Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Гордієнко Євгеній Олегович
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Ігровий Android-застосунок у жанрі «Shoot 'em up»
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2101098.01.01.ПЗ

Виконав студент III курсу, група ІПЗс-21-1 _____ Євгеній ГОРДІЄНКО
Підпис Ім'я, ПРІЗВИЩЕ

Керівник д-р фіз.-мат. наук, проф _____ Леонід БЕДРАТЮК
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

11 січня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
02 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Гордієнко Євгеній Олегович
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Ігровий Android-застосунок у жанрі «Shoot 'em up»

Керівник кваліфікаційної роботи Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 08.01.2024 р. № 6
2. Строк подання студентом роботи на кафедру 01.06.2024 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Три креслення:
1. Діаграма послідовності
2. Діаграма компонентів
3. Діаграма варіантів використання

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Перевірила Ю.І. канд. пед наук, доцент		
Антиплагіат	Форкун Юрій Вікторович, доцент	30.06.24	30.06.24

7. Дата видачі завдання « 02 » січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проектування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12– 31.12.2023	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2024	
3 Проектування програмного забезпечення	21.02 – 20.03 2024	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2024	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2024	
6 Попередній захист КвР	травень 2024	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2024	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2024	

Студент _____ Є.О. ГОРДІЄНКО
Підпис Ініціали, ПРІЗВИЩЕ

Керівник роботи _____ Л.П. Бедратюк
Підпис Ініціали, ПРІЗВИЩЕ

АНОТАЦІЯ

Автор роботи: Гордієнко Євгеній Олегович.

Керівник роботи: Бедратюк Леонід Петрович.

Пояснювальна записка: 108 с., 52 рис., 6 табл., 3 дод., 6 джерел.

Графічна частина: 18 презентаційних слайдів.

Ключові слова: ANDROID, ІГРОВИЙ РУШІЙ, МУЛЬТИПЛЕЄР, ПЛАГІН, СЕССІЇ, UNREAL ENGINE.

Метою роботи є створення мобільного Android-застосунку у жанрі «Shoot 'em up», який дозволяє користувачам ефективно проводити вільний час, граючи як самостійно, так і з іншими гравцями.

У кваліфікаційній роботі проведено аналіз предметної області, визначено вимоги до ігрового застосунку, спроектовано структуру застосунку та інтерфейс. Для розробки програмної системи використано ігровий рушій Unreal Engine та мову програмування C++ з плагінами AutoSettings та StarSphere.

Результатом проектування є реалізація однокористувацької та багатокористувацької частин ігрового застосунку. Практична значимість проекту полягає у створенні зручного та захоплюючого ігрового застосунку, який забезпечує користувачам можливість грати в будь-якому місці та в будь-який час. Ступінь впровадження проекту полягає у розробці повноцінного продукту, готового до випуску на ринок. Область застосування цього застосунку охоплює широку аудиторію мобільних гравців, що робить його потенційно комерційно привабливим продуктом.

Висновки та пропозиції щодо розвитку об'єкта розроблення: надалі можна розглянути можливість додавання нових режимів гри, покращення графічних елементів та розширення функціональності для залучення ще більшої аудиторії.

06.06.2024
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2101098.01.01.ПЗ	Пояснювальна записка	108		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
4	А4		Презентаційні матеріали	18		
			Графічні документи			
5	А3	КвРІПЗ.2101098.01.01.Е8	Діаграма послідовності	1		
6	А3	КвРІПЗ.2101098.01.01.Е8	Діаграма компонентів	1		
7	А3	КвРІПЗ.2101098.01.01.Е8	Діаграма варіантів	1		
			використання			

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2101098.01.01.ВД		
Виконав		Гордієнко Є. О.		06.06	Ігровий Android-застосунок у жанрі «Shoot 'em up» Відомість документів	Літ.	Арк. 1 / Аркушів 1
Керівник		Бедратюк Л.П		06.06			
Консультант		Говорущенко Т.О.		12.06			
Н. контр.		Праворська Н.І.		08.06		ХНУ, ІПЗс-21-1	
Зав. каф.		Бедратюк Л.П.		06.06			

ЗМІСТ

ВСТУП

У сучасному світі важко уявити людину без мобільного телефону, планшета, смартфона чи іншого портативного мультимедійного пристрою. Смартфони сьогодні є невід'ємною частиною нашого повсякденного життя, замінюючи собою безліч інших пристроїв, таких як класичні портативні ігрові системи, фотоапарати, MP3-плеєри та багато іншого. Їхня популярність як ігрових платформ обумовлена не лише портативністю, але й широкою доступністю, адже нині майже кожна людина користується смартфоном. Завдяки цьому мобільна ігрова індустрія стрімко розвивається, створюючи продукти різноманітних жанрів, що дозволяє кожному користувачеві знайти гру на свій смак.

Мобільні ігри завжди "під рукою", тому їх можна запускати будь-де і будь-коли, що робить їх особливо привабливими для сучасних користувачів. Вони стали невід'ємною частиною нашого дозвілля, дозволяючи відволіктися від повсякденних турбот та отримати задоволення у будь-який час і в будь-якому місці. Сьогоднішні смартфони мають достатньо потужності, щоб забезпечити високу якість графіки та продуктивність, порівняну з класичними портативними ігровими системами, такими як Nintendo DS або Playstation Portable. Ця портативність та доступність призвели до того, що мобільні ігри стали більш популярними, ніж будь-коли раніше.

Одним із популярних жанрів мобільних ігор є жанр «Shoot 'em up». Цей жанр, який виник ще в 1960-х роках, передбачає управління персонажем або технічним засобом, що бореться з великою кількістю ворогів за допомогою стрілянини. На піку своєї популярності жанр був відомий під назвою "shooter" і здебільшого реалізовувався у двовимірному форматі. З появою тривимірних ігор жанр «Shoot 'em up» розширився і став більш різноманітним. Нині, через високі вимоги до графічної потужності, такі ігри рідко виходять для персональних комп'ютерів, але залишаються популярними на портативних пристроях завдяки їх доступності та простоті.

Актуальність теми кваліфікаційної роботи обумовлена кількома чинниками. По-перше, жанр «Shoot 'em up» є класичним і має широку базу прихильників, які готові спробувати нові продукти у цьому жанрі. Багато гравців пам'ятають класичні ігри цього жанру з часів аркадних автоматів та перших домашніх консолей, що створює додатковий інтерес до нових ігор, які пропонують сучасний підхід до класичного геймплею. По-друге, популярність мобільних ігор забезпечує значний потенціал для залучення широкої аудиторії. Мобільні ігри завжди "під рукою", тому їх можна запускати будь-де і будь-коли, що робить їх особливо привабливими для сучасних користувачів. По-третє, сучасні технологічні можливості дозволяють створювати високоякісні та захоплюючі ігрові продукти навіть для мобільних платформ. Використання таких інструментів, як Unreal Engine, дозволяє створювати ігри з високою якістю графіки та складним геймплеєм, що раніше було доступно лише на стаціонарних платформах.

Метою даного проєкту є створення мобільної гри у жанрі «Shoot 'em up», головним завданням якої є забезпечення гравців захоплюючим ігровим досвідом та можливістю здобувати якомога більше очок. Ця гра повинна бути цікавою як для новачків, так і для досвідчених гравців, пропонуючи різні рівні складності та різноманітні режими гри. Для досягнення цієї мети необхідно виконати наступні завдання:

– Визначити специфіку предметної області та дослідити сучасні тенденції розвитку жанру «Shoot 'em up» на мобільних платформах. Це включає аналіз ринку мобільних ігор, вивчення попиту на ігри цього жанру та оцінку конкурентного середовища.

– Провести аналіз схожого програмного забезпечення, визначити їхні переваги та недоліки, щоб уникнути типових помилок та впровадити найкращі практики у розробку. Це дозволить визначити, які елементи геймплею та дизайну користувачі цінують найбільше, а також які аспекти можуть бути покращені.

– Підібрати та проаналізувати засоби для створення ігрових додатків, обґрунтувати вибір конкретних інструментів та технологій для реалізації проєкту. Використання сучасних технологій, таких як Unreal Engine та мова програмування C++, дозволить створити високоякісний продукт з використанням передових графічних та програмних можливостей.

– Спроектувати архітектуру гри, розробити інтерфейс користувача та продумати ігровий процес, забезпечивши інтуїтивність та зручність користування. Важливо створити такий інтерфейс, який буде зручним та інтуїтивно зрозумілим для гравців, а також забезпечити плавний та захоплюючий ігровий процес.

– Реалізувати проект програмним шляхом, використовуючи ігровий рушій Unreal Engine та мову програмування C++. Це включає створення основних механік гри, розробку графічних елементів, а також інтеграцію різних плагінів для забезпечення багатокористувацького режиму та інших функцій.

– Здійснити тестування ігрового додатку, виявити та виправити можливі помилки, забезпечити стабільність та продуктивність роботи гри на різних мобільних пристроях. Тестування є ключовим етапом розробки, оскільки воно дозволяє виявити та виправити помилки, забезпечити високу якість продукту та гарантувати, що гра буде працювати стабільно на різних пристроях.

У процесі реалізації даного проєкту буде використано сучасні методи та засоби інформаційних технологій, що дозволять створити якісний та конкурентоспроможний продукт. Ігровий рушій Unreal Engine, який забезпечує високу якість графіки та широкі можливості для розробки, а також мова програмування C++, що забезпечує ефективну роботу з ресурсами, дозволять реалізувати всі необхідні функції гри. Використання плагінів AutoSettings та StarSphere дозволить додати до гри додаткові функції, такі як налаштування параметрів гри та створення захоплюючих ігрових світів.

Практична значимість даної роботи полягає у створенні нового ігрового продукту, який може залучити широку аудиторію користувачів та стати

комерційно успішним. Мобільний застосунок, розроблений у рамках даного проєкту, забезпечить користувачам можливість грати як самостійно, так і з іншими гравцями, що підвищить його привабливість та конкурентоспроможність на ринку мобільних ігор. Додатково, можливість багатокористувацького режиму дозволить гравцям взаємодіяти між собою, змагатися та спільно проходити рівні, що значно підвищить інтерес до гри.

У підсумку, розробка нової мобільної гри у жанрі «Shoot 'em up» сприятиме розвитку мобільної ігрової індустрії, надаючи користувачам якісний та захоплюючий ігровий досвід. Подальший розвиток проєкту може включати додавання нових режимів гри, покращення графічних елементів та розширення функціональності, що дозволить залучити ще більшу аудиторію та забезпечити довготривалий успіх продукту на ринку. В майбутньому можна розглянути можливість додавання нових рівнів, персонажів, зброї та інших елементів, що робитиме гру ще більш захоплюючою та цікавою для користувачів.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, й структурних та функціональних особливостей

IT-фахівець - це широке поняття, що охоплює представників різних професій, які працюють у сфері інформаційних технологій, такі як програмісти, розробники, адміністратори мереж і баз даних, модератори, фахівці з робототехніки, інформаційної безпеки, web-дизайнери і 3D-аніматори. З розвитком інформаційних технологій у всіх сферах діяльності з'являються нові професії для IT-фахівців. Дослідження в галузі IT та думка аналітиків підтверджують, що фахівці з IT є високо затребуваними і залишатимуться такими і найближчим часом. Зокрема, на сьогоднішній день по всьому світу з'являється багато перспективних ігрових розробок і проектів.

GameDev (геймдев) - це скорочення від Game Development (розробка ігор). У процес розробки гри входить: розробка дизайну ігрового процесу, програмування ігрового рушія або використання готових рішень, розробка візуального концепту і його складових, створення графіки і моделювання об'єктів, створення музичного та звукового супроводу.

Відеогра - це програма або додаток, призначений для розваги або навчання, який може використовуватися як одним гравцем, так і декількома.

Сучасні ігри доступні на різних платформах, від персональних комп'ютерів до мобільних телефонів. Це різноманіття платформ породило велику кількість жанрів, які можуть бути характерними для певної платформи, і залежно від популярності жанру гра може бути портована на інші пристрої.

При виборі жанру для певної платформи, слід спочатку враховувати, наскільки зручно буде грати на наявних пристроях керування. Наприклад, жанр Shoot`em up спочатку з'явився на комп'ютерах, а потім був портований на телефони, де знайшов своє застосування.

Гра у жанрі shoot'em up для платформи Android – це вид аркадних відеоігор, в яких гравець контролює персонажа або транспортний засіб (наприклад, космічний корабель), що бореться з великим числом ворогів, зазвичай через стрілянину. Розробка такого ПЗ на платформі Android має ряд специфічних проблем та невирішених питань, які можна вирішити або покращити завдяки новим ідеям та технологіям. Нижче наведено опис основних проблем:

− проблеми продуктивності та оптимізації. Shoot'em up ігри часто вимагають великої кількості обчислень через велику кількість ворогів, снарядів і спецефектів, що може впливати на продуктивність пристрою, особливо на старих або бюджетних моделях;

− керування та інтерфейс користувача. Екран сенсорного пристрою має обмежену область для керування, що може ускладнювати точне керування персонажем або транспортним засобом;

− різноманітність контенту та реіграбельність. Shoot'em up ігри можуть швидко набриднути через одноманітність контенту та відсутність стимулів для повторного проходження;

− мультиплеєрні функції. Відсутність або обмеженість мультиплеєрних функцій може зменшувати інтерес гравців до гри.

Вирішення цих проблем допоможе створити успішну та популярну shoot'em up гру на платформі Android, яка буде цікавою та доступною для широкого кола гравців.

Для розробки якісного застусунку портрібно проаналізувати типи кінцевих користувачів, визначити кінцеву групи користувачів та їх інформаційні потреби:

− гравці-новачки - нові користувачі, які можуть не мати великого досвіду в іграх жанру shoot'em up. Інформаційні потреби - легкий вступ до гри, прості та інтуїтивно зрозумілі керування;

− досвідчені гравці - користувачі з досвідом у подібних іграх, які шукають нові виклики та глибший ігровий процес. Інформаційні потреби - високий рівень

складності, складні боси, різноманітні рівні, можливості для налаштування гри, досягнення та нагороди;

− казуальні гравці - користувачі, які грають у вільний час і не прагнуть досягати високих результатів. Інформаційні потреби - простота використання, швидкий доступ до гри, можливість гри в короткі сесії, доступність без значних вкладень часу або ресурсів;

− соціальні гравці - користувачі, які хочуть грати з друзями або змагатися з іншими гравцями. Інформаційні потреби - мультиплеєрні функції, наявність ігрових очок та відображення максимальної кількості зароблених очок.

Виходячи з проаналізованих кіцевих користувачів розроблювана гра буде в основному орієнтована на соціальних та казуальних гравців. Для соціальних гравців ми надаємо мультиплеєрні функції, відображення очок як під час гри так і відображення максимальної кількості зароблених. Казуальні гравці оцінять простоту використання, швидкий доступ до гри, можливість грати в короткі сесії та доступність без значних вкладень часу або ресурсів.

Також потрбіно проаналізувати процес переходу вхідної інформації у вихідну під час гри.

Збір вхідної інформації:

− керування гравця - вхідні дані від сенсорного екрану або контролерів (дотики, жести, натискання кнопок);

− ігрові події - дії гравця (стрілянина, ухилення), стан ворогів (позиція, атаки), стан рівня (перешкоди, бонуси);

− дані системи - інформація про продуктивність пристрою, поточний рівень ресурсоємності.

Обробка вхідної інформації:

− ігровий рушій - обробка даних керування, розрахунок руху гравця та ворогів, відстеження зіткнень та взаємодій;

− логіка гри - обчислення результатів дій гравця, зміни в ігровому середовищі (поява нових ворогів, зміни у рівні);

− графіка та звук - відповідне відображення візуальних і звукових ефектів на основі дій гравця та стану гри.

Вивід вихідної інформації:

− інтерфейс користувача - відображення поточного стану гри (позиція гравця, ворогів, снарядів), показник здоров’я, бали, досягнення;

− зворотній зв’язок - анімації, звукові ефекти, вібрація пристрою для створення більш занурюючого досвіду;

− статистика та аналітика - вивід статистичних даних, прогресу гравця, досягнень та результатів на екрані, збереження в локальну базу даних або в хмару для подальшого аналізу.

Цей процес забезпечує інтерактивність та динамічність гри, відповідаючи на інформаційні потреби різних категорій користувачів та забезпечуючи їм задоволення від ігрового процесу.

У сучасному світі кожен має доступ до Інтернету, і додатки з можливістю грати по мережі стають більш популярними серед широкої аудиторії. Тому актуальність даного проекту вища, ніж у аналогів.

1.2 Аналіз програмно-технічного забезпечення предметної області

"Shoot'em up" ігри, також відомі як "shmups" або "shooters", є жанром комп'ютерних ігор, де головною метою гравця є стріляти по ворожим силам і уникати їх атак. Це дуже популярний жанр з численними іграми, які пропонують широкий спектр геймплейних можливостей та стилів. Ось деякі з найвідоміших шутерів цього жанру:

− Space Invaders;

− Radiant;

− Чужий в космосі.

Гра Space Invaders.

Самою популярною shoot`em up являється Space Invaders, зображена на рисунку 3, яка була одна з перших ігор в цьому жанрі. Мета гри – знищити усіх ворогів на екрані. Спочатку гра мала просту графіку і показник очок життя, при досягненні нульового значення якого гра завершувалася, однак пізніше вона удосконалилася, а саме були введені нові кольорові схеми і показники очок, зображено на рисунку 1.1.

