МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-08

Дипломний проект

здобувача освіти денної форми навчання
РП.08.12.000.ДП

НІКОЛАЙЧУКА
МАКСИМА ГЕНАДІЙОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**

Освітньо-професійна програма: «**Розробка програмного забезпечення»**

Група: **4РП-08**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка мобільної 2D-гри "Bomber" з можливістю кооперативу у ICP Unity

Проектний матеріал складається з пояснювальної записки на 70 сторінках та графічного (презентаційного) матеріалу на 10 аркушах (слайдах)

Дипломник _____ (Ніколайчука М.Г.)

Керівник _____ (Шувалова І.О.)

Консультанти:

з економічного розділу _____ (Канський М.Ю.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділенням _____ (Краснокутська К.Г.)

Захист «26» 06 2025 р. Протокол ЕК № 2

Оцінка ЕК 4/85

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 Інженерія програмного забезпечення*

Освітньо-професійна програма *Розробка програмного забезпечення*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"12" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Ніколайчука Максиму Георгійовичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка мобільної 2D-гри "Bomber" з можливістю кооперативу у ICP Unity*

затверджена наказом по коледжу від "14" листопада 2024 р. № 246

2. Термін здачі закінченого проекту 16.06.2025

3. Вихідні данні до проекту *Забезпечити повний цикл розробки гри, включаючи проектування ігрового процесу, програмування логіки гри, створення графічного інтерфейсу та анімацій. Забезпечити тестування гри на різних пристроях для виявлення та виправлення помилок*

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

1. Аналіз та характеристика мобільних ігор. 2. Аналіз існуючих розробок. 3. Аналіз середовищ розробки та обґрунтування вибору технології розробки проекту. 4. Огляд інструментарію Unity. 5. Мультиплеєр PUN2. 6. Проектування та розробка гри. 7. Тестування гри 8. Охорона праці. 9. Висновки. 10. Список використаних джерел інформації.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Інтегрована архітектура гри; ER-діаграма структури збереження ігрових даних (гравець, кімната, стан). Діаграма станів ігрового персонажа; Діаграма потоків для підключення до мультиплеєрної гри; Діаграма взаємодії між клієнтами та сервером Photon (PUN2); Блок-схема логіки розміщення та вибуху бомби; Схема анімаційних станів у Animator Controller (Unity); Приклади візуального інтерфейсу гри; Файлова структура проєкту у Unity

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Шувалова І.О.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 05.04.2025

Керівник *Шувалова І.О.* _____
(підпис)

Завдання прийняв до виконання *Ніколайчук М.Г.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Підбір та вивчення літератури з теми дослідження*	*5.04.2025*	Виконав
2	*Аналіз ринку мобільних ігор та тенденцій розвитку*	*10.04.2025*	Виконав
3	*Аналіз існуючих ігор-аналогів*	*14.04.2025*	Виконав
4	*Визначення основних функціональних вимог до гри*	*21.04.2025*	Виконав
5	*Аналіз середовищ розробки*	*29.04.2025*	Виконав
6	*Вибір технологій для реалізації мультиплеєра*	*4.05.2025*	Виконав
7	*Проектування гри*	*7.05.2025*	Виконав
8	*Розробка графічного інтерфейсу користувача*	*10.05.2025*	Виконав
9	*Програмування ігрової логіки та механік.*	*20.05.2025*	Виконав
10	*Налаштування мережевої взаємодії*	*31.05.2025*	Виконав
11	*Тестування гри на різних пристроях.*	*5.06.2025*	Виконав
12	*Підготовка до малого захисту*	*10.06.2025*	Виконав
13	*Розробка питань з охорони праці*	*11.06.2025*	Виконав
14	*Підготовка роботи до захисту та тестування гри*	*14.06.2025*	Виконав
15	*Оформлення слайдів мультимедійної презентації*	*15.06.2025*	Виконав

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 08. 12 000. 00 ДП	Дипломний проект		
А4			РП 08. 12 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 08. 12 000. 00 ДП		
Розробив		Ніколайчук М.Г.		16.06	Розробка мобільної 2D-гри "Bomber" з можливістю кооперативу у ICP Unity	Літ.	Аркуш 4 / Аркушів 83
Перевірив		Шувалова І.О.		16.06.25		Н Д П	
Н. контр.		Петрашова В.І.		16.06.25	ВСП "ОТФК ОНТУ" гр.4РП-08		
Затверд.		Кривченко Ю.В.		16.06.25			

ЗМІСТ

ВСТУП

У сучасному світі мобільних технологій, де ігри стали невід'ємною частиною розваг та соціальної взаємодії, особливо популярними стають проекти, які поєднують простоту ігрових механік, захоплюючий геймплей та можливість змагатися з друзями. Розробка мобільних ігор є вкрай актуальною сферою, що вимагає ґрунтовних знань не лише у програмуванні, але й у дизайні, анімації, геймдизайні та тестуванні.

Цей дипломний проект присвячений створенню мобільної 2D-гри для платформи Android із застосуванням рушія Unity. Unity є потужним інструментом для розробки ігор, що надає розробникам широкий спектр можливостей для втілення творчих ідей. Використання Unity дозволяє створювати кросплатформені рішення з високою продуктивністю та якісною графікою. Крім того, рушій підтримує інтеграцію різноманітних компонентів, що значно спрощує процес розробки та тестування.

Гра, розроблена в межах цього проекту, належить до жанру аркад. У ній гравці встановлюють бомби, прагнучи знищити суперників та уникнути їхніх пасток, що додає елемент тактики та стратегії. Особливість цієї гри полягає в простоті механіки, що робить її доступною для широкого кола користувачів.

У роботі будуть висвітлені ключові етапи створення мобільної гри: розробка концепції та проєктування ігрового процесу, програмування основної логіки, створення анімацій, а також ретельне тестування та оптимізація готового продукту. Окрему увагу приділено реалізації багатокористувацького режиму за допомогою Photon PUN, що передбачає організацію серверної частини, налаштування підключення клієнтів, синхронізацію дій гравців і забезпечення стабільної роботи гри в мережевому середовищі.

Метою цього проекту є не лише створення функціональної та захопливої мобільної 2D-гри, але й здобуття практичного досвіду в розробці ігор на платформі Unity, що сприятиме професійному зростанню та подальшому розвитку у сфері програмної інженерії.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз та загальна характеристика мобільних ігор

Мобільні ігри нині є невід'ємною складовою цифрового простору, значно розширивши рамки індустрії розваг. Швидкий розвиток мобільних технологій та масове поширення смартфонів зробили ці ігри доступними для широкого кола користувачів. Вони створюються спеціально для мобільних пристроїв — смартфонів та планшетів — і забезпечують інтерактивний ігровий досвід, дозволяючи геймерам взаємодіяти з віртуальним світом через сенсорні екрани, акселерометри та інші можливості пристроїв.

Мобільні ігри охоплюють численні жанри — від простих казуальних розваг до складних стратегічних і рольових проєктів. При їх розробці необхідно враховувати обмежені ресурси мобільних пристроїв, такі як обчислювальна потужність, обсяг пам'яті та розмір екрану. Ключовими характеристиками таких ігор є короткі ігрові сесії, інтуїтивне керування та висока інтерактивність, що робить їх ідеальними для невеликої перерви в дорозі чи під час очікування.

Іноді мобільні ігри створюються як окремі проєкти, а інколи їх переносять із ПК або консолей. Проте останнім часом дедалі популярніші мобільні версії стають настільки відомими, що потім адаптуються під інші платформи.

Важливим чинником успіху мобільних ігор є їхня соціальна складова. Багато проєктів підтримують багатокористувацькі режими, що дає змогу гравцям змагатися або співпрацювати онлайн. Інтеграція з соціальними мережами та платформами, як-от Facebook або Google Play Games, дозволяє ділитися досягненнями, запрошувати друзів та брати участь у спільнотах.

Фінансова сторона мобільних ігор також є значущою для розвитку ринку: вони приносять істотні доходи через прямі продажі, рекламу та внутрішньоігрові покупки. Одна з найпоширеніших моделей монетизації — фріміум, коли базова версія гри безкоштовна, а додаткові функції чи контент відкриваються за плату.

Візуальне оформлення мобільних ігор може різнитися від простих піксельних графік до високоякісних 3D-зображень. Незважаючи на більші обмеження

мобільних пристроїв у порівнянні з ПК чи консолями, сучасні технології дозволяють створювати досить складні й візуально привабливі проєкти.

За класифікацією, мобільні ігри можна розділити за жанрами (казуальні, екшн, головоломки, рольові, стратегії, симулятори тощо), за режимами (одиночні або багатокористувацькі) та за графічним оформленням (2D чи 3D). Казуальні ігри зазвичай характеризуються простими механіками й короткими сесіями, що дозволяє грати кілька хвилин під час перерв. Екшн-ігри вимагають швидкої реакції й точного керування, тоді як головоломки зосереджуються на вирішенні задач — від доволі простих до вельми складних.

Розробка мобільних ігор — це комплексний процес, що включає проєктування ігрового процесу, програмування, створення графіки та анімацій, тестування й оптимізацію. Використання сучасних інструментів і технологій дозволяє створювати високопродуктивні ігри з якісною візуальною складовою. Це сприяє ефективному розробленню кросплатформених проєктів, які відповідають актуальним потребам користувачів та вимогам ринку.

Проаналізувавши різноманітні жанри та тенденції розвитку мобільних ігор, було вирішено створити двовимірну гру жанру аркадного екшену. Ігри цього типу характеризуються зрозумілою й простою механікою, що приваблює як казуальних гравців, так і тих, хто шукає більш інтенсивний ігровий досвід. Обрання 2D-графіки обумовлено її меншою вимогливістю до апаратних ресурсів та простотою реалізації, що дає змогу сконцентруватися на вдосконаленні ігрового процесу та забезпеченні високоякісного користувацького досвіду.

Отже, концепція обраної гри поєднує переваги аркадного екшену з простотою 2D-графіки, забезпечуючи захопливий геймплей і широке охоплення цільової аудиторії.

1.2 Аналіз існуючих розробок

Ігри, натхненні класичним «Bomberman», де гравці стратегічно ставлять бомби, щоб ліквідувати суперників і руйнувати перешкоди, і навіть за десятиліття після свого виходу в 1983 році продовжують приваблювати ґеймерів. Постійний інтерес до цього жанру пояснюється тим, що він поєднує прості, але надзвичайно

захопливі механіки, швидкий і динамічний геймплей, який вимагає блискавичної реакції та обдуманих тактичних рішень, а також можливість змагатися з іншими гравцями, що надає грі соціального відтінку.

Класичні представники жанру 8- та 16-бітної епохи:

Bomberman (1983). Розроблена компанією Hudson Soft, ця гра стала першою у своєму роді та заклала основні принципи ігрового процесу, які залишаються актуальними й донині. Користувач керує персонажем, який встановлює бомби з таймером: через певний проміжок часу вони вибухають, руйнуючи навколишні блоки і ворогів у радіусі дії. По дорозі гравець збирає бонуси, які можуть підвищити потужність вибуху, збільшити швидкість руху персонажа або додати можливість встановити більше бомб одночасно. Окрім одиночного режиму, до чотирьох людей могли грати на одному екрані, що зробило «Bomberman» улюбленою розвагою на вечірках і дружніх зібраннях.

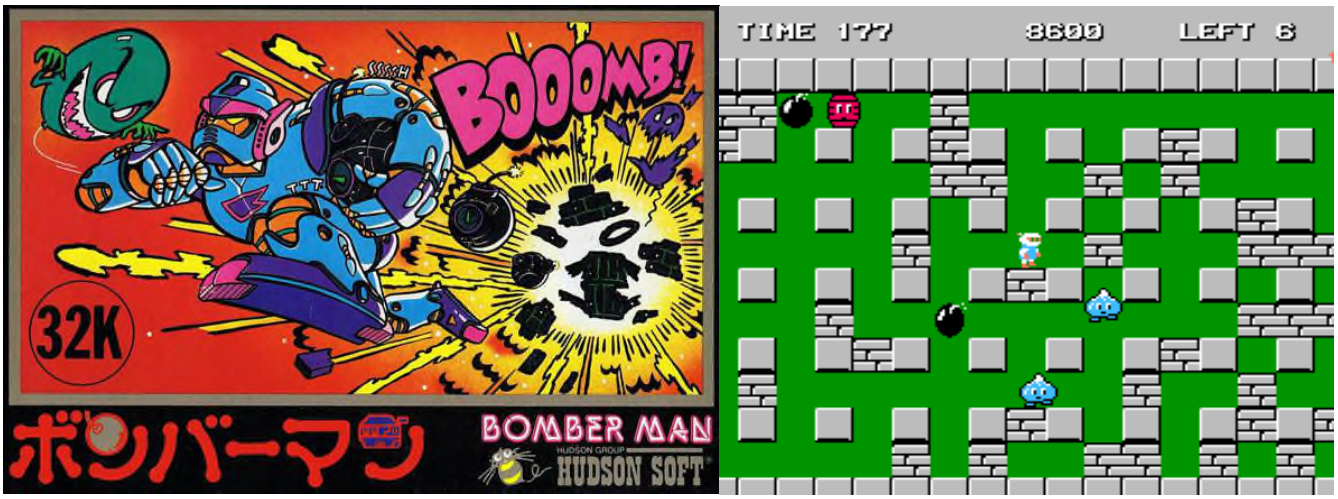


Рисунок 1.1 Bomberman 1983

Super Bomberman (1993). Ця серія для Super Nintendo Entertainment System (SNES) стала справжнім хітом завдяки яскравій графіці, привабливим персонажам і різноманітним ігровим режимам. Кожен бомбермен володів власними унікальними здібностями, що додавало стратегії та різноманітності. Нові рівні з різноманітними перешкодами, пастками та бонусами робили кожну гру неповторною й захопливою. Крім класичних матчів, у Super Bomberman з’явився

«Story Mode» зі сюжетом і кат-сценами, що значно розширило всесвіт і зробило гру ще цікавішою для фанатів.

Рисунок 1.2 Super Bomberman 1993

Сучасні інтерпретації та аналоги:

Bomberman Ultra (2009). Видана для PlayStation 3, ця гра стала однією з найвідоміших сучасних версій Bomberman. Вона зберігає класику геймплею, доповнюючи її більш сучасною візуальною складовою та розширеним мультиплеєром, у якому одночасно можуть брати участь до восьми гравців онлайн. Окрім цього, у Bomberman Ultra реалізували редактор персонажів, що дозволяє користувачам створювати власні унікальні образи для своїх бомберменів.

Рисунок 1.3 Bomberman Ultra 2009

Bombergrounds: Reborn (2020). Ця багато платформна гра від студії Gigantic Duck Games поєднує знайомий Bomberman-геймплей із механіками жанру Battle Royale та елементами колекційної карткової гри. Гравці змагаються на арені, що поступово звужується, збираючи підсилювачі, прокачуючи своїх героїв і

використовуючи бомби для знищення опонентів. У грі є кілька режимів: соло, дуети та командні бої, а також система прогресії, котра дає змогу відкривати нових персонажів і скіни.

Рисунок 1.4 Bombergrounds: Reborn 2020

1.3 Аналіз середовищ розробки та обґрунтування вибору технології розробки проекту

Вибір ігрового рушія — одне з ключових стратегічних рішень на старті розробки. Він задає не лише технічні можливості та обмеження проєкту, а й впливає на весь процес створення гри: від етапу дизайну та прототипування до тестування й релізу. Кожен рушій має власні переваги та недоліки, які слід враховувати відповідно до конкретних цілей і вимог проєкту.

1.3.1 Unreal Engine

Unreal Engine, розроблений і підтримуваний компанією Epic Games, вперше з’явився у 1998 році разом із виходом шутера від першої особи Unreal. Хоча спочатку рушій призначався переважно для створення шутерів, його гнучкість швидко виявилася, і в наступних версіях його застосовували в різних жанрах — від стелс-ігор та файтів до масових багатокористувацьких онлайн-ролівок.

Рисунок 1.5 Інтерфейс Unreal Engine

Unreal Engine, розроблений Epic Games, є потужною платформою для створення ігор, яка дозволяє розробникам звертатися до найрізноманітніших цільових пристроїв, починаючи від персональних комп'ютерів та консолей і закінчуючи мобільними гаджетами й пристроями віртуальної реальності. Щоб забезпечити високу продуктивність і якість проєктів, цей рушій підтримує роботу під операційними системами Windows (версії 7, 8 і 10, 64-біт), Linux (Ubuntu 15.04 і новіші) та macOS (10.13 і вище).

Для комфортної роботи з редактором рекомендується комп'ютер із процесором рівня Intel Quad-core або AMD із тактовою частотою від 2,5 ГГц, щонайменше 8 ГБ оперативної пам'яті (у середовищі Linux краще мати 16 ГБ) і відеокартою не нижче AMD Radeon 6870 HD чи NVIDIA GeForce GTX 470. Саме такий мінімальний набір апаратних ресурсів дає змогу розробнику безперебійно працювати над проєктом, хоча для створення компонентів з найбільшими вимогами до продуктивності (наприклад, складні шейдери чи великі відкриті світи) часто використовують ще більш потужні конфігурації.

Що стосується графічних можливостей, Unreal Engine дозволяє досягти фотореалістичності завдяки просунутим інструментам налаштування освітлення, тіней і системи рендерингу. Вбудовані механізми розподіленого освітлення дають змогу проєктувати як статичні, так і динамічні світлові ефекти з точним

відтворенням рефлексів, розсіяного світла та глобального освітлення. Завдяки цьому художники можуть реалізувати у своїх проєктах найвищі стандарти візуальної якості, які раніше були доступні хіба що на спеціалізованих графічних консолях.

Важливою особливістю Unreal Engine є система Blueprints — візуальний скриптовий інструмент, який дозволяє створювати складну ігрову логіку, не занурюючись безпосередньо у написання коду на C++. За допомогою Blueprints розробники можуть швидко прототипувати ігрові механіки, зв'язувати їх із подіями в редакторі й одразу бачити результати своїх змін у реальному часі. Це особливо корисно, коли потрібно швидко перевірити нову ідею або налаштувати поведінку персонажів без очікування компіляції. Водночас, якщо проєкт вимагає високооптимізованих рішень або роботи з низькорівневими алгоритмами, Unreal Engine надає повний доступ до вихідного коду рушія на C++. Завдяки цьому досвідчені програмісти мають змогу переписувати будь-які підсистеми рушія під свої потреби: від модифікації фізичного двигуна й оптимізації рендерингових підсистем до створення власних бібліотек і плагінів. Такий підхід гарантує максимальну гнучкість і контроль над усіма технічними аспектами проєкту.

У роботі із створенням ігрових ландшафтів Unreal Engine пропонує низку інструментів, що дозволяють художникам і дизайнерам швидко формувати місцевість і розміщувати рослинність. За допомогою вбудованих редакторів ландшафтів можна штучно формувати рельєф, накладати текстури, змінювати висоту поверхні й розставляти сотні чи тисячі дерев, кущів та трави в межах однієї сцени. При цьому рушій автоматично оптимізує віддалені елементи за допомогою LOD (Level of Detail), що дозволяє зберегти високу продуктивність навіть у великих відкритих світах. Для створення ефектів частинок розробники використовують редактор Cascade, який надає можливість конструювати візуальні ефекти будь-якої складності: від димових шлейфів і іскр до складних анімаційних композицій на основі фізики частинок.

Особливої уваги в Unreal Engine заслуговує редактор матеріалів із підтримкою PBR (Physically Based Rendering). Це означає, що художники можуть

створювати поверхні, які точно реагують на світлові умови, коректно відображають відблиски й розсіювання, враховують мікроперешкоди й властивості числа відбиттів. Редактор пропонує вузловий (node-based) інтерфейс, де кожна властивість матеріалу — колір, текстура, ступінь блиску, нормалі та інші — задається окремим елементом, а сам матеріал формується шляхом поєднання цих елементів у єдину мережу. Це дає змогу досягнути надзвичайної гнучкості: від реалістичних металевих поверхонь до картинних, стилізованих ефектів.

Інструмент Sequencer у Unreal Engine призначений для створення анімаційних послідовностей і кінематографічних сцен. За його допомогою можна налаштовувати рух камери, змінювати позиції персонажів і об’єктів, керувати візуальними та звуковими ефектами у певний момент часу. Sequencer підтримує ключові кадри для анімацій тіл і скелетних анімацій персонажів, що дає змогу створювати кінематографічну режисуру без додаткових зовнішніх програм. За допомогою цього інструменту режисери можуть відразу бачити результат і навіть редагувати анімацію в процесі відтворення.

У контексті підтримки віртуальної реальності Unreal Engine вже із самого початку надає вбудовані можливості для роботи з VR-пристроями. Рушій оптимізований для роботи з головними шоломами, такими як Oculus Rift, HTC Vive чи Valve Index, а також різними мобільними VR-пристроями. Він забезпечує коректне рендерення зображення з низькими затримками й автоматично підлаштовує параметри рендерингу під вимоги VR, що мінімізує ризик виникнення дискомфорту в користувачів.

