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

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітня програма: **«Комп'ютерна інженерія»**

Група: 2БКС-28

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

здобувача освіти денної форми навчання

БКС.28.25.000.КРБ

СПАТАРЬ
ПАВЛА
КОСТЯНТИНОВИЧА

м. Одеса
2024 р.

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

Спеціальність: **123 «Комп'ютерна інженерія»**
Освітня програма: **«Комп'ютерна інженерія»**
Група: **2БКС-28**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи бакалавра на тему: **«Розробка мобільної 2d гри під андроїд на платформі unity»**

Проектний матеріал складається з пояснювальної записки на 81 сторінках та графічного (презентаційного) матеріалу на 11 аркушах (слайдах).

Виконавець _____ (Спатарь П.К.)
Керівник проекту _____ (Гаджиєв М.М.)

Консультанти:
з з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)
з нормоконтролю _____ (Петрашова В.І.)
старший консультант _____ (Кривченко Ю.В.)

До захисту допущений
Завідувач кафедри _____ (Іванова Л.В.)
Завідувач відділення _____ (Скорнякова О.В.)

Захист «24» 06 2024 р. Протокол ЕК №_____
Оцінка ЕК 5 (відмінно) 95 б.

Секретар ЕК _____

АНОТАЦІЯ

У випускній кваліфікаційній роботі роботі метою було створення мобільної 2d гри під андроїд на платформі unity. Для досягнення мети було виконано ряд завдань, включаючи: аналіз сучасних мобільних ігор, розгляд існуючих розробок, також були досліджені різні середовища розробки ігор.

На основі цього було спроектовано та реалізовано ігрову механіку, графічний інтерфейс, систему керування та мережеву взаємодію.

Був описан процес проектування та розробки гри, включаючи створення рівнів з використанням Tilemap, розробку ігрового персонажа та створення анімацій, реалізацію механіки бомб та синхронізацію між гравцями за допомогою Photon. Розглянуто особливості використання компонентів Unity, а також створення та налаштування скриптів на C# для керування ігровою логікою та взаємодії з користувацьким інтерфейсом.

Для розробки відповідної програми використано ігровий рушій Unity 6 6000.0.1f1, що використовує мову C# та засобом програмування алгоритму є середа розробки Visual Studio 2022.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *Комп'ютерних систем* Кафедра *Комп'ютерної інженерії*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"15" 01 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

Здобувачеві освіти *Спатарь Павлу Костянтиновичу*

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Розробка мобільної 2d гри під андроїд на платформі unity*

затверджена наказом по коледжу від "2" 11 2023 р. № 244-А2-ОД

2. Термін здачі закінченого проекту (роботи) 10.06.2024

3. Вихідні данні до проекту (роботи) *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)

1. Ринок мобільних ігор.

2. Порівняння движків.

3. Photon Unity Networking.

4. Проектування гри.

5. Інтерфейс користувача.

6. Основні ігрові елементи.

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів, що їх стосуються

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

7. Дата видачі завдання _____

Керівник роботи *Гаджиєв М.М.* _____
(підпис)

Завдання прийняв до виконання *Спатарь П.К.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Відмітка про виконання
1	*Підбір та вивчення літератури з теми дослідження*	*5.04.2024*	
2	*Аналіз ринку мобільних ігор та тенденцій розвитку*	*10.04.2024*	
3	*Аналіз існуючих ігор-аналогів*	*14.04.2024*	
4	*Визначення основних функціональних вимог до гри*	*21.04.2024*	
5	*Аналіз середовищ розробки*	*29.04.2024*	
6	*Вибір технологій для реалізації мультиплеєра*	*4.05.2024*	
7	*Проектування гри*	*7.05.2024*	
8	*Розробка графічного інтерфейсу користувача*	*10.05.2024*	
9	*Програмування ігрової логіки та механік.*	*20.05.2024*	
10	*Налаштування мережевої взаємодії*	*31.05.2024*	
11	*Тестування гри на різних пристроях.*	*5.06.2024*	
12	*Підготовка до малого захисту*	*10.06.2024*	
13	*Розробка питань з охорони праці*	*11.06.2024*	
14	*Підготовка роботи до захисту та тестування гри*	*14.06.2024*	
15	*Оформлення слайдів мультимедійної презентації*	*15.06.2024*	

Здобувач освіти _____
(підпис)

Керівник роботи _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			2БКС 28.25 000.00 КРБ	Випускна робота		
А4			2БКС 28.25 000.00 КРБ ПЗ	Пояснювальна записка	1	

Змн.	Арк.	№ докум.	Підпис	Дата	БКС 28.25 000.00 КРБ ПЗ		
Розробив		Спатарь П.К		10.06	Розробка мобільної 2d гри під андроїд на платформі unity	Літ.	Аркуш 4 / Аркушів 81
Перевірив		Гаджиєв М.М.		10.06			
Н. Контр.		Петрашова В.І.		10.06		ВСП "ОТФК ОНТУ" гр. 2БКС-28	
Затверд.		Іванова Л.В.		10.06			

ЗМІСТ

ВСТУП

У сучасному світі мобільних технологій, де ігри стали невід'ємною частиною розваг та соціальної взаємодії, особливу популярність здобувають проекти, що поєднують простоту механіки, захопливий ігровий процес та можливість змагатися з друзями. Розробка мобільних ігор є актуальним напрямком, який вимагає глибоких знань не лише у програмуванні, але й у дизайні, анімації, гейм-дизайні та тестуванні.

Цей дипломний проект присвячений розробці мобільної 2D гри для операційної системи Android з використанням платформи Unity. Unity є потужним інструментом для створення ігор, що надає розробникам широкий спектр можливостей для реалізації їхніх творчих ідей. Використання Unity дозволяє створювати кроссплатформенні проекти з високою продуктивністю та якістю графіки. Платформа підтримує інтеграцію різних компонентів, що значно полегшує процес розробки та тестування.

Гра, створена в рамках цього проекту, належить до жанру аркад. У грі гравці встановлюють бомби, намагаючись підірвати суперників і уникнути їхніх бомб, що додає елемент стратегії та тактики. Особливістю даної гри є її проста механіка, яка робить її доступною для широкого кола користувачів.

У роботі буде розглянуто основні етапи створення мобільної гри, включаючи розробку концепції та проектування ігрового процесу, програмування основної логіки гри, створення анімацій, а також ретельне тестування і оптимізацію готового продукту. Важливим аспектом розробки є також реалізація багатокористувацького режиму за допомогою Photon PUN. Це передбачає створення серверної частини, налаштування підключення клієнтів, синхронізацію дій гравців та забезпечення стабільної роботи гри в мережевому середовищі.

Метою цього проекту є не лише створення функціональної та захоплюючої мобільної 2D гри, але й набуття практичного досвіду у розробці ігор на платформі Unity, що сприятиме професійному зростанню та подальшому розвитку у сфері розробки програмного забезпечення.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз та загальна характеристика мобільних ігор

Мобільні ігри стали невід'ємною частиною сучасного цифрового ландшафту, значно розширивши межі індустрії розваг. Завдяки швидкому розвитку мобільних технологій та широкому розповсюдженню смартфонів, мобільні ігри стали доступнішими для широкого кола користувачів. Вони є програмами, розробленими спеціально для мобільних пристроїв, таких як смартфони та планшети. Ці ігри забезпечують інтерактивний ігровий процес, дозволяючи користувачам взаємодіяти з віртуальним середовищем через сенсорні екрани, акселерометри та інші функціональні можливості мобільних пристроїв.

Мобільні ігри охоплюють широкий спектр жанрів, від простих казуальних ігор до складних стратегій і рольових ігор. Їх розробка часто передбачає створення оптимізованого ігрового процесу, який враховує обмежені ресурси мобільних пристроїв, такі як обчислювальна потужність, обсяг пам’яті та розмір екрану. Основними особливостями мобільних ігор є короткі ігрові сесії, просте управління та інтерактивність, що робить їх ідеальними для гри "на ходу".

Мобільні ігри можуть бути як самостійними проектами, так і портованими версіями популярних комп'ютерних та консольних ігор. Останнім часом також спостерігається зворотна тенденція, коли мобільні ігри стають настільки популярними, що згодом переносяться на інші платформи.

Важливою складовою успіху мобільних ігор є їх соціальна складова. Багато мобільних ігор підтримують багатокористувацькі режими, дозволяючи гравцям змагатися або співпрацювати один з одним через інтернет. Соціальні мережі та інтеграція з платформами, такими як Facebook або Google Play Games, дозволяють гравцям ділитися своїми досягненнями, запрошувати друзів до гри та брати участь у спільнотах.

Мобільні ігри також відіграють значну роль у розвитку мобільного ринку додатків, генеруючи значні доходи через продажі, рекламу та внутрішньоігрові покупки. Фріміум-модель, де базова версія гри безкоштовна, але доступ до

додаткових функцій чи контенту надається за плату, є однією з найпоширеніших моделей монетизації в мобільних іграх.

Візуальна складова мобільних ігор може варіюватися від простих піксельних зображень до високоякісної 3D-графіки. Хоча мобільні пристрої мають обмежені ресурси порівняно з ПК чи консолями, сучасні технології дозволяють створювати на них досить складні та візуально привабливі ігри.

Мобільні ігри можуть бути класифіковані за кількома критеріями:

- За жанром: казуальні, екшн, головоломки, рольові, стратегії, симулятори тощо.
- За режимом гри: одиночні та багатокористувацькі.
- За візуальною складовою: 2D та 3D ігри.

Казуальні ігри зазвичай мають прості механіки та короткі ігрові сесії, що робить їх ідеальними для гри в короткі перерви. Екшн-ігри вимагають від гравців швидких реакцій і точного управління, тоді як головоломки зосереджені на вирішенні завдань і можуть варіюватися від простих до дуже складних.

Розробка мобільних ігор є комплексним процесом, що включає проектування ігрового процесу, програмування, створення графіки та анімації, тестування та оптимізацію. Використання сучасних інструментів і технологій дозволяє розробляти високопродуктивні ігри з якісною графікою. Це сприяє ефективному створенню кросплатформених проектів, що відповідають сучасним вимогам користувачів та ринку мобільних ігор

Аналізуючи різні жанри та тенденції розвитку мобільних ігор, було вирішено створити двовимірну гру у жанрі аркадного екшену. Ігри цього жанру мають зрозумілу та просту механіку, що дозволяє залучити широку

аудиторію, включаючи як казуальних гравців, так і тих, хто шукає більш динамічний ігровий досвід. Вибір 2D-графіки обумовлений її меншою вимогливістю до апаратних ресурсів та простотою у розробці, що дозволяє зосередитися на поліпшенні ігрового процесу та наданні високоякісного користувацького досвіду.

Таким чином, обрана концепція гри має на меті поєднати переваги аркадного

екшену та доступності 2D-графіки, забезпечуючи захопливий ігровий процес та широкий охоплення цільової аудиторії.

1.2 Аналіз існуючих розробок

Ігри, натхненні класичним "Bomberman", де гравці стратегічно розміщують бомби для знищення суперників та перешкод, продовжують захоплювати гравців навіть через десятиліття після свого дебюту у 1983 році. Незмінна популярність цього жанру пояснюється його унікальним поєднанням простих, але захоплюючих механік, динамічного ігрового процесу, що вимагає швидкої реакції та стратегічного мислення, а також можливості змагатися з іншими гравцями, що додає соціальний аспект до гри.

Класичні представники жанру, що заклали фундамент: епоха 8- та 16-бітних консолей:

Bomberman (1983): Ця гра, розроблена Hudson Soft, стала основоположником жанру, представивши гравцям ключові елементи геймплею, які залишаються актуальними і досі. Гравці керують персонажем, який розміщує бомби з таймером, що вибухають через певний час, знищуючи блоки та ворогів у радіусі дії. Зібрані бонуси можуть збільшити силу вибуху, швидкість персонажа або дати додаткові бомби. Багатокористувацький режим дозволяв до чотирьох гравців змагатися на одному екрані, що зробило Bomberman популярною грою для вечірок та дружніх посиденьок.

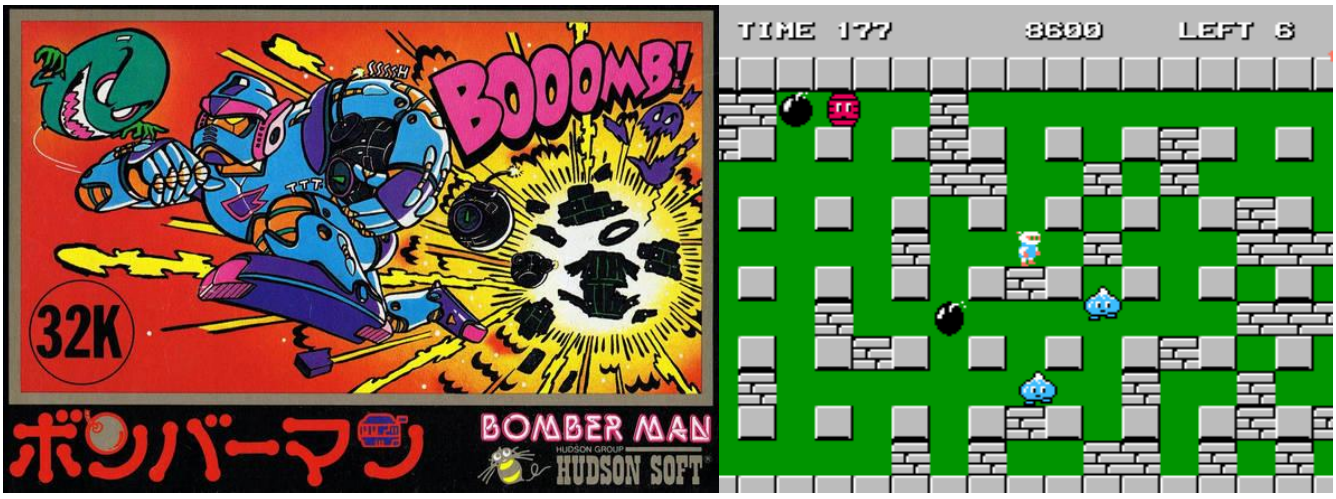


Рисунок 1.1. Bomberman 1983

Super Bomberman (1993): Ця серія ігор, випущена для Super Nintendo Entertainment System (SNES), стала культовою завдяки своїй барвистій графіці, харизматичним персонажам та різноманітним режимам гри. Кожен персонаж мав унікальні здібності, що додало глибини до ігрового процесу. Нові рівні з різними перешкодами, пастками та бонусами зробили кожне проходження унікальним та захоплюючим. Серія Super Bomberman також представила режим "Story Mode" з сюжетом та кат-сценами, що розширило всесвіт гри та зробило його більш цікавим для гравців.

Рисунок 1.2. Super Bomberman 1993

Сучасні інтерпретації та аналоги:

Bomberman Ultra (2009): Ця гра, випущена для PlayStation 3, є однією з найпопулярніших сучасних інтерпретацій Bomberman. Вона пропонує класичний геймплей з оновленою графікою та розширеним багатокористувацьким режимом, що підтримує до 8 гравців онлайн. Bomberman Ultra також має редактор персонажів, що дозволяє гравцям створювати унікальні образи для своїх бомберменів

Рисунок 1.3. Bomberman Ultra 2009

Bombergrounds: Reborn (2020): Ця багатоплатформова гра, розроблена Gigantic Duck Games, поєднує класичний геймплей Bomberman з елементами Battle Royale та колекційної карткової гри. Гравці змагаються на арені, яка поступово зменшується, збираючи бонуси, покращуючи своїх персонажів та використовуючи бомби, щоб знищувати ворогів. Гра має різні режими гри, включаючи соло, дуети та команди, а також систему прогресії, яка дозволяє гравцям розблоковувати нових персонажів та скіни.

Рисунок 1.4. Bombergrounds: Reborn 2020

1.3 Аналіз середовищ розробки та обґрунтування вибору технології розробки проекту

Вибір ігрового движка є одним з найважливіших стратегічних рішень на початку розробки гри. Він визначає не лише технічні можливості та обмеження проекту, але й впливає на весь процес створення гри, починаючи від дизайну та прототипування, закінчуючи тестуванням та випуском. Кожен движок має свої сильні та слабкі сторони, які потрібно враховувати, виходячи з конкретних потреб та цілей проекту.

1.3.1 Unreal Engine

Розроблений та підтримуваний компанією Epic Games, Unreal Engine вперше заявив про себе у 1998 році з виходом шутера від першої особи Unreal. Хоча спочатку двигун був орієнтований на шутери, його універсальність швидко стала очевидною, і наступні версії успішно застосовувалися у різноманітних жанрах, включаючи стелс-ігри, файтинги та MMORPG.