Рисунок 1.1 – Кадр з гри Space Invaders

Space Invaders, класична аркадна відеогра, відіграла значну роль у розвитку геймінгу та стала однією з найбільш впливових ігор свого часу. Ось кілька переваг і недоліків цієї гри:

Переваги:

− простота концепції: Space Invaders проста в освоєнні і розумінні. Гравцям не потрібно витрачати час на вивчення складних правил або механік гри.

− захоплюючий геймплей: гра пропонує захоплюючий геймплей, який швидко стає динамічним і вимагає швидких реакцій від гравця;

− висока ігральна вартість: незважаючи на свою простоту, Space Invaders має високий рівень ігральної вартості, оскільки гравці постійно стріляють

нарахунок ворожих кораблів, намагаючись вижити якомога довше і набрати найбільшу кількість очок;

− іконічний статус: Space Invaders стала символом ранньої ери відеоігор і її образи, включаючи ворожих інопланетян і захищаючий корабель, стали відомими в усьому світі.

Недоліки:

− обмежена різноманітність: гра має досить обмежений набір механік та рівнів, що може призвести до відчуття монотонності після тривалого часу гри;

− брак глибини: у порівнянні з сучасними іграми Space Invaders може здатися досить поверхневою, оскільки в ній відсутня складна сюжетна лінія або глибокі персонажі;

− неадаптована складність: складність гри з часом може виявитися надто високою для новачків, що може відлякувати нових гравців;

− брак мультиплеєра: оскільки гра розроблялася в еру одиночних аркад, відсутність можливості грати з друзями або онлайн може зменшувати її привабливість у сучасному світі ігор.

Space Invaders є іконічною грою, яка внесла значний внесок у розвиток геймінгу, але вона також має свої обмеження і недоліки порівняно з сучасними стандартами геймінгу.

Гра Radiant.

Аналог гри Space Invaders випущений у 2010 році. Мета гри – знищити всіх ворогів на екрані. Головна особливість гри – покращена 2D графіка, зображено на рисунку 5. По мірі збільшення часу гри збільшується і складність – з’являється більше ворогів.

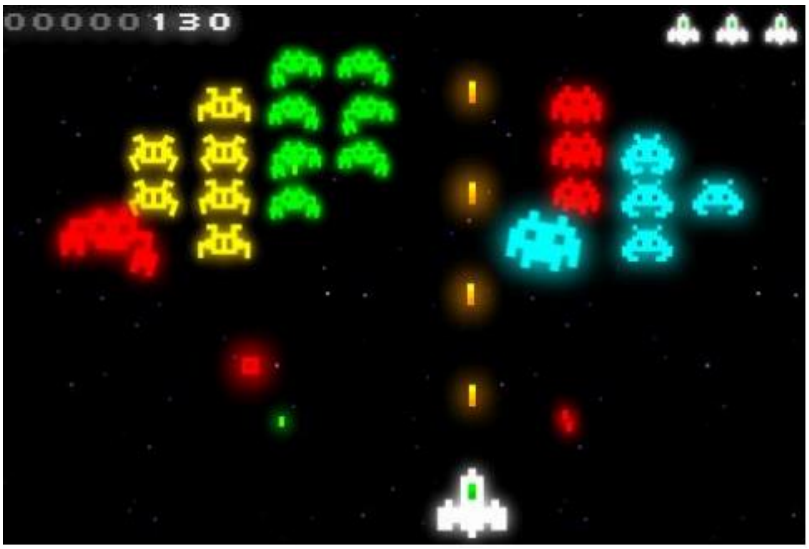


Рисунок 1.2 – Кадр з гри Radiant

"Radiant" - це ігра в жанрі shoot'em up, яка відрізняється від класичного "Space Invaders" та інших аркадних шутерів більшою глибиною геймплею та розвинутою системою прогресу. Ось деякі переваги і недоліки цієї гри:

Переваги:

− глибший геймплей: "Radiant" має більш розвинуту систему геймплею, включаючи різноманітність ворогів, зброї, апгрейдів та рівнів, що надає більше можливостей для стратегічного геймплею;

− естетика та дизайн: гра має стильний космічний атмосферний дизайн та яскраву візуальну графіку, що робить її привабливою для гравців;

− апгрейди та досягнення: "Radiant" пропонує систему апгрейдів для зброї та корабля, а також можливість отримання досягнень, що стимулює гравців до подальшого прогресу;

Недоліки:

− складність для новачків: у порівнянні з класичними аркадними шутерами, "Radiant" може бути складнішою для новачків через більшу кількість механік та ворогів;

− залежність від апгрейдів: в деяких випадках успіх у грі може залежати від рівня апгрейду зброї та корабля, що може створювати нерівність між гравцями;

− необов'язкові мікротранзакції: у деяких версіях гри можуть бути присутні мікротранзакції або реклама, що може впливати на досвід гри.

"Radiant" в цілому є цікавим та захоплюючим представником жанру shoot'em up, проте вона має свої особливості та може бути не для кожного гравця.

Гра «Чужий у космосі».

"Чужий у космосі" – це гра які була випущена у 2013 році. Хоробрий космодесантник повинен провести вчених в безпечні бункери, поки до них не дісталися місцеві жуки-переростки. Для виконання цієї непростої задачі гравцю дозволено користуватися всілякою зброєю: бензопилою, ракетами, кулями.

Гра має більш продуману графіку, а також систему покращення зброї (на модель гравця додаються нові елементи, а також змінюється модель кулі). Особливістю цієї гри є те, що у ній присутні рівні і складність змінюється в залежності від обраного рівня.

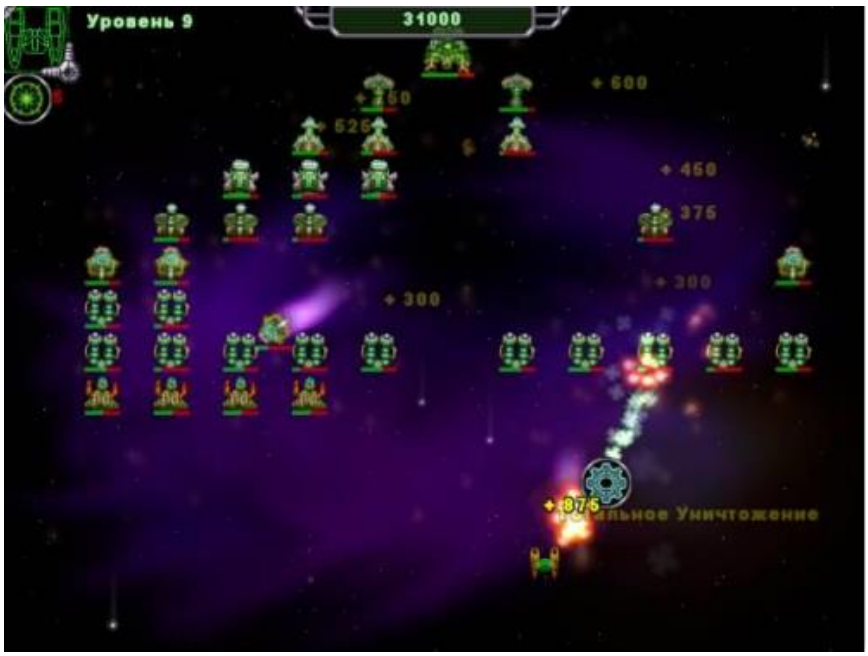


Рисунок 1.3 – Кадр з гри "Чужий у космосі"

"Чужий у космосі" (іноді відомий як "Alien Shooter") - це ігра в жанрі shoot'em up, яка відрізняється своїм атмосферним настроєм, інтенсивним

геймплеєм та наявністю різноманітних ворогів та зброї. Ось кілька переваг і недоліків цієї гри:

Переваги:

− атмосфера: "Чужий у космосі" створює напружену та атмосферну гру, яка відтворює атмосферу науково-фантастичного фільму про боротьбу за виживання в космосі проти нашестя інопланетян;

− різноманітність ворогів та зброї: гра пропонує різноманіття ворогів з різними характеристиками та різноманітність зброї, що дає гравцям можливість експериментувати з різними стратегіями боротьби;

− загальна ігрова вартість: "Чужий у космосі" має глибину і складність, що може захопити гравця на довгі години.

Недоліки:

− монотонність: несприятлива для гри може бути монотонність геймплею, особливо на тривалих рівнях;

− залежність від везіння: у деяких випадках успіх у грі може залежати від везіння або від того, яку зброю вдалося знайти.

− застаріла графіка та механіка: у порівнянні з сучасними іграми, "Чужий у космосі" може виглядати застарілою з точки зору графіки та геймплею.

"Чужий у космосі" залишається популярною грою в жанрі shoot'em up завдяки своїй атмосфері та геймплею, проте вона може не відповідати смакам кожного гравця через свої особливості.

Для систематизації даних, отриманих під час дослідження наявних рішень, корисно скористатися таблицями. Вони дозволяють швидко порівняти різні платформи за набором важливих функцій та можливостей. Для цього була створена порівняльна таблиця 1.1.

Таблиця 1.1 – Порівняльна таблиця схожих ігрових рішень.

Назва	Space Invaders	Radiant	Чужий у космосі
1	2	3	4

Змінний рівень складності	Ні	Так	Так

Продовження таблиці 1.1

1	2	3	4
Наявність різних моделей літаків з різними характеристиками	Ні	Так	Так
Зручність та зрозумілість інтерфейсу	Через вік гри інтерфейс її може здатися не зрозумілим та застарілим	Сучасніша гра, має портовану версію на Android, в якій адаптовано інтерфейс для даної платформи	Гра хоч нова але має застарілий інтерфейс
Мультиплеєр	Ні	Ні	Ні
Графіка	2D, проста, піксельна	2D, яскрава, стильна	2D/3D, деталізована, реалістична
Сюжет	Відсутній	Відсутній	Присутній, боротьба за виживання проти нашестя інопланетян
Механіка	Базова, обмежена	Розвинута з апгрейдами	Різноманітна, збільшена кількість зброї, апгрейди
Геймплей	Простий, лінійний	Динамічнй	Динамічний

1.3 Визначення функціональних та нефункціональних вимог до програмного забепечення

Виходячи з аналізу наявнивних ігор даного жанру можна сформулювати вимоги до програмного забезпечення гри в жанрі shoot'em up для платформи Android.

До функціональних вимог можна віднести:

− режими гри: режими гри включають одиночний режим та мультиплеєр (локальний та онлайн);

− зброя та апгрейди: можливість гравця збирати та використовувати різну зброю, а також апгрейди для покращення характеристик корабля;

− життя та бонуси: система життя, бонусів та спеціальних предметів, які випадають під час гри;

− система балів: нарахування балів за знищення ворогів та збирання бонусів;

− сенсорне керування: підтримка сенсорного керування для руху корабля, стрільби та використання спеціальних здібностей;

− головне меню: меню з опціями для старту гри, вибору режиму, налаштувань, перегляду досягнень та статистики;

− ігровий інтерфейс: відображення стану гравця;

− локальний мультиплеєр: підтримка гри на одному пристрої або через локальну мережу.

До нефункціональні вимог можна віднести:

− оптимізація: гра повинна бути оптимізована для плавної роботи на різних моделях пристроїв;

− час завантаження: час завантаження гри та рівнів не повинен перевищувати 5 секунд на середньому пристрої;

− платформи: підтримка широкого спектру Android пристроїв;

− інтуїтивний інтерфейс: інтерфейс повинен бути зрозумілим та інтуїтивним для користувачів будь-якого рівня;

− відсутність багів: гра повинна бути стабільною та без помилок, які можуть переривати геймплей.

Ці вимоги забезпечать створення якісного та цікавого продукту, який буде задовольняти потреби різних категорій гравців та відповідати сучасним стандартам мобільних ігор.

Діаграма варіантів використання вказує на залежність між різними варіантами використання та учасниками, що беруть участь у процесі. Варіанти використання, або прецеденти, показують поведінку програмної системи та її реакцію на зовнішні впливи. Створення такої діаграми допомагає краще зрозуміти логіку програмного забезпечення та сприяє полегшенню його розробки.

Наступним кроком буде опис акторів програмного продукту та варіантів його використання.

У таблиці 1.2 наведено опис акторів програмного продукту, а в таблиці 1.3 - варіанти використання.

Таблиція 1.2 – Актори ПЗ та їх функції

Актор	Доступні шляхи використання ПЗ
Користувач (гравець)	Початок гри, зміна налаштувань, вибір літального апарату, завантаження рівня, вихід,

Таблиція 1.3 – Опис варіантів використання програмної системи

Актор	Варіант використання	Опис ВВ
Користувач (гравець)	Початок одиночної гри	Користувач починає гру у одиночному режимі
	Початок гри мультплеєру	Користувач починає гру у мультиплеєрному режимі
	Вибір літального апарату	Користувач може вибрати апарат серед наявних
	Завантаження рівня	Відбувається після того як користувач розпочав гру
	Зміна налаштування	Яка включає в себе можливість змінити налаштування гучності ефектів та музики та і зміна налаштувань графіки
	Вихід	Користувач може покинути гру

На рисунку 1.4 зображену діаграму використання для Shoot`em up гри:

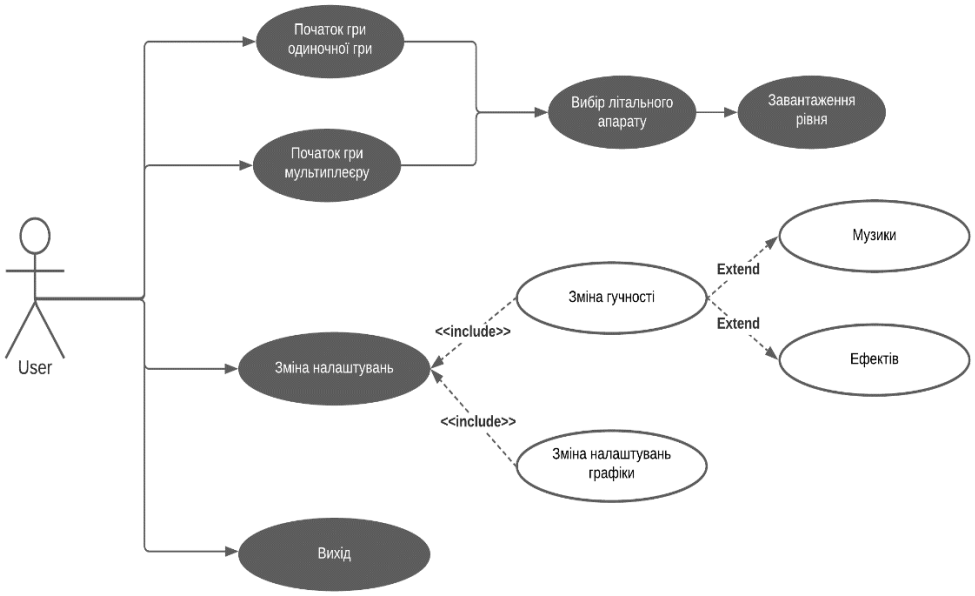


Рисунок 1.4 – Діаграма варіантів викоритсання для гравця

На основі діаграми варіантів використання можна скласти вимоги до інтерфейсу користувача для гри у жанрі shoot'em up.

Основні екрани:

− головне меню: екран з кнопками для початку одиночної гри, мультиплеєру, налаштувань та виходу з гри;

− екран вибору літального апарату: вибір та налаштування корабля перед початком рівня;

− екран завантаження рівня: відображення процесу завантаження рівня з індикатором прогресу;

− екран налаштувань: дозволяє змінювати гучність (музика та ефекти) та налаштування графіки;

− ігровий інтерфейс: відображення основної інформації під час гри (здоров'я, боєзапас, бали).

Форми взаємодії з користувачами:

− рух корабля: за допомогою свайпів або натискання на екран для переміщення корабля;

− стрільба: автоматична стрільба або натискання на спеціальну кнопку для стрільби.

− музика: слайдер для регулювання рівня гучності музики.

− ефекти: слайдер для регулювання рівня гучності звукових ефектів.

− якість графіки: вибір між різними рівнями якості графіки (низька, середня, висока) для оптимізації продуктивності на різних пристроях.

Вимоги щодо доступу до внутрішньої функціональності ПЗ:

− швидкий доступ до налаштувань: кнопка налаштувань доступна з головного меню та паузи під час гри;

− зміна налаштувань: можливість змінювати налаштування в будь-який час без перезавантаження гри;

− автозбереження: прогрес гри автоматично зберігається локально після завершення рівня або виходу з гри.

Цей аналіз забезпечить розробку інтерфейсу користувача, який буде зручним, інтуїтивно зрозумілим та доступним для всіх категорій гравців.

1.3 Висновки. Постановка завдання

В результаті огляду вище перерахованих ігрових рішень були виділені особливості жанру shoot`em up:

− набір місій або нескінченний режим гри;

− основна мета – набрати якомога більшу кількість очок;

− щоб збільшити кількість очок потрібно знищити модель ворога за допомогою випуску кулі (атаки);

− майже у всіх є показник очків здоров’я, який зменшується або при досягнені моделі ворога, або якоїсь певної точки.

Також серед розглянутих рішень жодне не надавало можливості грати по мережі.

Переважно всі ігри цього жанру виконані у 2D стилі з використанням різноманітних спрайтів.