Для організації поведінки штучного інтелекту в ігрових персонажів Unreal Engine пропонує розширену систему ШІ. Зокрема, рушій підтримує використання Behavior Tree — ієрархічної структури для побудови логіки прийняття рішень. Behavior Tree у поєднанні з Blackboard (контейнером даних для збереження змінних середовища та станів персонажа) дає змогу створювати складні поведінкові моделі: від базового патрулювання стежок і реакції на події до складних сценаріїв взаємодії з кількома персонажами та навколишнім середовищем. Крім того, Unreal Engine включає вбудовані навігаційні сітки

(NavMesh), які автоматично генеруються для рівнів і дозволяють персонажам обирати найкоротший шлях у просторі з урахуванням перешкод.

Що стосується ліцензування, то з 2015 року Unreal Engine доступний для всіх розробників безкоштовно. Компанія Epic Games стягує роялті лише у випадку, якщо створений проєкт приносить понад 3000 доларів США за один календарний квартал, причому ставка складає 5 % від суми доходу. Завдяки такому підходу дрібні інді-команди чи студенти можуть починати роботу без початкових витрат, а Epic Games отримує винагороду лише тоді, коли проєкт починає генерувати суттєві прибутки.

Для підтримки розробників існує офіційна документація, де представлено вичерпні покрокові інструкції з налаштування середовища, роботи з основними модулями рушія, а також численні туторіали від працівників Epic Games і спільноти. Крім того, Unreal Engine Marketplace пропонує широкий асортимент готових ассетів: моделі персонажів, текстури, звукові ефекти, плагіни для додаткових функцій, які розробники можуть придбати або завантажити безкоштовно. Це значно пришвидшує розробку, оскільки дає змогу не створювати всі компоненти з нуля, а використати готові рішення й адаптувати їх під власні потреби.

Таким чином, Unreal Engine є універсальним рушієм, що поєднує високопродуктивні інструменти для створення реалістичної графіки, зручні засоби візуального скриптингу, гнучкість C++-програмування, а також готові рішення для анімації, ШІ та VR. Однак щоб повною мірою скористатися усіма його можливостями, розробникам потрібен відповідний рівень знань і досвіду, оскільки рушій пропонує широкий спектр налаштувань, які вимагають розуміння принципів роботи графічного рендерингу, оптимізації продуктивності та архітектури програмного коду.

1.3.2 Godot Engine

Godot Engine — це сучасний рушій для розробки ігор, який відрізняється своєю простотою та водночас багатофункціональністю. Його основна архітектура побудована на основі сцен і вузлів (nodes), де кожен елемент гри—від персонажа

до фону чи кнопки інтерфейсу—представлений як окремий вузол, що у свою чергу може містити вкладені сцени. Такий підхід дозволяє розробнику гнучко комбінувати готові об’єкти, наслідувати їхню поведінку та швидко змінювати структуру проєкту без необхідності писати багато «клейового» коду. Зміни, внесені у батьківську сцену, автоматично поширюються на всі її копії, що значно прискорює роботу над великими проєктами.

Для написання коду у Godot Engine найчастіше використовують власну мову GDScript, яка за синтаксисом дуже нагадує Python. GDScript створена з огляду на потреби ігрової розробки, тому має мінімалістичний, але водночас функціонально повний набір інструментів для роботи з графікою, фізикою, анімацією та мережею. Система сигналів (Signals) у поєднанні з методом виклику функцій (callbacks) робить взаємодію між різними вузлами живою та зручною: наприклад, кнопка може відправити сигнал про натиск, і будь-який об’єкт, підписаний на цей сигнал, одразу виконає певну дію. Крім того, Godot підтримує написання скриптів на C# (через Mono) та навіть C++ за допомогою GDNative — це дає можливість використати вже знайомі розробнику технології або вбудувати високооптимізовані фрагменти коду.

Godot Engine є універсальною платформою для розробки ігор, яка працює на Windows, Linux, macOS, а також дозволяє експортувати проєкти у вигляді застосунків для Android, iOS та навіть веб-гри (HTML5). Завдяки мінімальним апаратним вимогам рушій може без проблем запускатися як на сучасних комп’ютерах середнього класу, так і на старіших чи малопотужних пристроях.

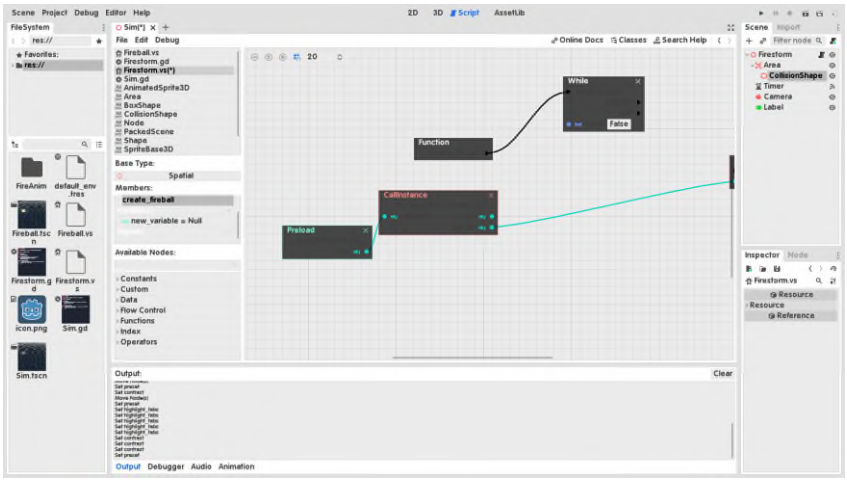


Рисунок 1.6 Інтерфейс Godot Engine

Однією з ключових особливостей Godot Engine є підтримка кількох мов програмування, що значно розширює можливості розробників. GDScript, власна скриптова мова рушія, наслідує стиль та синтаксис Python, але оптимізована під потреби створення ігрової логіки: вона дозволяє в декілька рядків коду реалізувати рух персонажа, перевірку колізій або взаємодію з інтерфейсом. Завдяки цьому новачки можуть дуже швидко впоратися з першим ігровим прототипом, а досвідченіші розробники — написати чистий і підтримуваний код без зайвих конструкцій. Якщо ж у команді є фахівці з C#, Godot пропонує повну інтеграцію з Mono, що дозволяє використовувати знайомі бібліотеки .NET і готові пакети NuGet. Крім того, за допомогою GDNative можна впроваджувати модулі власним C++, що часто буває необхідним у випадку високих вимог до продуктивності (наприклад, для реалізації складного штучного інтелекту чи обробки фізичних симуляцій).

У роботі з 2D-графікою Godot відзначається особливою зручністю. Вбудований редактор анімацій дозволяє не лише анімувати спрайти, а й синхронізувати рухи різних об’єктів, створювати послідовності подій (наприклад, поступове відкривання воріт або похиляння камери), а також задавати звукові ефекти під час окремих кадрів. Фізичний рушій для 2D-об’єктів обробляє колізії за кількома алгоритмами, що дає змогу задавати геометрію зіткнень за допомогою різноманітних колайдерів (прямокутники, кола, полігони). Система частинок у 2D-режимі дозволяє легко створювати дрібні візуальні ефекти — від диму і вогню до блискучих спалахів або розсипу листя. Завдяки цим інструментам можна швидко зібрати прототип 2D-платформера, головоломки чи аркади з динамічними ефектами без потреби використовувати сторонні бібліотеки.

Незважаючи на сильний акцент на 2D, Godot також пропонує розвинуті засоби для 3D-розробки. Рушій підтримує Physically Based Rendering (PBR), що дозволяє створювати реалістичні матеріали з точним відтворенням властивостей поверхонь — блискучих металів чи матових тканин. Вбудований рендерер другого покоління забезпечує апаратне прискорення, а також можливість підключати власні шейдери на мові, схожій на GLSL. Для роботи з навколишнім

освітленням Godot використовує як статичне (Lightmap), так і динамічне освітлення (OmniLight, SpotLight, DirectionalLight). Сцени можуть містити компоненти для генерації навігаційних сіток (NavMesh), що потрібні для коректного переміщення персонажів із штучним інтелектом. Хоча ще не всі 3D-функції Godot так само детально опрацьовані, як у вузько спеціалізованих 3D-рішеннях, цього цілком достатньо, щоб створити середній за складністю шутер або 3D-платформер.

У самій основі архітектури Godot лежить система сцен і вузлів (nodes), де кожен вузол відповідає за конкретну функцію — графіку, звук, фізику, обробку введення чи логіку. Сцени можуть вкладатися одна в одну:, наприклад, ви створюєте сцену «Головний герой» із набором анімацій, колайдером та логікою руху, а потім вкладаєте її в сцену «Рівень», яка вже містить тайли, ворогів і елементи інтерфейсу. Якщо у сцені «Головний герой» потрібно замінити анімацію стрибка, достатньо відредагувати батьківську сцену, і зміни автоматично застосуються до всіх сцен, де герой використовується. Такий підхід значно прискорює роботу над великими проєктами і знижує ймовірність помилок, адже ви оперуєте не «сторінками» коду, а окремими об’єктами із чітко визначеними функціями.

Для тих, хто не хоче або не вміє одразу писати код, Godot пропонує візуальний скриптовий редактор VisualScript. За допомогою нього можна у графічному інтерфейсі з’єднувати блоки логіки — події, умови, дії, цикли — і таким чином реалізувати поведінку об’єктів без жодного рядка текстового коду. VisualScript відмінно підходить для розробників, які звикли до візуальних середовищ або хочуть швидко зібрати прототип, однак у складних проєктах усе ж частіше використовують GDScript чи C#, адже текстовий код легше тестувати та відлагоджувати.

Ще однією перевагою рушія є його виняткова гнучкість: ви можете комбінувати GDScript і C# у межах одного проєкту, одночасно використовуючи переваги легкості GDScript для більшості ігрової логіки та високу продуктивність C# або C++ там, де потрібна максимальна швидкодія. Якщо потрібно розширити

рушій за межі його базових можливостей, можна скористатися GDNative і написати власні модулі на C++ або на будь-якій іншій мові, сумісній із C API. Таким чином, Godot підходить як для створення невеликих інді-ігор, так і для великих комерційних проектів.

З погляду ліцензування Godot Engine поширюється під умовами ліцензії MIT. Це означає, що рушій можна використовувати абсолютно безкоштовно, не сплачуючи жодних роялті чи ліцензійних відрахувань незалежно від того, чи ви робите навчальний проєкт, інді-гру, чи користуєтеся рушієм у виготовленні комерційної AAA-гри. Відкрите ліцензування також дає повну свободу змінювати вихідний код рушія: якщо вам потрібно додати специфічну функцію або виправити недолік, ви можете редагувати внутрішні файли, а потім спільнота зрадіє таким оновленням, адже ви можете внести свої зміни назад у загальний репозитарій.

Godot має активну та дружню спільноту, яка щодня зростає. На офіційному сайті рушія зібрана велика кількість документації: офіційний посібник крок за кроком, довідник по API, а також численні навчальні статті з прикладами та відеоуроки від волонтерів. У разі складнощів розробники можуть звернутися на офіційний форум, у Discord-чат або на Stack Overflow, де завжди знайдуться ті, хто готовий допомогти розібратися з тією чи іншою проблемою. Крім того, Godot має власну Asset Library — онлайн-каталог, де розробники розміщують безкоштовні моделі, звукові ефекти, скрипти чи плагіни, які можна швидко інтегрувати у свій проєкт. Це значно прискорює розробку, адже, наприклад, замість того, щоб писати власну систему керування камерою або складний шейдер для води, можна знайти готовий і налаштувати під свої потреби.

Загалом, Godot Engine уособлює ідею простоти без втрати функціональності: він дозволяє почати роботу практично без підготовки, але водночас надає всі інструменти, необхідні для професійної розробки як 2D-, так і 3D-проектів. Завдяки гнучкому підходу до скриптування, чудовій системі сцен і відкритому ліцензуванню цей рушій стає все популярнішим вибором серед інді-

розробників, студентів і навіть професійних студій, які цінують свободу й швидкість реалізації ідей.

1.3.3 Unity

Unity — провідний ігровий рушій, який активно використовується для створення найрізноманітніших проєктів — від простих мобільних додатків до масштабних AAA-ігор із високою якістю графіки та складною логікою. Його універсальність, кросплатформеність, розвинена екосистема плагінів, а також активна спільнота розробників роблять Unity надзвичайно привабливим вибором для професіоналів і початківців. Саме завдяки цим перевагам Unity було обрано як основу для реалізації дипломного проєкту, що передбачав створення ефективного, адаптивного застосунку для вивчення іноземних мов.

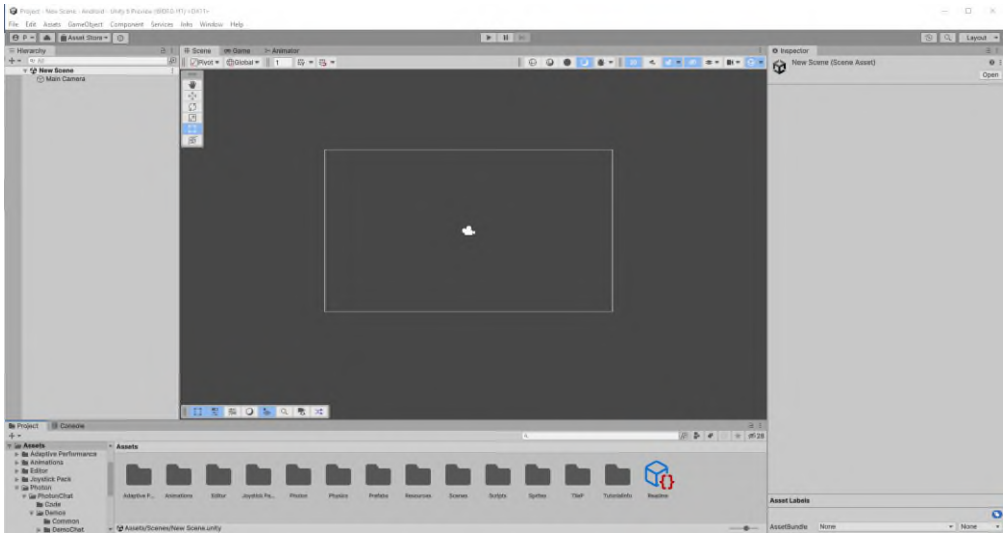


Рисунок 1.7 Інтерфейс Unity

Unity – це багатофункціональний рушій, який дає змогу створювати ігри для практично будь-якої платформи: від настільних комп’ютерів під керуванням Windows, macOS та Linux до мобільних пристроїв на Android та iOS, різноманітних консолей (PlayStation, Xbox, Switch), а також веб-ігор і проектів із підтримкою віртуальної реальності. Він спроєктований так, щоб працювати навіть на помірних конфігураціях, тому для більшості проєктів достатньо комп’ютера зі середнім процесором і кількома гігабайтами оперативної пам’яті, а отже, розпочати розробку можна без інвестицій у дорогі робочі станції. Переважно

ігрову логіку пишуть мовою C#, котра поєднує в собі строгість типів і простоту синтаксису, що спрощує навчання новачкам і водночас не обмежує досвідчених програмістів.

Щодо інструментального забезпечення, Unity зроблений так, щоб уже з перших кроків автор міг швидко зосередитися на ігровому процесі, а не на налаштуванні середовища. Рушій містить вбудований тайлмап-редактор, який значно спрощує створення двовимірних рівнів: достатньо намалювати тайли у зовнішньому графічному редакторі, а потім розставити їх у редакторі сцени, легко слідкуючи за вирівнюванням міри та розмірами спрайтів. Система 2D-анімацій дозволяє налаштовувати ключові кадри й переходи між ними, щоб персонажі рухалися плавно й природно, а фізичний рушій спеціально оптимізований для 2D: він сам обробляє зіткнення, відскоки та інші явища, необхідні для коректної роботи платформерів чи аркадних ігор. Ще однією зручністю є можливість створювати частинки безпосередньо в редакторі: задати швидкість, напрямок, розкид і час життя, після чого легко отримати ефекти вибухів, диму чи блиску, які додають грі візуальної виразності.

Unity дає змогу заощадити час і ресурси за рахунок великого магазину готових активів. Якщо треба додати персонажа, звуковий ефект або плагін, часто достатньо записатися в Asset Store, вибрати відповідний ресурс – а він може бути як безкоштовним, так і платним – і відразу інтегрувати в свій проєкт. Це особливо корисно для студентських чи інді-команд, які не мають власних художників чи звукоінженерів. Щоб програмісти не писали увесь код вручну, Unity підтримує візуальні графи на базі плагіна Bolt (згодом перейменованого на Visual Scripting), де замість текстових скриптів можна поєднувати блоки дій, умов і викликів. Завдяки цьому навіть люди без досвіду програмування можуть реалізувати прості механіки, а досвідчені розробники можуть швидко прототипувати вихідні версії ігор, перш ніж переходити до оптимізованого коду.

Важливою складовою довкола Unity є величезна спільнота розробників, які активно публікують статті, відеоуроки, готові шаблони та відповідають на запитання на форумах чи в соціальних мережах. Офіційна документація

регулярно оновлюється, а в інтегрованому середовищі розробки є підказки та напряму показані приклади використання API Unity, що значно спрощує вивчення специфіки рушія. Крім того, Unity пропонує власні сервіси – наприклад, Collaborate для спільної роботи над репозиторієм у хмарі, Cloud Build для автоматизації збірки проєкту на різні платформи й Analytics, який дозволяє відстежувати поведінку користувачів у готовій грі й підказувати, як можна покращити баланс або інтерфейс.

Для тих, хто планує заробляти на грі, Unity надає набори інструментів для монетизації: вбудовані SDK для показу реклами, можливість додавати внутрішньоігрові покупки та впроваджувати модель підписок. Це означає, що розробник може заробляти як на разовій покупці гри, так і на її подальшому розвитку – продаючи додатковий контент.

Стосовно ліцензування, Unity має безкоштовний персональний план, який дозволяє розробляти ігри без будь-яких виплат, поки дохід проєкту або фінансування розробника не перевищує встановлену компанією межу. Після того, як проєкт стає комерційно успішним, можна перейти на одну з платних підписок, що відкривають доступ до розширених функцій, таких як розширене профілювання, підтримка хмарних сервісів або спеціалізована допомога від технічної служби Unity.

Після всебічного аналізу можливостей і порівняння з іншими рушіями було прийнято рішення використовувати Unity для розробки цього дипломного проєкту. По-перше, в Unity уже з коробки є все необхідне для створення високоякісної 2D-гри: тайлмап-редактор полегшує створення арени, інструменти анімації гарантують плавні переходи між станами персонажів (біг, вибух, смерть), а фізичний рушій дозволяє коректно обробляти зіткнення бомб із межами карти й іншими об’єктами. По-друге, механізми профілювання та оптимізації, що вбудовані в редактор, допомагають уже під час розробки відстежувати навантаження на CPU й GPU, а таким чином вчасно виявляти й усувати вузькі місця, що особливо важливо при орієнтації гри на широкий спектр Android-пристроїв. Нарешті, велика спільнота та Asset Store дають змогу швидко знайти

готові приклади мультиплеєра, UI-елементів або спецефектів, що економить час і дозволяє зосередитися на авторських механіках гри. Завдяки цим причинам Unity став оптимальним вибором для реалізації дипломного 2D-аркадного проєкту.

1.4 Огляд інструментарію та технологій Unity для розробки ігор

Інструмент 2D Sprite Editor у Unity призначений для повного контролю над імпортом, налаштуванням і оптимізацією двовимірної графіки, яку в іграх зазвичай називають спрайтами. Як тільки ви перетягуєте будь-яке зображення у форматі PNG, JPG або навіть багатошарове PSD у проєкт, Unity автоматично розпізнає його як спрайт. При імпорті багатоосібкових PSD-файлів рушій зберігає інформацію про шари, що дає змогу працювати з кожним елементом окремо, без необхідності заздалегідь розрізати зображення у сторонньому редакторі.

У середовищі 2D Sprite Editor ви можете вибирати, як саме трактувати кожне зображення: чи це єдиний спрайт для цілого полотна, чи ж варто розбити його на набір окремих фрагментів. Якщо ж форма об’єкту сильно відрізняється від прямокутника, є можливість намалювати власний полігон, щоб контур спрайту облягав саме ту область, яка заповнена кольором, без зайвих прозорих пікселів. Параметр “Pixels Per Unit” дозволяє вказати, скільки пікселів зображення відповідатиме одній одиниці в ігровому просторі: це визначає масштаб спрайта відносно інших елементів сцени. За допомогою налаштування точки прив’язки (Pivot) ви вказуєте, яка саме точка спрайта служитиме центром обертання або зразком для позиціонування—наприклад, чи це середина зображення, чи будь-який інший вибраний вами піксель. Крім того, можна обирати, чи генерувати для спрайта прямокутну сітку, яка перекриває весь прямокутник зображення, або ж вкладати обмежувальний багатокутник, який максимально точно оточує справжню фігуру об’єкта.

Розбивка зображень на окремі спрайти (Sprite Slicing) реалізується автоматично через виявлення прозорих областей або вручну, задаючи власні межі для кожного фрагмента. У результаті натискання двох-трьох кнопок ви отримуєте набір окремих зображень, які потім підтягуються у анімації як окремі кадри або

розташовуються в тайлмап-редакторі задля формування рівня. Для кожного виявленого фрагменту окремо можна задати власну точку прив’язки, що суттєво спрощує точне вирівнювання різних елементів, особливо коли вони анімовані переходами.