Рисунок 1.5. Інтерфейс Unreal Engine

Технічні можливості:

- Кросплатформеність: Unreal Engine підтримує широкий спектр платформ та операційних систем, включаючи Windows, Linux, macOS, iOS, Android, консолі Xbox та PlayStation, Nintendo Switch та інші.
- Системні вимоги: Для комфортної роботи з редактором Unreal Engine рекомендується комп'ютер з такими мінімальними характеристиками:
  - ОС: Windows 7/8/10 64-bit, Linux Ubuntu 15.04+, macOS 10.13+
  - Процесор: Intel Quad-core або AMD (2.5 ГГц+)
  - Оперативна пам'ять: 8 ГБ (16 ГБ для Linux)
  - Відеокарта: AMD Radeon 6870 HD або NVIDIA GeForce 470 GTX+

Інструментарій та особливості:

- Фотореалістична графіка: Завдяки широким можливостям

налаштування тіней, освітлення та рендерингу, Unreal Engine дозволяє досягти вражаючої візуальної якості.

- Blueprint: Система візуальних скриптів Blueprint спрощує розробку ігрової логіки, дозволяючи створювати складні механіки без написання коду.
- C++: Для досвідчених розробників Unreal Engine надає повний доступ до вихідного коду на C++, забезпечуючи максимальну гнучкість та контроль.
- Інструменти для роботи з ландшафтом та рослинністю: Unreal Engine має вбудовані інструменти, що спрощують створення реалістичних природних сцен.
- Cascade: Редактор візуальних ефектів Cascade дозволяє створювати вражаючі ефекти частинок та інші візуальні елементи.
- Редактор матеріалів: Unreal Engine надає потужний редактор матеріалів з підтримкою PBR (Physically Based Rendering), що дозволяє створювати реалістичні матеріали з урахуванням фізичних властивостей поверхонь.
- Sequencer: Вбудований інструмент Sequencer дозволяє створювати катсцени та інші анімаційні послідовності.
- Підтримка VR: Unreal Engine має вбудовану підтримку віртуальної реальності з розширеними можливостями налаштування.
- Штучний інтелект: Розширена система ШІ дозволяє створювати складну поведінку персонажів та інших ігрових об'єктів.

Ліцензування та підтримка:

- Безкоштовна ліцензія: З 2015 року Unreal Engine доступний безкоштовно для всіх розробників.
- Роялті: Якщо проект, створений на Unreal Engine, приносить прибуток понад 3000 доларів за квартал, розробник повинен виплачувати Epic Games 5% роялті.
- Документація та навчальні матеріали: На офіційному сайті Unreal

Engine доступна велика кількість документації, туторіалів та інших навчальних матеріалів.

- Marketplace: Unreal Engine Marketplace пропонує широкий вибір готових ассетів, плагінів та інструментів для розробки.

Unreal Engine – це потужний і універсальний ігровий движок, який підходить для створення різноманітних ігор, особливо у жанрі 3D. Він пропонує широкий спектр інструментів та можливостей, але вимагає певного досвіду та знань для ефективного використання.

1.3.2 Godot Engine

Godot Engine – це безкоштовний ігровий движок з відкритим вихідним кодом, який стрімко набирає популярність завдяки своїй гнучкості, простоті використання та потужному інструментарію. Розроблений спільнотою ентузіастів, Godot Engine є чудовим вибором як для новачків, так і для досвідчених розробників.

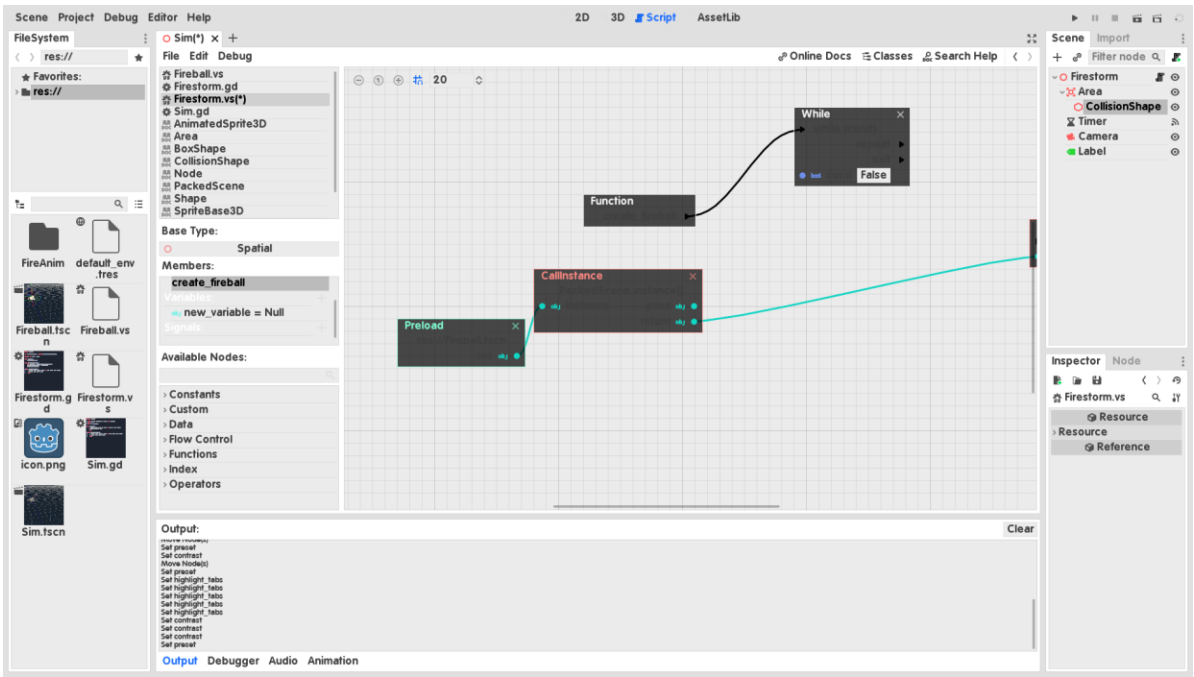


Рисунок 1.6. Інтерфейс Godot Engine

Технічні можливості:

- Кросплатформеність: Godot Engine підтримує широкий спектр платформ, включаючи Windows, Linux, macOS, Android, iOS та веб.

- Системні вимоги: Godot Engine має низькі системні вимоги та може працювати навіть на старіших або менш потужних комп'ютерах.
- Мови програмування: Godot Engine підтримує декілька мов програмування, включаючи:
  - GDScript: Вбудована мова, схожа на Python, спеціально розроблена для Godot Engine. Вона проста у вивченні та ідеально підходить для новачків.
  - C#: Популярна мова програмування, що використовується у багатьох інших ігрових движках, таких як Unity.
  - C++: Мова програмування низького рівня, що забезпечує максимальну продуктивність та контроль.

Інструментарій та особливості:

- 2D-орієнтованість: Godot Engine має потужний набір інструментів для створення 2D-ігор, включаючи редактор анімації, фізичний движок та систему частинок.
- 3D-можливості: Хоча Godot Engine більше орієнтований на 2D, він також має непоганий набір інструментів для створення 3D-ігор.
- Node-based система сцен: Унікальна система сцен Godot Engine дозволяє організовувати ігрові об'єкти у вигляді дерева вузлів, що робить розробку більш інтуїтивною та зрозумілою.
- Візуальний скриптинг: Godot Engine має вбудований візуальний редактор скриптів, що дозволяє створювати ігрову логіку без написання коду.
- Гнучкість: Godot Engine дозволяє розробникам використовувати різні підходи до розробки та вибирати найбільш зручні інструменти та мови програмування.

Ліцензування та підтримка:

- Безкоштовний та відкритий вихідний код: Godot Engine розповсюджується під ліцензією MIT, що дозволяє вільно використовувати, модифікувати та розповсюджувати движок та

створені на ньому ігри.

- Спільнота: Godot Engine має активну та дружню спільноту розробників, які готові допомогти та поділитися своїм досвідом.
- Документація та навчальні матеріали: На офіційному сайті Godot Engine доступна велика кількість документації, туторіалів та інших навчальних матеріалів.
- Asset Library: Godot Engine має власну бібліотеку ассетів, де розробники можуть знайти готові моделі, текстури, звуки та інші ресурси для своїх проектів.

1.3.3 Unity

Unity – це один з найпопулярніших ігрових движків у світі, що використовується для створення різноманітних ігор, від простих мобільних до складних AAA-проектів. Завдяки своїй універсальності, величезній спільноті та широкому спектру інструментів, Unity став вибором багатьох розробників по всьому світу.

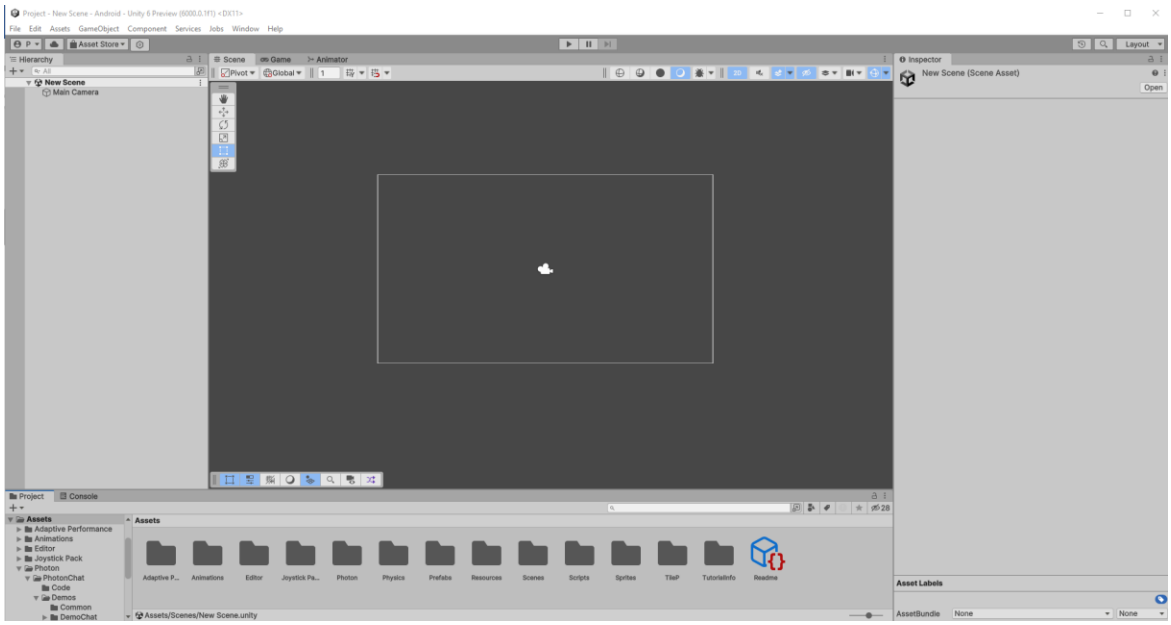


Рисунок 1.7. Інтерфейс Unity

Технічні можливості:

- Кросплатформеність: Unity підтримує вражаючу кількість платформ, включаючи Windows, macOS, Linux, Android, iOS, різноманітні консолі

(PlayStation, Xbox, Switch), а також веб та віртуальну реальність (VR).

- Системні вимоги: Unity має помірні системні вимоги, що дозволяє працювати з ним на більшості сучасних комп'ютерів. Проте, для великих та складних проектів може знадобитися потужніший апарат.
- Мова програмування: Основною мовою програмування в Unity є C#, що є популярною та відносно простою у вивченні мовою.

Інструментарій та особливості:

- Широкі можливості для 2D-розробки: Unity надає розробникам широкий спектр інструментів та можливостей, спеціально розроблених для створення 2D-ігор, включаючи 2D Tilemap Editor, 2D Animation та 2D Physics.
- Asset Store: Величезний магазин готових ассетів, плагінів та інструментів дозволяє швидко розпочати розробку та зекономити час на створенні власних ресурсів.
- Візуальний скриптинг: Unity підтримує візуальний скриптинг за допомогою Bolt, що дозволяє створювати ігрову логіку без написання коду.
- Спільнота та підтримка: Unity має величезну та активну спільноту розробників, які готові допомогти та поділитися своїм досвідом. На офіційному сайті Unity доступна велика кількість документації, туторіалів та інших навчальних матеріалів.
- Monetization: Unity надає інструменти для монетизації ігор, такі як реклама, покупки всередині програми та підписки.
- Сервіси Unity: Unity пропонує різноманітні сервіси для розробників, такі як Unity Collaborate для спільної роботи над проектами, Unity Cloud Build для автоматизації збірки та Unity Analytics для аналізу поведінки гравців.

Ліцензування:

- Безкоштовна версія: Unity пропонує безкоштовну версію з обмеженим функціоналом для особистого використання та невеликих проектів.

- Платні підписки: Для комерційного використання та доступу до розширеного функціоналу Unity пропонує різні платні підписки.

Після аналізу можливостей різних ігрових движків, вибір було зроблено на користь Unity як оптимального інструменту для розробки дипломного проекту. Цей вибір зумовлений сукупністю факторів, що роблять Unity ідеальним рішенням для створення 2D-гри подібного жанру. Unity надає широкий набір спеціалізованих інструментів, що прискорюють та спрощують процес створення гри. Вбудований тайловий редактор дозволяє швидко створювати ігрові рівні, гнучкі анімаційні системи забезпечують плавний рух, а фізичний движок, оптимізований для 2D, легко реалізує зіткнення та вибухи.

Гнучкість Unity відкриває широкі можливості для реалізації всіх необхідних ігрових механік. Крім того, наявність інструментів для профілювання та оптимізації продуктивності дозволяє досягти плавної роботи гри на різних пристроях Android

1.4 Огляд інструментарію та технологій Unity для розробки ігор

2D Sprite Editor – це вбудований інструмент Unity, який надає повний контроль над імпортом, налаштуванням та оптимізацією 2D-графіки (спрайтів) для створення гри.

Основні можливості:

1. Імпорт:

- Легко імпортуйте зображення різних форматів (PNG, JPG, PSD тощо) у ваш проект Unity.
- Автоматично налаштовує імпортовані зображення як спрайти.
- Підтримує імпорт багатошарових PSD-файлів, зберігаючи структуру шарів та дозволяючи працювати з ними окремо.

2. Налаштування:

- Sprite Mode: Вибір режиму спрайту:
  - Single: Один спрайт на все зображення.

- Multiple: Розбиття зображення на кілька спрайтів.
- Polygon: Створення спрайту з довільною формою за допомогою полігонів.

- Pixels Per Unit (PPU): Визначення масштабу спрайту у світі Unity.
- Pivot: Вибір точки прив'язки спрайту (центр, верхній лівий кут тощо).
- Mesh Type: Вибір типу сітки спрайту:
  - Full Rect: Прямокутна сітка, що охоплює весь спрайт.
  - Tight: Сітка, що щільно облягає непрозорі пікселі спрайту.

3. Розбиття спрайтів (Sprite Slicing):

- Automatic: Автоматичне розбиття зображення на спрайти за прозорими областями.
- Grid By Cell Size/Count: Розбиття зображення на спрайти за допомогою сітки з заданим розміром або кількістю комірок.
- Custom: Ручне розбиття зображення на спрайти за допомогою інструментів виділення.

4. Налаштування півотних точок (Pivot Points):

- Індивідуальне налаштування півотних точок для кожного спрайту.
- Використання півотних точок для точного позиціонування та анімації спрайтів.

5. Оптимізація:

- Compression: Вибір алгоритму стиснення зображень для зменшення розміру файлів.
- Mip Maps: Створення міп-мапів для покращення якості відображення спрайтів на різних відстанях.

Додаткові можливості:

- Sprite Atlas: Об'єднання кількох спрайтів в один атлас для оптимізації рендерингу.
- 9-Slice Scaling: Масштабування спрайтів без спотворення за допомогою 9-слайсового масштабування.
- Sprite Shape: Створення спрайтів з довільною формою за допомогою

сплайнів.

Unity надає потужний фізичний двигун, заснований на Box2D, для симуляції фізики в 2D-іграх. Це дозволяє об'єктам взаємодіяти між собою та з оточенням, реагуючи на сили, зіткнення та гравітацію.

Основні компоненти:

1. Rigidbody 2D:

- Ключовий компонент, що надає об'єкту фізичні властивості: масу, лінійну та кутову швидкості, тертя тощо.
- Дозволяє об'єкту рухатися під дією сил, зіштовхуватися з іншими об'єктами, реагувати на гравітацію.
- Різні типи Rigidbody 2D:
  - Dynamic: Об'єкт рухається під дією сил та зіткнень.
  - Kinematic: Об'єкт рухається лише під дією скриптів, не реагує на зіткнення.
  - Static: Об'єкт нерухомий, використовується для створення статичних перешкод.