Проєктування гри в жанрі shoot'em up для платформи Android потребує чіткого планування та розподілу завдань. Нижче наведено основні задачі та їх розбивка на конкретні підзадачі.

Основні задачі:

- аналіз вимог;
- проєктування;
- розробка інтерфейсу користувача (UI);
- розробка функціоналу гри;
- тестування та відлагодження.

Аналіз вимог:

- визначення функціональних вимог;
- визначення нефункціональних вимог;
- аналіз конкурентів та ринку;
- формування технічного завдання (ТЗ).

Проєктування:

- рівні складності та прогресія;
- проєктування механік гри, такі як: рух та стрільба, вороги та їхні типи, система очок та бонусів;
- проєктування системи апгрейдів, такі як: зброя та обладнання.

Розробка інтерфейсу користувача (UI):

- головне меню;
- екран налаштувань;
- екран вибору літального апарату;
- ігровий інтерфейс;
- екран паузи та завершення гри;

- створення анімацій для переходів та взаємодії;
- адаптація UI для різних розмірів екранів.

Розробка функціоналу гри:

- фізика руху та стрільби;
- реалізація системи апгрейдів та бонусів;
- інтеграція аудіо та візуальних ефектів;
- мультиплеєр.

Тестування та відлагодження:

- розробка тестових сценаріїв;
- функціональне тестування;
- тестування на різних пристроях;
- виправлення багів та оптимізація.

Потрібно реалізувати деякий “спавнер”, який буде займатися появою ворогів, а також передбачити ситуацію, щоб він не переходив максимальну границю частоти появи ворогів.

Так як гра буде нескінченною потрібно буде продумати як чинити з об’єктами, які вийшли за межі екрану, щоб їхня кількість не впливала на продуктивність апаратної частини.

Через те, що цільовою платформою є мобільні телефони на базі ОС Android, необхідно реалізувати адаптивний інтерфейс, який не буде залежати від роздільної здатності екрану поточного апарату користувача

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура та функціональна структура додатка

Для розробки програмного забезпечення були поставленні наступні завдання:

- створення проєкту;
- вибір необхідних додаткових інструментів;
- реалізація головного персонажу;
- реалізація супротивників;
- реалізація нескінченного режиму з наростаючою складністю;
- реалізація UI.

Для того, щоб створити проєкт, спочатку потрібно обрати засоби створення. Потрібно обрати ті засоби, які допоможуть реалізувати проєкт швидко і якісно, при цьому зробити його доступним і зручним для кінцевого користувача, а також допоможуть забезпечити швидкість виконання дій.

Головний персонаж буде представлений у вигляді космічного літака, який матиме можливість отримувати пошкодження, наносити пошкодження та підбирати бонуси та спеціальні “бусти”. Також кожен літак повинен мати свою гілку еволюції, наприклад, у кожного літака будуть різні види снарядів. Зростання по гілці еволюції дасть можливість змінювати тип снаряда, величину пошкодження від нього та кількість снарядів, які з’являються кожен певний проміжок часу.

Супротивники будуть представлені у вигляді космічних літаків з іншими моделями. Вони матимуть можливість переміщатися та отримувати пошкодження.

Для управління появи ворогів буде створено деякий об’єкт, який буде в собі зберігати інформацію про “хвилі” ворогів. Поступово наростаюча складність буде полягатиме у збільшенні швидкості появи “хвиль” ворогів.

Для усього цього повинен бути розроблений інтерфейс, який буде давати необхідну інформацію під час гри, таку як: поточне здоров'я літака гравця, поточні очки, а також можливість поставити гру на паузу. У головному меню користувач повинен мати можливість змінити налаштування графіки, гучності та обрати необхідний літак для початку гри.

Для відображення етапів проектування архітектури використовувалися UML діаграми. UML - це уніфікована мова моделювання, яка використовується у парадигмі об'єктно-орієнтованого програмування та є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. Це мова широкого профілю та відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, яка відома як UML-модель. UML був створений для визначення, візуалізації, проєктування та документування програмних систем.

Діаграма послідовності (Sequence Diagram) відображає взаємодії об'єктів, впорядкованих за часом. Зокрема, такі діаграми показують, які об'єкти беруть участь у взаємодії та яку послідовність повідомлень вони відправляють один одному.

На рисунку 2.1 зображена діаграма послідовностей яка відображає взаємодію об’єктів у процесі геймплею.

При перетині колізій літака головного персонажу і ворожого юніту, літаку головного персонажу наноситься пошкодження, та якщо очок здоров’я недостатньо, то літак користувача знищується і користувач отримує меню перегляду своїх результатів, після чого він має змогу покинути гру.

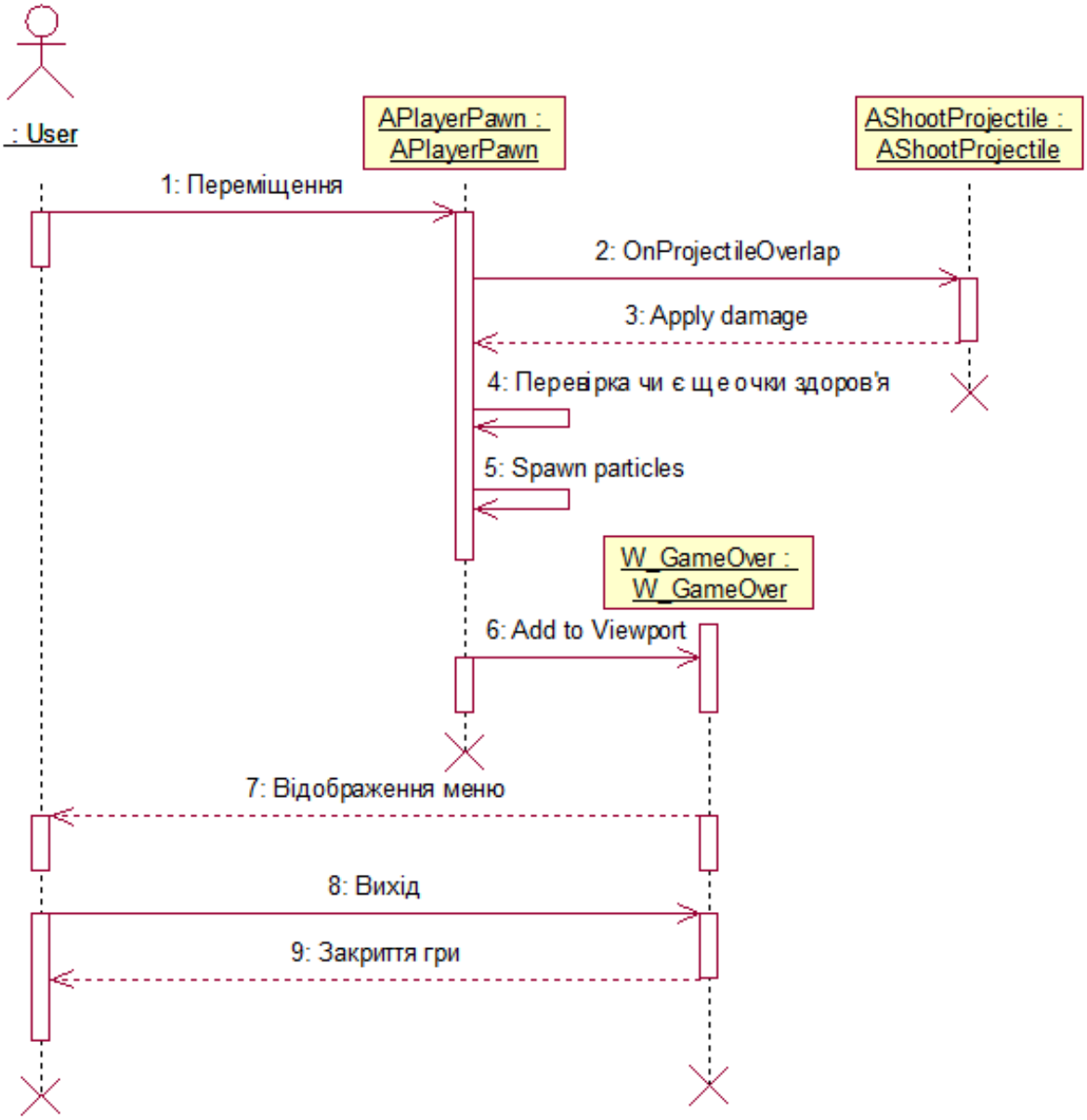


Рисунок 2.1 – Діаграма послідовностей у процесі геймплею

На рисунку 2.2 зображена діаграма послідовності, яка відображає взаємодію об’єктів у головному меню.

Спершу як користувач потрапляє на карту меню, він одразу бачить віджет головного меню W_MainMenu. Далі, щоб не обрав користувач, віджет головного меню буде знищено. Наступним кроком користувач потрапляє до меню взаємодії з об’єктом W_ChooseMainSpaceSheep, де він може обрати свій літак, який далі він отримає після завантаження гри

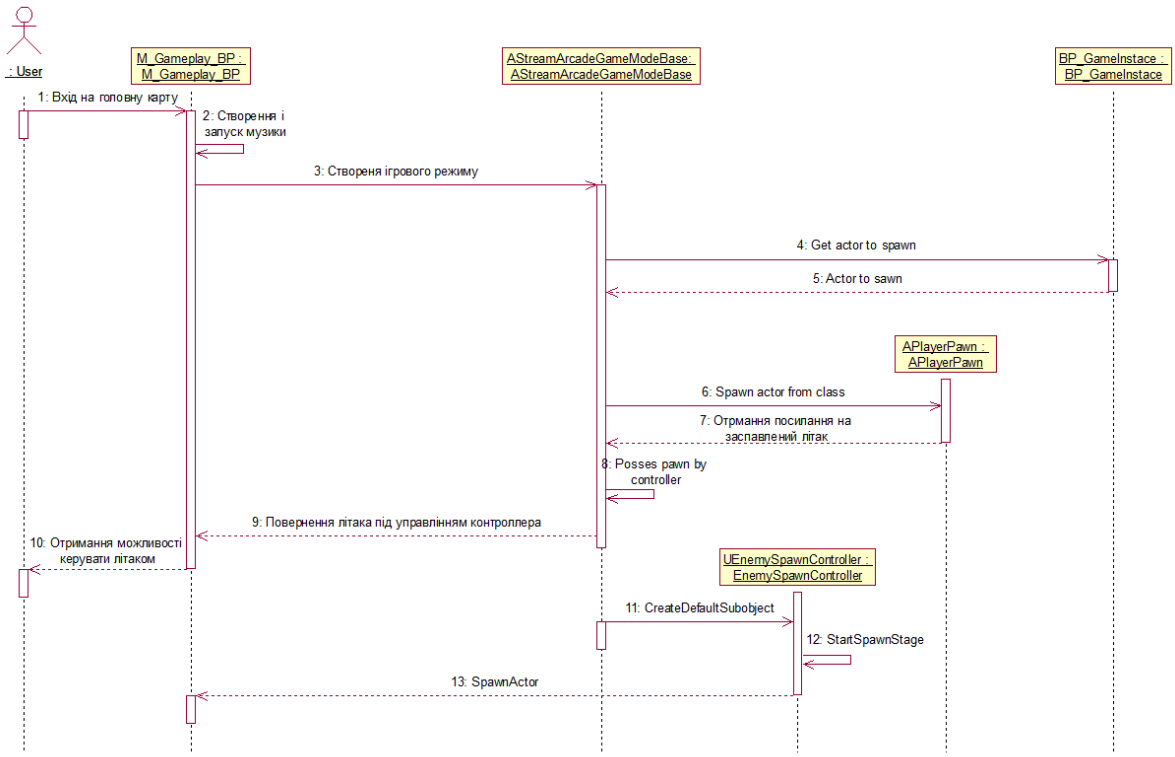


Рисунок 2.2 – Діаграма послідовностей у головному меню

Далі була створена діаграма класів, яка відображає структуру системи, включаючи класи, їх атрибути, методи та відношення між класами (асоціації, наслідування, агрегації), яка наведена на рисунку 2.3.

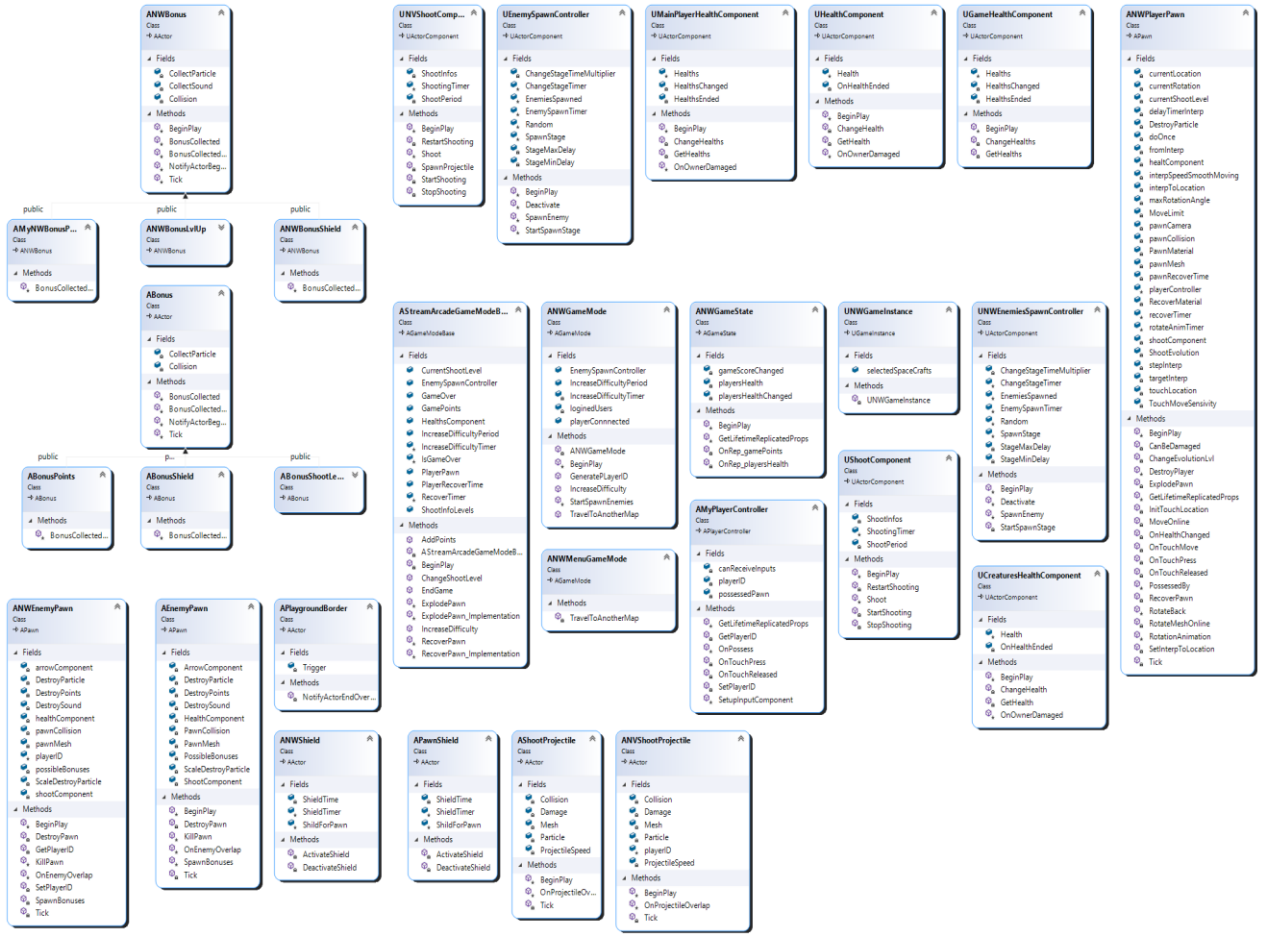


Рисунок 2.3 – Діаграма класів

Діаграма класів містить 32 класи. Нижче буде наведений опис найбільш значущих класів.

Клас StreamArcadeGameModeBase наслідуються від класу GameModeBase, який є класом ігрового режиму та відповідає за загальні ігрові правила – в даному випадку він відповідає за кількість життів гравця, накопичення очків та “еволюцію” рівня стрільби корабля головного гравця.

Клас AMyPlayerController наслідується від класу APlayerController, який являється класом контроллером. Даний клас передає вхідні дії користувача до об’єктів, якими він керує, також даний клас отримує кожен клієнт при підключенні до серверу і вже через нього клієнти можуть спілкуватися з сервером.

Клас NWGamaState наслідується від класу AGameState. Даний клас призначений для відображення поточних даних гри, які необхідні усім клієнтам для успішного функціонування.

Клас PlayerPawn наслідується від класу Pawn, який являється класом пішкою. Особливість даного класу в тому, що цим класом, як і ще одним, можна керувати. Даний клас відповідає за переміщення головного літака гравця, обробку подій отримання пошкоджень та набір компонентів, які будуть використані в даному класі.

Клас EnemyPawn відповідає за отримання пошкоджень, максимальну кількість очок здоров'я та кількість очок, яку буде надано за знищення даного роду противника.

Клас ShootComponent наслідується від класу ActorComponent, який являється класом компонентом. Даний клас відповідає за частоту пострілів, їх напрямок та кількість, а також тип снаряду для пострілу.