Коли суспільство мобільних й інших платформ вимагає компактних проєктів із швидким завантаженням, 2D Sprite Editor надає можливості стискати зображення за різними алгоритмами, зменшуючи їхній кінцевий розмір без втрати помітної якості. У разі потреби можна увімкнути генерацію міпмапів, які підтягують лише відповідний рівень деталізації в залежності від того, наскільки далеко камера знаходиться від спрайта, що покращує продуктивність на слабших пристроях. Ще однією суттєвою перевагою є підтримка створення атласів (Sprite Atlas), коли кілька окремих спрайтів групуються в одне велике зображення. Це дозволяє рушію значно зменшити кількість звернень до текстурного буфера на рівні відеокарти — у результаті рендер проходить швидше, особливо коли на сцені одночасно відображається багато різних спрайтів.

Крім плаского атласу, інколи треба масштабувати інтерфейсні елементи або графіку з неоднорідною формою без спотворення країв. Для цього існує техніка 9-Slice Scaling: дев’ять окремих частин зображення масштабуються по-різному, аби центрова ділянка могла розтягуватися без зміни товщини рамок по периметру. Так само редактор дозволяє використовувати сплайни Sprite Shape, коли необхідно створювати довільно вигнуті контури—скажімо, паркани, річки чи доріжки—за допомогою кількох контрольних точок, а рушій автоматично підтягує підходящу текстуру уздовж цих кривих.

Для керування фізичною взаємодією двовимірних об’єктів Unity використовує рушій Box2D, який реалізований через компоненти Rigidbody 2D, Collider 2D і набори додаткових допоміжних елементів. Якщо ви додаєте до спрайта компонент Rigidbody 2D, цей об’єкт починає реагувати на сили та гравітацію, може рухатися під дією фізичних імпульсів та зіштовхуватися з іншими активними елементами сцени. За замовчуванням це динамічний тип Rigidbody, що означає, що рушій лише після певної кількості кубічних

перетворень відстежує пришвидшення, масу й інерцію, а ви можете задавати ці параметри, змінюючи вагу або обмежуючи максимальну швидкість об’єкта. Якщо ж вам потрібно, щоб об’єкт рухався виключно за логікою скрипта, не реагуючи автоматично на зіткнення, достатньо змінити режим на “Kinematic”: тоді будь-які зміни його позиції чи обертання виконуються лише через код, а фізика Box2D ігнорує ці перетворення. А ось якщо об’єкт завжди має залишатися нерухомим (наприклад, земна поверхня, стіни або будь-які перешкоди, що «непорушні»), варто встановити його як “Static” — після цього рушій більше не керує ним узагалі, і немає потреби витрачати процесорний час на обрахунок його руху.

Форма зіткнення для кожного Rigidbody задається компонентом Collider 2D, а саме, колайдери бувають кількох типів: прямокутна рамка (Box Collider 2D), кругова межа (Circle Collider 2D), довільний багатокутник (Polygon Collider 2D) і навіть лінійний контур (Edge Collider 2D), що часто застосовується для платформи з нерівним рельєфом. Капсульний колайдер (Capsule Collider 2D) виглядає як поєднання прямокутника з двома півколами з боків і добре підходить для фізичної моделі персонажів у платформерах, адже він плавно ковзає по нерівних поверхнях. Для кожного з колайдерів можна вказати, чи працювати з ним у звичайному режимі фізичних зіткнень, коли оновлюється швидкість і напрям руху об’єкта, або зробити його “Trigger”. У разі тригера при зіткненні об’єкти не відштовхуються — натомість генерується подія, яку ви вже обробляєте у скрипті (наприклад, збір бонуса чи запуск анімації). Щоб коректно відштовхувати об’єкти один від одного з урахуванням тертя та упругості, можна вказати фізичний матеріал (Physics Material 2D), задавши силу тертя або відскоку, що дозволяє реалізувати різні поверхні — від слизького льоду до м’якої подушки.

Якщо в грі потрібно з’єднати об’єкти особливим чином — наприклад, зробити двері, що відкриваються як шарнір, або транспорт, у якого колеса повинні крутитися й одночасно підтримувати вагу кузова, — Unity пропонує набір Joint 2D. Distance Joint 2D тримає два об’єкти на фіксованій відстані, наче їх зв’язує невидима штанга. Hinge Joint 2D створює шарнірне з’єднання, яке дозволяє обертанню навколо однієї точки, як у дверної петлі. Якщо необхідно, щоб один

блок ковзав уздовж прямої лінії, підходить Slider Joint 2D. Для коліс авто існує спеціалізований Wheel Joint 2D, де можна задавати пружність підвіски, силі спротиву обертанню та інші параметри, а Spring Joint 2D поводиться як віртуальна пружина, тягнучи чи штовхаючи об’єкти один до одного відповідно до заданої жорсткості.

Система Animator у Unity відповідає за анімацію ігрових об’єктів та надає єдиний інтерфейс для налаштування різних станів руху, переходів між ними і способу керування цими переходами. Серцем цієї системи є Animator Controller — файл, у якому ви візуально розташовуєте різні анімаційні стани у вигляді вузлів на графі, а стрілки між ними демонструють, як відбувається перехід одного стану в інший. Кожен вузол пов’язаний із конкретним Animation Clip — послідовністю кадрів, у якій задається, як змінюється позиція або властивості об’єкта з плином часу. Наприклад, для ігрового персонажа це можуть бути окремі кліпи для бігу, стрибка, стояння чи атаки.

Перехід між двома анімаційними станами здійснюється не відразу: ви задаєте умови, що мають виконатися, аби рушійну систему переключити зі стану А у стан Б. Ці умови базуються на параметрах, які зберігаються безпосередньо в Animator Controller. Параметри бувають булевими, числовими або тригерами, і саме вони визначають, коли має відбутися перехід. Наприклад, якщо ваш скрипт в певний момент перевіряє швидкість персонажа і відсилає у Animator значення змінної “Speed” рівне більш ніж п’ять одиниць, а на графі є стрілка, яка спрацьовує за цієї умови, то анімація зміни кадрів просто перемикається з “Idle” на “Run” одним плавним рухом. Якщо ж для одноразової події (наприклад, підбір скрині з бонусом) використовують тригерний параметр, система спрацьовує на моментальний стрибок у певний стан, де виконується потрібний Animation Clip (відкриття скрині), після чого зазвичай повертається у базовий стан, коли тригер знову неактивний.

Коли один стан має відпрацювати повністю, перш ніж почати інший, можна налаштувати перехід із увімкненням опції “Exit Time”, яка забороняє рушію переключатися на іншу анімацію, доки поточний кліп не дійшов до свого кінця.

Якщо ж ви хочете змінювати анімації миттєво, як тільки параметр отримує необхідне значення, можна виключити Exit Time і встановити нульову затримку. Окрім цього, для кожного переходу можна вказати “Transition Duration” — тривалість плавного згасання попереднього кліпу та нарощування нового, щоб рухи персонажа чи об’єкта не виглядали різко обірваними.

Крім графу станів, Animator Controller дозволяє створювати підграфи та підмодулі, що дуже зручно для великих проєктів: наприклад, всі кліпи, пов’язані з рухом персонажа, можна винести в окремий підконтролер, а кліпи інтерфейсу — у свій власний блок. Це робить організацію анімацій більш зрозумілою й зручною для підтримки: розробники та аніматори можуть працювати незалежно, вносячи зміни лише у свої частини графу.

Щоби керувати параметрами Animator із коду, у C# ви отримуєте доступ до компоненту Animator і можете викликати методи типу SetBool, SetTrigger чи SetFloat, передаючи назву або хеш параметра, а рушій сам обробить потрібні переходи. Це дозволяє «зв’язати» внутрішню фізику чи логіку гри з візуальним виглядом: наприклад, коли ворог починає рухатися у напрямку персонажа, ви вмикаєте анімацію бігу, а коли рівень здоров’я наближається до нуля, рушієте параметр тригера “Die” і запускаєте аварійну анімацію.

Таким чином, поєднання 2D Sprite Editor для точного налаштування спрайтів, фізичного рушія Box2D для реалістичної взаємодії в 2D-просторі та Animator для гнучкого керування анімаціями дає змогу створювати складні, візуально привабливі й водночас оптимізовані двовимірні ігри в Unity. Ці інструменти працюють у тісному взаємозв’язку: від правильного обрізання й стиснення спрайтів залежить ефективність роботи фізики та плавність анімацій, а грамотно налаштовані Collider 2D і Rigidbody 2D гарантують, що персонажі та об’єкти реагуватимуть на події правильно, відображаючи свою поведінку в Animator без затримок і збоїв. Завдяки багатому функціоналу кожен етап розробки, від створення графіки до реалізації механік колізій і відтворення анімацій, залишається гнучким і контролюваним, що суттєво спрощує роботу над 2D-проєктами.

1.5 Середовище розробки Visual Studio

Кроме автоматической настройки проекта, Visual Studio обеспечивает мощные возможности для отладки кода, что крайне важно при разработке игр. При запуске проекта из Unity, Visual Studio позволяет устанавливать точки останова (breakpoints), отслеживать значения переменных в реальном времени и выполнять поэтапный пошаговый анализ логики игрового процесса. Это упрощает поиск ошибок в скриптах и помогает быстро выявить причину некорректного поведения элементов, например, когда коллайдеры неправильно взаимодействуют или анимации не отрабатывают ожидаемым образом.

Еще одним важным преимуществом является встроенная система автодополнения кода (IntelliSense), которая ускоряет написание скриптов, подсказывая доступные методы и параметры классов Unity API. Благодаря этому разработчик тратит меньше времени на запоминание точных имен функций и может сразу видеть список доступных свойств у объектов, таких как Rigidbody2D или Animator. Наличие шаблонов кода и встроенных инструментов для рефакторинга (например, переименование переменных или автоматическое исправление неиспользуемых пространств имен) облегчает поддержку чистой и читабельной структуры проекта даже по мере его роста.

1.5.1 Unity Tools for Visual Studio

Unity Tools for Visual Studio представляє собою спеціальний набір розширень, що суттєво покращують роботу з Unity у середовищі Visual Studio. Завдяки інтелектуальному автозаповненню розробник отримує миттєві підказки щодо методів, властивостей і класів Unity API, що значно пришвидшує написання коду: імена змінних та функцій доповнюються автоматично, а підказки щодо параметрів дозволяють уникнути помилок. Окрім того, Visual Studio оснащена окремим провідником Unity Project Explorer, який дає змогу легко переглядати ієрархію файлів проекту без необхідності переходу до Unity Editor, що особливо корисно при роботі над великими проєктами з великою кількістю папок та скриптів. Інструменти налагодження вбудовані безпосередньо в IDE: коли ви

запускаєте налагодження з Unity, Visual Studio дозволяє встановлювати точки зупинки, виконувати код покроково та в будь-який момент переглянути значення змінних, що спрощує відстеження причин некоректної роботи ігрової логіки чи раптових збоїв.

Крім інструментів для автозаповнення й налагодження, Visual Studio налаштована так, щоб бездоганно обробляти специфічні для Unity файли — C#-скрипти, Shader-файли та інші ресурси. Змінивши код у C#-скрипті та зберігши файл у Visual Studio, ви автоматично ініціюєте процес перезавантаження та компіляції в Unity, тож цикл “редагування — перевірка результату” стає надзвичайно швидким. Така інтеграція дозволяє не відволікатися на перемикання між редакторами та гарантує, що кожна правка відразу потрапляє в ігрове середовище на тестування.

Окремо варто відзначити можливість працювати з системами контролю версій, наприклад із Git, безпосередньо зі середовища Visual Studio. Інструменти для комітів, пушів і пулів змін вбудовані в IDE, що забезпечує злагоджену командну роботу й дозволяє відразу фіксувати зміни у проекті, не виходячи з редактора. Підсумовуючи, використання Visual Studio разом із Unity Tools робить процес розробки більш зручним і ефективним: автоматичне налаштування проєкту, розширені підказки коду, потужний налагодчик, зручне відтворення Unity-специфічних файлів і безшовна інтеграція з контролем версій створюють оптимальні умови для швидкої та якісної розробки мобільної 2D-ігри на Unity.

1.6 Мультиплеєр

Мультиплеєрні ігри утворили невід’ємну частину сучасної індустрії розваг, адже дають можливість гравцям з різних куточків світу взаємодіяти та змагатися у спільному віртуальному середовищі. Завдяки стрімкому розвитку мобільних технологій такі ігри отримали новий поштовх: тепер користувачі можуть грати разом будь-де й будь-коли. Серед мультиплеєрних проєктів вирізняють кілька класифікацій, що формуються залежно від механік взаємодії та характеру геймплею.

Перший тип можна назвати динамічним одиночним режимом із мережевими можливостями. У таких іграх основна частина геймплею передбачає самостійну гру, проте розробники додають функції, які поєднують соціальні елементи: з’являються глобальні таблиці рекордів, кнопки для швидкого поширення своїх результатів у соціальних мережах та регулярне оновлення контенту. Тобто, хоча гравець переважно проводить час у своєму власному світі, він водночас може порівнювати власні досягнення з досягненнями інших учасників, змагатися за місце у рейтингу та миттєво ділитися своїми успіхами з друзями.

Інший підхід — це покроковий мультиплеєр, коли за одним ігровим столом або в одній сесії бере участь кілька осіб, але їхні дії виконуються по черзі. Такий формат ігор ідеально підходить для стратегічних та настільних ігор, оскільки кожен гравець устигає обдумати свій хід, побудувати довгострокову тактику та відкоригувати стратегію, спостерігаючи за діями суперників. Завдяки цьому ігровий процес стає більш розміреним і інтелектуально насиченим, а часу на обдумування вистачає, навіть якщо між ходами кілька хвилин тиші.

Третій варіант — це мультиплеєр у реальному часі, коли кілька учасників (зазвичай до кількох десятків залежно від жанру й архітектури сервера) одночасно перебувають в одній ігровій сесії. Цей підхід вимагає від гравців блискавичної реакції, швидкої комунікації та узгодженості дій. Саме він лежить в основі ігор-шутерів, автоперегонів, динамічних аркад чи спортивних симуляторів, де кожна секунда може вирішувати результат. Завдяки високій інтенсивності та постійному зв’язку з іншими гравцями, такі проекти забезпечують неабияку порцію адреналіну й відчуття змагання.

Нарешті, існують ігри з постійним, одночасно існуючим для всіх учасників ігровим світом — так звані «Persistent Game Spaces». У цьому випадку незалежно від того, чи перебуває конкретний гравець в онлайні, світ продовжує розвиватися: будівлі можуть руйнуватися, ресурси накопичуватися, а інші учасники чинять вплив на загальні події. Такий формат характерний для масових багатокористувацьких онлайн-ігор (ММО), де сотні й тисячі гравців взаємодіють

у рамках єдиного всесвіту, творячи його історію власними діями. Тривалий ефект від дій кожного учасника створює відчуття безперервності та масштабності, адже поява нового контенту, зміна економіки чи політичних відносин відбувається в реальному часі й впливає на всіх одночасно.

У підсумку, класифікація мультиплеєрних ігор охоплює проекти, де гравець може грати поодинці з елементами соціальної взаємодії, пошагові стратегії з обдуманими діями учасників, динамічні бої та гонки в реальному часі й масштабні віртуальні світи з тривалим станом для всіх. Кожен із цих підходів знаходить своє місце на ринку та приваблює різні категорії гравців залежно від їхніх уподобань і стилю гри.

1.6.1 Вибір технології для реалізації мультиплеєра

Для створення мультиплеєрного режиму в нашій мобільній 2D-грі на Android було обрано Photon Unity Networking 2 (PUN2) через його зручність, продуктивність і готові інструменти для швидкого запуску мережевих функцій. Цей фреймворк інтегрується із середовищем Unity практично «з коробки»: після встановлення відповідного пакета в редакторі одразу можна побачити готові шаблони для створення кімнат, налаштування підключення та синхронізації об’єктів. У документації PUN2 міститься багато прикладів коду й покрокових інструкцій, які дозволяють навіть тому, хто вперше стикається з розробкою мережевих ігор, без значних зусиль розібратися в основних концепціях. Це означає, що замість довгих налаштувань серверної частини ви можете одразу переходити до написання власного ігрового сценарію, а не витрачати час на базові комунікаційні механізми.

Фундаментальною перевагою PUN2 є його оптимізована архітектура, яка забезпечує низьку затримку й економне використання ресурсів пристрою. На мобільних телефонах та планшетах, де доступна оперативна пам’ять і процесорна потужність обмежені, це дуже важливо: PUN2 лише передає необхідні оновлення стану гри між клієнтами (наприклад, координати персонажів чи події вибуху), уникаючи надмірного споживання трафіку та зайвих обчислень. Така архітектура

дає змогу підтримувати плавну роботу навіть у випадку одночасної взаємодії декількох гравців у режимі реального часу.

Ще один важливий аспект полягає в тому, що Photon Unity Networking 2 підтримує кросплатформені з'єднання, що дозволяє гравцям на різних пристроях — Android, iOS або ПК — одночасно перебувати в одній ігровій сесії. Це значно розширює потенційну аудиторію гри: ви не обмежені лише мобільними користувачами, а можете об'єднати фанатів із різних платформ у єдиному матчі. Крім того, сам Photon дозволяє обирати між хмарним сервером (Cloud Server) і власним виділеним сервером (Dedicated Server). У хмарному режимі достатньо використовувати безкоштовний або преміальний план Photon Cloud, щоб миттєво розгорнути інфраструктуру й не піклуватися про фізичні сервери. Якщо ж у вас передбачено велику кількість одночасних підключень або ви хочете детально контролювати мережеву топологію, можна налаштувати власний виділений сервер із потрібними параметрами масштабування.

Особливістю Photon є те, що спільнота розробників і сама команда Photon активно підтримують проєкти, що використовують цей фреймворк. В інтернеті легко знайти відповіді на найрізноманітніші питання: від налаштування базових з'єднань до реалізації складних механік синхронізації об'єктів і масштабованих лобі. Це дозволяє зосередитися безпосередньо на геймдизайні та оптимізації власної логіки, не витрачаючи годинник на «гугління» ігорних глюків чи пошуку рідкісних багів у бібліотеках.

Створена гра належить до категорії Real-Time Multiplayer (мультиплеєр у реальному часі), тому всі гравці одночасно перебувають у спільному ігровому просторі. Завдяки цьому ігровий процес набуває максимальної динаміки: учасники можуть одночасно рухатися по карті, ставити бомби, ухилятися від вибухів і миттєво реагувати на дії інших гравців. Така взаємодія стимулює соціальне змагання та взаємодію, адже ваш наступний крок залежить від того, що вже зробили суперники або союзники. Використання PUN2 у такому контексті гарантує стабільність під час високого навантаження, адже цей фреймворк дозволяє розподіляти трафік у мережі, підтримувати кілька кімнат одночасно й

забезпечувати ігрову сесію навіть при нестабільному мобільному з’єднанні. У результаті гравці отримують плавний і злагоджений ігровий досвід, а розробник — можливість легко збільшувати кількість одночасних користувачів у разі зростання популярності гри.

1.7 Проектування та розробка гри

1.7.1 Створення головного меню

Почнемо створення гри з найпершого елементу, з яким взаємодіє гравець при запуску — головного меню. Саме воно формує перше враження про гру та задає загальний стиль інтерфейсу. Для цього на першому етапі створимо нову сцену в Unity, яку назвемо відповідно до її призначення — "**MainMenu**". У подальшому ця сцена буде містити кнопки переходу до основних режимів гри (наприклад, "Грати", "Налаштування", "Вихід") та елементи візуального оформлення, що забезпечать зручну навігацію й приємний користувацький досвід.

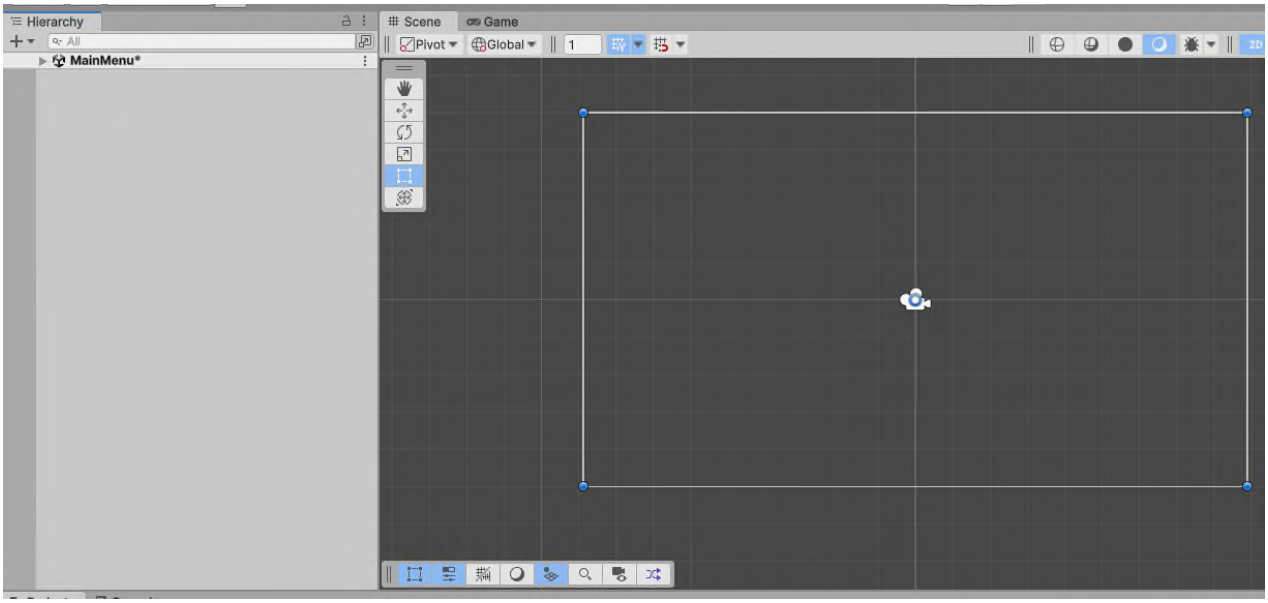


Рисунок 1.8 Створена сцена "MainMenu"

Наступним кроком додаю Canvas, який слугуватиме основою для розміщення всіх UI-елементів головного меню. У налаштуваннях Canvas у полі Render Camera вказую основну камеру сцени, щоб інтерфейс коректно відображався відносно її позиції. Для забезпечення адаптивності інтерфейсу на різних розмірах екранів, у параметрі UI Scale Mode обираю Scale With Screen Size,

а значення Match встановлюю на 0.5, щоб інтерфейс однаково підлаштовувався як під ширину, так і під висоту екрана.