2. Collider 2D:

- Визначає форму об'єкту для фізичних розрахунків.
- Різні типи Collider 2D:
  - Box Collider 2D: Прямокутна форма.
  - Circle Collider 2D: Кругла форма.
  - Polygon Collider 2D: Довільна багатокутна форма.
  - Edge Collider 2D: Лінійна форма (для створення платформ, стін тощо).
  - Capsule Collider 2D: Капсульна форма (поєднання прямокутника та двох півкіл).
- Можливість налаштування параметрів колайдера:
  - Is Trigger: Визначає, чи буде колайдер реагувати на зіткнення як тригер (без фізичної реакції).
  - Material: Визначає фізичний матеріал колайдера (тертя,

пружність тощо).

3. Joint 2D:

- Використовується для створення з'єднань між об'єктами.
- Різні типи Joint 2D:
  - Distance Joint 2D: Підтримує фіксовану відстань між двома об'єктами.
  - Hinge Joint 2D: Створення шарнірного з'єднання (як двері).
  - Slider Joint 2D: Дозволяє об'єктам ковзати вздовж осі.
  - Wheel Joint 2D: Створення колісного з'єднання.
  - Spring Joint 2D: Створення пружинного з'єднання.

4. Physics Material 2D:

- Визначає фізичні властивості поверхні колайдера: тертя, пружність тощо.
- Дозволяє створювати різні типи поверхонь: лід, пісок, гума тощо.

Animator - це компонент Unity, який дозволяє створювати, відтворювати та керувати анімаціями ігрових об'єктів. Він надає зручний інтерфейс для створення анімаційних станів, переходів між ними та налаштування параметрів анімації.

Основні концепції:

1. Animator Controller:

- Ключовий елемент Animator, який містить всі анімаційні стани, переходи та параметри для об'єкта.
- Створюється у вікні Animator, доступному через меню Window -> Animation -> Animator.
- Візуально представляє анімаційні стани у вигляді графу, де кожен вузол - це стан, а ребра - переходи між станами.

2. Animation State:

- Визначає конкретну анімацію, яку буде відтворювати об'єкт.
- Кожен стан містить посилання на анімаційний кліп (Animation Clip), який визначає послідовність кадрів анімації.
- Може мати налаштування швидкості відтворення, циклічності тощо.

3. Transition:

- Визначає умови переходу від одного анімаційного стану до іншого.
- Умови можуть бути різними:
  - Boolean Parameter: Перехід відбувається, коли булевий параметр має певне значення (true або false).
  - Trigger Parameter: Перехід відбувається один раз, коли тригерний параметр активується.
  - Float Parameter: Перехід відбувається, коли значення числового параметру досягає певного порогу.
  - Int Parameter: Перехід відбувається, коли значення цілочисельного параметру досягає певного значення.

4. Parameter:

- Змінна, яка використовується для керування переходами між анімаційними станами.
- Може бути булевого, тригерного, числового або цілочисельного типу.
- Значення параметрів можна змінювати зі скриптів або інших компонентів Unity.

1.5 Середовище розробки Visual Studio

Однією з основних причин вибору Visual Studio для розробки мобільної 2D гри на платформі Unity є її тісна інтеграція з Unity, що надає ряд переваг для розробників.

Unity автоматично налаштовує проект Visual Studio, що спрощує процес початкової конфігурації. Після створення нового проекту в Unity, середовище Visual Studio автоматично налаштовується для роботи з цим проектом. Це означає, що розробники можуть негайно розпочати написання коду без необхідності вручну налаштовувати параметри проекту або імпортувати бібліотеки.

1.5.1 Unity Tools for Visual Studio

Unity Tools for Visual Studio – це спеціальний набір інструментів, розроблений для покращення досвіду роботи з Unity у Visual Studio. Цей набір

включає:

- Інтелектуальне автозаповнення (IntelliSense): Ця функція допомагає розробникам швидко знаходити та використовувати методи, властивості та класи, доступні в Unity API. IntelliSense також надає підказки щодо параметрів методів та автоматично завершує імена змінних та функцій.
- Unity Project Explorer: Це розширення забезпечує зручний перегляд та навігацію по файлах проекту Unity безпосередньо з Visual Studio, що значно спрощує роботу з великими проектами.
- Unity Debugger: Вбудований відладчик дозволяє встановлювати точки зупинки (брекпоінти), виконувати покрокове виконання коду, переглядати значення змінних та виконувати інші операції налагодження. Це допомагає швидко знаходити та виправляти помилки в коді.

Visual Studio з легкістю обробляє специфічні для Unity файли, такі як скрипти C# та шейдери. Завдяки цьому, розробники можуть редагувати та тестувати ці файли безпосередньо в Visual Studio, не витрачаючи час на перемикання між різними інструментами. Наприклад, коли розробник зберігає зміни у файлі скрипту C#, Unity автоматично перезавантажує та компілює цей файл, що забезпечує швидкий цикл редагування та тестування.

Інтеграція Visual Studio з системами контролю версій, такими як Git, забезпечує зручне управління версіями проекту та полегшує співпрацю в команді. Розробники можуть використовувати вбудовані інструменти Visual Studio для комітів, пушів та пулів змін безпосередньо зі середовища розробки, що сприяє злагодженій командній роботі.

Використовуючи Visual Studio з Unity розробники отримують потужні інструменти, роблячи робочий процес зручнішим, що дозволяє значно підвищити ефективність розробки. Завдяки автоматичному налаштуванню середовища, інтелектуальному автозаповненню, зручному навігаційному інтерфейсу, вбудованому відладчику, підтримці специфічних для Unity файлів та інтеграції з

системами контролю версій, Visual Studio стає ідеальним вибором для розробки мобільних ігор на платформі Unity.

1.6 Мультиплеєр

Мультиплеєрні ігри стали однією з найважливіших складових сучасної ігрової індустрії. Вони дозволяють гравцям з різних куточків світу взаємодіяти та змагатися один з одним у віртуальному просторі. З розвитком мобільних технологій мультиплеєрні ігри отримали новий імпульс розвитку, надаючи користувачам можливість грати разом у будь-який час і в будь-якому місці. Існує кілька основних типів мультиплеєрних ігор, які розрізняються за своїми характеристиками та механіками. Нижче наведені чотири основні типи мультиплеєрних ігор.

Dynamic Single Player – це одиночні ігри з простими мережевими функціями, такими як соціальні дошки рекордів, кнопки "поділитися" та регулярні оновлення контенту. В таких іграх гравець переважно грає один, але має можливість порівнювати свої досягнення з іншими гравцями, брати участь у глобальних рейтингах та ділитися результатами через соціальні мережі.

Turn-Based Multiplayer – пошаговий мультиплеєр, у якому геймплей передбачає участь більше одного гравця, але дії виконуються по черзі. Цей тип ігор дозволяє гравцям обдумувати свої ходи та стратегії, що робить їх ідеальними для стратегічних і настільних ігор.

Real-Time Multiplayer – геймплей у реальному часі, де участь беруть одночасно кілька гравців (не більше 10/16/25/… ігроков). Такі ігри вимагають швидкої реакції та координації дій між гравцями. Вони можуть включати в себе різноманітні жанри, від шутерів до гонок, і забезпечують високий рівень інтерактивності та напруження.

Persistent Game Spaces – геймплей у реальному часі з загальним для всіх гравців станом гри. Цей тип ігор відрізняється тим, що світ гри існує незалежно від того, чи перебуває гравець в онлайн-режимі. Гравці можуть взаємодіяти з одним світом, де їхні дії мають тривалий вплив на ігрове середовище. Це характерно для

масових багатокористувацьких онлайн-ігор (MMO).

1.6.1 Вибір технології для реалізації мультиплеєра

Для реалізації мультиплеєрного режиму у мобільній 2D грі на платформі Android було обрано Photon Unity Networking 2 (PUN2). PUN2 є одним із найпопулярніших рішень для створення мультиплеєрних ігор на Unity, і ось чому:

- Photon Unity Networking 2 легко інтегрується з Unity, що дозволяє швидко почати роботу над мультиплеєрними функціями. Документація та численні приклади роблять процес освоєння цієї технології простим навіть для початківців.
- PUN2 забезпечує високу продуктивність завдяки своїй оптимізованій архітектурі. Це особливо важливо для мобільних ігор, де обмежені ресурси пристроїв вимагають ефективного використання обчислювальної потужності та пам'яті.
- Photon підтримує кросплатформеність, дозволяючи гравцям на різних пристроях (Android, iOS, ПК) взаємодіяти один з одним в одній грі. Це розширює аудиторію гри та підвищує її привабливість.
- Photon пропонує різні моделі підключення, включаючи Dedicated Server та Cloud Server, що дозволяє налаштувати ігрову інфраструктуру відповідно до потреб проекту. Це забезпечує високу гнучкість та можливість масштабування гри в залежності від кількості гравців.
- Photon має велику спільноту розробників та активну підтримку від команди розробників, що дозволяє швидко знаходити рішення на виникаючі питання та отримувати допомогу при розробці.

Створена мобільна 2D гра відноситься до категорії Real-Time Multiplayer. Гравці взаємодіють у реальному часі, що забезпечує динамічний ігровий процес і потребує від них швидкої реакції та координації. Обраний тип мультиплеєра дозволяє створити захопливий та інтенсивний ігровий досвід, який сприяє соціальній взаємодії та змаганню між гравцями. Використання Photon Unity Networking 2 для реалізації цього типу гри забезпечує стабільність, високу

продуктивність та можливість масштабування, що є важливими аспектами для успішної реалізації проекту.

1.7 Проектування та розробка гри

1.7.1 Створення головного меню

Почнемо створення гри з першого, що побачить гравець під час входу в гру - головного меню. Для початку створимо нову сцену давши відповідну назву "MainMenu".

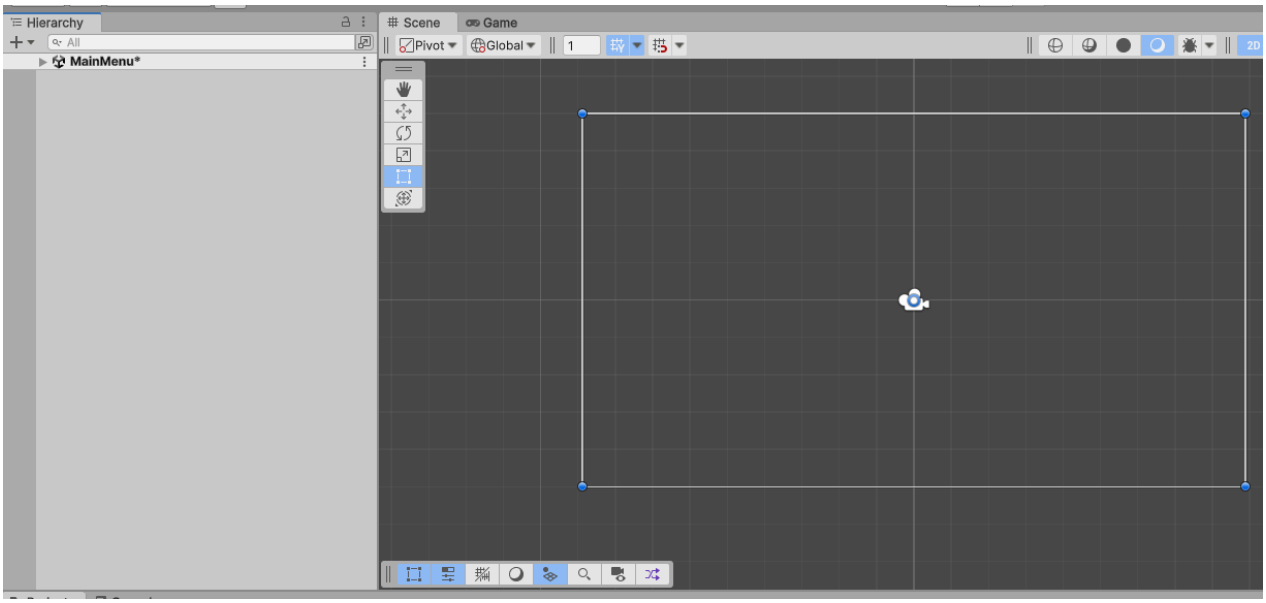


Рисунок 1.8. Створена сцена "MainMenu"

Далі створюю canavs, він буде основою для всіх елементів користувацького інтерфейсу (UI) у меню. У налаштуваннях Canvas у Render Camera прикріплюємо камеру, а в UI Scale Mode змінюємо на Scale With Screen Size та ставимо Match на 0.5. Додаю на Canvas Panel і назвав його "MainMenuPanel". Цей об'єкт буде контейнером для кнопок та інших елементів меню.

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

Ще потрібно буде додати Input Field, де буде ім'я гравця, і під час натискання його можна буде змінити, а також просто поле з текстом, де писатиметься інформація, наприклад, про те, є з'єднання із сервером чи ні.

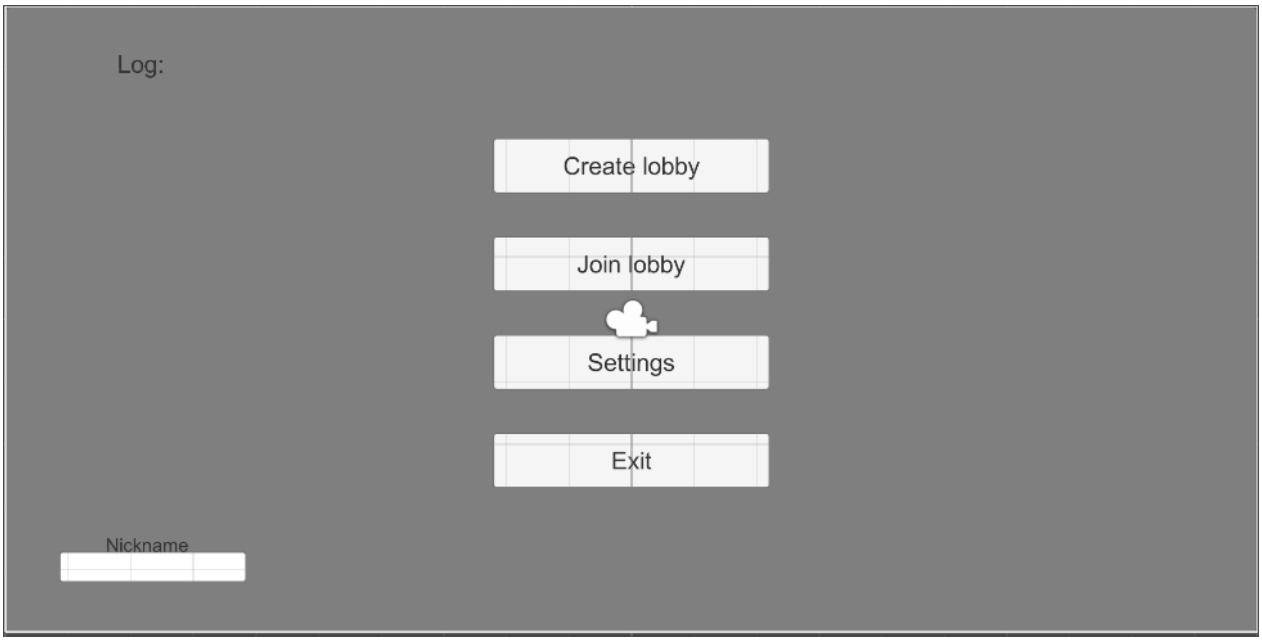


Рисунок 1.9. MainMenuPanel

Після цього створюю ще дві Panel назвавши їх Create Lobby Panel і Join Lobby і додам необхідні елементи.

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.