Клас Bonus являється базовим класом бонусом, який вже інші перевизначають за необхідності, також в ньому описані деякі базові речі, такі як переміщення на "тік" та базова поведінка при збиранні даного бонусу.

Діаграма станів зображує різні стани, через які проходить гра від запуску до завершення, включаючи взаємодію користувача з грою, яка наведена на рисунку 2.4.

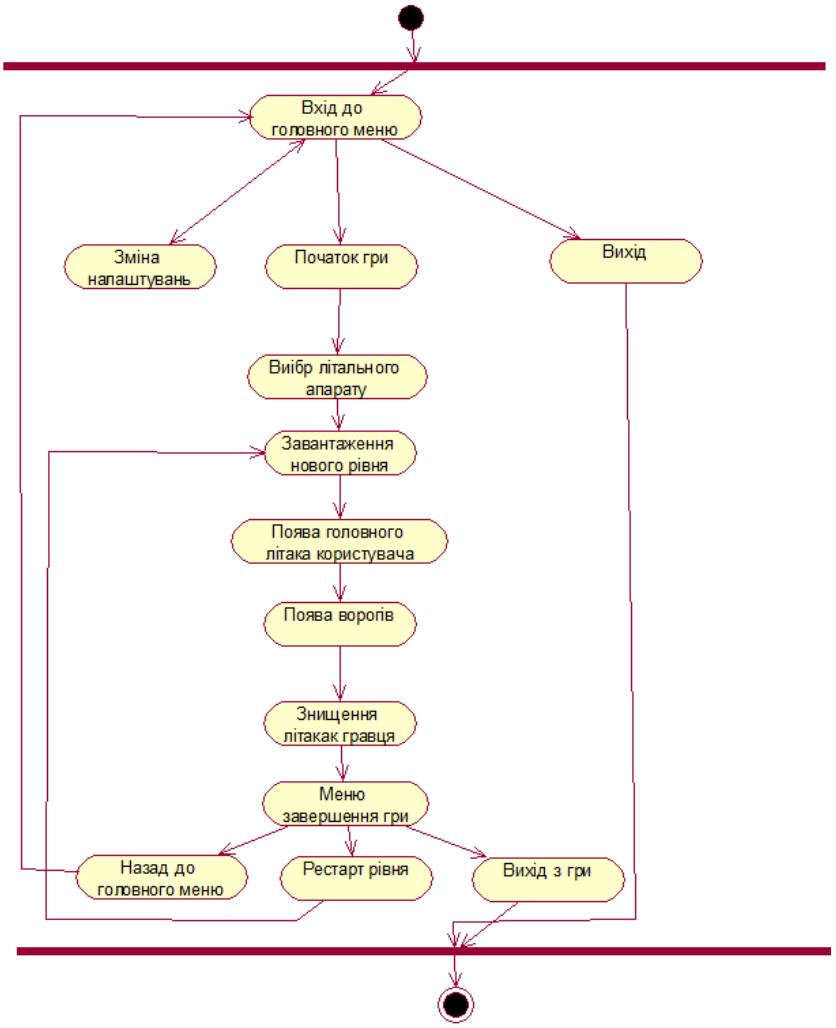


Рисунок 2.4 – Діаграма станів

Діаграма показує послідовність станів, через які проходить гравець від моменту запуску гри до її завершення. Включає основні дії, які може виконати гравець, такі як вибір літака, налаштування графіки, перегляд результатів та вихід з гри. Основний ігровий процес проходить у стані "Гра", який повторюється, поки у головного героя є життя. Після закінчення гри гравець може переглянути результати та повернутися до меню або вийти з гри.

Діаграма активностей зображує послідовність дій та переходи між ними в процесі гри, яка зображена на рисунку 2.5.

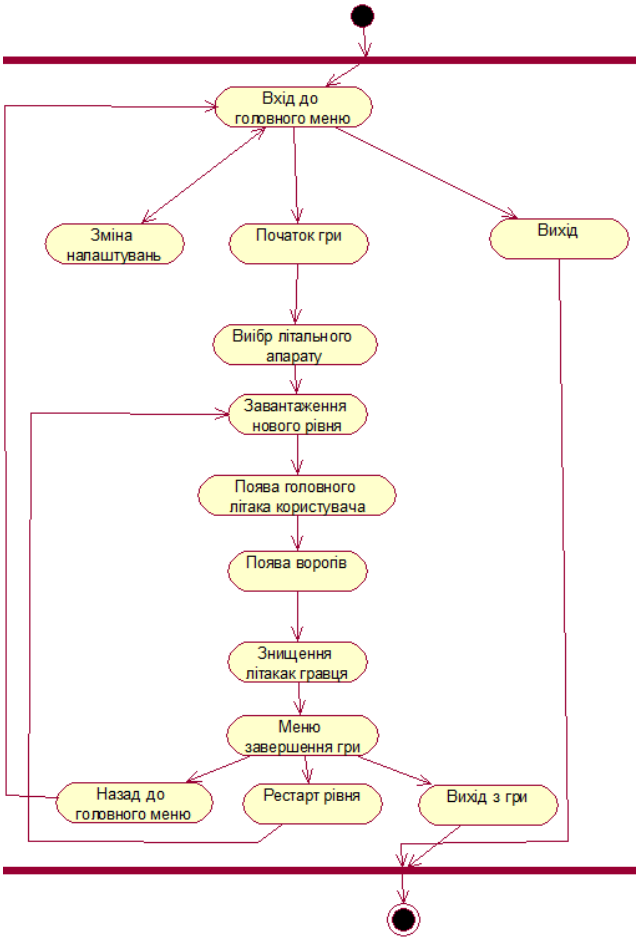


Рисунок 2.5 – Діаграма активностей

Діаграма активностей демонструє логічний порядок дій, які виконує гравець у процесі гри, починаючи з головного меню, вибору налаштувань, вибору літака, ігрового процесу та завершення гри. Кожна активність пов'язана з певними діями користувача та можливими переходами до інших активностей, що забезпечує плавний та послідовний ігровий досвід

Виходячи з обраної моделі мультиплеєру пристрій на якому буде запускатися гра буде представляти кілєнт і сервер до якого буде підключатися інший гравець. Даний процес ілюструє діаграма розгортання яка зображена на рисунку 2.13.

Діаграма розгортання, яка зображена на рисунку 2.6, демонструє основні компоненти системи та їх взаємодію в контексті гри в жанрі shoot'em up на

платформі Android, де один з клієнтів виконує роль сервера. Ось детальний опис взаємодії між компонентами системи:

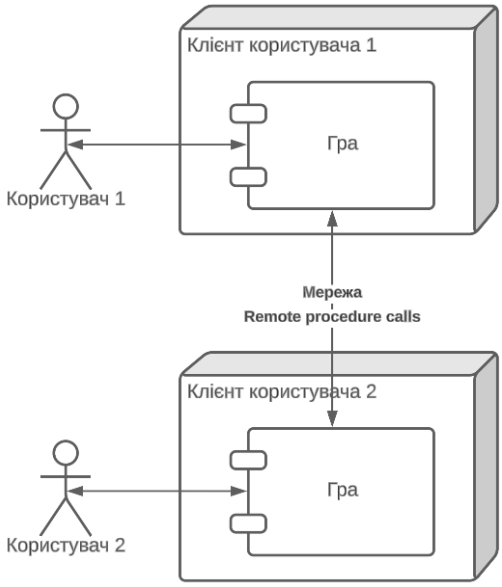


Рисунок 2.6 – Діаграма розгортання

Компоненти системи:

- клієнт Користувача 1;
- клієнт Користувача 2.

Взаємодія між компонентами відбувається наступним чином:

- клієнт Користувача 1 і клієнт Користувача 2 запускають ігровий додаток;
- клієнт Користувача 2 виступає в ролі сервера, встановлюючи початковий ігровий стан і чекаючи підключення іншого гравця;
- клієнт Користувача 1 надсилає запит на підключення до клієнта Користувача 2 через мережу;
- клієнт Користувача 2 приймає запит, підтверджує підключення і надсилає поточний стан гри клієнту Користувача 1;
- обидва клієнти синхронізують свої ігрові стани через мережу;
- клієнт Користувача 2 (сервер) відправляє оновлення про стан гри, включаючи позиції ворогів, проєктили, стан літаків та інші ігрові об'єкти до **Клієнта Користувача 1;

– гравці взаємодіють.

2.1 Обґрунтування вибору технологій для розробки

2.2.1 Вибір ігрового рушія

Проєкт являється грою, тому серед можливих варіантів доступних інструментів обиралися ті, за допомогою яких можна реалізувати дану задачу. Існує багато інструментів для створення ігор, вони носять назву ігрові рушії. Проте, оскільки однією з умов проєкту є створення мобільної гри, вибір стояв між Unity і Unreal Engine. Вони були обрані через те що вони є кросплатформенні, тобто під час процесу розробки гри можна користуватися ОС Windows та досить просто обрати під яку платформу для виконання потрібно зібрати проєкт.

Ігровий рушій Unreal Engine – ігровий рушій, який розробляється компанією Epic Games.

Перевагами Unreal Engine є:

− написаний на C++, що дозволяє створювати ігри під більшість доступних ОС, зокрема і на Android;

− в порівнянні з Unity, що потребує для своєї роботи встановлення безлічі плагінів (які часто є платними), Unreal Engine має дуже багато функціоналу, який вже присутній з “коробки”;

− можливість писати внутрішньоігрову логіку, комбінуючи C++ і Blueprints;

− готові інструменти для роботи з мультиплеєром;

− легке і зрозуміле налаштування проекту для збірки під іншу ОС.

Недоліки:

− так як Unreal Engine орієнтується в першу чергу на професіоналів, це досить сильно вплинуло на інтерфейс, точніше його ергономічності – в Unity в цьому плані все краще;

− також присутні проблеми з ІІ в NPC – якщо додати на локацію досить багато ІІ-істот, то спроби рушія опрацювати поведінку всіх призводить до падіння FPS.

Unity – багатоплатформений інструмент для розробки дво- та тривимірних додатків та ігор, що працює на операційних системах Windows і OS X. Створені за допомогою Unity застосунки працюють під системами Windows, OS X, Android, Apple iOS, Linux, а також на ігрових консолях PlayStation і XBox.

Недоліки:

− повільність – цей фактор часто випливає в роботі таких додатків, які мають на увазі масштабність, складні сцени. Доводиться використовувати додаткові утиліти, і все це впливає на продуктивність;

− відсутність «самостійності» – полягає в тому, що при використанні Unity 3D автоматично вбудовується їх логотип; позбутися його можна придбавши PRO-версію;

− обмеження – являє собою відсутність функцій управління «вшитими» параметрами, починаючи від камери і закінчуючи базою даних; це робить додаток «негнучким» і виключає роботу на максимальних можливостях.

Зважаючи на це все, ігровим рушієм було обрано Unreal Engine. Для створення налаштування у грі обрано плагін Autosettings, в порівнянні із стандартними методами в UE він дозволяє швидко створити налаштування для гри, такі як графіка, звук та інше.

2.2.2 Вибір інтегровного середовища розробки

Серед наявних IDE вибір не такий і великий, він лежить між двома продуктами - MS Visual Studio і Rider for Unreal Engine.

Rider for Unreal Engine – IDE створена спеціально для Unreal Engine компанією Jetbrains.

На рисунку 2.7 зображений інтерфейс програми Rider for Unreal Engine.

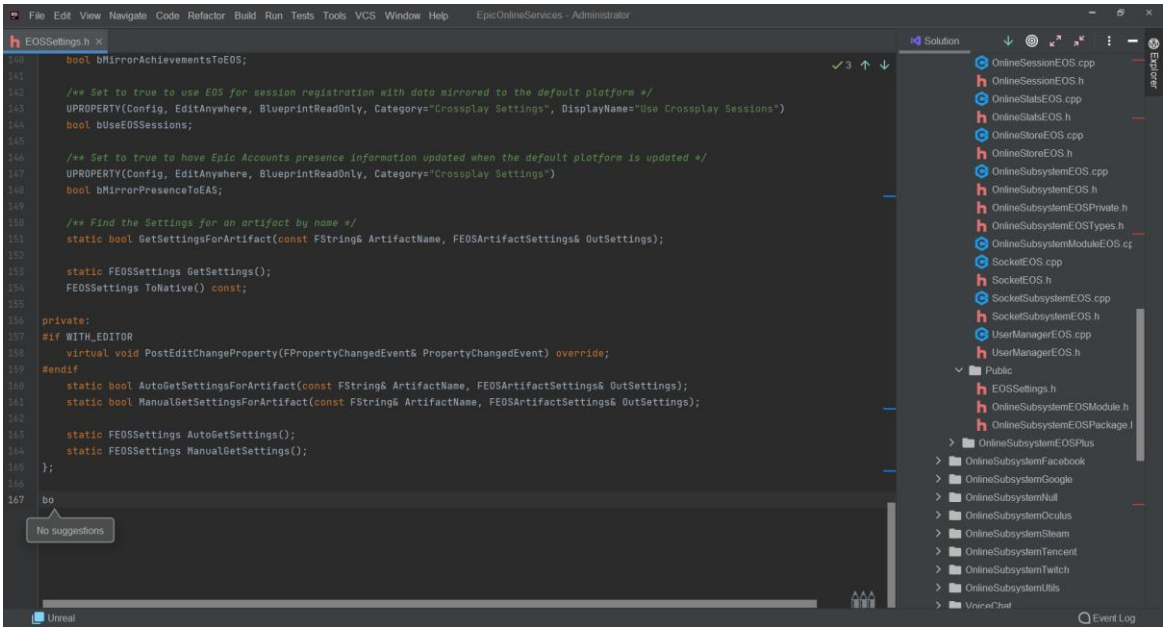


Рисунок 2.7 – Інтерфейс IDE Rider for Unreal Engine

Переваги:

– швидкий intelliSense – це є досить важливо для прискорення роботи, так як в такому рушію, де є багато залежностей, в інших IDE він працює значно повільніше;

– показує залежності в Blueprints;

– швидше завантаження проєкту.

Недоліки:

– не є IDE по замовчуванню, для того щоб нею користуватися потрібно завантажити необхідний плагін.

MS Visual Studio – IDE створена компанією Microsoft, яка підтримує розробку на багатьох мовах і в особливості на C++ під Unreal Engine.

На рисунку 2.4 зображений інтерфейс програми MS Visual Studio.

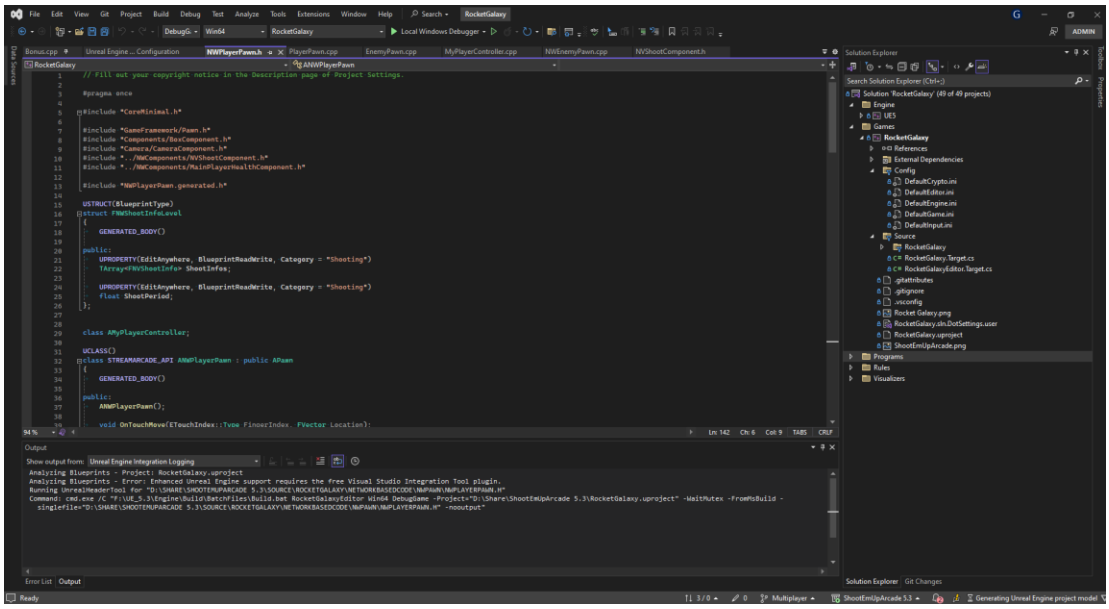


Рисунок 2.7 – Інтерфейс IDE MS Visual Studio

Недоліки:

– основним недоліком є швидкість роботи intelliSense, в Visual Studio він дуже повільний і не зрівняється по швидкості з intelliSense у Rider;

– для нормального використання потрібно встановлювати плагін ReSharper, без нього користуватися Visual Studio неможливо;

– довгий запуск проєкту.

Зважаючи на перелічене, IDE було обрано MS Visual Studio.

2.3 Проектування інтерфейсу користувача

Для точного відображення усіх меню системи було створено віщуальне відображення стукрутри інтерфейсу додатку (рисунок 2.8). Кожний прямокутник описує окреме меню гри.