Після цього всередині Canvas створюю новий елемент Panel, який називаю “MainMenuPanel”. Ця панель стане контейнером для усіх елементів головного меню — кнопок, логотипів, текстів тощо. Завдяки їй можна буде легко організовувати структуру інтерфейсу, змінювати стиль оформлення й групувати елементи для подальшої обробки.

Тепер створимо необхідні кнопки :

- Create Lobby - відкриває меню створення та налаштування параметрів лобі
- Join Lobby - Відкриває меню зі списком доступних лобі.
- Setting - Налаштування
- Exit - Вихід

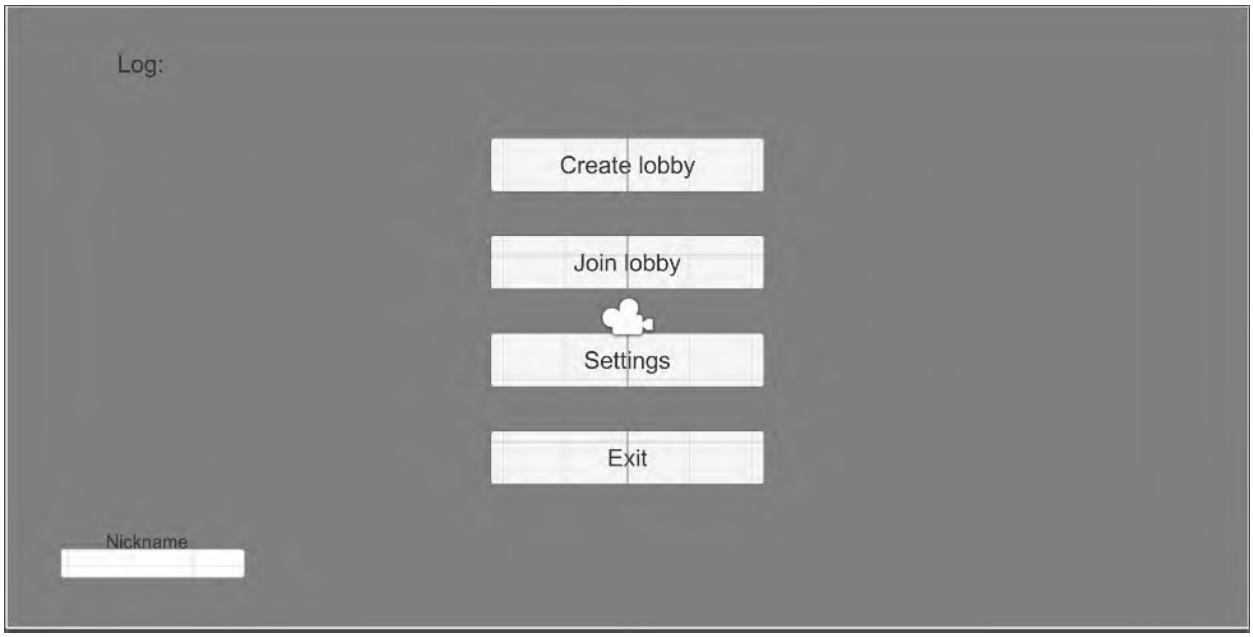


Рисунок 1.9 MainMenuPanel

Для зручності взаємодії з користувачем у сцену гри було додано елемент інтерфейсу — поле введення (Input Field), призначене для введення імені гравця. Це поле активне для редагування: користувач може самостійно змінити ім’я, натиснувши на нього та ввівши нове значення з клавіатури. Збереження імені може бути реалізовано за допомогою механізмів PlayerPrefs, щоб при наступному запуску гри дані залишалися актуальними.

Окрім цього, на екрані передбачено текстове поле, що відображає службові повідомлення, наприклад, інформацію про стан з’єднання з сервером, повідомлення про помилки чи успішне збереження даних. Таке поле автоматично оновлюється під час роботи програми, забезпечуючи зворотний зв’язок.

Після цього створюю ще дві панелі, надаючи їм відповідні назви — Create Lobby Panel і Join Lobby Panel. Вони слугуватимуть окремими інтерфейсними блоками для створення та приєднання до лобі. У кожну з панелей додаю необхідні елементи управління: текстові поля для введення імені або коду кімнати, кнопки для підтвердження дії, а також інформаційні написи, що пояснюють призначення кожного поля. Завдяки цьому інтерфейс стає зрозумілим, інтуїтивно зручним і готовим до взаємодії з серверною частиною гри.

1. CreateLobbyPanel:

– Input Field: Для введення назви лобі.

– Dropdown: Для вибору максимальної кількості гравців.

– Toggle: Для визначення типу кімнати (відкрита/закрита).

– Button "Create Room": Для створення лобі з вказаними параметрами.

– Button "back": Для закриття панелі створення лобі та повернення до головного меню.

2. JoinLobbyPanel:

– Scroll View: Для відображення списку доступних кімнат (лобі).

– Input Field : Для введення імені лобі.

– Button "Join": Для приєднання до вибраного лобі.

– Button "Join Random Lobby": Для приєднання до випадкової кімнати.

– Button "Back": Для закриття панелі приєднання до лобі та повернення до головного меню.

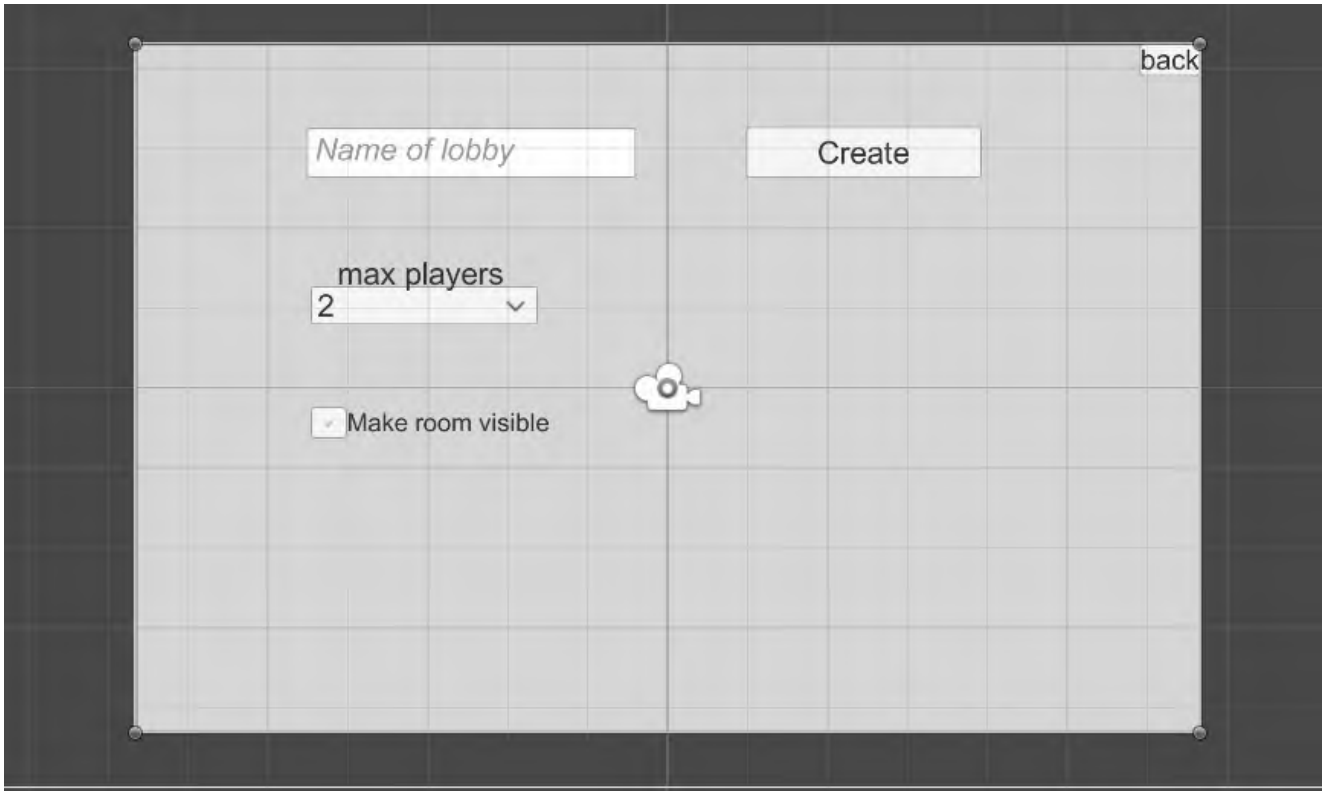


Рисунок 1.10.CreateLobbyPanel

Рисунок 1.11 JoinLobbyPanel

Далі додаю фонове зображення, яке слугуватиме загальною візуальною основою для головного меню, створюючи відповідну атмосферу гри. Також

імпортую підготовлені спрайти для кнопок та інших елементів інтерфейсу, що дозволяє надати меню стильний, цілісний вигляд. Застосування унікального дизайну не лише покращує естетику, а й забезпечує кращу впізнаваність проєкту, роблячи взаємодію з гравцем приємнішою та більш професійною.

Рисунок 1.12 MainMenuPanel після додавання спрайтів

Рисунок 1.13 CreateLobbyPanel після додавання спрайтів

Рисунок 1.14 JoinLobbyPanel після додавання спрайтів

На наступному етапі створюємо два C#-скрипти: UIManager і PhotonNetworkManager. У Unity скрипти пишуться на C# і додаються до ігрових об’єктів як компоненти. Вони мають доступ до всіх інших компонентів цього об’єкта, можуть змінювати їхні властивості, викликати методи та реагувати на різні події. Саме завдяки скриптам ігрові об’єкти отримують свою логіку й взаємодіють між собою в сцені.

- UIManager відповідає за взаємодію з інтерфейсом користувача. Цей скрипт контролюватиме показ і приховання різних панелей (головне меню, створення лобі, приєднання до лобі), збиратиме введені гравцем дані (наприклад, нікнейм чи назву кімнати) та передаватиме їх у PhotonNetworkManager для подальших мережевих операцій.
- PhotonNetworkManager реалізує всю мережеву логіку. Він встановлюватиме з’єднання з серверами Photon, керуватиме створенням і приєднанням до кімнат, а також оброблятиме події мережевої взаємодії (наприклад, успішне підключення чи оновлення списку доступних кімнат).

Нижча діаграма ілюструє, як Unity виконує й повторно викликає функції подій протягом життєвого циклу скрипта.

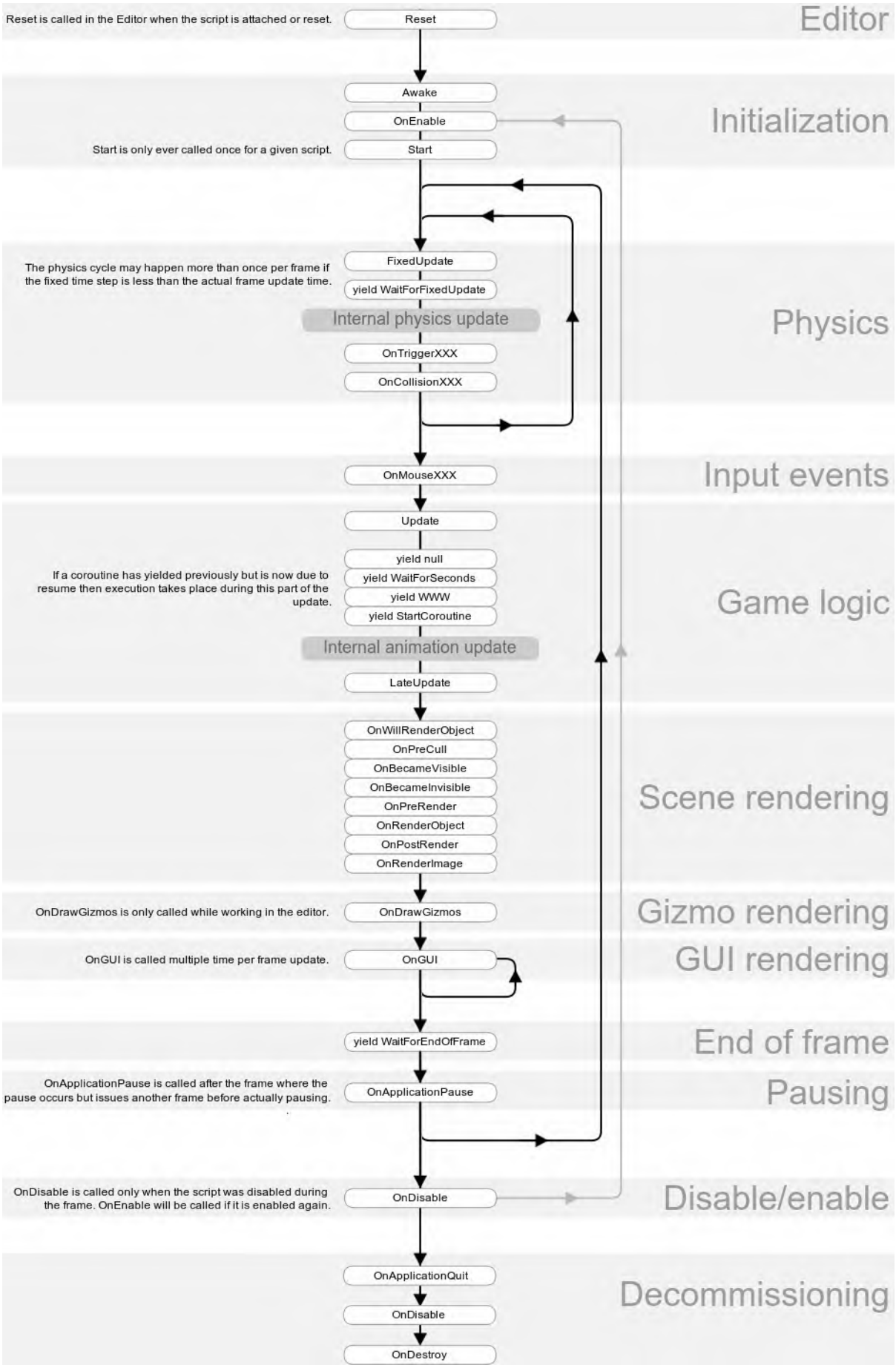


Рисунок 1.15 Діаграма життєвого циклу скрипта

У скрипті PhotonNetworkManage у методі Strart(), що викликається один раз при створенні екземпляра скрипта, пишемо наступний код:

```
  void Start()
{
    PhotonNetwork.AutomaticallySyncScene = true;
    PhotonNetwork.GameVersion = "1.0";
    PhotonNetwork.ConnectUsingSettings();
}
```

Отже, підключення до серверів Photon виконується за допомогою налаштувань, що зберігаються у файлі PhotonServerSettings. У цьому файлі вказані всі необхідні параметри для зв'язку з сервером – зокрема App ID, вибраний регіон та інші ключові опції.

Рисунок 1.16 PhotonServerSettings

Додамо метод CreateRoom() він відповідатиме за створення нової кімнати на сервері Photon і прийматиме три параметри maxNumberPlayers, isRoomVisible, roomName.

– maxNumberPlayers - максимальна кількість гравців у лобі
– isRoomVisible - видно в списку доступних кімнат
– roomName - ім'я кімнати

public void CreateRoom(int maxNumberPlayers, bool isRoomVisible, string roomName)

{

RoomOptions roomOptions = new RoomOptions();

roomOptions.MaxPlayers = maxNumberPlayers;

roomOptions.IsVisible = isRoomVisible;

roomOptions.EmptyRoomTtl = 0;

PhotonNetwork.CreateRoom(roomName, roomOptions);

}

Наступний медот буде призначений для встановлення нікнейму гравця у Photon.

public void SetPlayerNickname(string nickname)

{

PhotonNetwork.NickName = nickname;

}

А цей метод при викликі прийматиме рядок, що містить ім'я кімнати, і підключатиме гравця до кімнати із зазначеним ім'ям.

public void ConnectToRoom(string roomname)

{

PhotonNetwork.JoinRoom(roomname);

}

При підключенні до лобі гравець повинен переходити на нову сцену, створимо її та назвемо game. Далі перевизначимо метод OnJoinedRoom із класу MonoBehaviourPunCallbacks

public override void OnJoinedRoom()

{

PhotonNetwork.LoadLevel(1);

}

Перейду до написання коду кнопок у скрипті UIManager

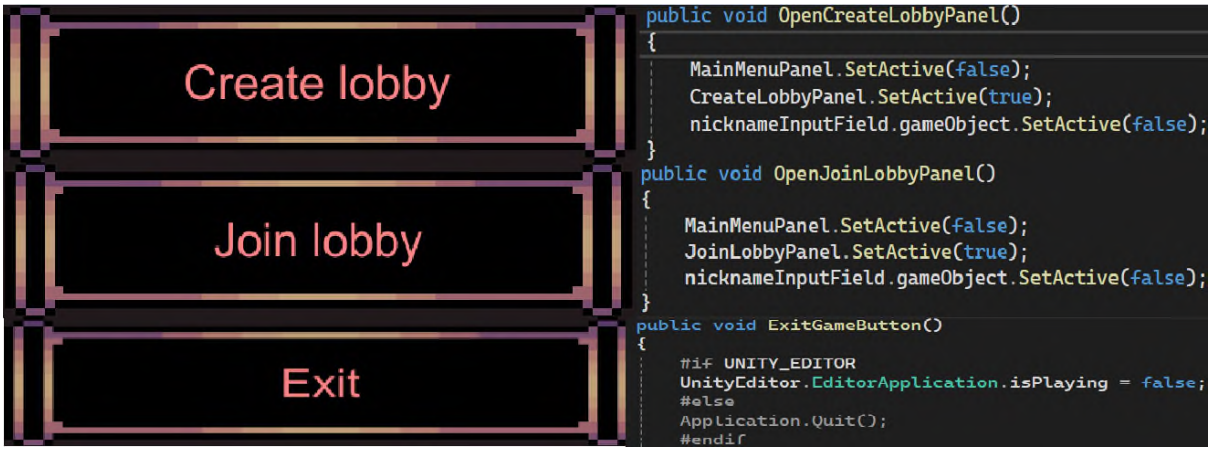


Рисунок 1.17 Код кнопок

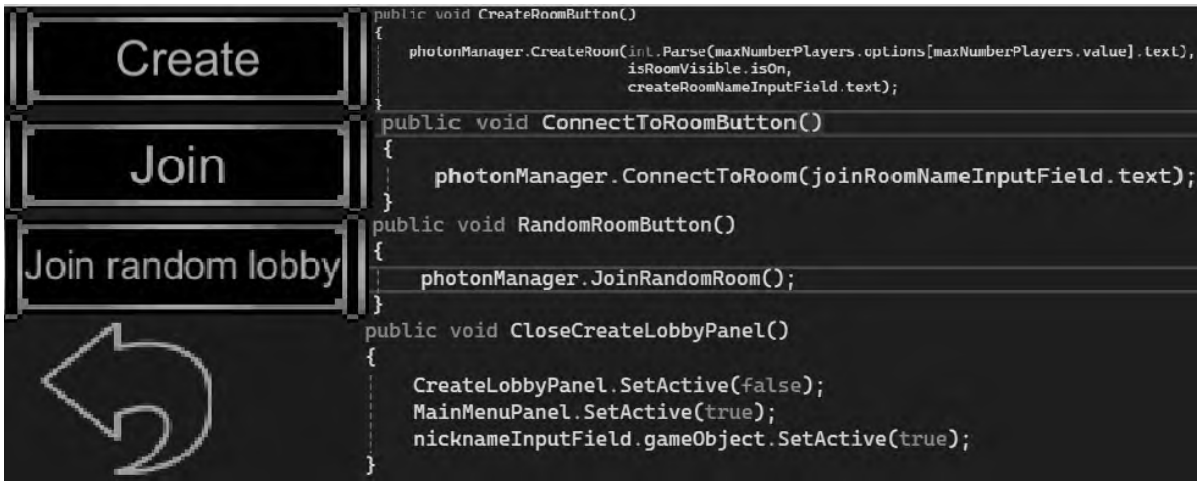


Рисунок 1.18 Код кнопок

Введений нікнейм в Input Field повинен зберігатися між сесіями для цього використовуємо PlayerPrefs. Цей клас дозволяє зберігати дані гравця між сесіями гри. Дані зберігаються у спеціальному файлі на пристрої гравця.

public void SaveNickname()

{

// Сохраняем никнейм игрока в PlayerPrefs

PlayerPrefs.SetString("PlayerNickname", nicknameInputField.text);

photonManager.SetPlayerNickname(nicknameInputField.text);

}

Та встановлю цей метод на подію On End Edit у Input Field. Так нікнейм буде зберігатися при його зміні.