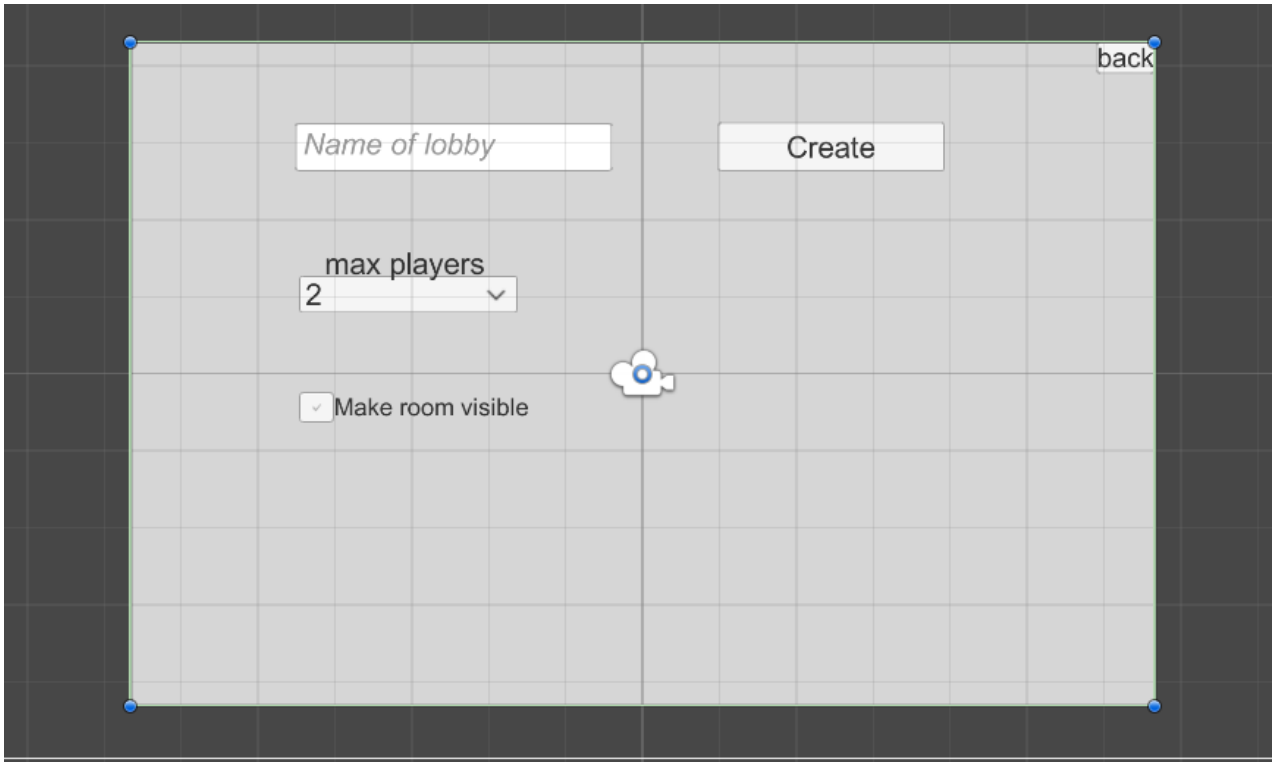


Рисунок 1.10. CreateLobbyPanel

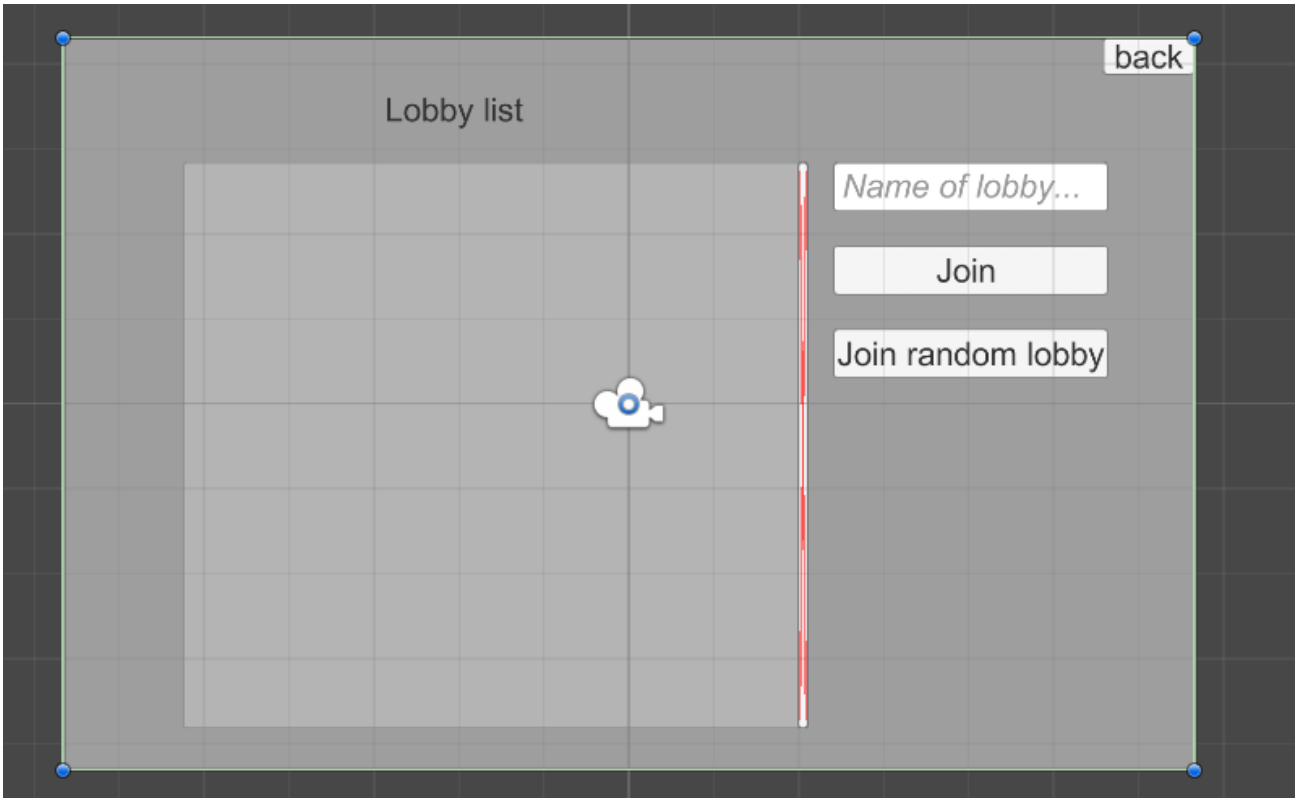


Рисунок 1.11. JoinLobbyPanel

Далі додам фон та підготовлені спрайти для кнопок та інших елементів інтерфейсу.

Рисунок 1.12. MainMenuPanel після додавання спрайтів

Рисунок 1.13. CreateLobbyPanel після додавання спрайтів

Рисунок 1.14. JoinLobbyPanel після додавання спрайтів

Наступним кроком створимо два скрипти UIManager та PhotonNetworkManager. Скрипти в Unity пишуться на мові програмування C# і прикріплюються до ігрових об'єктів як компоненти. Вони мають доступ до всіх інших компонентів об'єкта та можуть змінювати їхні властивості, викликати методи та реагувати на події. Таким чином, скрипти є "мозком" ігрових об'єктів, визначаючи їх поведінку та взаємодію з іншими об'єктами у сцені.

UIManager буде відповідати за взаємодію з інтерфейсом користувача. Керувати відображенням різних панелей (головне меню, створення лобі, приєднання до лобі), отримувати дані від гравця (наприклад, нікнейм, назву кімнати) та передавати їх у PhotonNetworkManager для виконання мережевих операцій.

PhotonNetworkManager необхідний обробки мережевої логіки. Цей скрипт буде встановлювати з'єднання з серверами Photon, управляти створенням і приєднанням до кімнат, обробляти події мережевої взаємодії (наприклад, підключення та оновлення списку кімнат).

На діаграмі нижче показано, як Unity упорядковує і повторює функції подій протягом усього часу існування скрипта.