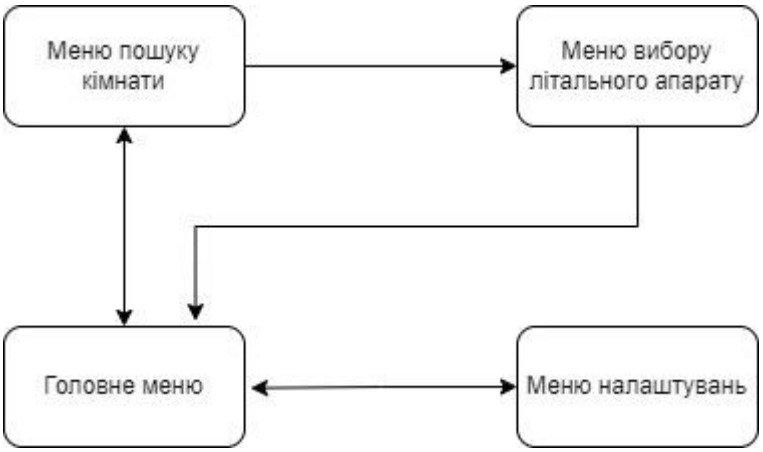


Рисунок 2.8 – Структура інтерфейсу ігрового додатку

Перед початком проектування інтерфейсу було

Для проектування інтерфейсу було використано онлайн платформу “Figma”, за допомогою якої було створено шаблони інтерфейсу для подальшої реалізації.

Шаблони були створені для головного меню, меню вибору корабля та меню налаштувань. Використання даних шаблонів суттєво полегшує подальшу реалізацію інтерфейсу.

Рисунок 2.9 – Шаблон головного меню

Шаблон головного меню, зображений на рисунку 2.9, буде відображатися тільки на початку гри і містить у собі необхідні елементи керування, щоб перейти до наступного пункту меню.

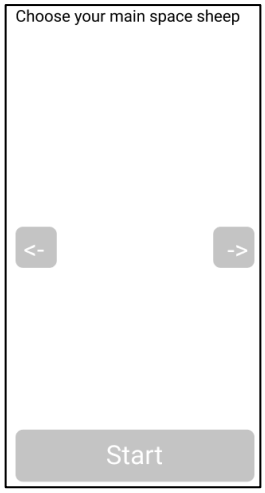


Рисунок 2.10 – Шаблон меню вибору літака

Шаблон меню вибору літака, зображений на рисунку 2.10, призначений для вибору користувачем літака, який у подальшому повинен з’явитися в наступному рівні.

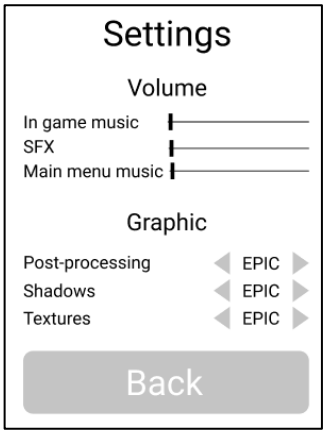


Рисунок 2.11 – Шаблон меню налаштувань

Якщо користувач потрапить на шаблон меню налаштувань, який зображений на рисунку 2.11, то він зможе налаштувати гучність

внутрішньоігрових ефектів, музику як у головному меню, так і під час гри. Також можна буде налаштувати графіку, змінивши тим самим якість пост-оброки, тіней та текстур.

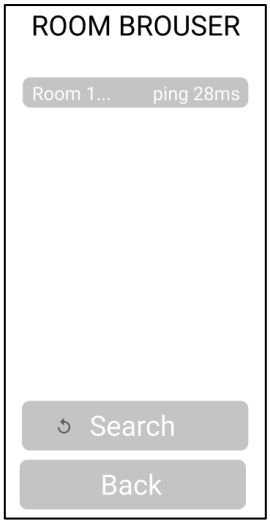


Рисунок 2.12 – Шаблон меню пошуку кімнати

Дане меню, яке наведена на рисунку 2.12, користувач буде бачити коли він захоче знайти нову кімнату для гри.

При проектуванні інтерфейсу було взято до уваги особливості платформи, під яку ведеться розробка – так як це мобільна платформа, то потрібно робити елементи UI досить великими і зручними, і зважати на те, що користувач на смартфоні може оперувати лише «тапами» і «свайпами».

2.3 Проектування алгоритмів

Для стрільби буде використаний певний компонент, який, використовуючи дані зі структури, буде створювати список з елементів даної структури і постріли будуть з’являтися на основі певного алгоритму, який поданий у вигляді блок-схеми на рисунку 2.13.

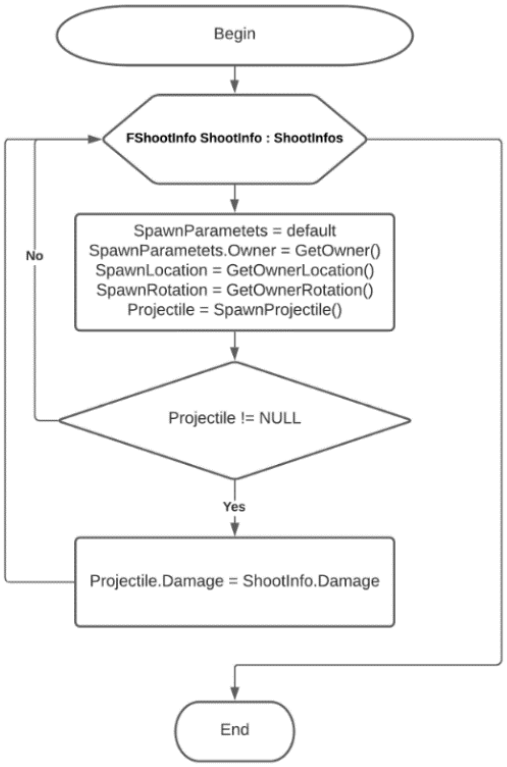


Рисунок 2.13 – Блок-схема появи пострілів

Для нескінченної появи ворогів створено "спавнер", який буде спавнити ворогів за алгоритмом зображеним на рисунку 2.14, і використовувати дані з структури EnemySpawnInfo щоб вороги з’являлися у потрібному місці з певними параметрами.

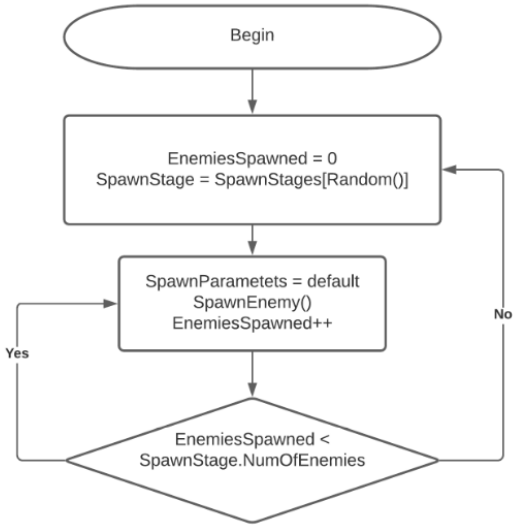


Рисунок 2.14 – Блок-схема роботи “спавнера”

Кожна стадія обирається випадково з масиву, далі спавняться усі вороги, які занесені у дану стадію, і поки кількість не досягне заданої.

Також з можливих проблем це те що дані алгоритми повинні працювати в мультиплеєрному режимі також. Щоб цього досягти без втрати їх ефективності потрібно щоб усі дії були репліковані відповідно до потреб. Поява пострілів має вираховуватися на сервері і репліковуватися на клієнти, так само з “спавнером”.

“Спавнер” має на сервері вирахувати відповідну стадію, на своїй стороні створити ворога, який має в свою чергу реплікуватися на всіх клієнтах.

2.4 Проектування мультиплеєру

Мультиплеєр у Unreal Engine здійснюється на основі взаємодії клієнт-сервер, де сервер є головним, а клієнти підключаються до нього. Також клієнти мають обмежений доступ до класів на рівнях, тобто їм не доступні усі класи – деякі є тільки на сервері.

У Unreal Engine є 2 типи серверів:

− виділений сервер – представлений у вигляді консолі і без «в’юпорту» для гравця;

− сервер-слухач – один з гравців є і сервером.

Для даного проекту буде використаний тип серверу слухач.

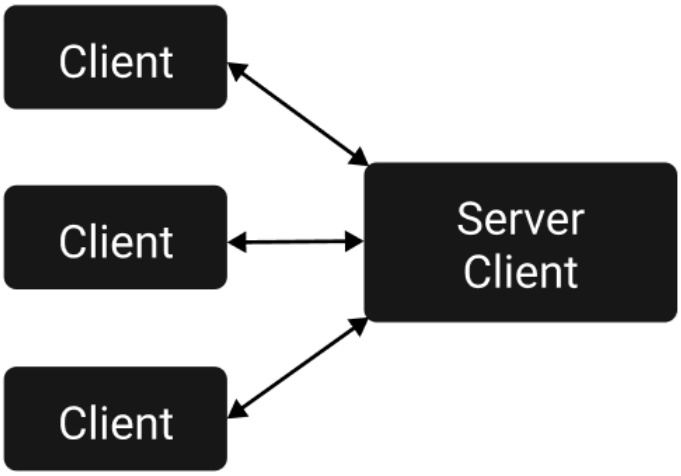


Рисунок 2.15 – Схема роботи серверу

Згідно схеми зображеній на рисунку 2.12, усі клієнти підключаються до серверу де сервер обробляє підключення і видає кожному свій контроллер і вже клієнти можуть спілкуватися з сервером через виданий контролер.

Так як для мультиплеєрного режиму буде використано функціонал Unreal Engine, яка має назву Unreal Engine networking. Вона в собі несе весь функціонал з підключення, підтримання з’єднання та його відновлення. На рахунок безпеки, то єдиною небезпекою для даної гри являються шахраї, які використовую зовнішні програми для отримання переваги в грі. Так як гра мультиплеєрна, використано підхід що усі елементи під керуванням гравців обчислюються на сервері, тому будь-які підміни даних на локальних пристроях не змінять нічого.

2.5 Висновки

В результаті проектування були визначені осовні завдання для розробки ігрового додатку. Також було діаграми послідовностуй для розуміння логіки роботи додатку.

Після чого було обрано необхідний ігровий рушій та інтегроване середовище розробки.

Було спроектовано необхідні алгоритми для роботи основних механік з відповідними блок-схемами.

Спроектована структура інтерфейсу додатку та зроблені шаблони меню для кожного пункту.

Визначено тип потрібного серверу для мультиплеєрної складової додатку.

В майбутньому бажано розширити геймплейну складову ігрового застосунку шляхом додавання різних рівнів, босів та більшої кількості різноманітних супротивників.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Конфігурація проєкту

Так як для розробки використовується кроссплатформений ігровий рушій, який має всі необхідні інструменти щоб писати програмні продукти під велику кількість наявних платформ. Так як цільовою платформою є Android потрібно встановити необхідні інструменти для розробки.

Щоб розробляти під Android потрібно встановити всі необхідні пакети, які можна отримати з офіційного сайту Android Studio, і скачати звідти Android Studio Flamingo, яка зображена на рисунку 3.1.

Рисунок 3.1 – необхідна Android Studio

Далі в завантаженій Android Studio потрібно вибрати необхідні пакети Android SDK, які зображені на рисунку 3.2.

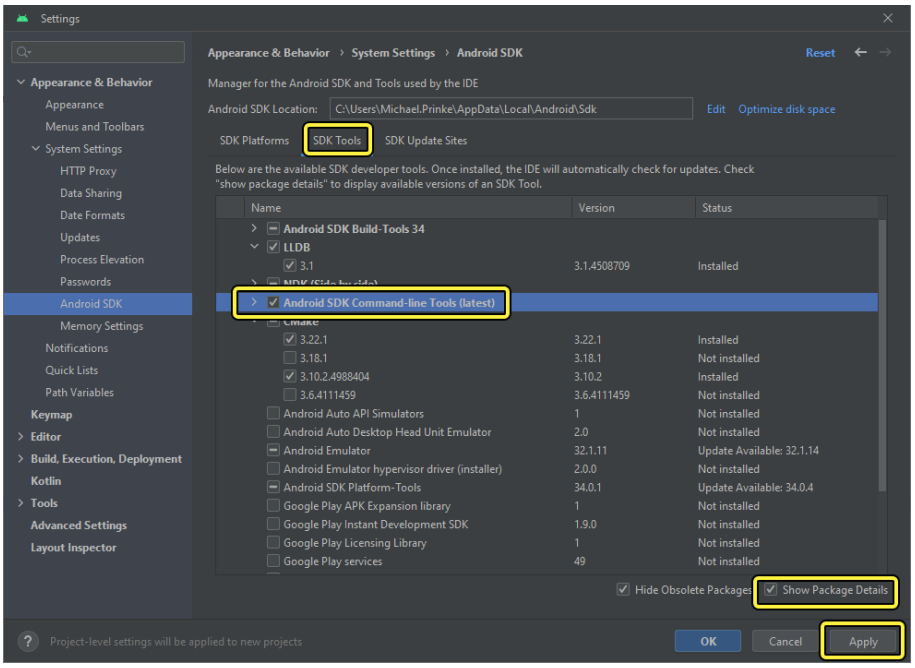


Рисунок 3.2 – необхідні пакети SDK

Після чого у вікні створеного проекту у вкладці Edit / Project Settings / Android SDK, зображено на рисунку 3.3, обрати всі необхідні файли, які були завантажені за допомогою Android Studio.

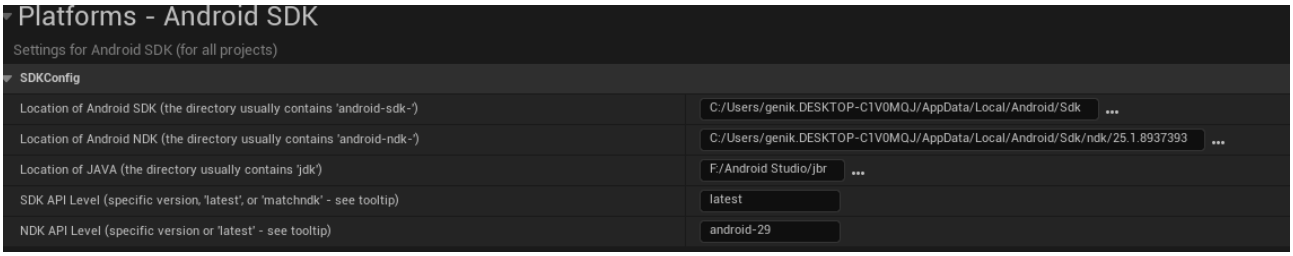


Рисунок 3.3 – конфігурація проєкту Unreal Engine

Далі потрібно встановити необхідні плагіни для роботи - це можна зробити з вкладки Settings / Plugins і обрати вказані плагіни зображені на рисунку 3.4.

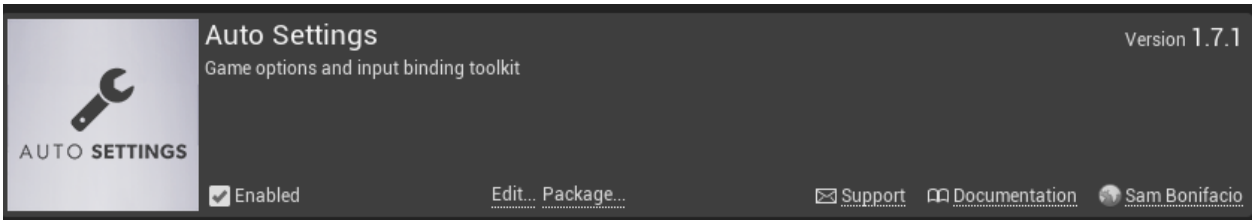


Рисунок 3.4 – Встановлення необхідних плагінів для проєкту

Для того щоб збірка пройшла успішно, потрібно перевірити чи усі плагіни тим чи іншим чином впливають на гру, щоб вони знаходилися в локальній папці з проектом також, потрібно щоб файл проетку *.uproject зображений на рисунку 3.5 виглядав приблизно так само.


```
{
    "FileVersion": 3,
    "EngineAssociation": "5.3",
    "Category": "",
    "Description": "",
    "Modules": [
        {
            "Name": "StreamArcade",
            "Type": "Runtime",
            "LoadingPhase": "Default",
            "AdditionalDependencies": [
                "Engine"
            ]
        }
    ],
    "Plugins": [
        {
            "Name": "AutoSettings",
            "Enabled": true,
            "MarketplaceURL": "com.epicgames.la
        },
        {
            "Name": "ModelingToolsEditorMode",
            "Enabled": true,
            "TargetAllowList": [
                "Editor"
            ]
        }
    ]
}
```


Рисунок 3.5 – вигляд файлу *.uproject

Для того щоб процес розробки був зручніший, проект розроблявся з використанням системи контролю версій GIT. Так як Unreal Engine має влаштовану підтримку розробки GIT, яка зображена на рисунку 3.6, завантаживши git bash і git LFS, можна підключити проект до git bash:

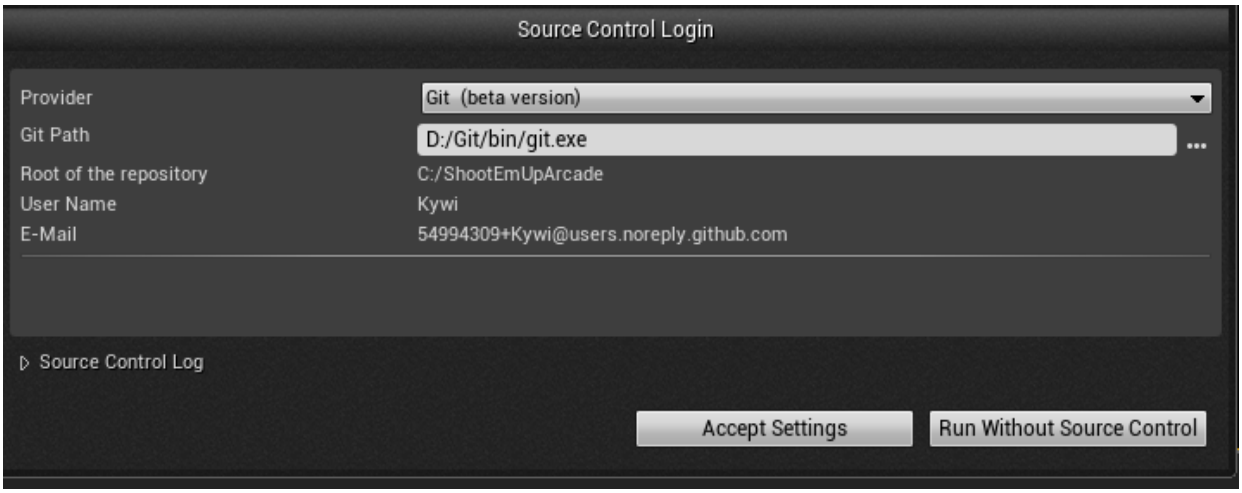


Рисунок 3.6 – Приклад підключення Unreal Engine до GIT

Це допоможе у подальшому якщо щось піде не так і буде можливість повернутися до попередньої версії.