Збереження імені готове, тепер необхідно, щоб при старті гри збережений нікнейм відображався в Input Field і встановлювався в Photon Network.

Цей метод скрипту UIManager буде викликатися один раз під час запуску сцени.

```
void Start()
{
    string Nickname = PlayerPrefs.GetString("PlayerNickname", "");
    if (!string.IsNullOrEmpty(Nickname))
    {
        nicknameInputField.text = Nickname;
    }
    else
    {
        Nickname = "Player" + Random.Range(1, 1000);
        nicknameInputField.text = Nickname;
    }
    photonManager.SetPlayerNickname(Nickname);
}
```

Спочатку ми пробуємо дістати значення з PlayerPrefs за ключем «PlayerNickname». Якщо воно існує, використовуємо його, інакше працюємо з порожнім рядком. Далі перевіряємо, чи отриманий рядок не є порожнім; якщо він містить текст, відразу підставляємо його в InputField. Якщо ж нічого не знайдено, створюємо випадковий нік у форматі «PlayerXXX» (де XXX — число від 1 до 999) і вже його відображаємо в полі вводу. Врешті викликаємо метод SetPlayerNickname з нашого скрипта PhotonNetworkManager, щоб передати в мережу або завантажений, або щойно створений нік.

Рисунок 1.19 Нікнейм гравця

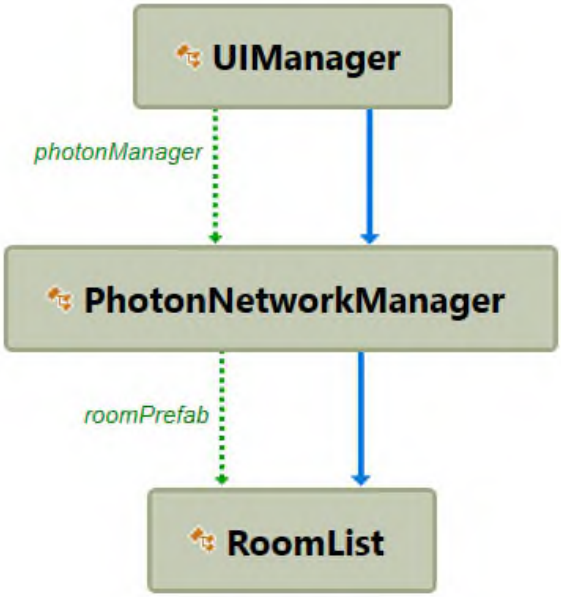


Рисунок 1.20 Діаграма класів, що відповідають за інтерфейс користувача та мережеву логіку

1.7.2 Створення рівня гри

Для створення ігрового рівня використовувався Tilemap, що дозволило оперативно збудувати 2D-місцевість, комбінуючи спрайти, які зображають підлогу, стіни та інші об’єкти довкілля. Tilemap забезпечує легке налаштування порядку шарів, що гарантує коректне накладання елементів один на одного, і дає змогу підключити Tilemap Collider для автоматичного створення колайдерів, аби персонажі та інші об’єкти могли взаємодіяти з навколишнім середовищем. Для зручності розміщення й організації всіх цих спрайтів на сцені було створено Tile Palette. Цей інструмент дозволяє зібрати набір потрібних графічних елементів у вигляді палітри, з якої їх можна легко вибирати та малювати безпосередньо на Tilemap.

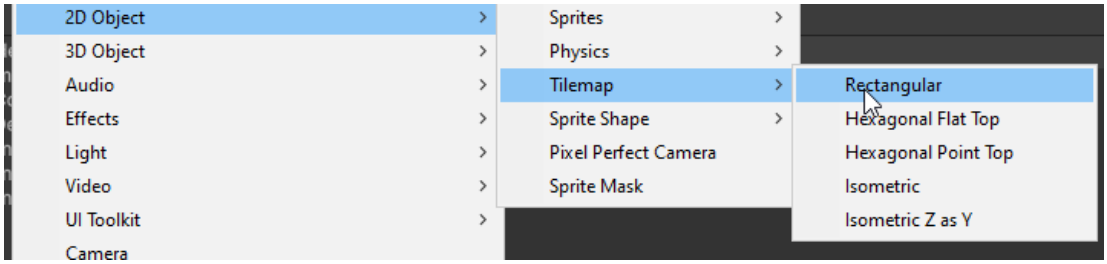


Рисунок 1.21 Створення Tilemap

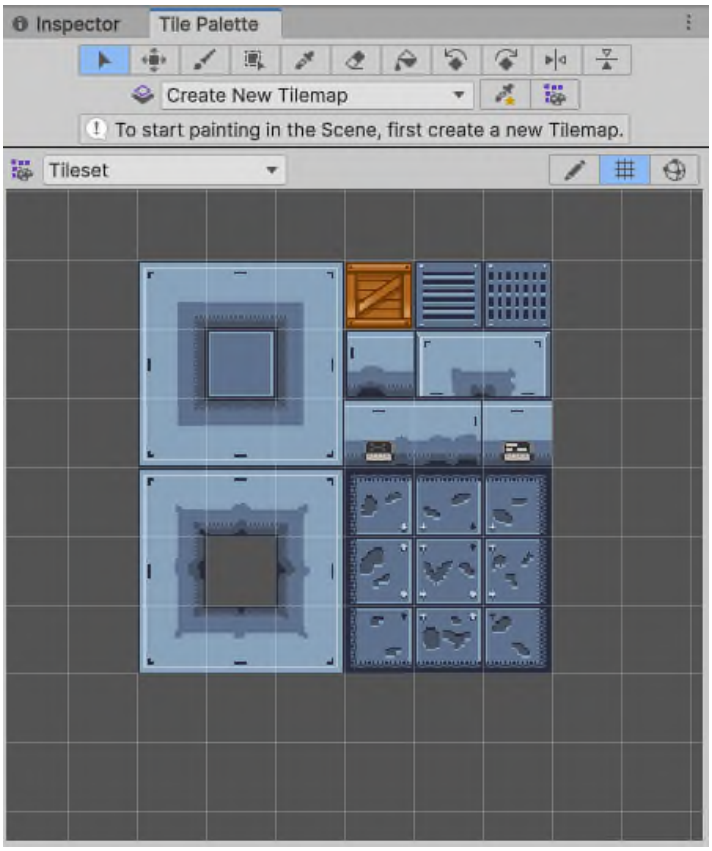


Рисунок 1.22 Створення Tile Palette

На рівні використовуються два Tilemap:

– "Indestructible": Призначений для об'єктів, що не руйнуються (земля, стіни). У компоненті Tilemap Renderer цього Tilemap встановлено значення Order in Layer рівним 0.

– "Destructible": Призначений для об'єктів, що піддаються руйнуванню бомбами (наприклад, ящики). У компоненті Tilemap Renderer цього Tilemap встановлено значення Order in Layer рівним 1.

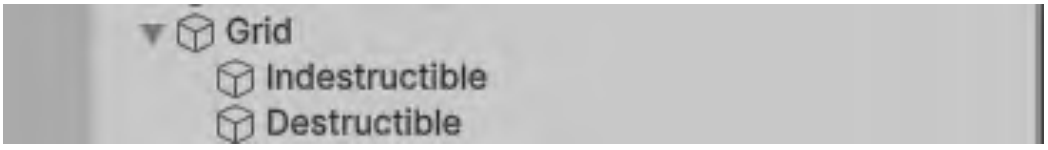


Рисунок 1.23 Tilemap та "Indestructible" та "Destructible"

До обох Tilemap додаю компонент Tilemap Collider 2D, що дозволить об'єктам на рівні мати фізичні властивості та взаємодіяти з іншими об'єктами, що мають колайдери.

Використовуючи раніше створену Tile Palette, створюю рівень.

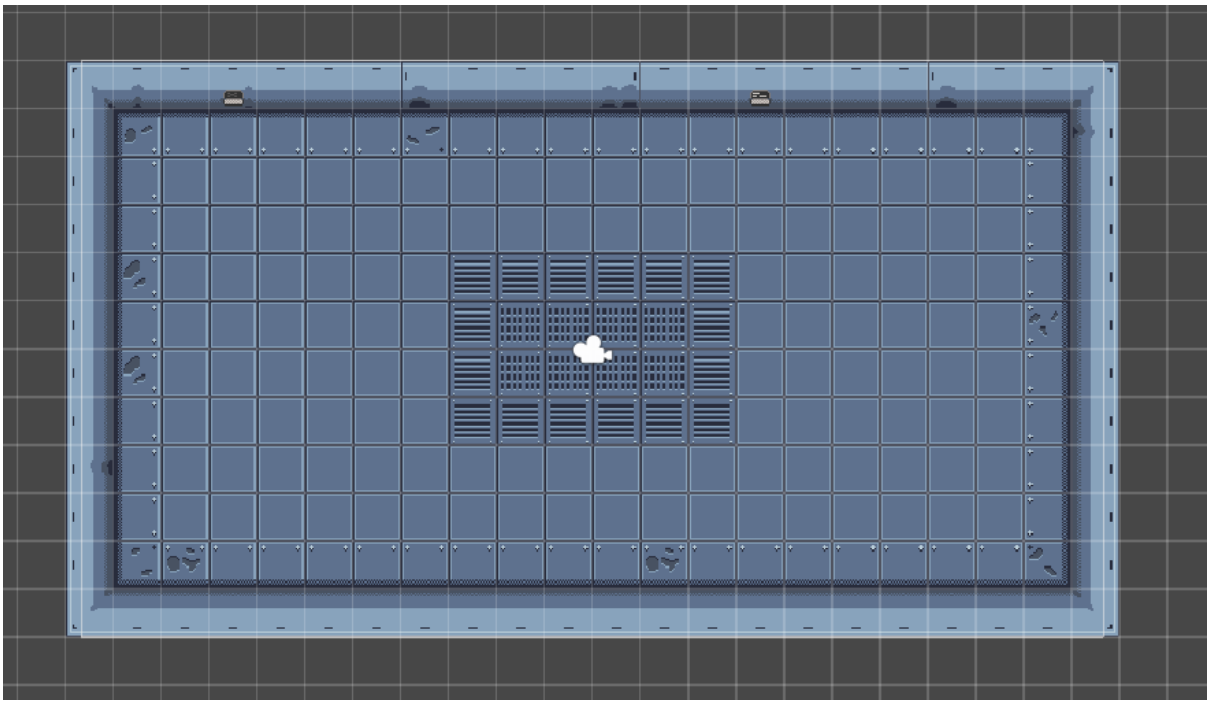


Рисунок 1.24 Tilemap "Indestructible"

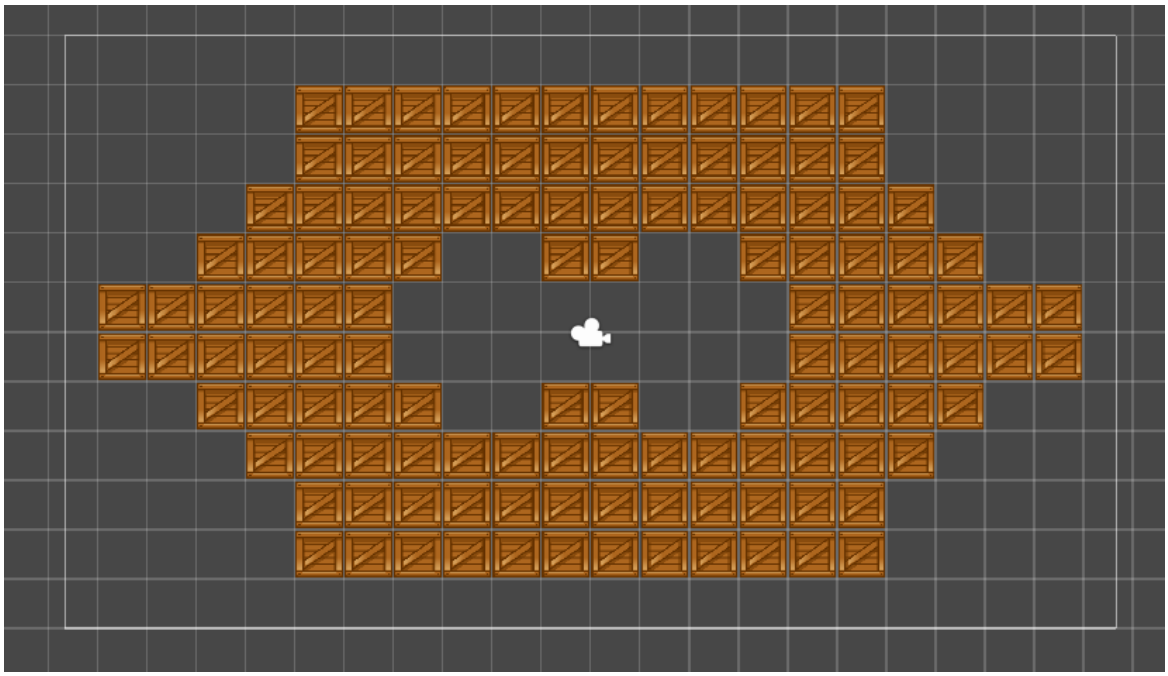


Рисунок 1.25 Tilemap "Destructible"

У створеній Tile Palette у спрайтів, що представляють стіни та ящики, встановлено тип колайдера "Sprite". Це забезпечить точну фізичну взаємодію цих

об'єктів з іншими елементами на рівні.

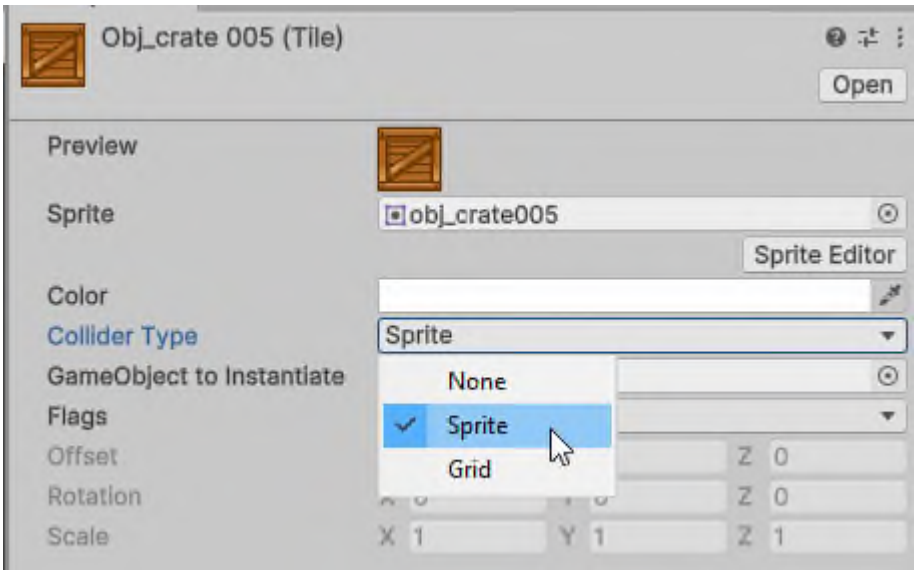


Рисунок 1.26 Зміна типу колайдера

1.7.3 Створення ігрового персонажа

Створимо новий порожній об'єкт GameObject на сцені та назвемо його «Player». Для реалізації фізики, руху й анімації додамо на нього такі компоненти:

Rigidbody 2D - дає змогу гравцеві взаємодіяти з фізичним світом гри, реагувати на сили та зіткнення.

Circle Collider 2D - визначає форму гравця для розрахунку зіткнень.

Animator - дасть змогу створювати та відтворювати анімації гравця, такі як рух, смерть тощо.

Рисунок 1.27 Компоненти об'єкта - «Player»

Тепер потрібно створити 5 анімації для гравця:

1. анімація спокою (Idle)

2.Ходьба вгору (WalkUp)

3.Ходьба вниз(WalkDown)

4.Ходьба вліво(WalkLeft)

5.Ходьба вправо(WalkRight)

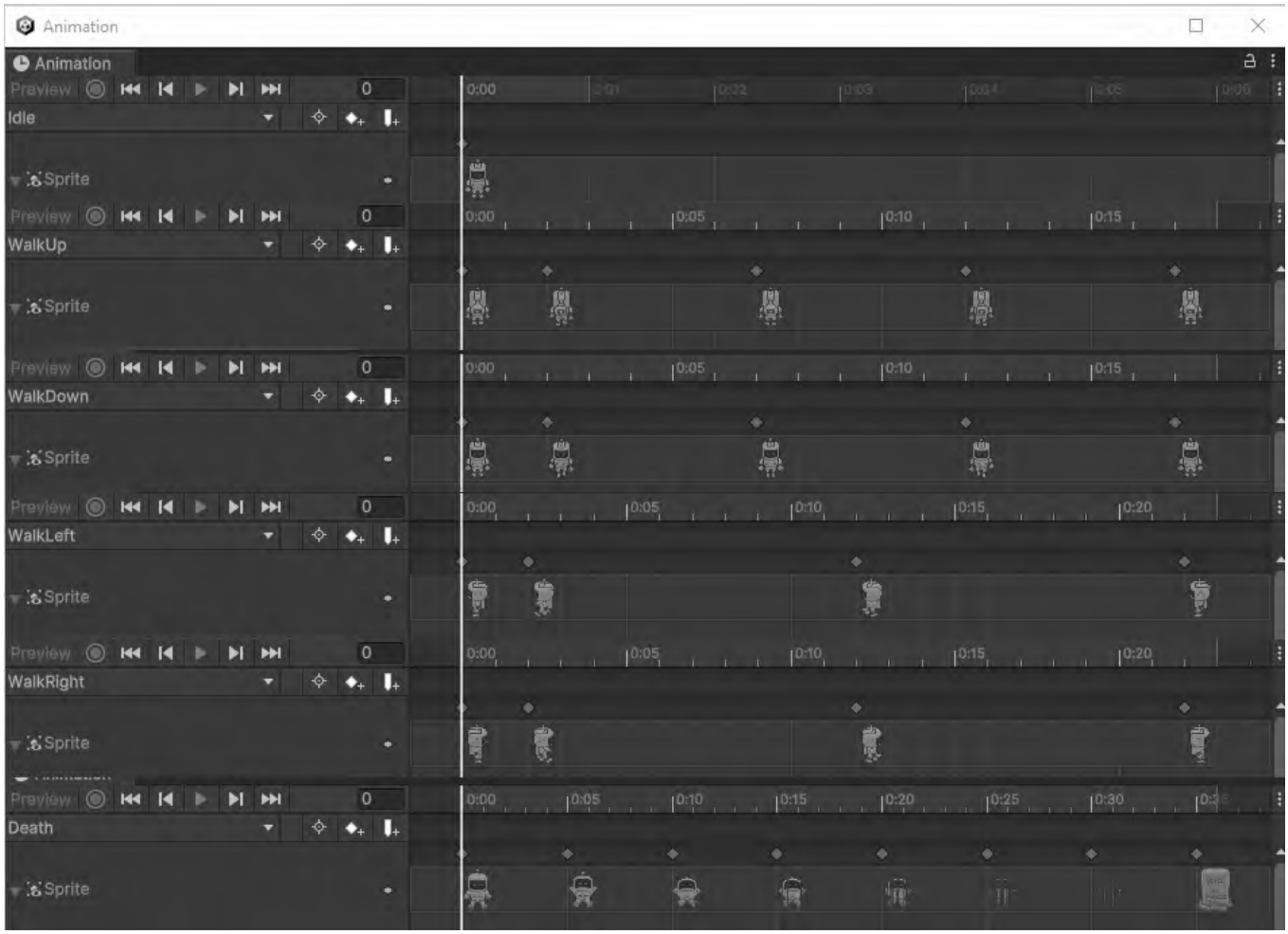


Рисунок 1.28 Анімації гравця

Перейдемо до налаштування аніматора. Додамо такі параметри:

- Vertical (float): Вертикальна складова руху (вгору/вниз).
- Horizontal (float): Горизонтальна складова руху (вліво/вправо).
- Speed (float): Швидкість руху персонажа.
- Rip (int): Цілочисельний параметр, який буде використовуватися для визначення стану смерті персонажа.

В аніматорі створимо Blend Tree, обравши тип 2D Simple Directional. Цей тип Blend Tree дозволяє плавно змішувати анімації залежно від напрямку руху персонажа.

Призначимо раніше створені параметри Vertical і Horizontal у Blend Tree та налаштуємо напрямки руху. Наприклад, вектор (0, 1) відповідатиме руху вгору, (0, -1) - руху вниз, (-1, 0) - руху вліво, (1, 0) - руху вправо.