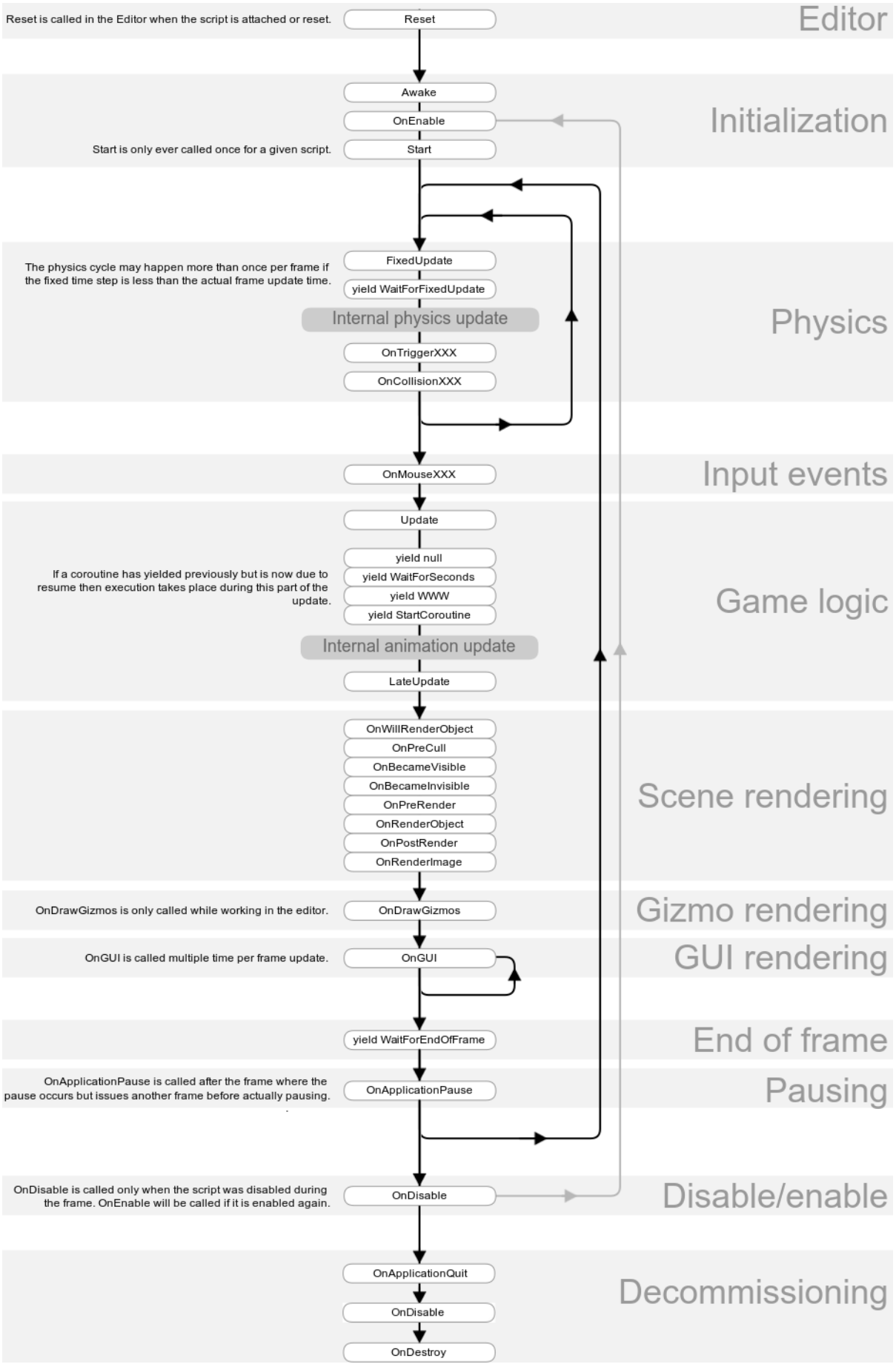


Рисунок 1.15. Діаграма життєвого циклу скрипта

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();
}
```

таким чином ми ініціюємо підключення до серверів Photon, використовуючи дані з файлу для зберігання конфігурації PhotonServerSettings. Цей файл містить основні налаштування підключення, такі як App ID, регіон сервера та інші параметри.

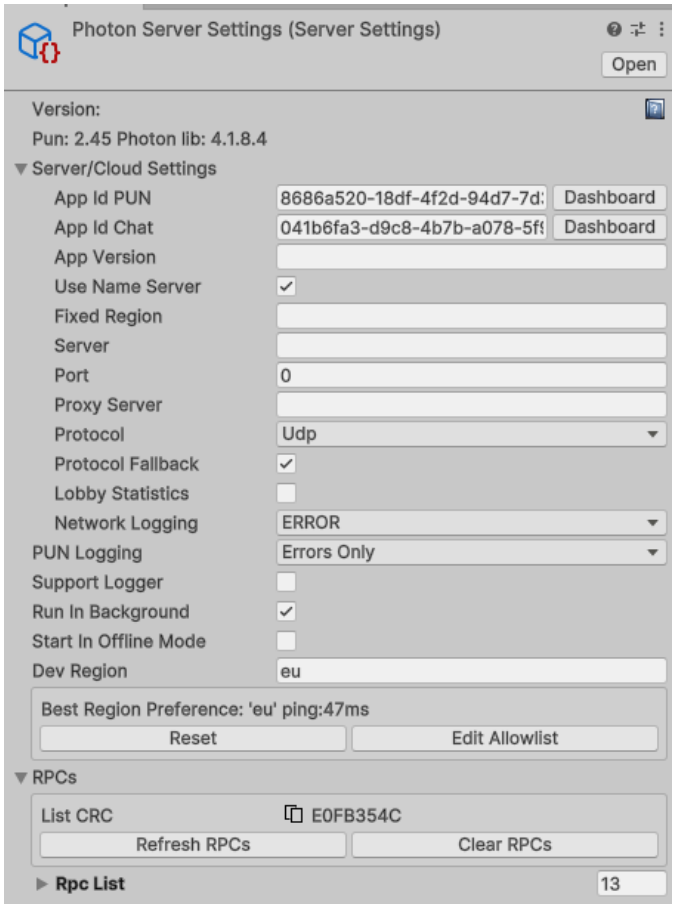


Рисунок 1.16. PhotonServerSettings

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

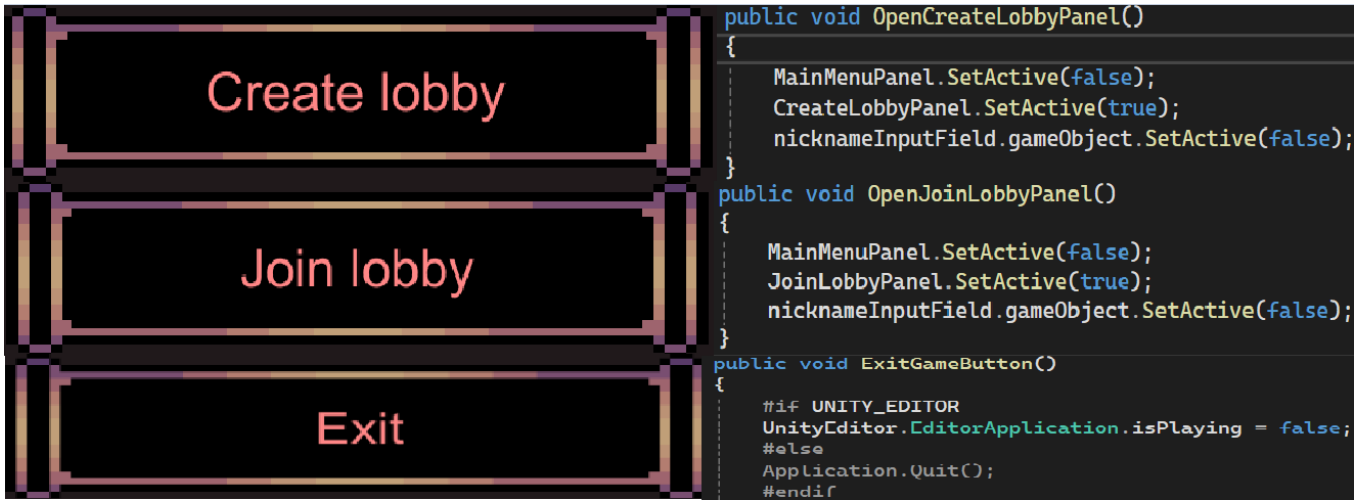


Рисунок 1.17. Код кнопок

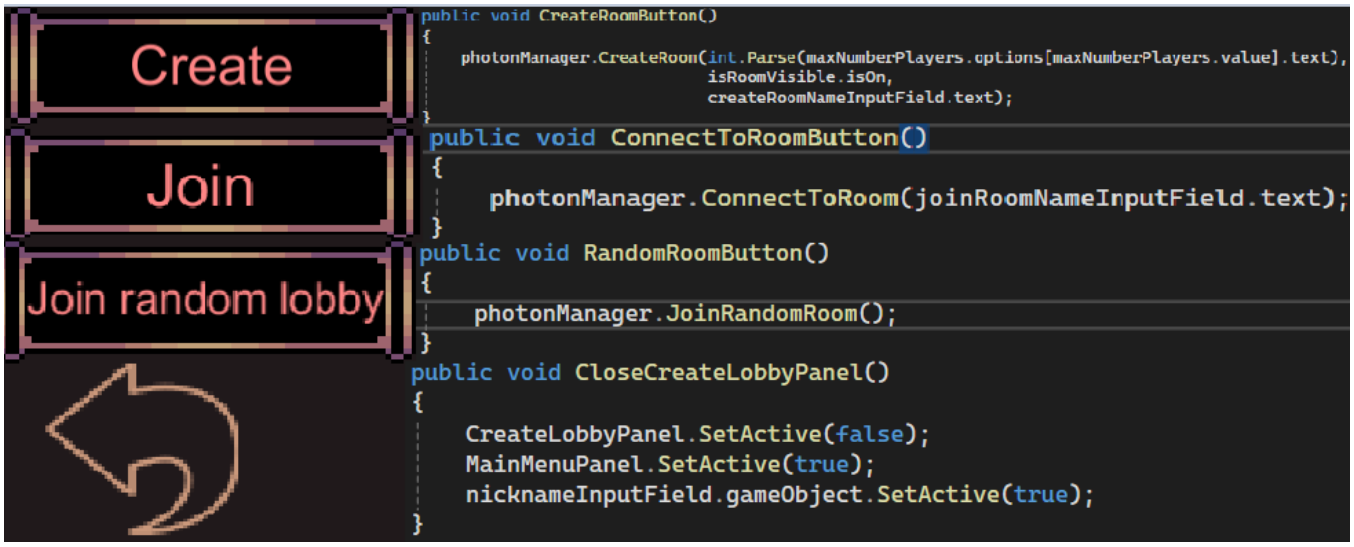


Рисунок 1.18. Код кнопок

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 йде спроба отримати рядок із ключем "PlayerNickname" із PlayerPrefs, і, якщо такий ключ не знайдений, повертається порожній рядок (""). Потім йде перевірка, чи порожній цей рядок, і, якщо ні він встановлюється в полі введення InputField, якщо ж отриманий рядок порожній, то генерується випадковий нікнейм у форматі "PlayerXXX", де XXX - випадкове число від 1 до 999, і вже після цього встановлюється в InputField. Ну у кінці викликається метод SetPlayerNickname скрипта PhotonNetworkManager, щоб встановити згенерований (або завантажений з PlayerPrefs) нікнейм Photon Network.

Рисунок 1.19. Нікнейм гравця

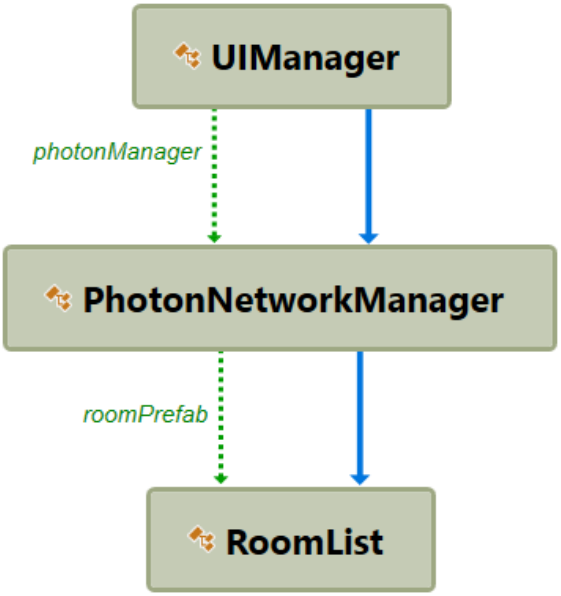


Рисунок 1.20. Діаграма класів, що відповідають за інтерфейс користувача та мережеву логіку.

1.7.2 Створення рівня гри

Для створення ігрового рівня був використан Tilemap. Це дозволило швидко створити та компонувати 2D рівень за допомогою комбінації спрайтів, що представляють різні елементи оточення (підлога, стіни, тощо). Tilemap також надає зручні можливості для налаштування порядку шарів, що важливо для правильного відображення елементів рівня, та додавання колайдерів за допомогою компонента Tilemap Collider, що забезпечує взаємодію персонажів та об'єктів з оточенням.

Для зручного управління спрайтами та їх розміщення на сцені було створено Tile Palette. Цей інструмент дозволяє організувати спрайти у вигляді палітри, з якої їх можна легко вибирати та розміщувати на Tilemap.

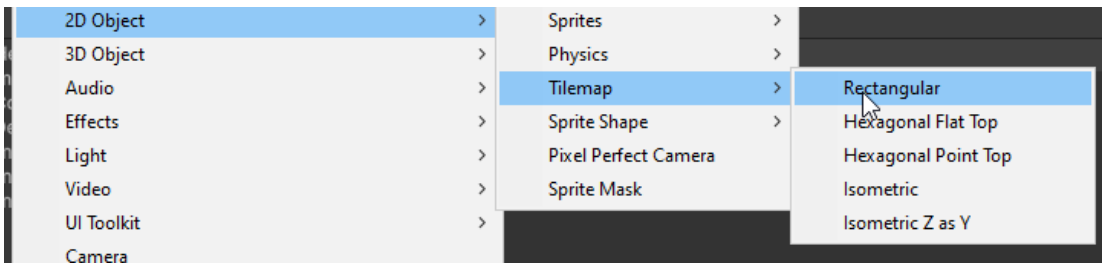


Рисунок 1.21. Створення Tilemap

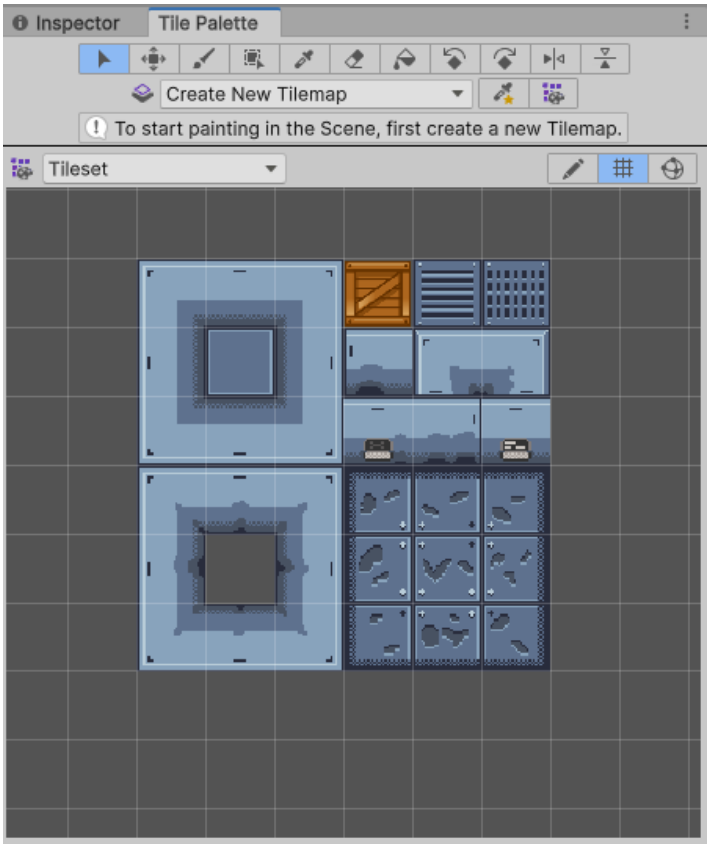


Рисунок 1.22. Створення Tile Palette

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.

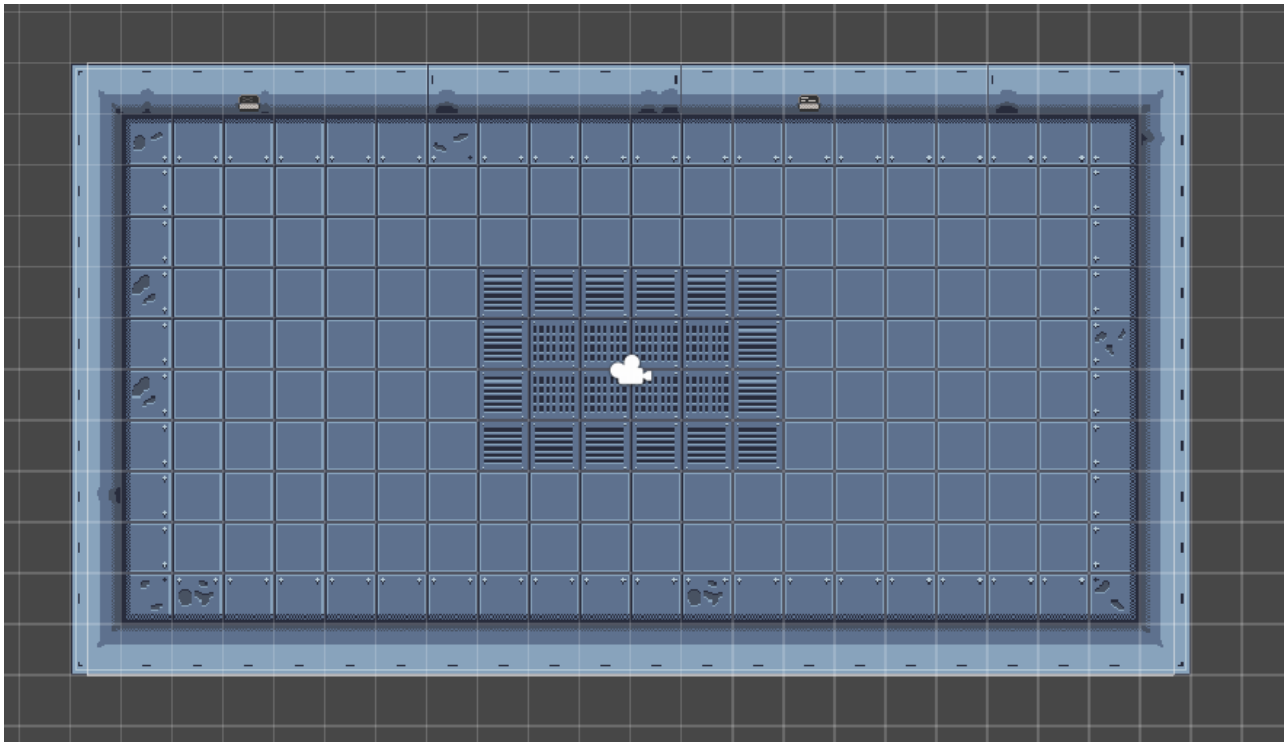


Рисунок 1.24. Tilemap "Indestructible"

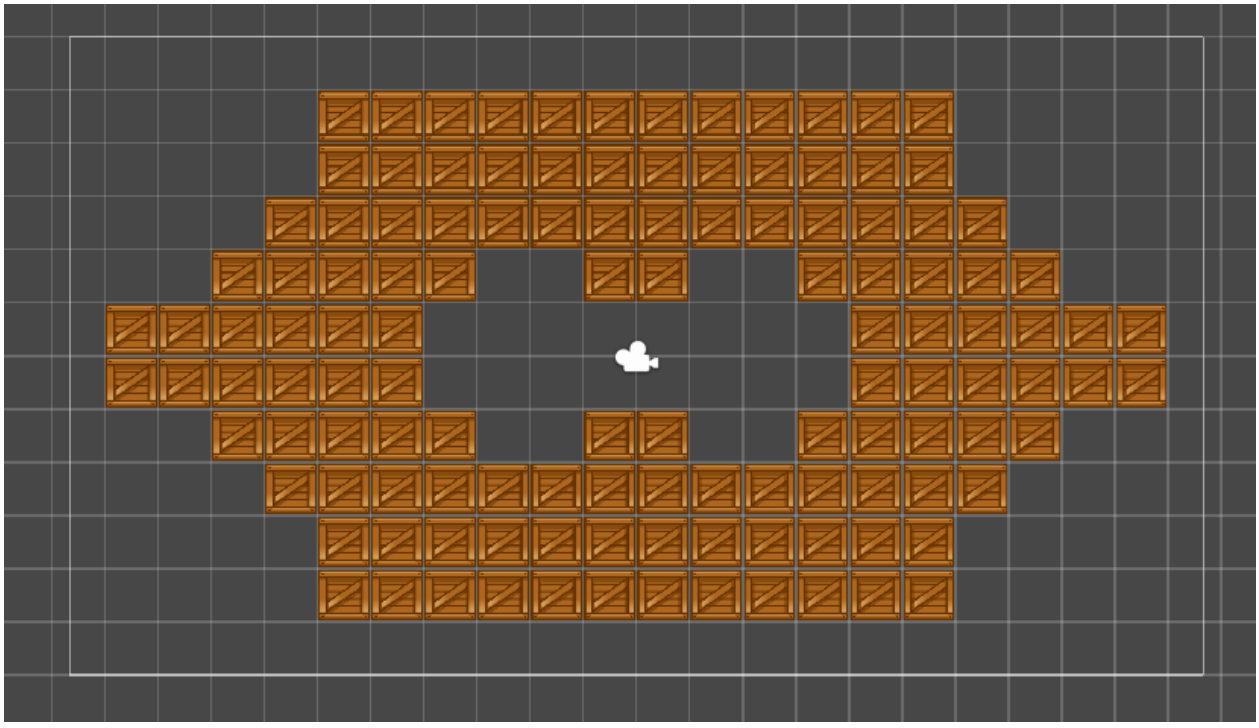


Рисунок 1.25. Tilemap "Destructible"

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.

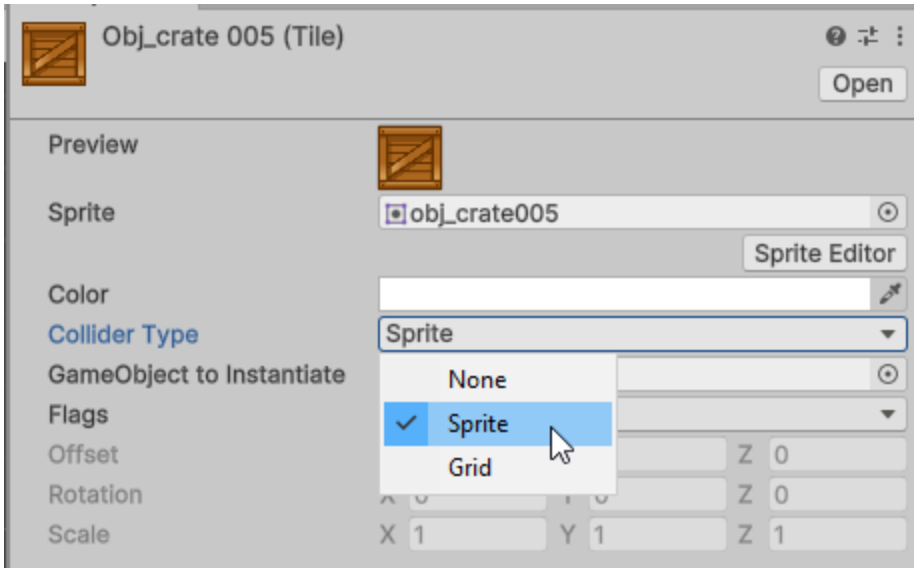


Рисунок 1.26. Зміна типу колайдера

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.

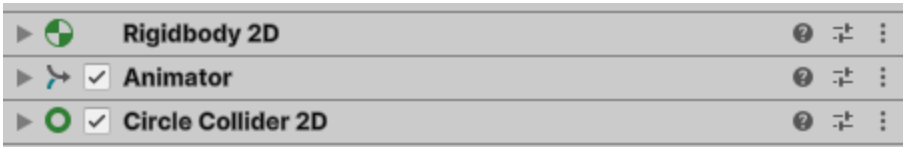


Рисунок 1.27. Компоненти об'єкта - «Player»

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)

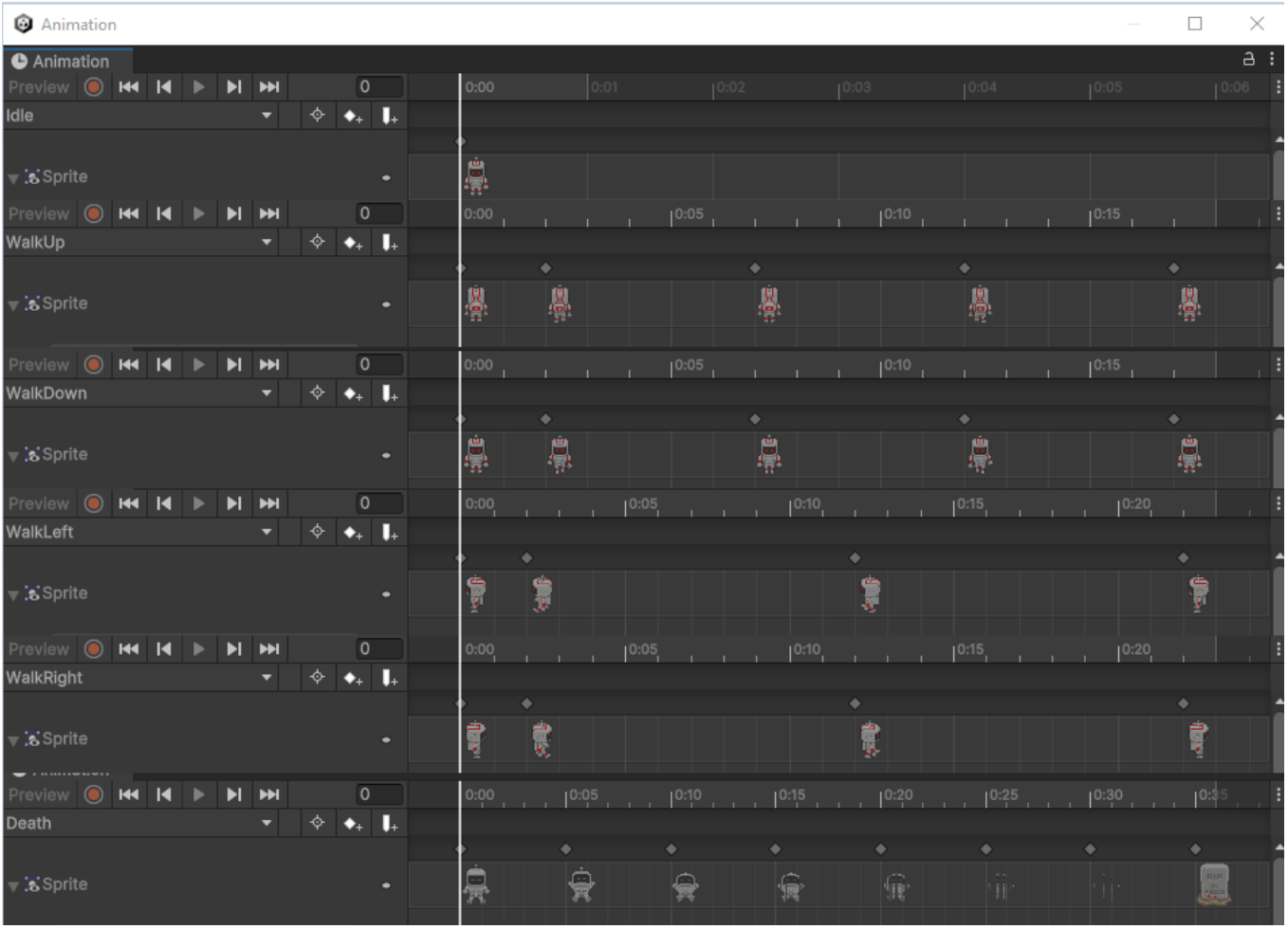


Рисунок 1.28. Анімації гравця

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.

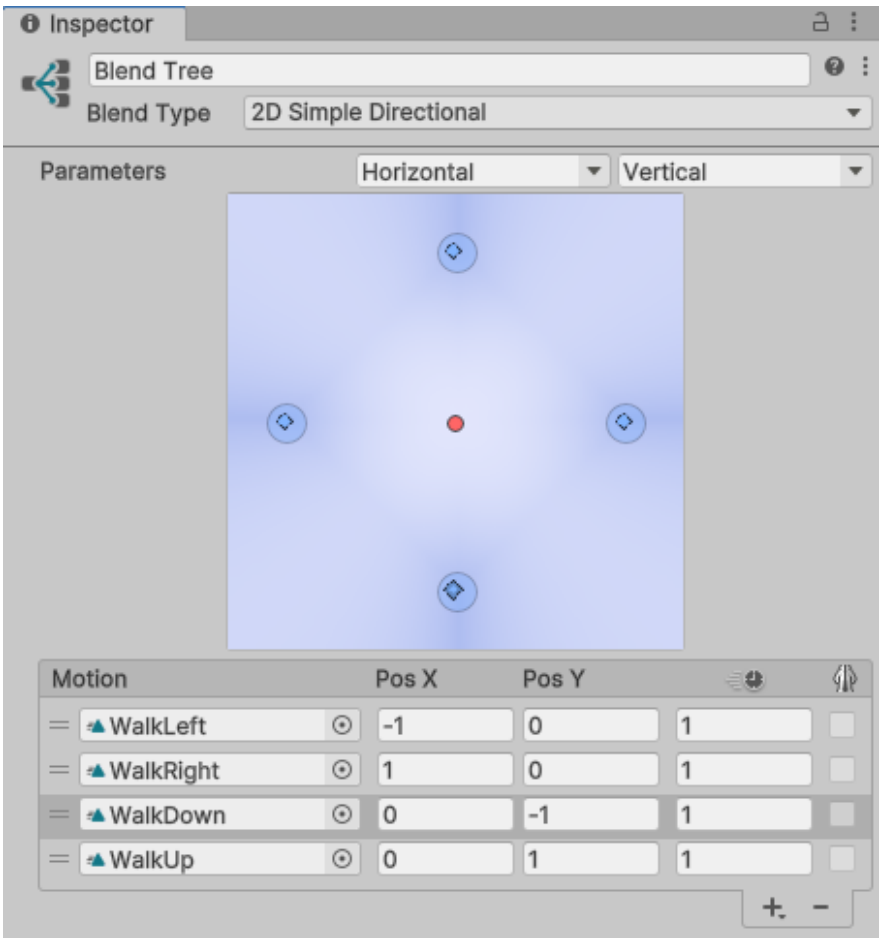


Рисунок 1.29. Blend Tree

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.

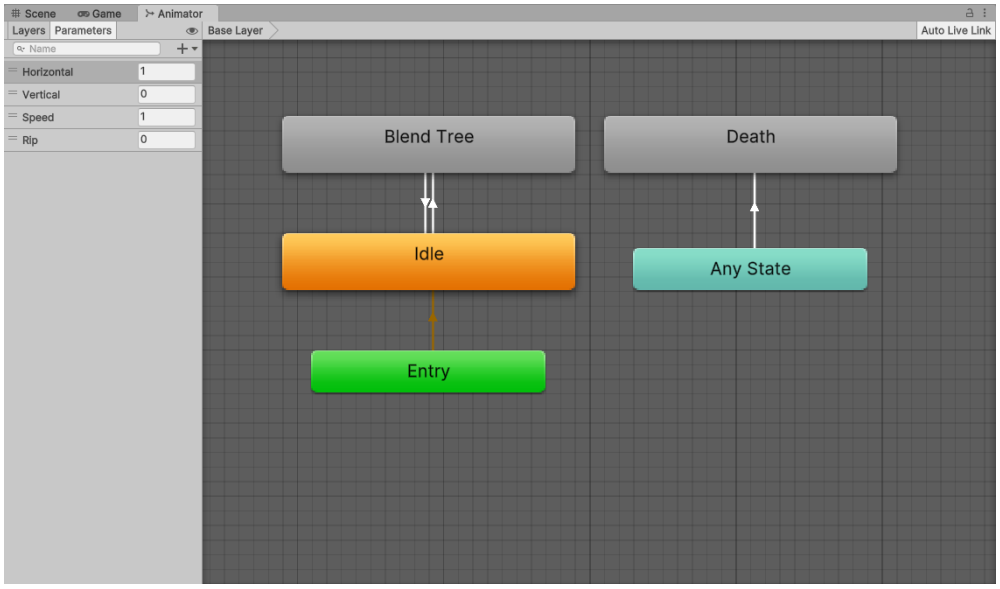


Рисунок 1.30. Animator

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());
        }
    }
```