3.2 Реалізація логіки розробки

Для написання коду у Unreal Engine потрібно створити C++ класи. Класи C++ створюються прямо з вікна редактора зображеного на рисунку 3.7.

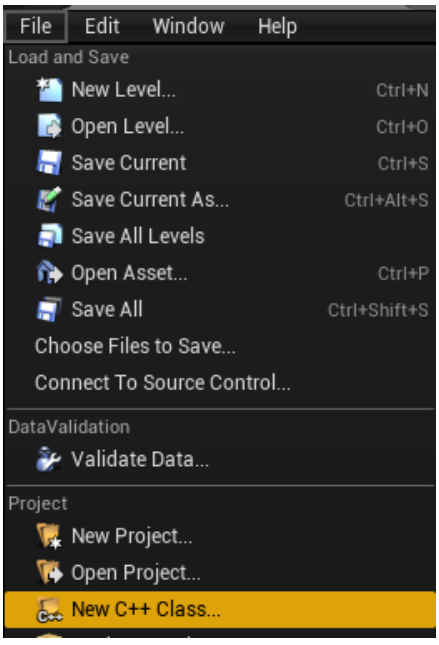


Рисунок 3.7 – Створення C++ класу

Після створення класів їх можна знайти у content браузері, у папці C++ Classes зображеній на рисунку 3.8.

Рисунок 3.8 – Вигляд папки з C++ классами

Для головного гравця створено клас C++ APlayerPawn, в якому визначено його поведінку на дотик по екрану, на отримання пошкоджень, на вихід за межі ігрового поля. Далі його вже наслідують Blueprint класи, зображені на рисунку 3.9, в яких визначений StaticMeshComponent, поставлена камера, підібрані розміри.

Рисунок 3.9 – Приклад Blueprint класів літака головного героя

Для ворогів аналогічним способом створено C++ клас EnemyPawn і наслідувані Blueprint класи які зображені на рисунку 3.10.

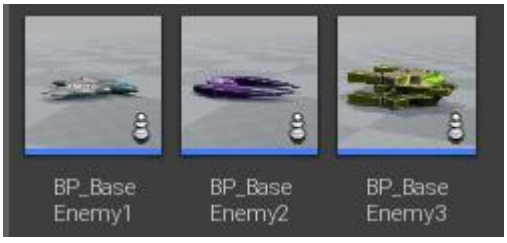


Рисунок 3.10 – Приклад Blueprint класів ворогів

Для стрільби був створений базовий клас на C++ ShootComponent, суттю якого є спавн снарядів, які і будуть наносити пошкодження.

```
  for (FShootInfo ShootInfo : ShootInfos)
    {
        FActorSpawnParameters SpawnParameters;
        SpawnParameters.Owner = GetOwner();
        SpawnParameters.SpawnCollisionHandlingOverride =
ESpawnActorCollisionHandlingMethod::AlwaysSpawn;
        FVector SpawnLocation =
            GetOwner()->GetActorLocation()
            +
            GetOwner()->GetActorRotation().RotateVector(ShootInfo.Offset);
        FRotator SpawnRotation = GetOwner()->GetActorRotation();
        SpawnRotation.Add(0.f, ShootInfo.Angle, 0.f);
        AShootProjectile* Projectile = GetWorld()-
>SpawnActor<AShootProjectile>(ShootInfo.ProjectileClass, SpawnLocation,
SpawnRotation, SpawnParameters);
        if (Projectile) Projectile->Damage = ShootInfo.Damage;
    }
```

Для управління очками здоров’я був створений клас HealthComponent, який при отриманні пошкоджень буде зменшувати своє здоров’я

```
void UGameHealthComponent::ChangeHealths(int ByValue)
{
      Healths += ByValue;
      HealthsChanged.Broadcast(ByValue);
      if (Healths <= 0) {
            HealthsEnded.Broadcast();
      }
}
```

Далі для реалізації нескінченного спавну ворогів створено клас EnemySpawnController, який являється компонентом і є складовою класу StramArcadeGamemodeBase. Для такого роду спавну використовується метод StartSpawnStage.

```
void UEnemySpawnController::StartSpawnStage()
{
   SpawnStage = SpawnStages[Random.RandRange(0, SpawnStages.Num()-1)];
   EnemiesSpawned = 0;
   SpawnEnemy();
   float ChangeStageTime = Random.RandRange(StageMinDelay, StageMaxDelay) *
ChangeStageTimeMultiplier;
```

```
   GetWorld()->GetTimerManager().SetTimer(ChangeStageTimer, this,
&UEnemySpawnController::StartSpawnStage, ChangeStageTime, false);
}
void UEnemySpawnController::SpawnEnemy()
{
   FActorSpawnParameters SpawnParameters;
   GetWorld()->SpawnActor<AEnemyPawn>(SpawnStage.EnemyClass,
SpawnStage.SpawnTransform, SpawnParameters);
   EnemiesSpawned++;
   if (EnemiesSpawned < SpawnStage.NumOfEnemies) {
      GetWorld()->GetTimerManager().SetTimer(EnemySpawnTimer, this,
&UEnemySpawnController::SpawnEnemy, SpawnStage.SpawnDelay, false);
   }
}
```

Даний компонент автоматично активується і починає спавнити ворогів, як тільки в ігровому режимі створений компонент даного класу, при початку гри.

Так як у грі повинні бути бонуси для них створено клас ABonus, який релізує базовий функціонал перетину колізії.

```
void ABonus::NotifyActorBeginOverlap(AActor* OtherActor)
{
    Super::NotifyActorEndOverlap(OtherActor);

    UE_LOG(LogTemp, Log, TEXT("Bonus overlap"));
    if (!OtherActor) return;
    if (!Cast<APlayerPawn>(OtherActor)) return;

    UE_LOG(LogTemp, Log, TEXT("Bonus CHAR overlap"));
    BonusCollected();
}
```

Щоб визначити RPC функцію потрібно використати макрос і вказати у ньому відповідні налаштування.

```
UFUNCTION(NetMulticast, Reliable)
      void ExplodePawn();
```

Ця декларація дозволить даному методу викликатися на всіх клієнтах, де існує даний актор. Далі для визначення функції потрібно написати ім’я функції_Implemetation.

```
void ANWPlayerPawn::ExplodePawn_Implementation()
{
      SetActorEnableCollision(false);
      bCanBeDamaged = false;
      shootComponent->StopShooting();
```

```
	for (UActorComponent* Component :
GetComponentsByClass(UParticleSystemComponent::StaticClass()))
	{
		Component->Activate(false);
	}

	pawnMesh->SetMaterial(0, RecoverMaterial);
	if (DestroyParticle)
		UGameplayStatics::SpawnEmitterAtLocation(GetWorld(), DestroyParticle,
GetActorTransform(), true);
}
```

Даний метод призначений для спавну ефекту та зміни матеріалу при отриманні пошкоджень гравцем.

Для бонусів з різними властивостями створено різні класи, і для зручності всі ці класи віднаслідувані, використовуючи Blueprint зображені на рисунку 3.11.

Рисунок 3.11 – Приклад Blueprint класів бонусів

Майже кожен C++ клас має наслідуваний Blueprint клас, так як конфігурувати його у Blueprint набагато зручніше та швидше.

Кожна цільова платформа має свої особливості, для коректної роботи Android додатку потрібно запитати в пристрою необхідні дозволи « android.permission.ACCESS_NETWORK_STATE» та «android.permission.INTERNET». Перевірити на наявність відповідних дозволів та запросити їх можна використовуючи Blueprint, зображений на рисунку 3.12.

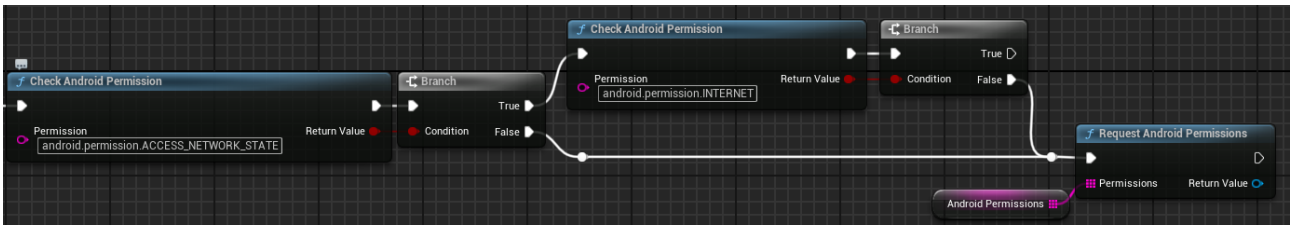


Рисунок 3.12 – Blueprint запиту необхідних дозволів

Застосунки які зібрані під платформу Android мають особливість, що там стоїть обмеження на 30 кадрів/с потрібного його зняти і виключити вертикальну синхронізацію, це зображено на рисунку 3.13.

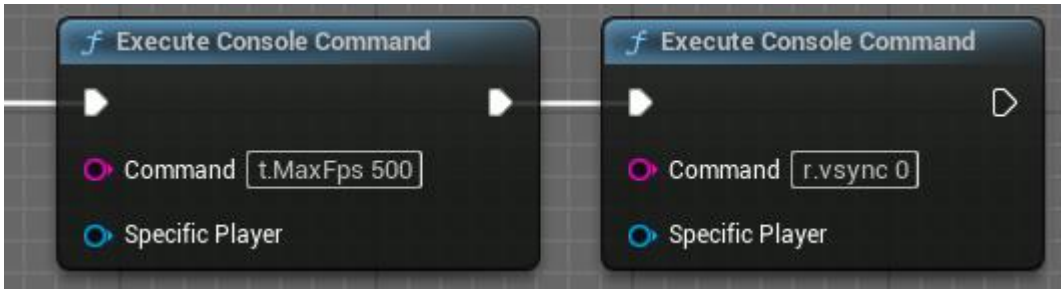


Рисунок 3.13 – Blueprint зміни налаштувань

3.3 Інтерфейс користувача

Для роботи інтерфейсу на Android платформі потрібно в налаштуваннях проєкту обрати пункт «Use mouse for touch», який зображено на рисунку 3.14.

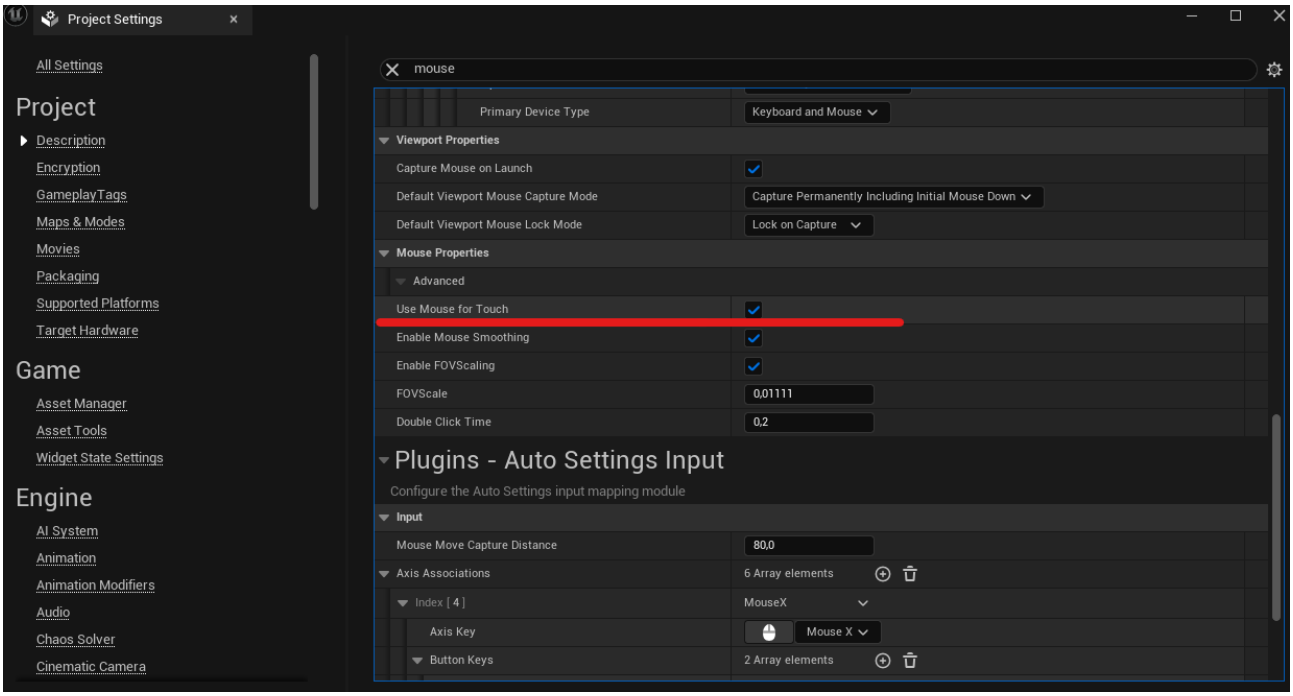


Рисунок 3.14 – Налаштування проєкту для Android

Після того як даний пункт обрано усі дотики на екрані смартфонів будуть працювати.

Інтерфейс програми буде реалізовано готовими методами Unreal Engine, які називаються Widget, в поєднанні з плагіном Autosettings, що дасть можливість швидко та зручно створити UI разом з налаштуваннями графіки та звуку.

Для цього у проекті було створено окрему папку, яка має назву UI, де зберігаються всі UI елементи, яка зображена на рисунку 3.15.

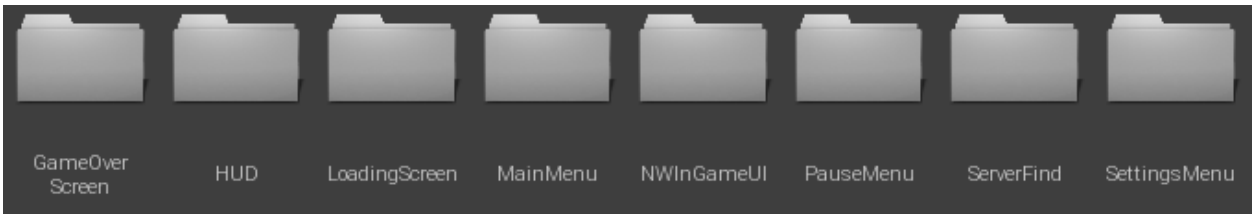


Рисунок 3.15 – Перелік усіх створених UI компонентів

Далі для кожного типу UI був створений клас WidgetBlueprint, який зображений на рисунку 3.16, після створення Blueprint класу даного типу і його

подальшого відкриття відкривається дизайнер, де можна зручно розмістити елементи керування і надати їм необхідний стиль.

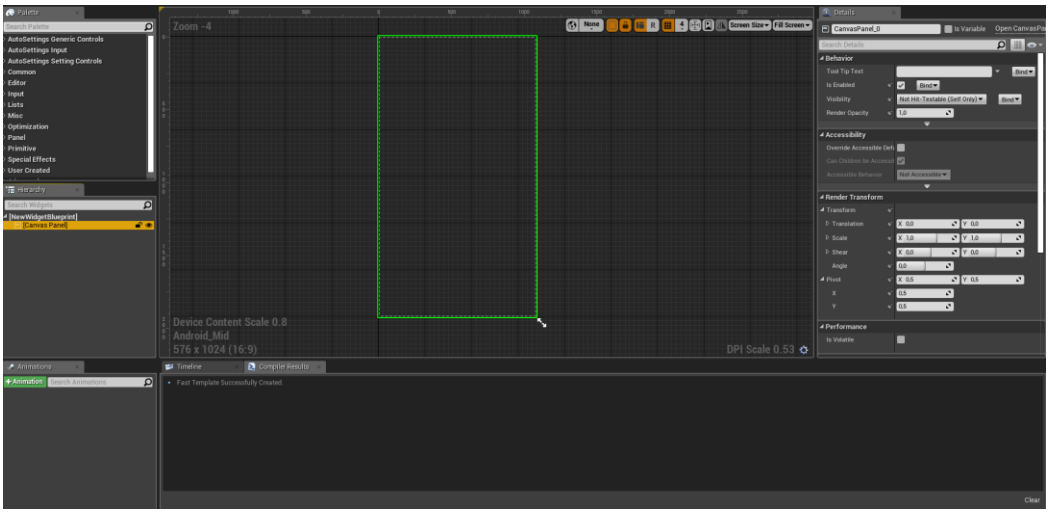


Рисунок 3.16 – Приклад вигляду дизайнера Widget

Усі елементи інтерфейсу знаходяться в контейнерах,

Після додавання необхідних елементів і налаштування їхніх стилів отримується наступний вигляд віджету головного меню, який зображений на рисунку 3.17.