Далі перемістимо раніше підготовлені анімації на відповідні їм вектори напрямку у Blend Tree. Таким чином, аніматор зможе автоматично вибирати та

плавно змішувати анімації залежно від значень параметрів Vertical та Horizontal, що визначають напрямок руху персонажа.

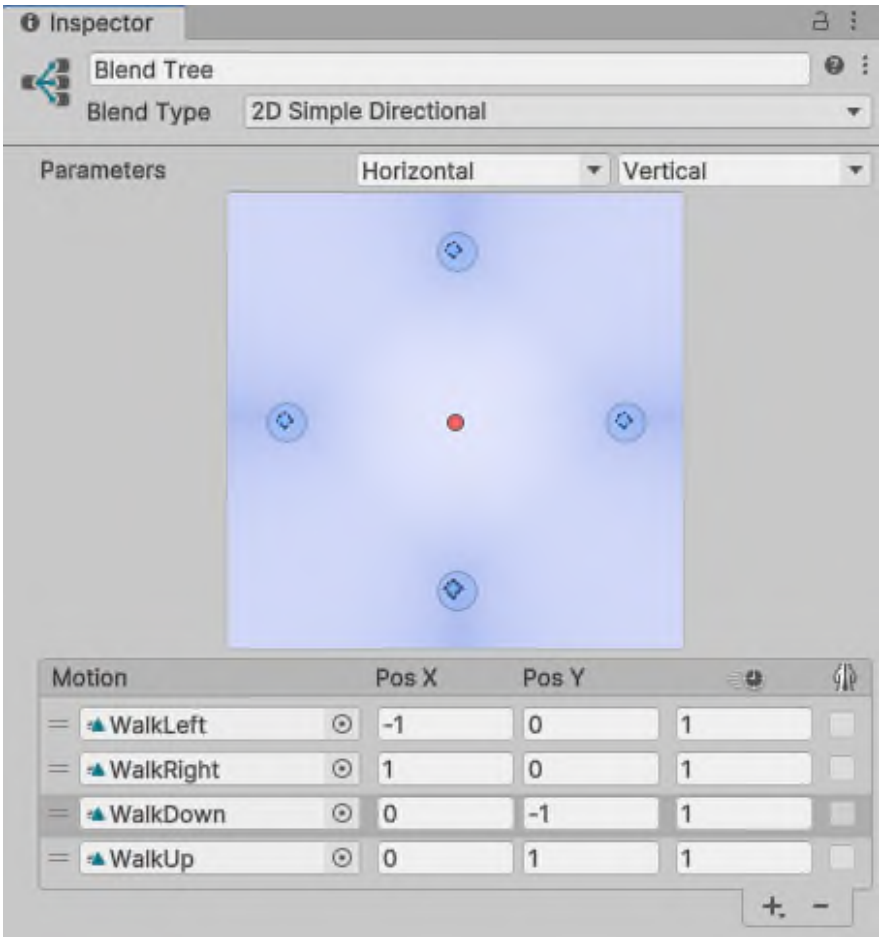


Рисунок 1.29 Blend Tree

Умови переходу:

- Idle -> Blend Tree: перехід від анімації Idle до Blend Tree з умовою Speed > 0.1. Це означає, що коли швидкість персонажа перевищить 0.1, він почне рухатися, і анімація перейде до Blend Tree.
- Blend Tree -> Idle: перехід від Blend Tree до Idle з умовою Speed < 0.1. Це означає, що коли швидкість персонажа стане меншою за 0.1, він зупиниться, і анімація повернеться до Idle.
- Any State -> Rip: перехід від будь-якого стану (Any State) до анімації Rip з умовою Rip == 1. Це означає, що коли параметр Rip буде встановлено в 1 (при смерті персонажа), анімація перейде до Rip незалежно від поточного стану.

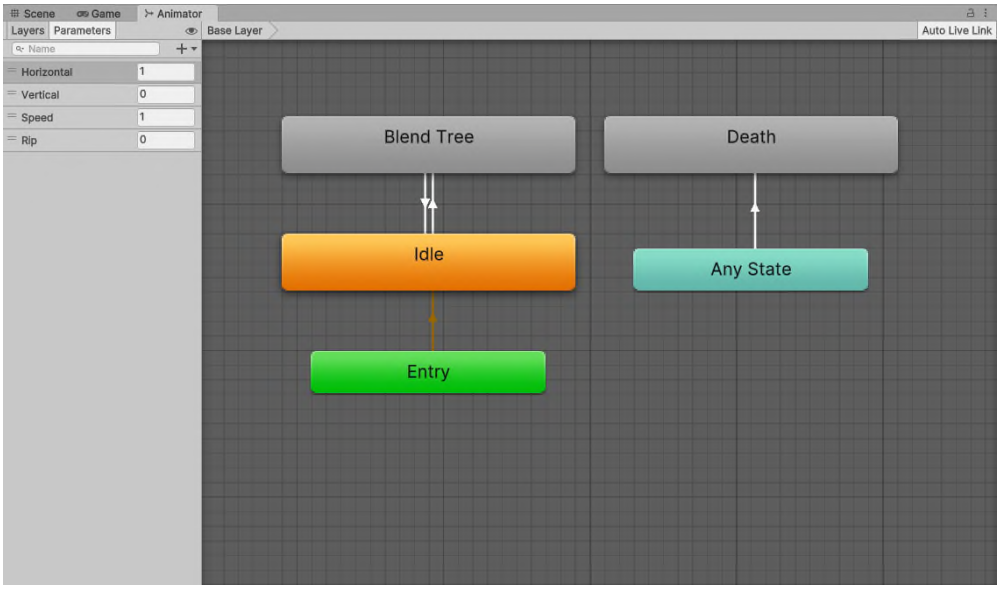


Рисунок 1.30 Animator

Тепер для керування рухом персонажа та анімаціями створю скрипт PlayerController і додамо його до об'єкта гравця.

Наступний код відповідатиме за перемикання анімацій а також зберігатиме напрямок персонажа:

```
private void SetDirection(Vector2 newDirection)
{
   animator.SetFloat("Horizontal",newDirection.x);
   animator.SetFloat("Vertical", newDirection.y);
   animator.SetFloat("Speed", newDirection.sqrMagnitude);
   direction = newDirection;
}
```

Також щоб була можливість синхронізації анімації та пересування додам компонент Photon View до об'єкта гравця.

Наступним кодом реалізую логіку пересування персонажа:

```
private void Update()
{
   if (view.IsMine)
   {
      if (joystick.Vertical > 0.7)
      {
```

```
            SetDirection(Vector2.up);
        }
        else if (joystick.Vertical < -0.7)
        {
            SetDirection(Vector2.down);
        }
        else if (joystick.Horizontal < -0.7)
        {
            SetDirection(Vector2.left);
        }
        else if (joystick.Horizontal > 0.7)
        {
            SetDirection(Vector2.right);
        }
        else
        {
            SetDirection(Vector2.zero);
        }
    }
    else
        SmoothNetMovement();
}

private void FixedUpdate()
{
    Vector2 targetVelocity = direction * speed;
    Vector2 newPosition = (Vector2)transform.position + targetVelocity *
Time.fixedDeltaTime;
    // Обновляем позицию персонажа
    transform.position = newPosition;
```

```
}
```

Для синхронізації використовую метод OnPhotonSerializeView, який відіграє ключову роль у синхронізації стану гравця між клієнтами у Photon PUN.

```
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if (stream.IsWriting)
        {
            stream.SendNext(transform.position);
            stream.SendNext(animator.GetFloat(«Horizontal»));
            stream.SendNext(animator.GetFloat(«Вертикаль»));
stream.SendNext(animator.GetFloat(«Вертикаль»));
            stream.SendNext(animator.GetFloat(«Speed»));
        }
        else
        {
            selfpos = (Vector3)stream.ReceiveNext();
            animator.SetFloat(«Горизонталь», (float)stream.ReceiveNext());
            animator.SetFloat(«Vertical», (float)stream.ReceiveNext());
            animator.SetFloat(«Speed», (float)stream.ReceiveNext());
        }
    }
```

Метод OnPhotonSerializeView автоматично викликається Photon PUN для кожного компонента PhotonView, що прикріплено до об’єкта. Він відповідає за серіалізацію (запис) та десеріалізацію (зчитування) даних гравця, щоб гарантувати їхню синхронну передачу між усіма клієнтами.

Щоб Photon міг розгорнути екземпляри гравця на всіх підключених клієнтах за допомогою PhotonNetwork.Instantiate, потрібен відповідний префаб. Для цього у проекті Unity створіть папку **Resources**, а потім перетягніть у неї об’єкт “Player” із ієрархії сцени. Після цього Unity автоматично згенерує префаб “Player” у папці

Resources, і Photon зможе використовувати його для створення однакових копій цього об’єкта на всіх клієнтах, забезпечуючи коректну мережеву синхронізацію та взаємодію.

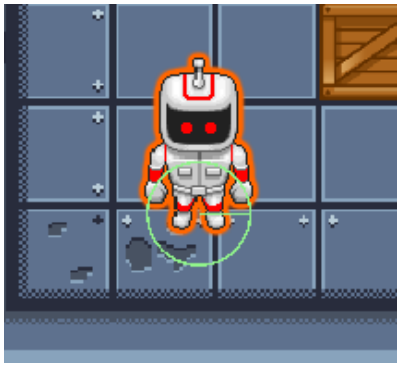


Рисунок 1.31 Префаб гравця на сцені

1.7.4 Реалізація ігрових механик

Створимо префаб бомби аналогічно до того, як це було зроблено з гравцем: після налаштування бомби на сцені, з усіма потрібними компонентами, зберігаємо її як префаб і поміщаємо у папку **Resources** для подальшого динамічного завантаження під час гри. До бомби обов’язково додаємо компонент **Photon View**, що дозволяє синхронізувати її стан між усіма гравцями в мережевій грі. Завдяки цьому кожен гравець бачитиме появу, активацію та вибух бомби в реальному часі, що є критично важливим для коректної роботи мультиплеєра.

Рисунок 1.32 Префаб бомби на сцені

Щоб реалізувати поведінку бомби, створюється скрипт PhotonBomb**,** який додається до префабу бомби. Цей скрипт відповідає за повний життєвий цикл бомби: він керує її активацією після розміщення, запускає таймер затримки перед вибухом, ініціює сам вибух із подальшою генерацією вибухової хвилі у всіх напрямках, а також обробляє зіткнення з об'єктами на сцені. У випадку, якщо об’єкти підпадають під зону ураження, скрипт забезпечує їхнє знищення або приховання, залежно від типу — зруйновані стіни, вороги чи інші ігрові

елементи. Таким чином реалізується базова механіка бомбардування в стилі класичних аркадних ігор.

```
public IEnumerator ActiveBomb()
{
    Vector2 position;
    yield return new WaitForSeconds(bombFuseTime - 0.1f);
    position = transform.position;
    position.x = Mathf.Round(position.x);
    position.y = Mathf.Round(position.y);
    Explosion explosion = Instantiate(explosionPrefab, position, Quaternion.identity);
    explosion.SetActiveRenderer(explosion.start);
    explosion.DestroyAfter(explosionDuration);
    Explode(position, Vector2.up, explosionRadius);
    Explode(position, Vector2.down, explosionRadius);
    Explode(position, Vector2.left, explosionRadius);
    Explode(position, Vector2.right, explosionRadius);
    Debug.Log("PhotonBomb");
    Destroy(gameObject);
}
```

Метод ActiveBomb() запускає корутину, яка відповідає за затримку перед детонацією бомби. Спочатку очікується час bombFuseTime мінус 0.1 секунди, щоб показати ефект запалювання, а потім бомба вибухає у своїй позиції. На сцені створюється інстанс explosionPrefab та встановлюється explosionRadius, після чого викликається метод Explode(), що розповсюджує вибух у чотирьох напрямках і перевіряє, чи є поруч об’єкти, які потрібно зруйнувати. Після виконання вибуху сам об’єкт бомби видаляється.

Для управління встановленням бомб гравцем створений скрипт BombController, який додається до об’єкта персонажа. Він відстежує натискання кнопки встановлення бомби й при потребі створює її екземпляр на всіх клієнтах,

водночас контролюючи, щоб гравець не перевищував ліміт доступних бомб. Крім того, BombController налаштовує параметри кожної створеної бомби (радіус вибуху, тривалість вибуху, час, поки горить запал) і активує її корутину ActiveBomb(), щоб запустити процес детонації.

```
[PunRPC]
    void RPC_ActiveBomb(int bombViewID)
    {
        // Находим объект бомбы по его ViewID
        PhotonView bombView = PhotonView.Find(bombViewID);
        if (bombView != null)
        {
            bomb = bombView.gameObject;
            photonBomb = bomb.GetComponent<PhotonBomb>();
            if (photonBomb != null)
            {
                photonBomb.explosionRadius = explosionRadius;
                photonBomb.explosionDuration = explosionDuration;
                photonBomb.bombFuseTime = bombFuseTime;
                StartCoroutine(photonBomb.ActiveBomb());
            }
            else
            {
                Debug.LogError("PhotonBomb component not found on bomb");
            }
        }
        else
        {
            Debug.LogError("Bomb View not found");
        }
    }
```

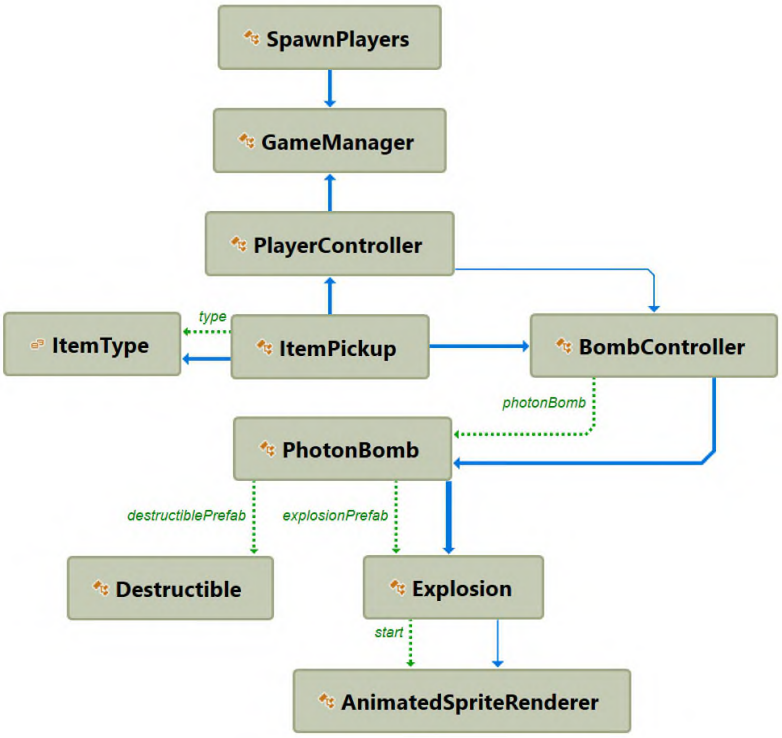


Рисунок 1.33 Діаграма класів, що відповідають за реалізації ігрової логіки

Метод RPC_ActiveBomb виконує налаштування та запускає корутину ActiveBomb() компонента PhotonBomb, яка відповідає за активацію та детонацію бомби.

1.7.5 Керування персонажем

Управління персонажем у грі здійснюється за допомогою віртуального джойстика та кнопки для встановлення бомб.

Рисунок 1.34 Віртуальний джойстик

Рисунок 1.35 Кнопка встановлення бомби

1.7.6 Тестування

Функціональне тестування – це процес перевірки функціональності гри для підтвердження того, що вона працює згідно з визначеними вимогами та очікуваннями. Метою функціонального тестування є перевірка того, що всі ігрові механіки та функції працюють коректно.

Таблиця 1.1 Функціональне тестування

№ Тесту	Дія тестувальника	Очікуваний результат	Проходження тесту
1	Запуск гри	Успішний запуск гри, відображення головного меню.	Так
2	Створення та приєднання до кімнати	Успішне створення кімнати з заданими параметрами або приєднання до існуючої кімнати.	Так
3	Рух персонажа за допомогою віртуального джойстика	Плавний та коректний рух персонажа у всіх напрямках відповідно до дій гравця.	Так
4	Встановлення бомби	Успішне встановлення бомби у позиції гравця. Початок відліку часу до вибуху.	Так
5	Вибух бомби	Вибух бомби через заданий час. Знищення руйнівних об'єктів у радіусі вибуху. Поява бонусів з деяких знищених ящиків.	Так
6	Збір бонусів	Збільшення сили вибуху, кількості бомб або швидкості пересування персонажа після збору відповідного бонусу.	Так
7	Смерть персонажа	Активація анімації смерті персонажа після знищення вибухом або зіткнення з вогнем. Видалення персонажа з гри.	Так
8	Перевірка синхронізації між гравцями	Позиції, анімації та стани персонажів синхронізовані. Відсутність помітних затримок або розсинхронізації під час гри.	Так
9	Перевірка роботи елементів інтерфейсу	Усі елементи інтерфейсу відображаються коректно та реагують на дії користувача.	Так

Ця таблиця демонструє основні функціональні тести, проведені під час розробки гри. Усі тести пройшли успішно, що підтверджує коректну роботу реалізованих функцій та механік

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

Цей дипломний проект присвячений створенню мобільної 2D-гри для платформи Android із застосуванням рушія Unity. Гра пройшла повне функціональне тестування, яке підтвердило правильність роботи основних механік: руху персонажів, встановлення й вибуху бомб, збору бонусів, мережевої синхронізації та інтеграції з користувацьким інтерфейсом. Прототип успішно протестовано на різних Android-пристроях, що підтверджує його стабільність і сумісність.

У цьому розділі ми представляємо розрахунок вартості розробленого програмного забезпечення (ПЗ).

Ефективність ПЗ залежить не лише від його якісних характеристик, а й від ефективності процесу розробки. Якість ПЗ оцінюється за такими критеріями: наскільки ПЗ відповідає їхнім очікуванням, ефективність, з якою ПЗ використовує системні ресурси, чи відповідає ПЗ встановленим стандартам та специфікаціям. З точки зору користувача, важливою складовою оцінки якості є витрати на розробку, зокрема обсяг залучених трудових ресурсів та загальна вартість створення продукту.

2.2 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки залежить від ряду факторів, включаючи обсяг робіт, рівень складності, кваліфікацію команди розробників та часові обмеження, зумовлені ринковими умовами. Для оцінки масштабу програмного продукту використовується метод структурної аналогії, що передбачає звернення до спеціалізованих каталогів схожих проєктів. На їхній основі визначається кількість умовних машинних команд, необхідних для реалізації аналогічного програмного рішення (у тисячах команд).

Таблиця 2.1 Каталог аналогів

Найменування ПП	**Обсяг функції ПП – V_o, умовн. машинних командах.**
1. ПП автоматизації засобів по каталогу	680 – 7000
2. ПП автоматизованих розрахунків	1300 – 8600
3. ПП оптимізації розрахунків	1300 – 4200

У таблиці 2.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт. Для нашого варіанта виділено сірим кольором.

Вибравши аналог ПЗ, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл.2.2

Таблиця.2.2 Норма часу

Обсяг ПЗ, тис.умов.машинних команд	**Норма часу, люд/год**
1.00	229
2.00	244
3.00	262

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): Тар = 229 х 0,7 = 160,3 (люд/годин).

Трудомісткість програмного продукту визначається окремо для кожного етапу розробки, виходячи з показників трудомісткості відповідного аналога. При цьому враховують рівень складності розробки, ступінь інноваційності та частку використання стандартних модулів. Розрахунки проводяться згідно з наступними формулами:

$$T_{ТЗ} = T^a p \times L_1 \times K_H \qquad (2.1)$$

$$T_{ТП} = T^a p \times L_2 \times K_H \qquad (2.2)$$

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \qquad (2.3)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 2.3.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.4.);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.5)

Таблиця 2.3 Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП

Код стадії	**Ступінь новизни**		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Для нашого варіанта виділено сірим кольором.

Таблиця 2.4 Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	**Ступінь новизни**	**Значення $К_н$**
А	Принципово нові ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП маючий аналог	0,7

Для нашого варіанта виділено сірим кольором.

Таблиця 2.5 Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	**Значення $К_т$**
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Для нашого варіанта виділено сірим кольором.

Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$T_{тз}=T^a * L_1 * K_н = 160,3 * 0,12 * 0,7 = 13,46 \text{ (люд/годин)} \qquad (2.4)$$

Трудомісткість розробки технічного проекту

$$T_{тп}=T^{а} *L_2*К_н =160,3 \ *0,11*0,7 = 12,34 \text{ (люд/годин)} \quad (2.5)$$

Трудомісткість розробки робочого проекту

$$T_{рп}=T^{а} *L_3*К_н*К_т=160,3 \ *0,61*0,7*0,6= 41,07 \text{ (люд/годин)} \quad (2.6)$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}$= 2 (стр), розробка ТП $N_{тп}$=25 (стр), розробка робочого проекту $N_{рп}$=7(стр), пояснювальна записка відповідно $N_{пз}$=22(стр) Розрахунок зведений у таблицю 2.6.