Цей метод є зворотним викликом, який Photon PUN автоматично викликає для кожного компонента PhotonView, прикріпленого до об'єкта. Він використовується для серіалізації та десеріалізації даних гравця, щоб забезпечити їх синхронізацію між усіма клієнтами.

Щоб Photon міг створити екземпляр гравця на всіх клієнтах за допомогою PhotonNetwork.Instantiate, необхідно створити префаб гравця. Для цього створюю папку з назвою "Resources" у директорії проекту Unity та перетягую об'єкт гравця ("Player") з ієрархії сцени в папку Resources. Це автоматично створить префаб гравця, який Photon Network зможе використовувати для створення копій об'єкта на всіх підключених клієнтах, забезпечуючи синхронізацію та мережеву взаємодію.

Рисунок 1.31. Префаб гравця на сцені

1.7.4 Реалізація ігрових механик

Створимо префаб бомби так само як і з гравцем помістивши її в папку Resources і додавши на неї компонент Photon View.

Рисунок 1.32. Префаб бомби на сцені

Для реалізації логіки роботи бомби створюю скрипт PhotonBomb та додамо його до префабу бомби. Цей скрипт буде відповідає за активацію бомби, створення вибуху та знищувати зруйновані об'єкти.

```
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 активацію та детонацію бомби. Після затримки (bombFuseTime - 0.1f) для візуального ефекту запалювання, бомба вибухає у своїй поточній позиції. Створюється об'єкт вибуху (explosionPrefab) і встановлюється радіус вибуху(explosionRadius), який потім поширюється у чотирьох напрямках методом Explode(), перевіряючи наявність руйнівних об'єктів. Після цього бомба знищується.

Для керування встановленням бомб гравцем створюю скрипт BombController та додаю його до об'єкта гравця. Цей скрипт відповідає за обробку натискання кнопки встановлення бомби, створення екземпляра бомби на всіх клієнтах, обмеження кількості доступних бомб, а також за налаштування параметрів створеної бомби (радіус вибуху, тривалість вибуху, час горіння запалу) та активацію її корутини ActiveBomb(), яка відповідає за детонацію.

