBottom;
    private Transform crossLeft;
    private Transform crossRight;
    private float crosshairDistance = 0f;
    private float previousOffset = 0f;
    internal Player player;
    private PlayerShoot playerShoot;
    private PlayerController controller;
    private WeaponManager weaponManager;
    private RMF_RadialMenu radialMenuController;
    public static bool isCommandOn = false;
    private float timeToCommandMenu = 0.2f;
    private float commandTimer = 0f;
    private void Start()
    {
        PauseMenu.IsOn = false;
        rootScaler.referenceResolution = new Vector2(Screen.currentResolution.width, Screen.currentResolution.height);
        radialMenuController = commandMenu.GetComponentInChildren<RMF_RadialMenu>();
        GameManager.instance.onPlayerKilledCallback += OnKill;
        MatchMonitor.instance.onTeamScoreChangedCallback += OnTeamScoreChanged;
        MatchMonitor.instance.onMatchEndCallback += OnMatchEnd;
        playerShoot = player.GetComponent<PlayerShoot>();
        SetCrosshairParts();
        firstTeamProgressBar.GetComponent<Image>().color = GameManager.instance.matchSettings.firstTeamColor;
        secondTeamProgressBar.GetComponent<Image>().color = GameManager.instance.matchSettings.secondTeamColor;
```

```
            firstTeamProgressBar.GetComponent<Image>().fillAmount = 0f;
            secondTeamProgressBar.GetComponent<Image>().fillAmount = 0f;
        }
        private void OnTeamScoreChanged(){
            SetKillCounts();
        }
        private void OnMatchEnd(){
            InGameUI.SetActive(false);
            Cursor.lockState = CursorLockMode.Confined;
            if (!MatchMonitor.instance.winnerTeamName.Equals(String.Empty))
                gameEndStatus.text = player.team == MatchMonitor.instance.winnerTeamName ? "VICTORY" : "DEFEAT";
            else gameEndStatus.text = "DRAW";
            EndGameUI.SetActive(true);
        }
        void Update() {
            SetFuelAmount(controller.getThrusterFuelAmount());
            SetHealthAmount(player.GetHealth());
            SetAmmoAmount(weaponManager.getCurrentWeapon().ammoInCurrentClip, weaponManager.getCurrentWeapon().ammoLeft,
weaponManager.currentThrowableAmount);
            SetMatchTimer();
            UpdateCrosshairPosition();
            if (Input.GetKeyDown(KeyCode.Escape))
                TogglePauseMenu();
            if (Input.GetKey(KeyCode.Q))
                if (isCommandOn == false) {
                    commandTimer += Time.deltaTime;
                    if (commandTimer > timeToCommandMenu)
                        ToggleVoiceCommandMenu();
                }
            if (Input.GetKeyUp(KeyCode.Q)){
                commandTimer = 0f;
                if (isCommandOn == true)
                    ToggleVoiceCommandMenu();
            }
            if (Input.GetKeyDown(KeyCode.Tab))
                scoreboard.SetActive(true);
            else if (Input.GetKeyUp(KeyCode.Tab))
                scoreboard.SetActive(false);
        }
        private void SetKillCounts(){
            firstTeamProgressBar.GetComponent<Image>().fillAmount = ((float)(MatchMonitor.instance.GetFirstTeamKills()) /
((float)MatchMonitor.instance.GetStandardKillCount()));
            secondTeamProgressBar.GetComponent<Image>().fillAmount = ((float)(MatchMonitor.instance.GetSecondTeamKills()) /
((float)MatchMonitor.instance.GetStandardKillCount()));
        }
        public void TogglePauseMenu(){
            pauseMenu.SetActive(!pauseMenu.activeSelf);
            crosshair.SetActive(!crosshair.activeSelf);
            PauseMenu.IsOn = pauseMenu.activeSelf;
        }
        public void SetPlayer(Player _player){
            player = _player;
            controller = player.GetComponent<PlayerController>();
```