Рисунок 3.17 – Вигляд віджету головного меню

Далі потрібно зробити меню вибору літального апарату, яке зображене на рисунку 3.18.

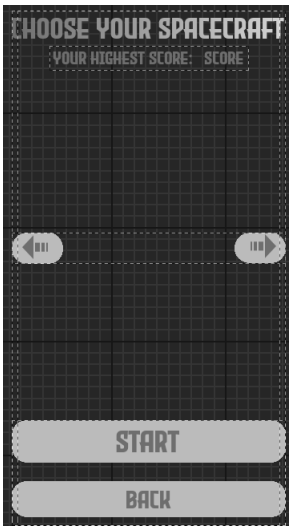


Рисунок 3.18 – Вигляд віджету меню вибору літального апарату

Наступним кроком потрібно зробити меню паузи в яке гравець може потрапити під час самої гри, яке зображена на рисунку 3.19.

Рисунок 3.19 – Вигляд віджету меню паузи

Для якісного ігрового досвіду гравця обов’язково потрібне меню налаштувань, де можна буде налаштувати:

– загальну гучність;

- гучність музики під час гри;
- гучність внутрішньоігрових ефектів;
- гучність музики у меню;
- налаштування графіки.

Меню налаштувань зображене на рисунку 3.20.

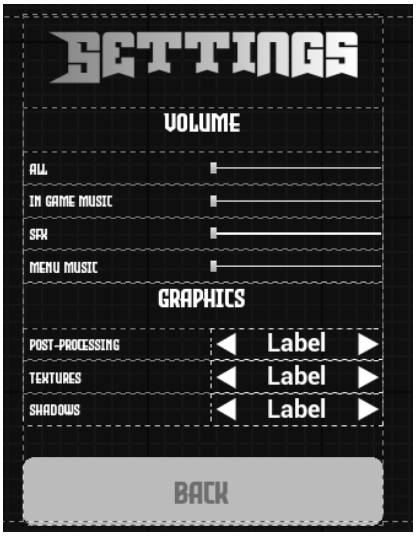


Рисунок 3.20 – Вигляд віджету меню паузи

Аналогічно і для наступних віджетів.

Усі готові ассети, які були використані, поміщені у спеціальну папку, яка називається Assets. В ній вже в залежності від категорії ассет попадає у відповідну папку. Такий спосіб групування полегшує подальшу розробку так як ассети завжди знаходяться в одному місці.

3.4 Технічні характеристики програмного забезпечення

Для коректної роботи додатка потрібно щоб Android пристрій відповідав наступним характеристикам:

- об’єм ОП – не менше 4 ГБ;
- процессор не нижче Qualcomm Snapdragon 636;

− операційна система – Android (версії 9 і вище);

− наявність підключення до локальної мережі (для використання можливостей мультиплеєру).

Протестовані пристрої зі схожими характеристиками наведені у таблиці 3.1.

Таблиця 3.1 – Протестовані пристрої.

Назва	Середня частота кадрів (кадри/секунду)
Xiaomi Redmi Note 5	50 к/с
Motorola Moto G7 Plus	58 к/с
Nokia 6.1 Plus	44 к/с
Asus Zenfone 5	66 к/с

На основі результатів протестованої працездатності гри на пристроях зі схожими технічними характеристиками до рекомендованих, можна зробити висновок що технічні вимоги для апаратного пристрою підібрані корректно.

3.4 Інструкція користувача

Діаграма розгортання (Deployment diagram) — діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах, яка зображена на рисунку 3.21.

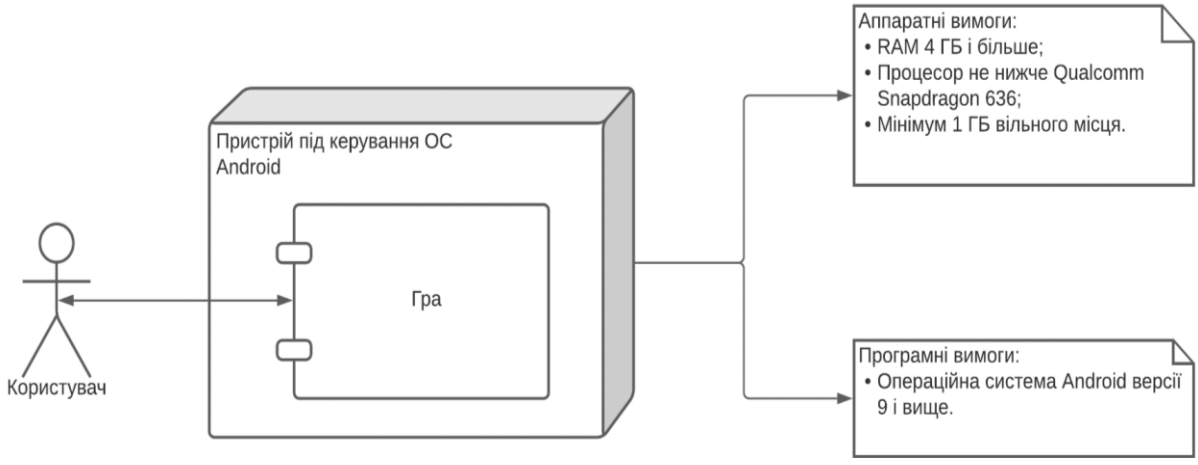


Рисунок 3.21 – Діаграма розгортання

Вузлом діаграми або ж фізичним ресурсом є пристрій під керуванням ОС Android, на якому завантажена програма.

Кінцевий користувач отримує *.apk файл, зображений на рисунку 3.22, який йому потрібно встановити на свій Android девайс.

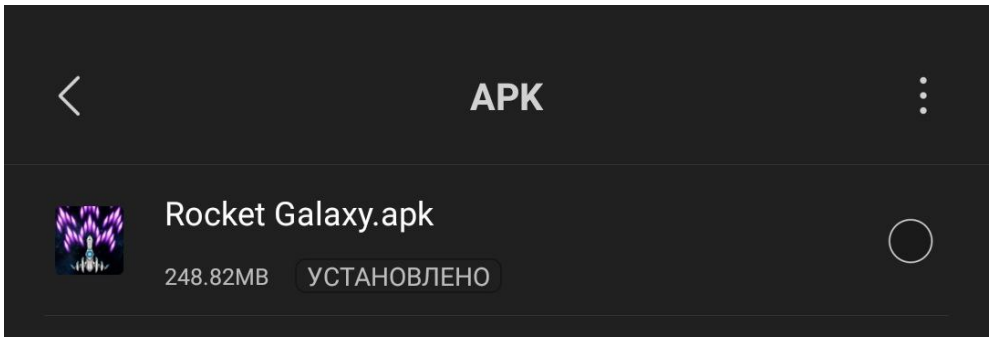


Рисунок 3.22 – Вигляд *.apk файлу

Після встановлення даного *.apk файлу можна виявити що на робочому столі з’явилася гра.

На рисунку 3.23 зображено встановлений додаток.

Рисунок 3.23 – Вигляд встановленого додатку

Після того як встановлено додаток він готовий до експлуатації

Після запуску додатку на виконання можна спостерігати вигляд головного меню, зображеного на рисунку 3.24.

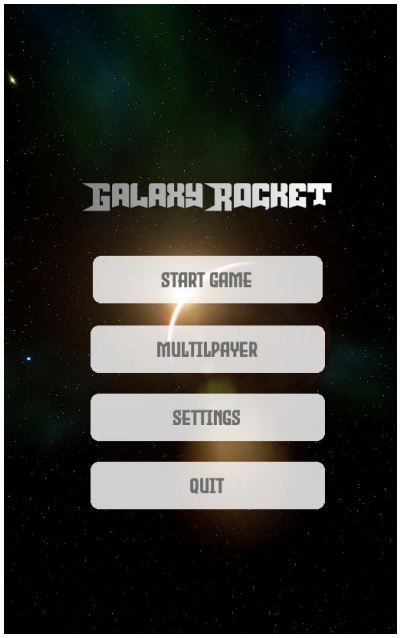


Рисунок 3.24 – Вигляд головного меню проєкту

Далі можна змінити налаштування, натиснувши кнопку “Settings”, де можна буде змінити як налаштування графіки, так і гучності звуку.

На рисунку 3.25 зображено інтерфейс меню налаштувань.

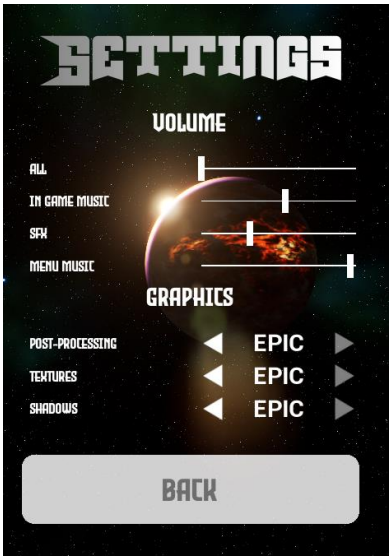


Рисунок 3.25 – Вигляд меню налаштувань

Після усіх проведених налаштувань, можна повернутися до головного меню, з якого потім вже натиснувши кнопку “Start Game”, отримати можливість вибору космічного літака серед доступних, а також переглянути рекордну кількість очок.

На рисунку 3.26 зображено меню вибору літака.

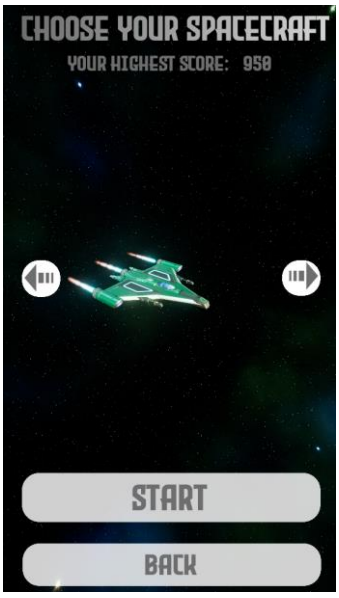


Рисунок 3.26 – Вигляд меню вибору літака

Після того як користувач натискає кнопку “Start”, завантажується рівень, де саме і відбувається головний геймплей, який зображений на рисунку 3.27.

Рисунок 3.27 – Приклад геймплею

Під час гри гравець може поставити гру на паузу, яка зображена на рисунку 3.28, під час якої гравець також може змінити налаштування, продовжити гру чи почати рівень спочатку.

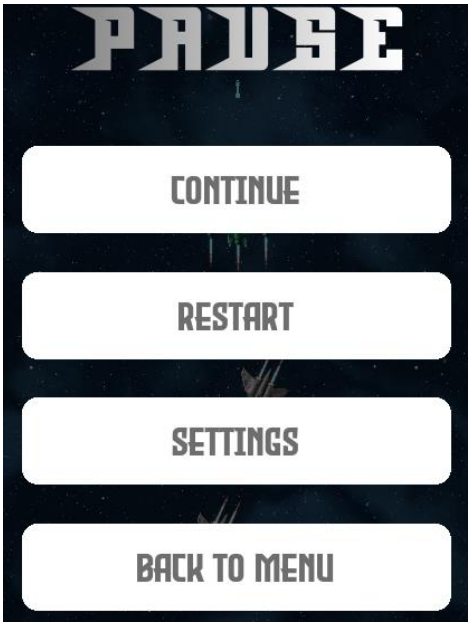


Рисунок 3.28 – Вигляд меню паузи

Як тільки у гравця закінчуються очки здоров’я, його корабель знищується і на екрані з’являється повідомлення, яке зображене на рисунку 3.29, де гравець може побачити свої накоплені очки, та почати рівень спочатку або повернутися до меню.

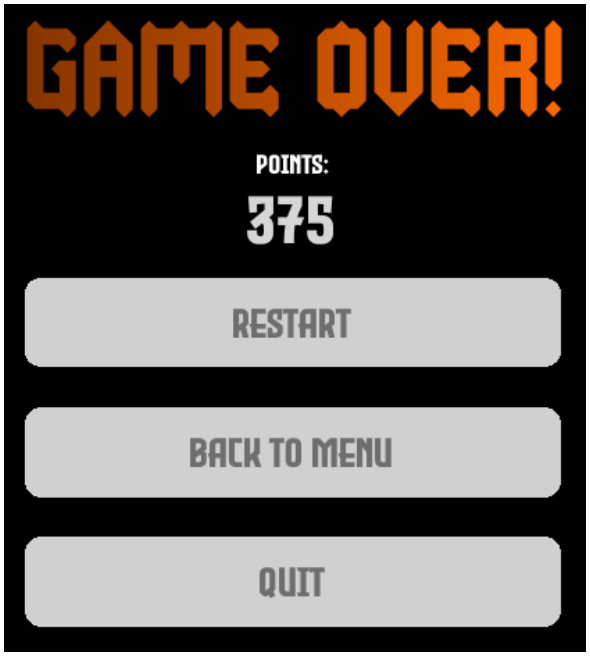


Рисунок 3.29 – Вигляд меню поразки

Отже, під час програмної реалізацій було описано кроки з налаштування проекту, налаштування середовища для розробки під Android пристрої. Після чого було підключено необхідні плагіни для розробки. Після чого було детально описано процес розробки додатку як одиночного режиму так і мультиплеєрної складової. Також були реалізовано інтерфейс меню користувача. Останнім кроком було написано інструкцію користувача, яка допоможе з налаштування Android додатку на правильним його функціонуванням.

Код програми подано у додатку Б.

3.5 Вибір та обґрунтування методів тестування додатку

Для мови програмування C++ існує багато способів тестування програмного забезпечення. Автоматизовані тести використовуються для спрощення та прискорення процесу перевірки коду. Одні з найвідоміших методів включають модульне та інтеграційне тестування. Модульне тестування (unit

testing) дозволяє тестувати окремі компоненти програми, що сприяє швидкому виявленню та усуненню помилок. На відміну від нього, інтеграційне тестування спрямоване на перевірку взаємодії між різними компонентами системи, що не дозволяє ізолювати конкретні помилки окремих модулів. Важливо зазначити, що для розробки цього додатку був використаний ігровий рушій Unreal Engine, який надає власний набір інструментів для написання модульних тестів.

Крім того, unit-тестування дозволяє автоматизувати процес перевірки, що значно скорочує час, необхідний на тестування, і підвищує загальну ефективність розробки. Автоматизовані тести можуть бути повторно використані під час подальших циклів розробки, що забезпечує безперервний контроль якості програмного забезпечення.

Вибір unit тестування для ігрового Android-застосунку у жанрі «Shoot ’em up» є важливим з кількох причин:

– забезпечення стабільності коду: Unit тестування допомагає виявляти помилки на ранніх стадіях розробки. Це особливо важливо для ігрових застосунків, де навіть невеликі баги можуть значно вплинути на ігровий процес і користувацький досвід;

– модульність та підтримка коду: використання unit тестування сприяє розробці більш модульного коду.

– автоматизація перевірок: Unit тести можуть бути виконані автоматично при кожному оновленні коду, що знижує ризик введення нових помилок і дозволяє розробникам швидко перевіряти зміни;

– продуктивність і швидкість розробки: Unit тести зазвичай виконуються дуже швидко і дозволяють розробникам швидше отримувати зворотний зв'язок щодо коду. Це сприяє більш швидкому циклу розробки і підвищенню продуктивності команди.

Впровадження unit тестування у розробку ігрового застосунку дозволяє забезпечити високу якість коду, стабільність гри та покращити користувацький досвід.

Окрім цього, значну увагу слід приділити інтерфейсу користувача, оскільки його функціональність безпосередньо впливає на загальну якість ігрового додатку. Відсутність помилок та зручність використання інтерфейсу є ключовими факторами, що визначають задоволення користувачів і їхній досвід від взаємодії з грою.

3.6 Налаштування проекту для створення модульних тестів

Так як проєкт розроблено на ігровому рушії Unreal Engine, то в ньому вже реалізований інструментарій для написання модульних тестів. Для початку потрібно підготувати проєкт. Потрібно створити папку Tests, далі в неї потрібно скопіювати необхідну карту та ассети. Карта повинна мати специфічну назву, початок імені карти повинен починатися з “FTEST_”, далі потрібно створити об’єкти, які наслідуються від класу FunctionalTest. Як тільки клас створено, його потрібно поставити на сцену тестового рівня, як зображено на рисунку 3.30.