```
[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");
        }
    }
}
```

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

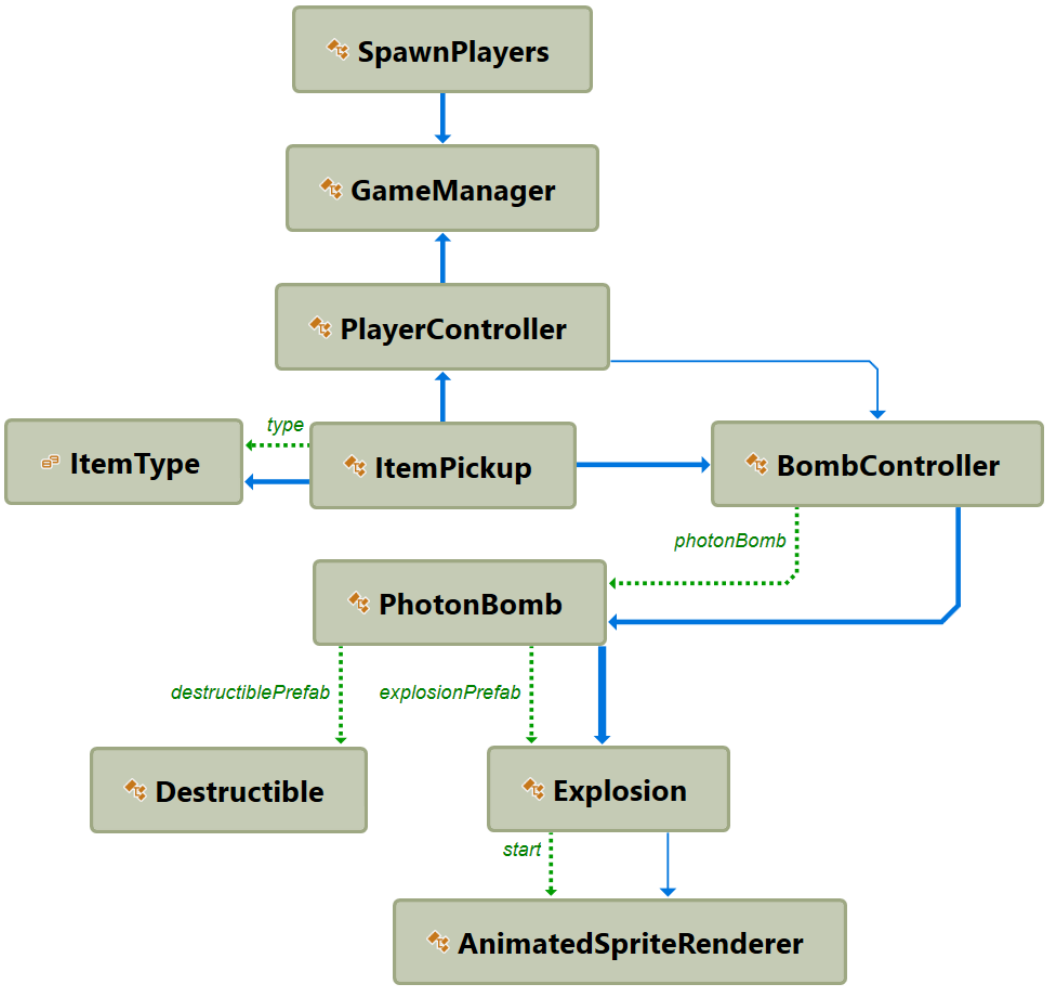


Рисунок 1.33. Діаграма класів, що відповідають за реалізації ігрової логіки.

1.7.5 Керування персонажем

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

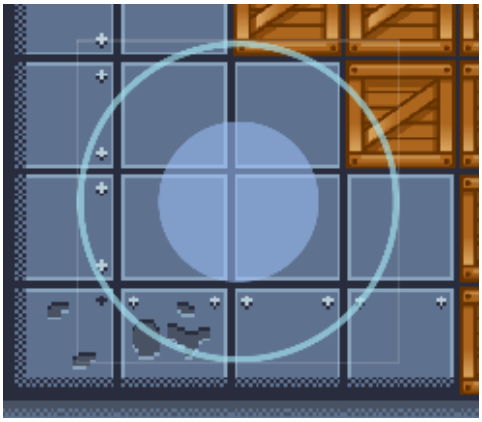


Рисунок 1.34. Віртуальний джойстик

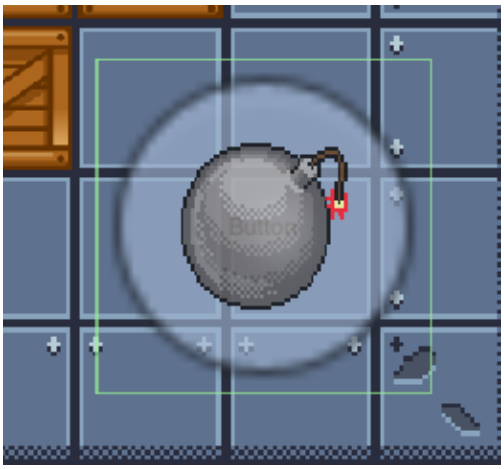


Рисунок 1.35. Кнопка встановлення бомби

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	Успішне створення кімнати з	Так

	приєднання до кімнати	заданими параметрами або приєднання до існуючої кімнати.	
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. Інтерфейс зручний для використання на різних мобільних пристроях з різними роздільними здатностями екрану.	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 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

2.1 Вступ

Охорона праці - це система заходів, покликана забезпечити збереження життя і здоров'я людей в процесі виконання ними своїх трудових обов'язків.

Охорона праці містить безліч аспектів таких як норми освітлення, рівня шуму, мікроклімату, розмір робочого місця і безліч інших, не менш важливих пунктів.

Нині комфорт та безпеку робітників забезпечують нормативно- правові акти, закони та положення. Адже, якщо працівник працюватиме у негативній для нього обстановці або під впливом негативних факторів і на робочому місці не забезпечуватиметься належні умови праці, тоді цей працівник буде швидше втомлюватися, допускати помилки, що може стати причиною розвитку професійної хвороби або виробничої травми.

В дипломному проєкті охорона праці розглядається безпека праці на робочому місці програміста, який займається розробкою програми з використанням персонального комп'ютера.

2.2 Аналіз та безпека умов праці працівника на робочому місці

Шкідливими факторами на робочому місці програміста можуть стати чинники що приведенні у таблиці нижче

Таблиця 2.1. Шкідливи чинники

Джерело	Чинники	
	Щкідливі	Небезпечні
1.Робота за персональним комп’ютером.	1. Підвищена або знижена температура повітря робочої зони. 2. Підвищена або знижена рухливість повітря.	1. Підвищене значення напруги в електричному ланцюзі, замикання якої може статися через тіло людини.

	3. Підвищена або знижена іонізація повітря 4. Статичні перевантаження кістково-м'язового апарату і динамічні локальні перевантаження м'язів кистей рук. 5. Перенапруження зорового аналізатора. 6. Монотонність праці.	

Ця робота не вимагає будь-яких фізичних навантажень і належить до категорії 1а. Під час виконання роботи використовується персональний комп'ютер.

2.3 Організація робочого місця.

Робочий стіл з урахуванням характеру виконуваної роботи повинен мати достатній розмір для раціонального розміщення монітора (дисплея), клавіатури, іншого використовуваного обладнання та документів, поверхня, що володіє низькою відбивною здатністю. Клавіатура розташовується на поверхні столу таким чином, щоб простір перед клавіатурою було достатнім для опори рук працівника (на відстані не менше ніж 300 мм від краю, зверненого до працівника). Щоб забезпечувалося зручність зорового спостереження, швидке і точне зчитування інформації, площина екрану монітора розташовується нижче рівня очей працівника переважно перпендикулярно до нормальної лінії погляду працівника (нормальна лінія погляду - 15° вниз від горизонталі).

Для виключення впливу підвищених рівнів електромагнітних випромінювань відстань між екраном монітора і працівником має становити не менше 500 мм (оптимальне 600-700 мм). Робочий стілець (крісло) повинен бути стійким, місце сидіння повинно регулюватися по висоті, а спинка сидіння - по висоті, кутах нахилу, а також відстані спинки від переднього краю сидіння. Для

тих, кому це зручно, передбачається підставка для ніг.

2.4 Мікроклімат

Згідно до ДСанПіН 3.3.6--42-99 мікроклімат приміщення повинен відповідати нормативам.

Таблиця 2.2. Мікроклімат приміщень

Пора року	Температура повітря у градусах Цельсія.	Відносна вологість повітря у відсотках.	Швидкість руху повітря у метрах за секунду.
Холодна	22-24	40-60	0,1
Тепла	23-25	40-60	0,1

Це оптимальні умови для забезпечення максимально комфортного теплового балансу температури тіла людини і його терморегуляції. Якщо температура вище норми, кровоносні судини розширюються і тепловіддача в навколишнє середовище зростає. При зниженні температури кровоносні судини відповідно звужуються приплив крові до тіла сповільнюється і тепловіддача зменшується.

На терморегуляцію організму впливає також вологість повітря. Занадто висока вологість (більше 85 %) ускладнює терморегуляцію, а занадто низька (менше 20%) викликає пересихання слизових, причому не тільки дихальних шляхів, але і очей.

Не менш важлива оптимальна вологість в приміщенні: чим вона вища, тим слабкіше вплив електростатичних і електромагнітних полів, рівень випромінювання яких в приміщенні, де встановлений комп'ютер, завжди підвищений.

Таблиця 3.3. Рівень іонів у повітрі

Рівні	Кількість на сантиметр кубічний.	
	n+	n-
Мінімальне	400	600
Оптимальне	1500-300	3000-5000
Максимальне	50000	50000

Рівень іонів у повітрі що повинні відповідати санітарно-гігієнічним нормам № 2152-80.

Для підтримки мікроклімату слід використовувати кондиціонери та зволожувачі повітря або інші прилади.

2.5 Освітлення

Робоче місце з використанням ПК розміщується таким чином, щоб природне світло падало збоку (бажано зліва). Для зниження яскравості в полі зору при природному освітленні застосовуються регульовані жалюзі, щільні штори. Світильники загального і місцевого освітлення повинні створювати нормальні умови освітленості і відповідний контраст між екраном і навколишнім оточенням з урахуванням виду роботи і вимог видимості з боку працівника.

Освітленість на поверхні столу в зоні розміщення робочого документа повинна становити 300-500 люкс. Можливі відблиски на екрані монітора і іншому обладнанні, що заважають відображенню, усуваються шляхом відповідного розміщення екрану, обладнання, розташування світильників місцевого освітлення. При рядному розміщенні робочих столів розташування екранів відеомоніторів назустріч один одному через їх взаємне відображення не допускається

У разі використовувані штучного освітлення слід використовувати люмінесцентні лампи. Допускається використання металогалогенних ламп потужністю 250Вт. Також допускається використовування ламп розжарювання для місцевих світильників. Використовування будь яких світильників без розсіювачів заборонено.

2.6 Електробезпека

Конструктивні заходи забезпечують захист від випадкового дотику до струмопровідних частин за допомогою захисних оболонок і ізоляції струмоведучих частин.

Схемно-конструктивні заходи призначені для забезпечення захисту від ураження електричним струмом при дотику до металевих оболонок, які можуть опинитися під напругою в ре-зультаті аварії. У даному приміщенні в комп'ютерах застосовується занулення. Біля монітора передбачена подвійна ізоляції.

Необхідно дотримуватися правил техніки безпеки при роботі з високою напругою і наступних запобіжних засобів:

- монтаж, обслуговування, ремонт і наладка ЕОМ, заміна деталей, пристосувань, блоків повинна здійснюватися тільки при повному відключенні живлення;
- заземлення конструкції приміщення мають бути надійно захищені діелектричними щитками або сітками від випадкового дотику.

2.7 Пожежна безпека

Пожежна безпека входить в комплекс заходів з охорони праці, і організаційна робота в цій сфері на об'єктах господарювання включає широкий спектр заходів, а саме:

- створення умов для безпечної праці,
- мінімізації ризику виникнення пожеж,
- своєчасне і повноцінне забезпечення технічними засобами для запобігання займання та усунення самих пожеж та їх наслідків,
- контроль дотримання протипожежних вимог і норм законодавства,
- розробка і впровадження регламентів по гасінню пожеж, евакуації тапорятунку з місць пожежі й задимлення людей і майна (матеріальних цінностей)
- внутрішнє і зовнішнє навчання співробітників.

Джерелами займання у виробничих приміщеннях з ПЕОМ можуть бути: Іскра при розряді статичної електрики; іскри від електробладнання; іскри від удару і тертя; відкрите полум'я.

Первинні засоби пожежогасіння застосовуються для боротьби з пожежами на початковій стадії. До них належать: пожежні кран-комплекти, вогнегасники, пожежний інвентар (резервуари з водою, ящики з піском, пожежні відра, лопати), а також різний переносний пожежний інструмент (кирки, сокири, багри, ломи і т. ін.).

Коли від пожежі захищаються приміщення з персональними комп'ютерами,

то слід враховувати специфіку вогнегасних речовин у вогнегасниках, які призводять під час гасіння до псування обладнання. Ці приміщення рекомендується оснащувати вуглекислотними вогнегасниками з урахуванням гранично допустимої концентрації вогнегасної речовини.

Вуглекислотні вогнегасники вважаються найефективнішими для гасіння електрообладнання та електроприладів. З недоліків вуглекислоти можна відзначити тільки шкідливе випаровування цієї речовини. Саме тому забороняється гасити електроустановки в закритих приміщеннях.

- Вуглекислота не залишає слідів після випаровування і в той же час не пошкоджує запалене електрообладнання. Це особливо важливо при гасінні комп'ютерної техніки або загорівся телевізора.
- Вуглекислотними вогнегасниками можна гасити електроустановки під напругою до 10000 Вольт (10 кВ).

Також при роботі з ПЕОМ, на робочому місці забороняється мати вогненебезпечні речовини. Перш за все, не можна допускати накопичення паперових відходів поблизу ПЕОМ і їх несвоєчасного прибирання.

У приміщеннях забороняється: запалювати вогонь; включати електрообладнання, якщо в приміщенні пахне газом; курити; сушити що-небудь на опалювальних приладах; закривати вентиляційні отвори в електроапаратурі. Підключення ПЕОМ до електромережі необхідно проводити через мережеві фільтри. Мережеві фільтри повинні підключатися до електромережі через розетки з заземлюючими контактами.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто поставленої мети – створено мобільну 2D-гру під Android на платформі Unity , використовуючи сучасні технології та інструменти, такі як Unity, C# та Photon PUN. Цей проект демонструє ефективність використання даних технологій для створення високоякісних та захопливих мобільних ігор.

При виконанні роботи було вивчено жанри мобільних ігор і проведено аналіз тенденцій ринку мобільних ігор, що дозволило виявити зростаючий попит на багатокористувацькі ігри з динамічним ігровим процесом. На основі отриманих даних було прийнято рішення створити мультиплеєрну гру у жанрі аркадного екшену. Аналіз існуючих розробок допоміг визначити ключові особливості та механіки, які мають бути реалізовані в грі, щоб забезпечити її конкурентоспроможність та привабливість для гравців.

Було здійснено детальне проєктування та реалізацію ігрової механіки, включаючи правила гри, поведінку персонажів, взаємодію з об'єктами та бомбами. Розроблено інтуїтивно зрозумілий графічний інтерфейс, адаптований під сенсорні екрани мобільних пристроїв, що забезпечує зручність та комфортність керування грою. За допомогою мережевого Photon PUN було створено стабільну та синхронізовану мережеву взаємодію між гравцями. Це дозволило реалізувати гру в режимі реального часу, де гравці можуть бачити дії один одного, змагатися та взаємодіяти в рамках ігрового процесу.

Гра пройшла ретельне функціональне тестування, яке підтвердило коректну роботу всіх ігрових механік, включаючи рух персонажів, встановлення та вибух бомб, збір бонусів, синхронізацію між гравцями та взаємодію з інтерфейсом користувача. Прототип успішно протестований на різних мобільних пристроях під управлінням Android, що гарантує його стабільну роботу та сумісність.

Розроблена гра може слугувати основою для подальшого розвитку та вдосконалення, додавання нових рівнів, персонажів, режимів гри та інших функцій.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Васильєв О.М. Алгоритми. К.: Ліра-К, 2022 рік, 424 с.

2. Бублик В.В Об’єктно-орієнтоване програмування: ІТкнига, 2015. – 624 с.

3. Геврик Є. О. Охорона праці Навч. посіб. для студ. вищ. навч. закл. К. : Ніка-Центр, 2005. - 294 с.

4.Троелсен Ендрю, Джетікс Філіп. Мова програмування C# 7 і платформи .NET і .NET Core, 2018. Видавництво Вільямс. 1328 с.

5. Роберт Мартін. Чистий код: створення, аналіз, рефакторинг: Фабула, 2019, 416 с.

6. Goldstone W. Unity Game Development Essentials. / W Goldstone. Birmingham: Packt Publishing Ltd. – 2009. – 316 с.

7. Thorn A. Mastering Unity Scripting / Alan Thorn., 2015. – 380 с.

8. Hocking J. Unity in Action. Multiplatform game development in C# with Unity 5 / Joseph Hocking., 2015. – 352 с

9. Jared Halpern, Developing 2D Games with Unity: Independent Game Programming with C#, 2018. Apress; 1st ed. Edition. 410 с.

10. Paris Buttfield-Addison Unity Game Development Cookbook: Essentials for Every Game, 2019. O'Reilly Media; 1st edition. 408 с.

12. Unity Documentation https://docs.unity.com/

13. document about: PUN 2 https://doc.photonengine.com/pun/current/getting-started/pun-intro

14. https://stackoverflow.com/

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"));
        }
        else
        {
            selfpos = (Vector3)stream.ReceiveNext();

            animator.SetFloat("Horizontal", (float)stream.ReceiveNext());
            animator.SetFloat("Vertical", (float)stream.ReceiveNext());
            animator.SetFloat("Speed", (float)stream.ReceiveNext());
        }
    }
}
```

Файл BombController.cs

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

public class BombController : MonoBehaviour
{
    public Button bombButton;
    [Header("Bomb")]
    public GameObject photonBombPrefab;
    public float bombFuseTime = 3f;
    public int bombAmount = 1;
    private int bombsRemaining;

    [Header("Explosion")]
    public float explosionDuration = 1f;
    public int explosionRadius = 1;

    private PhotonView view;
    private PhotonBomb photonBomb;
    private GameObject bomb;

    private void Start()
    {
        GameObject find = GameObject.FindGameObjectWithTag("Boom");
        bombButton = find.GetComponent<Button>();
        view = GetComponent<PhotonView>();
        bombButton.onClick.AddListener(PlaceBombButton);

    }

    private void OnEnable()
    {
        bombsRemaining = bombAmount;
    }

    public void PlaceBombButton()
    {
        if (view.IsMine)
        {
            Debug.Log("бомба поставлена");
```

```
            if (bombsRemaining > 0)
            {
                StartCoroutine(PlaceBomb());
            }
        }
    }

    private IEnumerator PlaceBomb()
    {
        float offset = 0.5f;
        Vector2 position = transform.position;
        position.x = Mathf.Round(position.x);
        position.y = Mathf.Round(position.y - offset);
        bomb = PhotonNetwork.Instantiate(photonBombPrefab.name, position,
Quaternion.identity);
        StartCoroutine(DelayedRPCActivation());

        IEnumerator DelayedRPCActivation()
        {
             yield return new WaitForSeconds(0.1f);
             view.RPC("RPC_ActiveBomb", RpcTarget.All,
             bomb.GetComponent<PhotonView>().ViewID);
        }
        bombsRemaining--;
        yield return new WaitForSeconds(bombFuseTime);
        bombsRemaining++;
    }

    public void AddBomb()
    {
        bombAmount++;
        bombsRemaining++;
    }

    private void OnTriggerExit2D(Collider2D other)
    {
        if (other.gameObject.layer == LayerMask.NameToLayer("Bomb"))
        {
            other.isTrigger = false;
        }
    }
    [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");
        }
    }
}
```

Файл PhotonBomb.cs

```
using Photon.Pun;
using System.Collections;
using UnityEngine;
using UnityEngine.Tilemaps;

public class PhotonBomb : MonoBehaviour
{

    [Header("Bomb")]
    public float bombFuseTime = 3f;

    [Header("Explosion")]
    public Explosion explosionPrefab;
    public LayerMask explosionLayerMask;
    public float explosionDuration = 1f;
    public int explosionRadius = 1;

    [Header("Destructible")]
    private Tilemap destructibleTiles;
    public Destructible destructiblePrefab;
    private PhotonView view;

    private void Start()
    {
        GameObject destructibleObject = GameObject.Find("Destructible");
        view = GetComponent<PhotonView>();
        // Получить компонент Tilemap из найденного объекта
        destructibleTiles = destructibleObject.GetComponent<Tilemap>();

    }

    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);
    }

    private void Explode(Vector2 position, Vector2 direction, int length)
    {
        if (length <= 0)
        {
            return;
```

```
        }

        position += direction;

        if (Physics2D.OverlapBox(position, Vector2.one / 2f, 0f,
explosionLayerMask))
        {
            ClearDestructible(position);
            return;
        }

        Explosion explosion = Instantiate(explosionPrefab, position,
Quaternion.identity);
        explosion.SetActiveRenderer(length > 1 ? explosion.middle : explosion.end);
        explosion.SetDirection(direction);
        explosion.DestroyAfter(explosionDuration);
        Explode(position, direction, length - 1);

    }

    private void ClearDestructible(Vector2 position)
    {
        Vector3Int cell = destructibleTiles.WorldToCell(position);
        TileBase tile = destructibleTiles.GetTile(cell);


        if (tile != null)
        {
            PhotonNetwork.Instantiate(destructiblePrefab.name, position,
Quaternion.identity);
            destructibleTiles.SetTile(cell, null);
            Debug.Log("Клетка уничтожена");
        }

    }

}
```

Файл Destructible.cs

```
using Photon.Pun;
using UnityEngine;

public class Destructible : MonoBehaviour
{
    public float destructionTime = 1f;
    [Range(0f, 1f)]
    public float itemSpawnChance = 0.2f;
    public GameObject[] spawnableItems;

    private void Start()
    {
        Destroy(gameObject, destructionTime);
    }

    private void OnDestroy()
    {
        // Только мастер-клиент выполняет спавн предметов
        if (PhotonNetwork.IsMasterClient)
        {
            if (spawnableItems.Length > 0 && Random.value < itemSpawnChance)
            {
                int randomIndex = Random.Range(0, spawnableItems.Length);
                PhotonNetwork.Instantiate(spawnableItems[randomIndex].name,
transform.position, Quaternion.identity);
```

```
            }
        }

    }

}
```

Файл SpawnPlayers.cs

```
using UnityEngine;
using Photon.Pun;


public class SpawnPlayers : MonoBehaviourPunCallbacks
{
    public GameObject playerPrefab;
    public Transform[] spawnPoints;
    public BombController BombController;
    void Start()
    {
            Spawn();
    }

    void Spawn()
    {
        Transform spawnPoint = spawnPoints[Random.Range(0, spawnPoints.Length)];
        GameObject player = PhotonNetwork.Instantiate(playerPrefab.name,
      spawnPoint.position, Quaternion.identity);
        GameManager.Instance.RegisterPlayer(player);
    }

}
```

Файл PickupItem.cs

```
using UnityEngine;

public class PickupItem : MonoBehaviour
{
    public enum ItemType
    {
        ExtraBomb,
        BlastRadius,
        SpeedIncrease,
    }

    public ItemType type;

    private void OnItemPickup(GameObject player)
    {
        switch (type)
        {
            case ItemType.ExtraBomb:
                player.GetComponent<BombController>().AddBomb();
                break;

            case ItemType.BlastRadius:
                player.GetComponent<BombController>().explosionRadius++;
                break;

            case ItemType.SpeedIncrease:
                player.GetComponent<PlayerController>().speed += 0.5f;
                break;
        }
```