Таблиця 2.6 Розрахунок трудомісткості ПП

Найменування етапів	**Розрахунок, годин**		
1.ТЗ	$T_{Ртз}$=13,46	$T_{кк}=0,7*N_{тз}=0,7*3=2,1$	$T_{нк}=0,15*N_{тз}=0,15*3=0,45$
2.Розробка ТП	$T_{Ртп}$=12,34	$T_{кк}=0,7*N_{тп}=0,7*25=17,5$	$T_{нк}=0,15*N_{тп}=0,15*25=3,75$
3.Розробка РП	$T_{Ррп}$= 41,07	$T_{кк}=0,7*N_{рп}=0,7*7=4,9$	$T_{нк}=0,15*N_{рп}=0,15*7=1,05$
4.Розробка ПЗ	$T_{пз}=1,5*N_{пз}=1,5*22=33$	$T_{кк}=0,7*N_{тз}=0,7*22=15,4$	$T_{нк}=0,15*N_{пз}=0,15*22 =3,3$
Усього, в т.ч.:	148,32		
- на розробку	$\Sigma T_р$=99,87		
- контроль керівника		$\Sigma T_{кк}$=39,9	
- нормоконтроль			$\Sigma T_{нк}$=8,55

2.3 Розрахунок ціни програмного продукту

Для оцінки вартості програмного продукту розглядаються основна заробітна плата виконавців, матеріальні витрати та загальні витрати на розробку ПЗ. Детальний розрахунок основної заробітної плати наведено у таблиці 2.7. Згідно зі статтею 8 «Закону про Державний бюджет України на 2025» з 1 січня 2025 року встановлено мінімальну місячну заробітну плату у розмірі 8000 гривень, а також мінімальну погодинну тарифну ставку – 48,00 грн.

Таблиця 2.7 Розрахунок основної заробітної плати виконавців

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн.	Розрахунок, грн.
1.Розробка ПП	99,87	48,00	4793,76
2.Контроль керівника	39,9	95,00	3790,50
3.Нормоконтроль	8,55	90,00	769,5
Усього	-	-	ΣЗо= 9353,76

Зробимо розрахунок матеріальних витрат на розробку ПП (табл.2.8).

Таблиця 2.8 Розрахунок матеріальних витрат на розробку ПЗ

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	57	5.0	285,00
Разом	-	-	-	$В_{м\,і}$ =285,00
Транспортно – заготівельні Витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$ = 0,1*285=28,5
Усього				$В_м=В_{мі}+В_{тр\_з=}$ 313,50

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 2.9.

Таблиця 2.9 Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	313,5	$В_м$ (див. табл. 2.8.)
2. Основна заробітна плата	9353,76	$З_о$ (див. табл. 2.7.)
3.Додаткова заробітна плата	935,37	$Зд = 0{,}1 \times Зо$ = 9353,76*0,1
4.Відрахування до єдиного фонду соціального внеску	2263,61	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$ =0,22* (9353,76+935,37)
5. Накладні витрати	3741,5	$Внак. = 0{,}4 \times Зо$ =0,4 * 9353,76
6. Повна собівартість	16607,74	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.= 313,5+9353,76+935,37+2263,61 +3741,5

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

$П=(С_{пов}*Р)/100=(16607,74*15)/100=2491,16$ грн (2.7)

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$Ц_о = С_{пов}+П$ = 16607,74+2491,16 = 19098,9 грн (2.8)

Ціна реалізації розробленого програмного забезпечення становитиме:

$Ц_р=Ц_о+ПДВ$ = 19098,9 + 19098,9 *0.2 = 22918,68 грн; (2.9)

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Основним нормативно-правовим актом, що регулює відносини у сфері охорони праці, є Закон України "Про охорону праці" : «Цей Закон визначає основні положення щодо реалізації конституційного права працівників на охорону їх життя і здоров'я у процесі трудової діяльності, на належні, безпечні і здорові умови праці, регулює за участю відповідних органів державної влади відносини між роботодавцем і працівником з питань безпеки, гігієни праці та виробничого середовища і встановлює єдиний порядок організації охорони праці в Україні» Дотримання його положень є обов'язковим для всіх учасників процесу.

3.1 Вимоги до організації робочого місця працівника

Основні вимоги до приміщення: мінімальна площа на одну особу має бути 6 м², що гарантує достатній простір для продуктивної роботи; об’єм приміщення не менше 20 м³, що сприяє належній циркуляції повітря та зменшує ризик накопичення шкідливих речовин; висота приміщення становить 2.5 м та більше, що дозволяє забезпечити оптимальні умови освітлення та розміщення обладнання; вентиляційна система включає природну та примусову вентиляцію, що гарантує стабільний рівень повітрообміну, видалення зайвого тепла від електронної техніки та підтримку комфортного мікроклімату.

Основні вимоги до робочого середовища: освітленість – 300–500 лк (змішане освітлення: природне + штучне); мікроклімат – температура 20–24°C, вологість 40–60%, забезпечені системою вентиляції та кондиціонування; шум – рівень ≤ 50 дБ, що відповідає нормам для комфортної розумової діяльності; електромагнітне випромінювання – рівень OLED-дисплея та Arduino не перевищує допустимих норм, що гарантує безпеку для користувача.

3.2 Ергономіка робочого місця

Освітлення робочого місця: оптимальне освітлення сприяє зменшенню напруги очей, тому робоче місце повинно мати достатнє, рівномірне та несліпуче освітлення. При цьому необхідно уникати прямих сонячних променів і відблисків

від освітлювальних приладів на екрані монітора ПК та на дисплеї пристрою моніторингу.

Розташування монітора ПК: Монітор ПК повинен бути розташований на відстані 60-70 см від очей користувача. Верхній край екрана має знаходитись на рівні очей чи трохи нижче, щоб уникнути надмірного нахилу голови, під кутом 90-100 градусів до вікон, забезпечуючи бокове падіння світла.

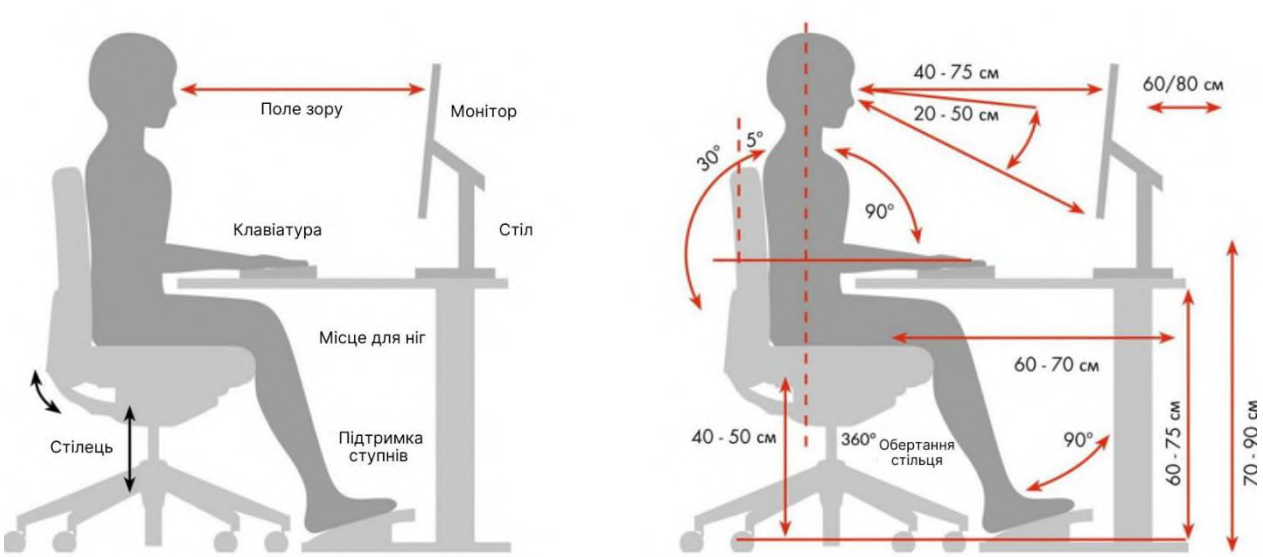


Рисунок 3.1 Правильне робоче місце

Розташування пристрою моніторингу: Дисплей пристрою діагностики (OLED-дисплей) має бути розміщений у зручному для спостереження місці, яке не вимагає зайвих рухів голови чи очей, забезпечуючи швидке візуальне зчитування даних "з першого погляду". Важливо, щоб він не було додаткових відблисків.

Ергономіка робочого столу: робочий стіл повинен мати достатні розміри для розміщення комп'ютера, пристрою моніторингу, клавіатури, миші та необхідних інструментів, забезпечуючи свободу рухів. Бажано, щоб стіл мав можливість регулювання висоти (в межах 680-800 мм), що дозволяє адаптувати його під індивідуальні особливості користувача.

Ергономіка робочого стільця: Робочий стілець має бути оснащений підйомно-поворотним пристроєм для регулювання висоти сидіння та спинки, а також

кута її нахилу. Спинка стільця повинна забезпечувати надійну опору для попереку, а сидіння — бути зручним. Наявність підлокітників, що регулюються, є бажаною для підтримки рук.

Правильна робоча поза: При роботі за комп'ютером та з електронними компонентами важливо підтримувати правильну робочу позу. Спина повинна бути прямою, опираючись на спинку крісла. Ноги повинні стояти на підлозі чи на спеціальній підставці під кутом приблизно 90%. Руки повинні бути розслаблені, а зап'ястя - прямими, розташовані на одному рівні з клавіатурою.

Розташування клавіатури та миші: Клавіатуру слід розташовувати на поверхні столу на відстані 100-300 мм від краю, зверненого до працюючого. Миша повинна знаходитись поруч з клавіатурою, забезпечуючи мінімальне напруження для руки.

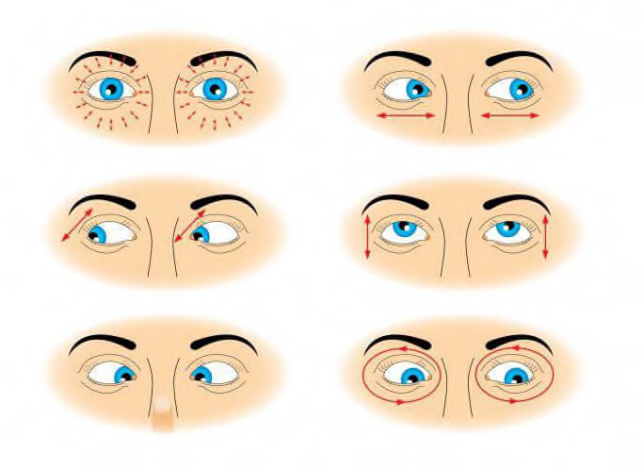


Рисунок 3.2 Гімнастика для очей

Регулярні перерви: Необхідно робити короткі, регулярні перерви (кожні 45-60 хвилин роботи — 5-10 хвилин відпочинку). Під час перерв рекомендується виконувати вправи для очей, легку розминку, зміну положення тіла для зняття напруги з м'язів.

3.3 Пожежна безпека

Забезпечення пожежної безпеки приміщень, де є електричні мережі та електронне обладнання, регламентується Правилами пожежної безпеки в Україні. Хоча

пристрій моніторингу використовує низькі напруги, ризик виникнення пожежі через короткі замикання, перегрів компонентів та несправність обладнання завжди існує. Робоче місце, де виконуються роботи з електронною технікою, зазвичай відповідає категорії Д пожежної безпеки (приміщення, де знаходяться негорючі речовини та матеріали в холодному стані).

Як запобігти пожежі:

Контролювати технічний стан компонентів: регулярно проводити візуальний огляд усіх компонентів пристрою (Arduino Nano, OLED-дисплея, проводів) на предмет можливих пошкоджень, ознак перегріву, оплавлення чи нетипового запаху. У разі виявлення таких ознак, пристрій слід негайно відключити від живлення.

Уникати перевантажень: слідкувати за відповідністю споживаної потужності пристрою можливостям джерела живлення (USB-порту ПК). Забороняється підключати до мікроконтролера Arduino та USB-порту ПК споживачі, які перевищують максимально допустимий струм, оскільки це може призвести до перегріву та загоряння.

Дотримуватися чистоти на робочому місці: тут повинно не повинно бути сторонніх предметів, пилу та сміття, оскільки накопичення пилу на електронних компонентах може призвести до порушення тепловідведення, перегріву та, як наслідок, до коротких замикань чи загоряння.

Не зберігати горючі матеріали: заборонено зберігати легкозаймисті та горючі матеріали (папір, тканина, аерозолі, рідини) безпосередньо поблизу працюючого електронного обладнання чи у зоні можливого перегріву.

Відключати обладнання від мережі: відключати все електрообладнання від мережі, залишаючи робоче місце навіть на короткий час.

Оснащення первинними засобами пожежогасіння та пожежною сигналізацією: на робочому місці чи в безпосередній близькості повинні бути доступні первинні засоби пожежогасіння. Для гасіння електроустановок, що знаходяться під напругою (до 1000 В), слід використовувати вуглекислотні (ВВ) чи порошкові (ВП)

вогнегасники. Забороняється використовувати водні та пінні вогнегасники для гасіння електрообладнання, оскільки це може призвести до ураження електричним струмом. У приміщеннях, де розташоване робочі місця, бажано наявність установки автоматичної пожежної сигналізації.

Рисунок 3.3 Вогнегасник порошковий

Отже, розробка та використання пристрою моніторингу комп’ютера не пов’язана з підвищеним рівнем ризику за умови дотримання норм техніки безпеки, ергономіки та санітарно-гігієнічних вимог. Всі види робіт виконувалися у відповідних умовах, що дозволяє вважати проєкт безпечним для розробника та користувача.

ВИСНОВКИ

У ході виконання дипломного проєкту було досягнуто основної мети – розроблено мобільну 2D-гру для платформи Android із використанням сучасних інструментів і технологій, зокрема Unity, мови програмування C# та Photon PUN. Цей проєкт наочно демонструє ефективність таких технологій для створення якісних та захопливих мобільних ігор.

У процесі розробки було проаналізовано жанрову різноманітність мобільних ігор, а також актуальні тенденції на ринку, що дозволило виявити зростаючий інтерес користувачів до багатокористувацьких ігор з активним, динамічним ігровим процесом. З огляду на ці результати було обрано концепцію створення мультиплеєрної гри у жанрі аркадного екшену. Дослідження існуючих рішень допомогло визначити ключові елементи ігрової механіки, необхідні для забезпечення привабливості та конкурентоспроможності проєкту.

Було проведено детальне проєктування ігрових елементів: розроблено правила гри, логіку поведінки персонажів, систему взаємодії з об’єктами та бомбами. Також створено зручний та інтуїтивно зрозумілий графічний інтерфейс, оптимізований для сенсорного керування на мобільних пристроях. Завдяки інтеграції Photon PUN вдалося реалізувати надійне мережеве з’єднання між гравцями та забезпечити синхронізовану взаємодію в реальному часі.

Гра пройшла повне функціональне тестування, яке підтвердило правильність роботи основних механік: руху персонажів, встановлення й вибуху бомб, збору бонусів, мережевої синхронізації та інтеграції з користувацьким інтерфейсом. Прототип успішно протестовано на різних Android-пристроях, що підтверджує його стабільність і сумісність.

Створена гра має потенціал для подальшого розвитку: можна додати нові рівні, персонажів, режими гри та розширені функції, що сприятиме її вдосконаленню й популяризації серед користувачів.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Васильєв, М. П. Unity 2021. Розробка ігор для початківців / М. П. Васильєв. – Київ : Діалектика, 2021. – 328 с.
2. Коваленко, І. І. Основи розробки мобільних ігор: Android + Unity / І. І. Коваленко. – Харків : ФОП Бровін О. В., 2022. – 276 с.
3. Шевченко, Л. П. Програмування мовою C# в середовищі Unity / Л. П. Шевченко. – Львів : Новий Світ, 2023. – 192 с.
4. Андрущенко, Т. В. Інженерія програмного забезпечення: інструменти та середовища / Т. В. Андрущенко. – Київ : Ліра-К, 2020. – 356 с.
5. Юрченко, О. Б. Мережеві технології в мобільній розробці: Photon, PUN2 / О. Б. Юрченко. – Чернівці : Техносфера, 2023. – 198 с.
6. Unity Technologies. Unity Manual. Documentation Version 2022.3 LTS [Електронний ресурс]. – URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 12.06.2025).
7. Gigantic Duck Games. Bombergrounds: Reborn – Game Design and Mechanics [Електронний ресурс]. – URL: https://bombergrounds.com/ (дата звернення: 13.06.2025).
8. Microsoft. Photon Unity Networking 2 (PUN2) Documentation [Електронний ресурс]. – URL: https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro (дата звернення: 14.06.2025).
9. Schell, J. The Art of Game Design: A Book of Lenses / *Jesse Schell.* – 4th ed. – Boca Raton : CRC Press, 2023. – 600 p.
10. Gregory, J. Game Engine Architecture / Jason Gregory. – 4th ed. – Boca Raton : CRC Press, 2021. – 1240 p.

ДОДАТОК А. Лістинг програми

Файл UIManager.cs

```
using UnityEngine;
using UnityEngine.UI;

public class UIManager : MonoBehaviour
{
    [Header("Main Menu Panel")]
    [SerializeField] private GameObject MainMenuPanel;
    [Header("Join Lobby Panel")]
    [SerializeField] private GameObject JoinLobbyPanel;
    [SerializeField] private InputField joinRoomNameInputField;
    [Header("Create Lobby Panel")]
    [SerializeField] private GameObject CreateLobbyPanel;
    [SerializeField] private InputField createRoomNameInputField;
    [SerializeField] private Toggle isRoomVisible;
    [SerializeField] private Dropdown maxNumberPlayers;
    [Header("Other")]
    [SerializeField] private InputField nicknameInputField;
    [SerializeField] private PhotonNetworkManager photonManager;

    void Start()
    {
        string Nickname = PlayerPrefs.GetString("PlayerNickname", "");
        if (!string.IsNullOrEmpty(Nickname))
        {
            nicknameInputField.text = Nickname;
        }
        else
        {
            Nickname = "Player" + Random.Range(1, 1000);
            nicknameInputField.text = Nickname;
        }
        photonManager.SetPlayerNickname(Nickname);
    }
    public void OpenJoinLobbyPanel()
    {
        MainMenuPanel.SetActive(false);
        JoinLobbyPanel.SetActive(true);
        nicknameInputField.gameObject.SetActive(false);
    }

    public void CloseJoinLobbyPanel()
    {
        JoinLobbyPanel.SetActive(false);
        MainMenuPanel.SetActive(true);
        nicknameInputField.gameObject.SetActive(true);
    }

    public void OpenCreateLobbyPanel()
    {
        MainMenuPanel.SetActive(false);
        CreateLobbyPanel.SetActive(true);
        nicknameInputField.gameObject.SetActive(false);
    }

    public void CloseCreateLobbyPanel()
    {
        CreateLobbyPanel.SetActive(false);
        MainMenuPanel.SetActive(true);
        nicknameInputField.gameObject.SetActive(true);
    }
```

```
    public void ExitGameButton()
    {
        #if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
        #else
        Application.Quit();
        #endif
    }
    public void SaveNickname()
    {
        // Сохраняем никнейм игрока в PlayerPrefs
        PlayerPrefs.SetString("PlayerNickname", nicknameInputField.text);
        photonManager.SetPlayerNickname(nicknameInputField.text);
    }

    public void CreateRoomButton()
    {

photonManager.CreateRoom(int.Parse(maxNumberPlayers.options[maxNumberPlayers.value].
text),
                                isRoomVisible.isOn,
                                createRoomNameInputField.text);
    }

    public void RandomRoomButton()
    {
        photonManager.JoinRandomRoom();
    }
    public void ConnectToRoomButton()
    {
        photonManager.ConnectToRoom(joinRoomNameInputField.text);
    }
}
```