```
        weaponManager = player.GetComponent<WeaponManager>();
    }
    void SetFuelAmount(float _amount){
        thrusterFuelFill.localScale = new Vector3(_amount, 1f, 1f);
    }
    void SetMatchTimer(){
        int time = Convert.ToInt32(MatchMonitor.instance.GetMatchTime());
        matchTimer.text = (time / 60).ToString() + ":" + (time % 60).ToString("00");
    }
    void SetHealthAmount(float _amount){
        healthText.text = ((int)_amount).ToString();
    }
    void SetAmmoAmount(int _amount, int _maxAmount, int _throwablesLeft){
        ammoText.text = _amount.ToString() + " / " + _maxAmount.ToString();
        throwableText.text = _throwablesLeft.ToString();
    }
    public void ToggleVoiceCommandMenu(){
        if (isCommandOn == true)
        {
            Button attachedButton = radialMenuController.elements[radialMenuController.index].button;
            if (attachedButton != null)
                attachedButton.onClick.Invoke();
            else Debug.Log("Attached button is null");
        }
        isCommandOn = !commandMenu.activeSelf;
        Cursor.lockState = CursorLockMode.Confined;
        Cursor.visible = !Cursor.visible;
        commandMenu.SetActive(!commandMenu.activeSelf);
        crosshair.SetActive(!crosshair.activeSelf);
    }
    public void OnKill(string _killer, string _killerTeam, string _source, string _victim,
        string _victimTeam, bool isObserver) {
        if (PauseMenu.IsOn)
            TogglePauseMenu();
    }
    private void SetCrosshairParts(){
        crossTop = crosshairContainer.Find("Top");
        crossBottom = crosshairContainer.Find("Bottom");
        crossLeft = crosshairContainer.Find("Left");
        crossRight = crosshairContainer.Find("Right");
        crosshairDistance = crossTop.localPosition.y;
    }
    private void UpdateCrosshairPosition(){
        float fireModifier = playerShoot.CalculateCurrentSpreadAngle();
        if (fireModifier >= 0.0005f){
            fireModifier = Mathf.Lerp(previousOffset, fireModifier * 15, 0.1f);
            crossTop.localPosition = new Vector3(0f, crosshairDistance + fireModifier, 0f);
            crossBottom.localPosition = new Vector3(0f, -crosshairDistance - fireModifier, 0f);
            crossLeft.localPosition = new Vector3(-crosshairDistance - fireModifier, 0f, 0f);
            crossRight.localPosition = new Vector3(crosshairDistance + fireModifier, 0f, 0f);
            previousOffset = fireModifier;
        }
        else
```

```
            {
                if (previousOffset > 0f){
                    previousOffset = Mathf.Lerp(previousOffset, 0f, 0.1f);
                    crossTop.localPosition = new Vector3(0f, crosshairDistance + previousOffset, 0f);
                    crossBottom.localPosition = new Vector3(0f, -crosshairDistance - previousOffset, 0f);
                    crossLeft.localPosition = new Vector3(-crosshairDistance - previousOffset, 0f, 0f);
                    crossRight.localPosition = new Vector3(crosshairDistance + previousOffset, 0f, 0f);
                }
            }
        }
        public void AttackCall(){
            controller.voiceController.PlaySound(VoiceCallType.Attack);
        }
        public void AffirmativeCall(){
            controller.voiceController.PlaySound(VoiceCallType.Affirmative);
        }
        public void AmmoCall(){
            controller.voiceController.PlaySound(VoiceCallType.Ammo);
        }
        public void NegativeCall(){
            controller.voiceController.PlaySound(VoiceCallType.Negative);
        }
        public void HelpCall(){
            controller.voiceController.PlaySound(VoiceCallType.Help);
        }
        public void MedicCall(){
            controller.voiceController.PlaySound(VoiceCallType.Medic);
        }
        public void ThankCall(){
            controller.voiceController.PlaySound(VoiceCallType.Thank);
        }
        public void TauntCall(){
            controller.voiceController.PlaySound(VoiceCallType.Taunt);
        }
    }
```