```
        Destroy(gameObject);
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player")) {
            OnItemPickup(other.gameObject);
        }
    }

}
```

Файл GameManager.cs

```
using Photon.Pun;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviourPunCallbacks
{
    [SerializeField] private GameObject PlayerSpawner;
    public static GameManager Instance { get; private set; }
    private List<GameObject> players = new List<GameObject>();
    private void Awake()
    {
        if (Instance != null)
        {
            DestroyImmediate(gameObject);
        }
        Instance = this;
        DontDestroyOnLoad(gameObject);
    }
    public void RegisterPlayer(GameObject player)
    {
        players.Add(player);
    }
    void Start()
    {
        PlayerSpawner.SetActive(true);
    }


    public void CheckWinState()
    {
        int aliveCount = 0;

        foreach (GameObject player in players)
        {
            if (player != null && player.activeSelf)
            {
                aliveCount++;
            }
        }

        if (aliveCount <= 1)
        {
            photonView.RPC(nameof(NewRound), RpcTarget.All);
        }
    }

    [PunRPC]
    private void NewRound()
```

```
    {
            Destroy(Instance.gameObject);
            Instance = null;
            PhotonNetwork.LoadLevel(SceneManager.GetActiveScene().buildIndex);
    }

}
```

Файл AnimatedSpriteRenderer.cs

```
using UnityEngine;

[RequireComponent(typeof(SpriteRenderer))]
public class AnimatedSpriteRenderer : MonoBehaviour
{
    private SpriteRenderer spriteRenderer;

    public Sprite idleSprite;
    public Sprite[] animationSprites;

    public float animationTime = 0.25f;
    private int animationFrame;

    public bool loop = true;
    public bool idle = true;

    private void Awake()
    {
        spriteRenderer = GetComponent<SpriteRenderer>();
    }

    private void OnEnable()
    {
        spriteRenderer.enabled = true;
    }

    private void OnDisable()
    {
        spriteRenderer.enabled = false;
    }

    private void Start()
    {
        InvokeRepeating(nameof(NextFrame), animationTime, animationTime);
    }

    private void NextFrame()
    {
        animationFrame++;

        if (loop && animationFrame >= animationSprites.Length) {
            animationFrame = 0;
        }

        if (idle) {
            spriteRenderer.sprite = idleSprite;
        } else if (animationFrame >= 0 && animationFrame < animationSprites.Length)
{
            spriteRenderer.sprite = animationSprites[animationFrame];
        }
    }

}
```

Файл Explosion.cs

```
using UnityEngine;

public class Explosion : MonoBehaviour
{
    public AnimatedSpriteRenderer start;
    public AnimatedSpriteRenderer middle;
    public AnimatedSpriteRenderer end;

    public void SetActiveRenderer(AnimatedSpriteRenderer renderer)
    {
        start.enabled = renderer == start;
        middle.enabled = renderer == middle;
        end.enabled = renderer == end;
    }

    public void SetDirection(Vector2 direction)
    {
        float angle = Mathf.Atan2(direction.y, direction.x);
        transform.rotation = Quaternion.AngleAxis(angle * Mathf.Rad2Deg,
Vector3.forward);
    }

    public void DestroyAfter(float seconds)
    {
        Destroy(gameObject, seconds);
    }

}
```

Файл LobbyManager.cs

```
using Photon.Pun;
using Photon.Realtime;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class LobbyManager : MonoBehaviourPunCallbacks
{

    [SerializeField] private Button[] lobbyButtons;
    [SerializeField] private Text[] playerNameTexts;
    [SerializeField] private GameObject lobbyPanel;
    [SerializeField] private GameObject background;
    [SerializeField] private GameObject GameManager;
    [SerializeField] private GameObject joystick;
    [SerializeField] private GameObject escPanel;
    [SerializeField] private GameObject esc;

    [PunRPC]
    private void StartGameForAll()
    {
        GameManager.SetActive(true);
        joystick.SetActive(true);
        esc.SetActive(true);
        lobbyPanel.SetActive(false);
        background.SetActive(false);
    }

    public void EscButton()
    {
        esc.SetActive(false);
        joystick.SetActive(false);
```

```
    escPanel.SetActive(true);
}

public void ContinueButton()
{
    esc.SetActive(true);
    joystick.SetActive(true);
    escPanel.SetActive(false);
}

public void StartGameButton()
{
    if (PhotonNetwork.IsMasterClient)
    {
        // Вызываем RPC метод для всех клиентов
        photonView.RPC("StartGameForAll", RpcTarget.All);
    }
}

void Start()
{
    int maxPlayers = PhotonNetwork.CurrentRoom.MaxPlayers;

    // Активируем только соответствующее количество кнопок лобби
    for (int i = 0; i < lobbyButtons.Length; i++)
    {
        lobbyButtons[i].gameObject.SetActive(i < maxPlayers);
    }

    UpdateLobbyUI();
}
void Update()
{

}
public void Leave()
{
    Debug.Log("Button Clicked");
    PhotonNetwork.LeaveRoom();
}

public override void OnLeftRoom()
{
    SceneManager.LoadScene(0);
}

public override void OnPlayerEnteredRoom(Player otherPlayer)
{
    UpdateLobbyUI();
    Debug.LogFormat("Player {0} entered room", otherPlayer.NickName);
}

public override void OnPlayerLeftRoom(Player otherPlayer)
{
    UpdateLobbyUI();
    Debug.LogFormat("Player {0} left room", otherPlayer.NickName);
}

void UpdateLobbyUI()
{
    int playerCount = PhotonNetwork.CurrentRoom.PlayerCount;
    for (int i = 0; i < lobbyButtons.Length; i++)
    {
        // Если игрок существует на данной позиции, отображаем его никнейм
```

```
            if (i < playerCount)
            {
                playerNameTexts[i].text = PhotonNetwork.PlayerList[i].NickName;
            }
            // В противном случае, устанавливаем текст на "Empty slot"
            else
            {
                playerNameTexts[i].text = "Empty slot";
            }
        }
    }
}
```

Файл ListItem.cs

```
using UnityEngine;
using UnityEngine.UI;
using Photon.Realtime;
using Photon.Pun;

public class ListItem : MonoBehaviour
{
    [SerializeField] Text textName;
    [SerializeField] Text textPlayerCount;

    public void SetInfo(RoomInfo info)
    {
        textName.text = info.Name;
        textPlayerCount.text = info.PlayerCount + "/" + info.MaxPlayers;
    }

    public void OnButtonClicked()
    {
        PhotonNetwork.JoinRoom(textName.text);
        Debug.Log("имя комнаты:" + textName.text);
    }
}
```

ДОДАТОК Б

Слайди мультимедійної презентації

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

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

за спеціальністю: 123 «Комп'ютерна інженерія»

на тему: «Розробка мобільної 2d гри під андроїд на платформі unity»

Розробив: Спатарь Павел
гр.2БКС-28
Керівник роботи: Гаджиєв М.М.

м. Одеса 2024 р.

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.

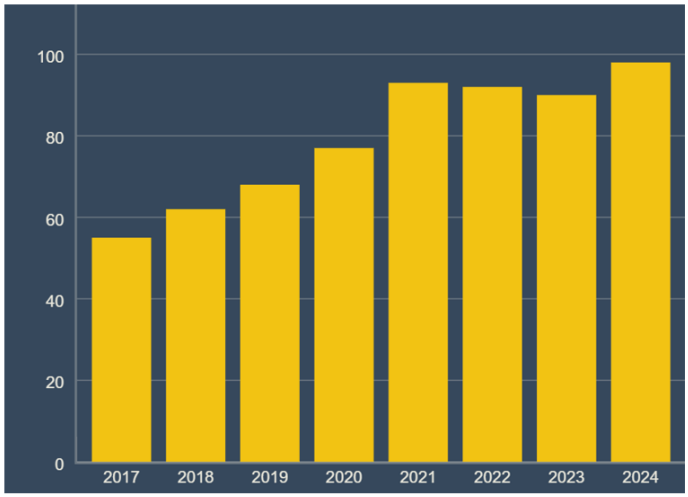
Метою даної кваліфікаційної роботи є розробка мобільної 2d гри під андроїд на платформі unity.

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

- Аналіз ринку мобільних ігор та існуючих розробок;
- Вибір оптимальних інструментів та технологій;
- Проектування 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

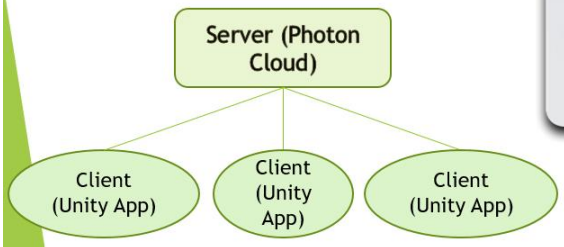
•Platform-native DLLs

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:

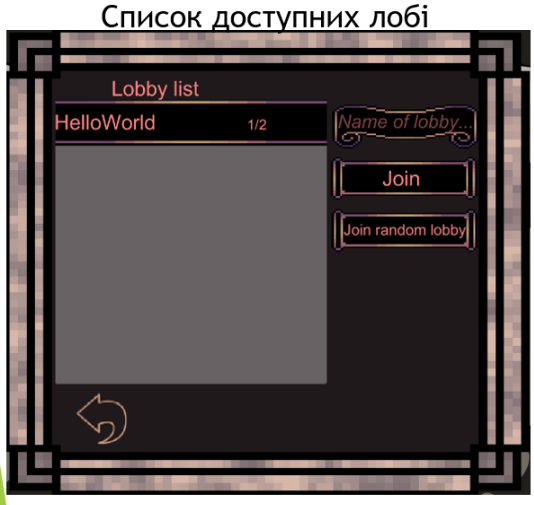
- Гравці пересуваються по ігровому полю та встановлюють бомби.
- Бомби вибухають через певний час, знищуючи перешкоди та інших гравців на своєму шляху.
- Знищені перешкоди можуть містити бонуси, що покращують характеристики гравця(швидкість, кількість бомб, силу вибуху).
- Перемагає останній гравець,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

Інтерфейс користувача

Головне меню

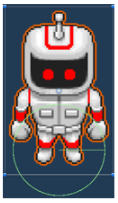
Створення лобі

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 кваліфікаційної 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 проекту.
- 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 даних та можливість гри в режимі реального часу.

Створена гра може слугувати основою для подальшого розвитку та вдосконалення, додавання нових рівнів, персонажів, режимів гри та інших функцій.

Дякую за увагу

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника про кваліфікаційну роботу бакалавра

Спатарь Павло Костянтинович

(прізвище, ім'я та по батькові здобувача/здобувачки освіти)

Освітньо-професійна програма *«Комп'ютерна інженерія»*

Спеціальність *123 «Комп'ютерна інженерія»*

Тема кваліфікаційної роботи

Розробка ігрового мобільного додатку під андроїд на Unity

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) обсяг і якість виконання роботи (розрахунково-пояснювальної записки)

Бакалаврська робота виконана якісно, у достатньому обсязі, на 68 стор. друкованому тексту, графічна частина у 18 слайдах відповідно до стандартам і ДСТУ. Розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у бакалаврської роботи. Презентація виконана якісно, у достатньому обсязі. Презентація наочно демонструє результати роботи. Список літератури складено в достатньому обсязі та відповідає темі кваліфікаційної роботи.

б) самостійність роботи над кваліфікаційною роботою

Студент самостійно обрав напрям та тематику кваліфікаційної роботи. Провів аналіз існуючих рішень та зробив необхідні висновки для реалізації проекту. Основні аналітичні та практичні результати студентом отримані самостійно

в) теоретична підготовка бакалавра

Відповідає вимогам, що надаються до бакалавра зі спеціальності «Комп'ютерна інженерія»

г) вміння розв'язувати виробничі та конструкторські питання _____

Дипломник вміє поставити реальні виробничі завдання досить високого технічного рівня і вирішити їх із застосуванням досягнень науки і техніки в області комп'ютерних наук, апаратних засобів і програмування. При цьому дипломник досить добре орієнтується в питаннях мережевих технологій і програмного забезпечення. Знайомий в останніх досягненнях в галузі комп'ютерних технологій і мережних систем.

Добре володіє умінням і знанням в області сучасних технологій, компонентів РЕА та їх застосуванням для вирішення завдань технікік і науки.

Оцінка розрахункової частини **5 (відмінно)**

Оцінка графічної (презентаційної) частини **5 (відмінно)**

Загальна оцінка **5 (відмінно)**

Прізвище, ім'я, по батькові керівника роботи

д.т.н., проф. Гаджиев М.М.

Місце роботи і посада керівника роботи

проф. каф. ІПЗ ДУІТЗ

«17» 06 2024 р. _____ **Гаджиев М.М.**

(підпис) (прізвище та ініціали керівника)

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра
відділення комп'ютерних систем

Спатарь Павло Костянтинович

(прізвище, ім`я та по батькові)

Напрямку підготовки *123 «Комп'ютерна інженерія»*

Керівник кваліфікаційної роботи ***Гаджиєв М.М.***

(прізвище, ім`я та по батькові)

Тема кваліфікаційної роботи ***«Розробка ігрового мобільного додатку під андроїд на платформі*** **Unity»**

Обсяг пояснювальної записки 67 сторінок

Обсяг графічної (презентаційної) частини проекту 17 аркушів (слайдів)

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) заключення про ступінь відповідності виконаної роботи завданню

Представлена на рецензію випускна кваліфікаційна робота відповідає затвердженій темі та виконаний відповідно до технічного завдання. Випускна робота має актуальну тематику щодо аналізу та практично створення мобільного ігрового додатка під андроід на платформи Unity.

б) характеристика виконання кожного розділу роботи

Пояснювальна записка складається з основного розділу, розділу охорони праці та додатку. Основний розділ пояснювальної записки містить підрозділи, що поетапно охоплюють аналітичну частину, реалізацію суті роботи, дослідження ефективності прийнятих рішень. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора ЕОТ

в) оцінка якості виконання графічної (презентаційної) частини роботи і пояснювальної записки

Графічна частина складається з 18 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять креслення та ілюстративні схеми, таблиці, графіки, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм оформлення документів. Якість виконання графічної частини роботи та пояснювальної записки висока, розробку виконано у повному обсязі

г) перелік позитивних якостей роботи _____

Проаналізовано саме актуальні методи створення мобільного ігрового додатка;
У роботі виконано практичну реалізацію методів створення мобільного ігрового додатка під андроїд на платформі Unity;
Розроблені рекомендації щодо їх подальшої модернізації.

д) основні недоліки роботи _____

З тексту пояснювальної записки не дуже зрозуміло, наскільки сильно рекомендований методи та ігровий додаток відрізняються від аналогів за якісними параметрами;
У розділі охорони праці наведені відомі нормативні вимоги загального плану замість конкретних розрахунків освітлення приміщення, вентиляції, рівня шуму.

Оцінка розрахункової частини 5 (відмінно)

Оцінка графічної (презентаційної) частини 5 (відмінно)

Загальна оцінка 5 (відмінно)

Прізвище, ім'я та по батькові рецензента доц. Кільдішев В.І.

Місце роботи і посада рецензента

доц.каф. КБ та ТЗІ ДУІТЗ

«18» 06 2024 р. _____ доц. Кільдішев В.І.

(підпис) (прізвище та ініціали рецензента)

ПІДПИС ПОСВІДЧУЮ
НАЧАЛЬНИК ВІДДІЛУ
КАДРІВ ДУІТЗ
18.06.2024

ВІДДІЛ КАДРІВ

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

ID перевірки:
1016376764

Дата перевірки:
19.06.2024 23:32:34 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
19.06.2024 23:36:14 EEST

ID користувача:
100011688

Назва документа: **2БКС-28_Спатарь**

Кількість сторінок: **45** Кількість слів: **6719** Кількість символів: **50875** Розмір файлу: **2.40 MB** ID файлу: **1016184921**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

1.43%
Схожість

Найбільша схожість: **0.22%** з Інтернет-джерелом (https://forum.photonengine.com/discussion/comment/42702)

1.43% Джерела з Інтернету	86	..... Сторінка 47

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	6

Підозріле форматування	12 сторінок

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Спатарь Павло Костянтинович,
здобувач освіти гр. 2БКС-28, та

Гаджиєв Матін Магсудович,
керівник випускної кваліфікаційної роботи,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи бакалавра на тему:

«Розробка мобільної 2d гри під андроїд на платформі unity» (автор роботи – Спатарь П.К., керівник роботи – Гаджиєв М.М.)

виконаної у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 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 _____ / Спатарь П.К. /

Керівник _____ / Гаджиєв М.М. /

«13» червня 2024 р.