Файл PhotonManager.cs

```
using Photon.Pun;
using Photon.Realtime;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class PhotonNetworkManager : MonoBehaviourPunCallbacks
{
    [Header("Connect Info")]
    [SerializeField] private Text isConnected;
    [SerializeField] private Text Server;
    [Header("Update rooms")]
    [SerializeField] private ListItem itemPrefab;
    [SerializeField] private Transform content;
    List<RoomInfo> allRoomsInfo = new List<RoomInfo>();
    void Start()
    {
        PhotonNetwork.AutomaticallySyncScene = true;
        PhotonNetwork.GameVersion = "1.0";
        PhotonNetwork.ConnectUsingSettings();
    }
    public void CreateRoom(int maxNumberPlayers, bool isRoomVisible, string roomName)
    {
        RoomOptions roomOptions = new RoomOptions();
        roomOptions.MaxPlayers = maxNumberPlayers;
        roomOptions.IsVisible = isRoomVisible;
        roomOptions.EmptyRoomTtl = 0;
        PhotonNetwork.CreateRoom(roomName, roomOptions);
```

```
}
public void ConnectToRoom(string roomname)
{

    if (PhotonNetwork.IsConnected)
    {
        PhotonNetwork.JoinRoom(roomname);
    }
    else
    {
        Debug.Log("Not connected to Photon Master Server");
    }
}

public void SetPlayerNickname(string nickname)
{
    PhotonNetwork.NickName = nickname;
}
public override void OnConnectedToMaster()
{
    isConnected.text = null;
    Server.text = null;
    isConnected.text = "Connected to server";
    Server.text = "Connected to region: " + PhotonNetwork.CloudRegion;
    if (!PhotonNetwork.InLobby)
    {
        PhotonNetwork.JoinLobby();
    }
}
public override void OnDisconnected(DisconnectCause cause)
{
    if (isConnected != null && Server != null)
    {
        isConnected.text = null;
        Server.text = null;
        isConnected.text = "disconnected from server";
    }
}
public override void OnJoinRoomFailed(short returnCode, string message)
{
    Debug.Log("Join failed because" + message);
}
public override void OnCreatedRoom()
{
    Debug.Log("Room created, room name:" + PhotonNetwork.CurrentRoom.Name);
}

public override void OnCreateRoomFailed(short returnCode, string message)
{
    Debug.Log("failed to create a room" + message);
}
public void JoinRandomRoom()
{
    PhotonNetwork.JoinRandomRoom();
}
public override void OnJoinedRoom()
{
    PhotonNetwork.LoadLevel(1);
}

public override void OnRoomListUpdate(List<RoomInfo> roomlist)
{
    foreach (Transform child in content.transform)
    {
```

```
            Destroy(child.gameObject);
        }

        // Создаем новые элементы списка для каждой комнаты
        foreach (RoomInfo info in roomlist)
        {
            // Проверяем, что в комнате есть игроки
            if (info.PlayerCount > 0)
            {
                ListItem listItem = Instantiate(itemPrefab, content);
                if (listItem != null)
                {
                    listItem.SetInfo(info);
                }
            }
        }
    }
}
```

Файл PlayerController.cs

```
using Photon.Pun;
using UnityEngine;

public class PlayerController : MonoBehaviour, IPunObservable
{
    private Rigidbody2D rb;
    private Vector2 direction = Vector2.down;
    public float speed = 5f;
    [SerializeField] private Animator animator;
    private Joystick joystick;
    private PhotonView view;
    public float smoothTime = 0.3f;
    private Vector3 selfpos;

    private void Start()
    {
        joystick = GameObject.FindFirstObjectByType<Joystick>();
        rb = GetComponent<Rigidbody2D>();
        view = GetComponent<PhotonView>();
        PhotonNetwork.SerializationRate = 30;
        PhotonNetwork.SendRate = 30;

    }

    private void Update()
    {
        if (view.IsMine)
        {
            if (joystick.Vertical > 0.7)
            {
                SetDirection(Vector2.up);
            }
            else if (joystick.Vertical < -0.7)
            {
                SetDirection(Vector2.down);
            }
            else if (joystick.Horizontal < -0.7)
            {
                SetDirection(Vector2.left);
            }
            else if (joystick.Horizontal > 0.7)
            {
                SetDirection(Vector2.right);
            }
```

```
            else
            {
                SetDirection(Vector2.zero);
            }
        }
        else
            SmoothNetMovement();
    }

    private void FixedUpdate()
    {
        Vector2 targetVelocity = direction * speed;
        Vector2  newPosition  =  (Vector2)transform.position  +  targetVelocity  *
Time.fixedDeltaTime;

        // Обновляем позицию персонажа
        transform.position = newPosition;

    }

    private void SetDirection(Vector2 newDirection)
    {
        animator.SetFloat("Horizontal",newDirection.x);
        animator.SetFloat("Vertical", newDirection.y);
        animator.SetFloat("Speed", newDirection.sqrMagnitude);
        direction = newDirection;
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.layer == LayerMask.NameToLayer("Explosion"))
        {
            DeathSequence();
        }
    }
    private void DeathSequence()
    {
        enabled = false;
        GetComponent<BombController>().enabled = false;
        animator.SetInteger("Rip", 1);
        Invoke(nameof(OnDeathSequenceEnded), 1.25f);
    }
    private void OnDeathSequenceEnded()
    {
        gameObject.SetActive(false);
        if (PhotonNetwork.IsMasterClient) // Только мастер-клиент должен проверять
состояние игры
        {
            GameManager.Instance.CheckWinState();
        }
    }
    private void SmoothNetMovement()
    {
        transform.position = selfpos;
    }
    public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if (stream.IsWriting)
        {
            stream.SendNext(transform.position);

            stream.SendNext(animator.GetFloat("Horizontal"));
            stream.SendNext(animator.GetFloat("Vertical"));
            stream.SendNext(animator.GetFloat("Speed"));
```

ДОДАТОК Б. Слайди мультимедійної презентації

ВСТУП

Мобільні ігри стали невід'ємною частиною сучасного життя, надаючи розваги та соціальну взаємодію мільйонам користувачів по всьому світу. Ринок мобільних ігор демонструє стрімке зростання, а технології розробки постійно вдосконалюються, відкриваючи нові можливості для створення захопливих та інноваційних ігрових продуктів.

Метою даної дипломної роботи є розробка мобільної 2d гри під андроїд на платформі unity.

Для досягнення цієї мети було виконано такі **завдання**:

•Аналіз ринку мобільних ігор та існуючих розробок;

•Вибір оптимальних інструментів та технологій;

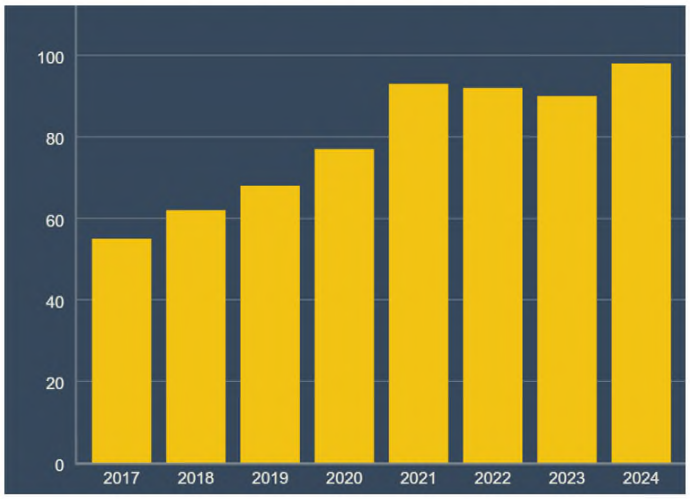
•Проєктування ігрової механіки;

•Розробка графічного інтерфейсу та системи керування;

•Реалізація мережевої взаємодії;

Ринок мобільних ігор

Зростання ринку мобільних ігор (млрд дол. США)



Найпопулярніші жанри

Порівняння движків

	Unity	Unreal Engine	Godot
Мова програмування	C#	C++	GDScript, C#, C++
Перший реліз	2005	1996	2014
Розробник платформи	Unity Technologies	Epic Games	The Godot Foundation
Підходить для	Мобільних ігор, 2D-проектів, інді-ігор, мультиплатформенних проектів	AAA-проектів, високобюджетних ігор, ігор з упором на графіку	2D-ігор, невеликих 3D-проектів, інді-ігор
Модулі та плагіни	Asset Store	Marketplace	Asset Library
Лицензия	Unity пропонує ліцензії Personal (безкоштовна, дохід до $200 тис.), Pro (дохід понад $200 тис.) та Industry (дохід понад $1 млн.).	Безкоштовна ліцензія але з роялті в розмірі 5 % для ігор, які заробили понад мільйон доларів США.	Повністю безкоштовний і з відкритим вихідним кодом
Спільнота	Величезна, безліч ресурсів та документації	Велика, активна спільнота	Зростаюча, активна спільнота

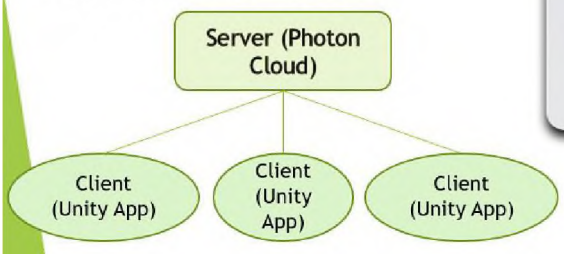
Photon Unity Networking-PUN2

PUN2 побудований на

- Високорівневому API
- Low-level Networking calls
- Platform-native DLLs

Підтримує

- Networked objects in scenes
- RPC та прямий обмін повідомленнями
- Користувацькі повідомлення



Проектування гри

Концепція гри:

- Гравці змагаються на арені, використовуючи бомби для знищення перешкод та суперників.
- Метою гри є залишитися останнім гравцем на полі бою.

Ігрова механіка:

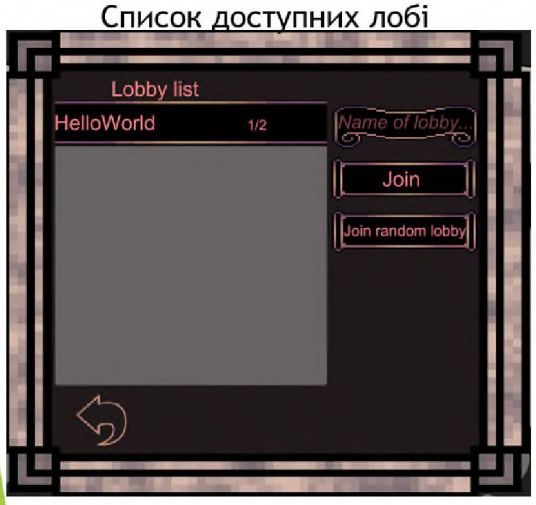
- Гравці пересуваються по ігровому полю та встановлюють бомби.
- Бомби вибухають через певний час, знищуючи перешкоди та інших гравців на своєму шляху.
- Знищені перешкоди можуть містити бонуси, що покращують характеристики гравця(швидкість, кількість бомб, силу вибуху).
- Перемагає останній гравець, що залишився живим.

Інтерфейс користувача

Головне меню

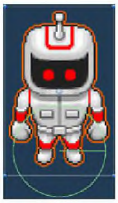
Створення лобі

Список доступних лобі



Лобі

Основні ігрові елементи



Персонаж, яким керує користувач. Завдання гравця - переміщуватися по ігровому полю, встановлювати бомби та знищувати суперників.

Бомба:
Основна зброя гравця. Вибухає через певний час, знищуючи перешкоди та інших гравців на своєму шляху.

Руйнівний об'єкт (ящик): Перешкода на ігровому полі, яку можна знищити бомбою.

Усилення (Power-ups): Спеціальні предмети, що випадають з деяких знищених ящиків. Усилення покращують характеристики гравця.

Збільшення кількості бомб: Дозволяє гравцеві встановлювати більше бомб одночасно.

Збільшення швидкості пересування: Дозволяє гравцеві рухатися швидше.

Збільшення радіусу вибуху:
Збільшує область дії вибуху бомби.

Висновки

У результаті виконання дипломної роботи поставлену мету було досягнуто — створено мобільну 2D-гру для Android на платформі Unity.

У процесі розробки було успішно вирішено ряд завдань, включаючи:
➤ Аналіз ринку мобільних ігор та існуючих розробок у жанрі, що дозволило визначити ключові вимоги та особливості проєкту.
➤ Проєктування і реалізація ігрової механіки, включно з правилами гри, поведінкою персонажів, взаємодією з об'єктами і бомбами.
➤ Розробку графічного інтерфейсу, адаптованого під сенсорні екрани мобільних пристроїв.
➤ За допомогою Photon PUN реалізовано стабільну мережеву взаємодію між гравцями, забезпечуючи синхронізацію дій, передачу

Дякую за увагу

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Ніколайчука Максима Генадійовича

(прізвище, ім`я та по батькові)

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітня програма *«Розробка програмного забезпечення»*

Керівник дипломного проекту (роботи) *Шувалова Ірина Олегівна*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка мобільної 2D-гри "Bomber" з можливістю кооперативу у ICP Unity*

Обсяг розрахунково-пояснювальної записки *72* сторінок

Обсяг графічної (презентаційної) частини *12* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений на рецензію дипломний проект відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячений Розробка мобільної 2D-гри "Bomber" з можливістю кооперативу у ICP Unity та складається з пояснювальної записки, додатку з програмним кодом та мультимедійної презентації, що містить приклади роботи програми.

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу (аналізу предметної області, проектування застосунку, реалізації застосунку, тестування застосунку), економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора КТ. Економічний розділ проекту містить розрахунок витрат на НДР та реалізацію проекту.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 13 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять ілюстративні схеми, скріншоти роботи програмного застосунку, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання графічної частини проекту та пояснювальної записки добра, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту *від аналізу середовищ і вибору технологій через проектування ігрового процесу, програмування логіки та реалізацію мультиплеєрної взаємодії до тестування на різних Android-пристроях .*

Забезпечено реалізацію мережевого режиму з використанням Phyton PUN,

д) основні недоліки дипломного проекту *У розділі тестування не перевірено відставання обробки одного гравця від іншого. Недостатня обробка мережевих подій. Присутні помилки оформлення пояснювальної записки*

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Добре*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові рецензента *к.т.н. Шибаєва Наталя Олегівна*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис: _____

«23» 06 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Ніколайчука Максима Генадійовича

(прізвище, ім`я та по батькові)

Спеціальність: *121 «Інженерія програмного забезпечення»*

Освітня програма: *«Розробка програмного забезпечення»*

Тема дипломного проекту: *Розробка мобільної 2D-гри "Bomber" з можливістю кооперативу у ICP Unity*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка до дипломного проекту містить 70 сторінок. У пояснювальній записці описано етапи Розробки мобільної 2D-гри "Bomber" з можливістю кооперативу у ICP Unity . Графічна частина складається з окремих слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра.*

б) самостійність роботи над проектом: *Протягом виконання дипломного проекту здобувач освіти Ніколайчук Максим поступово та послідовно виконував всі етапи, проявив ініціативу в створенні загальної концепції та реалізації роботи. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти Ніколайчук Максим під час роботи над дипломним проектом вивчив достатньо багато літературних та інтернет-джерел за даною тематикою.*

Вважаю, що теоретична підготовка дипломника достатня і він готовий до захисту проекту.

г) вміння розв'язувати виробничі та конструкторські питання *Під час виконання дипломного проекту здобувач освіти Ніколайчук Максим показав вміння організовано працювати над поставленим завданням, застосовувати знання у галузі програмування та математики, розробляти, встановлювати та налаштовувати спеціалізоване програмне забезпечення, оформлювати слайди та складати презентації, користуючись сучасними комп'ютерними програмними засобами, такими як Microsoft Visual Studio, Microsoft PowerPoint, Microsoft Visio, Unity та ін.*

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Задовільно*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові керівника дипломного проекту *Шувалова Ірина Олегівна*

Місце роботи і посада керівника дипломного проекту *ВСП «Одеський технічний фаховий коледж ОНТУ», викладач спецдисциплін циклової комісії комп'ютерної техніки та програмної інженерії*

Підпис

«16» 06 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Ніколайчук Максим Георгійович
здобувач освіти гр. 4РП-08, та

Шувалова Ірина Олегівна,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка мобільної 2D-гри “Bomber” з можливістю кооперативу у ICP Unity»
(автор роботи – Ніколайчук М.Г., керівник роботи – Шувалова І.О.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Ніколайчук М.Г. /

Керівник _____ / Шувалова І.О. /

«16» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4РП-08

Ніколайчука Максима Георгійовича

на тему *Розробка мобільної 2D-гри "Bomber" з можливістю кооперативу у ICP Unity*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до дипломного проєкту виконана з деякими порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *20.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 19.06.2025 р. значення коефіцієнту подібності в роботі становить 24,71%, коефіцієнт цитування – 1,64%.*

(підпис) *20.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Ніколайчука М.Г.* (П.І.Б.)

проведена « *20* » *червня* 2025 р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/19/2025
Дата редагування 6/19/2025

Документ прийнятий

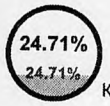
Звіт подібності

метадані

Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка мобільної 2D-гри “Bomber” з можливістю кооперативу у ICP Unity
Автор Науковий керівник / Експерт
Ніколайчук Максим ГеоргійовичШувалова Ірина Олегівна
підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1

КЦ

25
Довжина фрази для коефіцієнта подібності 2

14891
Кількість слів

119288
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		23
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки		0
Парафрази (SmartMarks)	a	173

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	ДИПЛОМ (Гезалов) — плагиат [redacted] 6/2/2025 State University of Intellectual Technologies and Communications (Кафедра інженерії програмного забезпечення)	166 1.11 %
2	ДИПЛОМ (Гезалов) — плагиат [redacted] 6/2/2025 State University of Intellectual Technologies and Communications (Кафедра інженерії програмного забезпечення)	109 0.73 %

3	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	100 0.67 %
4	ДИПЛОМ (Гезалов) — плагиат [redacted] 6/2/2025 State University of Intellectual Technologies and Communications (Кафедра інженерії програмного забезпечення)	93 0.62 %
5	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	82 0.55 %
6	ДИПЛОМ (Гезалов) — плагиат [redacted] 6/2/2025 State University of Intellectual Technologies and Communications (Кафедра інженерії програмного забезпечення)	82 0.55 %
7	ДИПЛОМ (Гезалов) — плагиат [redacted] 6/2/2025 State University of Intellectual Technologies and Communications (Кафедра інженерії програмного забезпечення)	79 0.53 %
8	ДИПЛОМ (Гезалов) — плагиат [redacted] 6/2/2025 State University of Intellectual Technologies and Communications (Кафедра інженерії програмного забезпечення)	78 0.52 %
9	https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download	77 0.52 %
10	ДИПЛОМ (Гезалов) — плагиат [redacted] 6/2/2025 State University of Intellectual Technologies and Communications (Кафедра інженерії програмного забезпечення)	74 0.50 %

з домашньої бази даних (0.21 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка мобільного застосунку-помічника майстра манікюру [redacted] 6/18/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	32 (3) 0.21 %

з програми обміну базами даних (11.81 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	ДИПЛОМ (Гезалов) — плагиат [redacted] 6/2/2025 **State University of Intellectual Technologies and Communications (Кафедра інженерії програмного забезпечення)**	**1736 (57) 11.66 %**
2	СИСТЕМА УПРАВЛІННЯ ЕНЕРГОСПОЖИВАННЯМ У ПОБУТІ ЗА ДОПОМОГОЮ IoT ТА ШТУЧНОГО ІНТЕЛЕКТУ [redacted] 6/14/2025 National University of Shipbuilding named after Admiral Makarov (National University of Shipbuilding named after Admiral Makarov)	10 (1) 0.07 %
3	Розробка програми «Клавіатурний тренажер» [redacted] 6/3/2024 Zhytomyr Agricultural Technical Professional College (Zhytomyr Agricultural Technical Professional College)	8 (1) 0.05 %

4	598986 [redacted] 11/23/2023 University of Southern Denmark (SDU backlog)	5 (1) 0.03 %

3 Інтернету (12.69 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/bitstreams/0e72a3b9-bdd7-4711-a3c6-dedc1d4287cc/download	459 (47) 3.08 %
2	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content**	**355 (8) 2.38 %**
3	**https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download**	**149 (5) 1.00 %**
4	https://card-file.ontu.edu.ua/bitstreams/12d5c0ab-e979-48f2-a8ec-d5fc31f71fd5/download	144 (16) 0.97 %
5	**https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download**	**104 (4) 0.70 %**
6	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content**	**77 (5) 0.52 %**
7	**https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download**	**76 (2) 0.51 %**
8	https://rama.unimal.ac.id/id/eprint/310/1/BagasAuliaAlfasyam_190180087_DistribusiSkripsi.pdf	71 (9) 0.48 %
9	**https://studfile.net/preview/2508489/**	**62 (2) 0.42 %**
10	**https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download**	**56 (3) 0.38 %**
11	**https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download**	**56 (3) 0.38 %**
12	**https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download**	**54 (5) 0.36 %**
13	**https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download**	**53 (3) 0.36 %**
14	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/995bdcec-4e4d-4321-8070-4d6badcb8e49/content**	**46 (2) 0.31 %**
15	https://card-file.ontu.edu.ua/bitstreams/e4afae26-0a7e-4a4d-afc2-94341838de2a/download	40 (3) 0.27 %
16	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	19 (1) 0.13 %
17	https://ela.kpi.ua/bitstream/123456789/34398/1/Stavcev_bakalavr.pdf	17 (2) 0.11 %
18	https://ir.nmu.org.ua/bitstream/handle/123456789/161861/121-18-1_%D0%A1%D0%B0%D0%B2%D0%BE%D1%81%D1%82%D1%8F%D0%BD%D0%B5%D0%BD%D0%BA%D0%BE.pdf?sequence=1	13 (2) 0.09 %
19	http://dspace.tnpu.edu.ua/bitstream/123456789/32692/1/Melnyk_Vasylenko.pdf	13 (1) 0.09 %
20	https://cyberleninka.ru/article/n/osnovnye-printsipy-razrabotki-obraza-igrovogo-povedeniya-i-kastomizatsii-personazha-v-ramkah-geym-dizayna	12 (1) 0.08 %
21	https://card-file.ontu.edu.ua/bitstreams/55e2b8f2-7d3c-4235-99fc-2be51199b96d/download	7 (1) 0.05 %
22	https://safety.org.ua/zakon-pro-ohoronu-pratsi-sfera-zastosuvannya/	6 (1) 0.04 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)