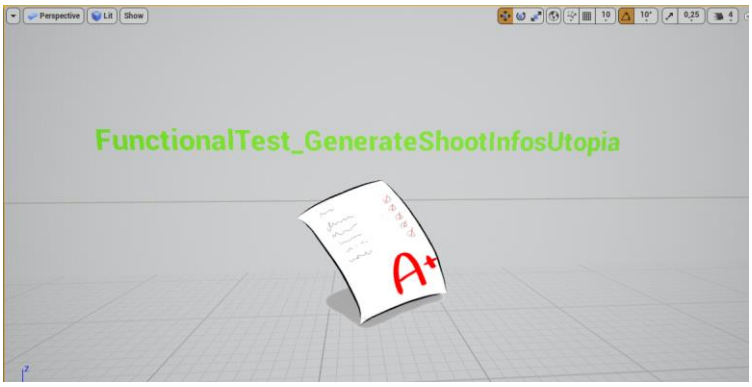


Рисунок 3.30 – Приклад розміщення тесту на рівні

Наступним кроком заходимо Window / Developer Tools / Session Frontend / Automation, далі можна побачити створені раніше тести, які зображені на рисунку 3.31.

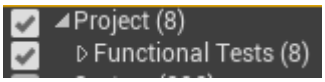


Рисунок 3.31 – Вигляд створених тестів

Для запуску створених тестів достатньо обрати потрібні тести і натиснути кнопку початку чи зупинки тестів, яка зображена на рисунку 3.32.

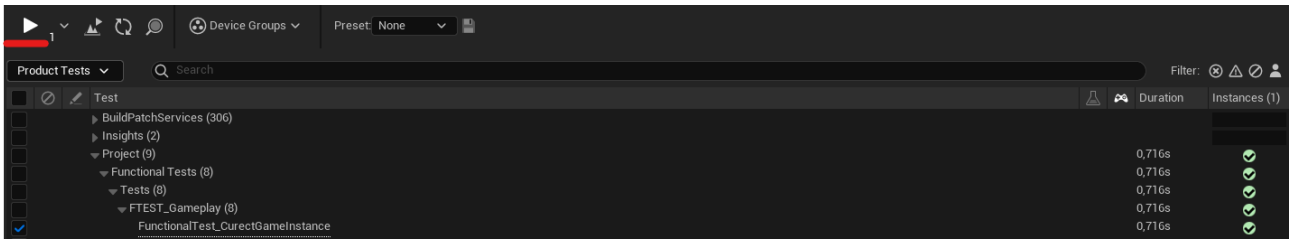


Рисунок 3.32 – Вигляд запущених тестів

3.7 Доведення працездатності програми

Для перевірки працездатності програмного забезпечення планується застосовувати unit-тестування. Цей підхід дозволяє тестувати окремі компоненти програми, забезпечуючи точну локалізацію помилок. У нашому випадку особливо важливо переконатися, що використовуються правильні об'єкти глобальних класів, літальні апарати з'являються коректно, а управління ними передається належним чином.

У таблиці 4.1 наведено сценарії проведення модульних тестів, які включають різні аспекти функціонування програми. Зокрема, увага приділяється перевірці коректного створення та ініціалізації об'єктів, належного функціонування методів і класів, а також відповідності їхньої роботи вимогам, заданим у технічному завданні. Це допоможе виявити і виправити будь-які невідповідності або помилки на ранніх етапах розробки.

Таблиця 4.1 – Сценарії для проведення модульних тестів

Команда	Опис	Вхідні дані	Очікуваний результат
GetGameMode	Взяття поточного ігрового режиму	Наявний об'єк ігрового режиму	Отриманий валідний об'єкт є валідний і має тип StreamArcadeGameMode
GetGameInstance	Взяття поточного об’єкту гри	Наявиний об'єкт ігрові сессії	Отриманий валідний об'єкт та відповідає типу MyNWGameInstance
GenerateColosusShootInfo	Створення еволюції для літального апарату типу “Colosus”	Згенерований масив з структур яка відповідає еволюції для літального апарату	Створено коректну структуру для еволюції літального апарату відповідного типу “Colosus”.
GenerateUtopiaShootInfo	Створення еволюції для літального апарату типу “Utopia”	Згенерований масив з структур яка відповідає еволюції для літального апарату	Створено коректну структуру для еволюції літального апарату відповідного типу “ Utopia ”.
GeneratePelicanShootInfo	Створення еволюції для літального апарату типу “Pelican”	Згенерований масив з структур яка відповідає еволюції для літального апарату	Створено коректну структуру для еволюції літального апарату відповідного типу “Pelican”.

Результат запуску даних тестів зображено на рисунку 3.33.

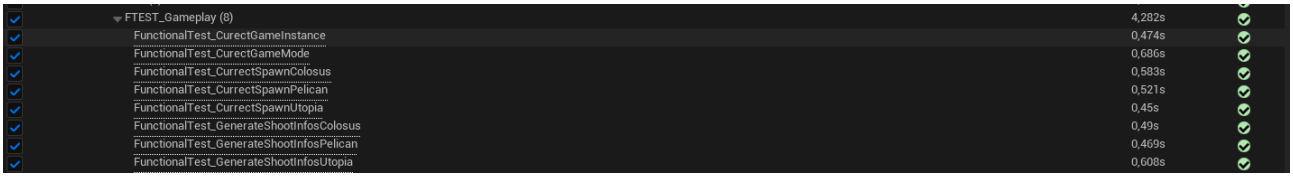


Рисунок 3.33 – Результат запуску модульних тестів

Тестовий сценарій – це послідовність дій по перевірці системи і відповідних результатів на її виході. Дана інформація може подаватися у вигляді таблиць, в якому кожен рядок є стан системи, вплив на неї та отриманий результат. Ці сценарії охоплюють різні аспекти взаємодії користувача з програмою, включаючи перевірку правильності відображення елементів інтерфейсу, реакцію системи на дії користувача та відповідність інтерфейсу встановленим вимогам. Наприклад, для кожної дії користувача, такої як натискання кнопки або введення даних у форму, зазначаються очікувані результати, що дозволяє швидко виявити та виправити будь-які недоліки.

Таблиця 4.2 – Сценарій для тестування інтерфейсу користувача.

Дія	Реакція програми	Результат
1. Натиснути кнопку «Settings»	Відкриється меню налаштувань	Вірно
2. Змінити налаштування гучності	Гучність музики в головному меню змінюється	Вірно
3. Вийти з меню налаштувань натиснувши кнопку «Back»	Відкриється головне меню	Вірно
4. Зайти знову до меню налаштуваню натиснувши кнопку «Settings»	Відкриється меню налаштувань і змінені налаштування мають бути збережені і відображаться	Вірно

У цьому розділі демонструється процес тестування розробленого програмного забезпечення та здійснюється його детальний аналіз. Ігровий додаток успішно пройшов етап тестування, що включав перевірку функціональності, стабільності та зручності використання.

ВИСНОВКИ

Для реалізації проєкту необхідно виконути наступні завдання:

– визначено спицифіку предметної області;

– проведено аналіз схожого та наяавного програмного забезпечення та визначені їх переваги та недоліки;

– проведено аналіз наявних інструментів для створення ігрових додатків;

– реалізовано проект програмним шляхом;

– проведене тестування ігрового додатку.

Також було комплексно описано процес розробки архітектури, розроблено інтерфейс застосунку з урахуванням особливостей роботи на мобільних платформах, розрбка необхідних алгоритмів та обгрунтування вибору мультиплеєрної моделі.

Для розробки застосунку був обраний ігровий рушій Unreal Engine з мовою програмування C++ разом з системою сценаріїв Blueprints. Для розробки на C++ використано MS Visual Studio.

Реалізовано декілька різних видів космічних літаків, якими може керувати користувач, кожен має свій тип снарядів та гілку “еволюції”. Ворогів також реалізовано декілька типів і їх нескінченний спавн. Також реалізовано збереження максимальної кількості очок. Була проведена робота зі звуком, а саме фонова музика та SFX звуки.

Також з використанням віджетів було реалізовано зручне меню, у якому можливо обрати літак для початку гри, змінити налаштування як графіки, так і звуку.

Реалізовані бонуси (бусти). Вони випадають з ворогів при знищенні, що додає геймплею різноманіття.

Реалізовано режим мультиплеєру, в якому присутні всі особливості однокористувацької гри.

У ході тестування програми було описано кроки для налаштування проекту специфічні для використаних інструментів, описані тест-кейси для виявлення проблем та сценарії для тестування графічного інтерфейсу.

Проведене тестування показало що всі функції даного застосунку працюють коректно.

Отже, в рамках виконання дипломного проекту був створений ігровий додаток для Android, який повністю відповідає всім критеріям та вимогам, визначеним у процесі дослідження предметної області. Від самого початку розробки основною метою було забезпечення високої якості продукту, що включає як технічну, так і функціональну відповідність.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Документація по ігровому рушію Unreal Engine : веб-сайт. URL: https://docs.unrealengine.com/en-US/index.html (дата звернення 11.04.2024)

2. Документація по плагіну Autosettings : веб-сайт. URL: https://acren.github.io/AutoSettingsDocs/ (дата звернення 20.03.2024)

3. Документація по плагіну StarSphere : веб-сайт. URL: http://unreal.loomman.com/starsphere/ (дата звернення 25.03.2024)

4. Форум розробників Unreal Engine 4 : веб-сайт. URL: https://forums.unrealengine.com/ (дата звернення 25.04.2024)

5. Unreal Engine : веб-сайт. URL: https://en.wikipedia.org/wiki/Unreal_Engine (дата звернення 10.03.2024)

6. Навчальний : веб-сайт. URL: https://www.unrealengine.com/en-US/learn (дата звернення 15.04.2024)

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проєкту розробки мобільного додатку «Rocket Galaxy» для ОС Android. Технічне завдання розроблено у відповідності до стандарту ГОСТ 19.201–78.

1 Підстава для розробки

Підставою для розробки є «Завдання на кваліфікаційну роботу», затверджене завідувачем кафедри інженерії програмного забезпечення.

Найменування розробки: «Ігровий Android-застосунок у жанрі «Shoot ’em up»».

2 Призначення розробки

2.1 Функціональне призначення

Геймплей та управління:

– реалізація основного геймплею "shoot'em up" зі здійсненням стрільби ворожими силами та униканням їх атак;

– риведення ігрового персонажу на екран;

– реалізація системи керування, яка дозволяє гравцеві маневрувати, стріляти та виконувати спеціальні дії.

Ворожі сили:

– створення ворожих об'єктів, таких як кораблі, снаряди;

– реалізація алгоритмів поведінки ворожих сил, включаючи їх рух, стрільбу та тактичні атаки.

Система зброї та апгрейдів:

– введення різних типів зброї, які може використовувати гравець;

– реалізація системи апгрейдів для зброї та корабля гравця, таких як збільшення потужності вогню, швидкості.

Рівні та прогресія:

– створення ігрових рівнів з унікальними ворожими формаціями;

– реалізація системи прогресії, що дозволяє гравцеві розблоковувати нові рівні, зброю та інші відкриття під час гри.

Графіка та звук:

– створення візуально привабливих об'єктів, фонів та анімацій;

– реалізація аудіо ефектів, музики та звукового супроводу, що відтворюється під час гри.

Інтерфейс користувача:

– розробка інтуїтивного інтерфейсу користувача з ігровими елементами, такими як відображення життя, кількості апгрейдів тощо;

– реалізація елементів меню, таких як головне меню, налаштування гри, вибір рівнів тощо.

2.2 Експлуатаційне призначення

Програма повинна експлуатуватися на мобільних пристроях під керуванням ОС Android.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати можливості виконання таких функцій:

– функція початку гри;

– функція зміни літального апарату;

– функція початку мультиплеєрної гри;

– функція підключення до створеної мультиплеєрної гри;

– функція редагування налаштувань.

3.2 Вимоги до надійності

Ігровий застосунок повинен працювати стабільно, повинні бути відсутні графічні артефакти.

3.3 Умови експлуатації та вимоги до технічних засобів

Додаток призначено для використання на смартфонах, що працюють під управлінням операційної системи Android.

Для успішного використання додатка необхідно, щоб пристрій відповідав таким рекомендованим вимогам:

– ОС Android 10.0 або вище;

– будь-який чотирьохядерний процесор;

– 3 ГБ ОЗП;

– 300 МБ постійної пам’яті.

3.4 Вимоги до інформаційної та програмної сумісності

При розробці додатку використовуватиметься ігровий рушій Unreal Engine, який використовує об’єктно орієнтована мову C++, який призначений для створення ігрових додатків під Android, iOS, Windows.

4 Вимоги до програмної документації

У момент здачі проекту замовнику надається наступний набір документів:

– текст програми;

– опис програми;

– технічне завдання;

– керівництво користувача.

4 Стадії розробки

Стадія та період	Етап	Зміст
1	2	3
Технічне завдання, січень	Обґрунтування необхідності розробки програми	Вибір теми дипломного проєкту, короткий опис ПЗ, вимоги до ПЗ, етапи розробки ПЗ
Ескізний проєкт, січень-лютий	Розробка ескізного проекту	Створення базової структури системи, вибір платформи та технологій
Технічний проєкт, лютий-березень	Розробка технічного проекту	Уточнення структури, вибір кінцевих технологій для розробки та проектування дизайну ПЗ
Робочий проєкт, квітень	Розробка програмного забезпечення	Програмна розробка ПЗ, створення додатку, що відповідає всім стандартам
Розробка документації, травень	Розробка документації для програмного забезпечення	Підготовка супровідної документації для проекту, створення

		ілюстративних матеріалів

1	2	3
Тестування, травень	Проведення тестування програмного забезпечення	Тестування окремих модулів і системи в цілому, усунення виявлених неузгоджень у програмному забезпеченні
Впровадження, червень	Підготовка і передача програми	Підготовка необхідної документації для затвердження роботи, включаючи відгуки та рецензії.
Захист, червень	Розробка документації	Захист КвР

7 Порядок контролю та приймання

Контроль та затвердження виконуються користувачами Android додатку та керівниками проекту. Це включає перевірку функціональності програмного забезпечення і відповідність документації.

ДОДАТОК Б

(обов'язковий)

ФРАГМЕНТ КОДУ ПРОГРАМНОЇ СИСТЕМИ

StreamArcadeGamemodeBase.h

```
#pragma once

#include "CoreMinimal.h"

#include "GameFramework/GameModeBase.h"
#include "StructsCollection/StructsCollection.h"

#include "Components/EnemySpawnController.h"
#include "Components/GameHealthComponent.h"

#include "StreamArcadeGameModeBase.generated.h"

DECLARE_DYNAMIC_MULTICAST_DELEGATE(FGameOverEvent);

UCLASS()
class STREAMARCADE_API AStreamArcadeGameModeBase : public AGameModeBase
{
    GENERATED_BODY()

    AStreamArcadeGameModeBase();

    virtual void BeginPlay() override;

public:

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Enemies")
    UEnemySpawnController* EnemySpawnController;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Game Health")
    UGameHealthComponent* HealthsComponent;

    UPROPERTY(BlueprintAssignable, Category = "Game")
    FGameOverEvent GameOver;

    UFUNCTION(BlueprintCallable, Category = "Game")
    void EndGame();

    UFUNCTION(BlueprintCallable, Category = "Game")
    void IncreaseDifficulty();

    UFUNCTION(BlueprintCallable, Category = "Game")
    void AddPoints(int Points);

    UFUNCTION(BlueprintCallable, Category = "Game")
    bool ChangeShootLevel(bool Up);

    UPROPERTY(BlueprintReadWrite, Category = "Game")
    float PlayerRecoverTime;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Game")
    float IncreaseDifficultyPeriod;

    UPROPERTY(BlueprintReadOnly, Category = "Game")
    class APlayerPawn* PlayerPawn;

    UPROPERTY(BlueprintReadOnly, Category = "Game")
    int GamePoints;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Shooting")
    TArray<FShootInfoLevel> ShootInfoLevels;

    UPROPERTY(BlueprintReadOnly, Category = "Shooting")
```

```
    int CurrentShootLevel;


protected:
    UFUNCTION(BlueprintNativeEvent, Category = "Game")
    void ExplodePawn();
    void ExplodePawn_Implementation();

    UFUNCTION(BlueprintNativeEvent, Category = "Game")
    void RecoverPawn();
    void RecoverPawn_Implementation();

    FTimerHandle RecoverTimer;
    FTimerHandle IncreaseDifficultyTimer;

    bool IsGameOver;
};
```

StreamArcadeGamemodeBase.cpp

```
#include "StreamArcadeGameModeBase.h"
#include "Pawns/PlayerPawn.h"

#include "Engine/World.h"
#include "TimerManager.h"
#include "Kismet/GameplayStatics.h"


AStreamArcadeGameModeBase::AStreamArcadeGameModeBase():
    PlayerRecoverTime(3),
    IncreaseDifficultyPeriod(4.f),
    CurrentShootLevel(-1)
{
    EnemySpawnController =
CreateDefaultSubobject<UEnemySpawnController>(TEXT("EnemySpawnController"));
    HealthsComponent = CreateDefaultSubobject<UGameHealthComponent>(TEXT("HealthsComponent"));
}

void AStreamArcadeGameModeBase::BeginPlay()
{
    Super::BeginPlay();
    HealthsComponent->HealthsEnded.AddDynamic(this, &AStreamArcadeGameModeBase::EndGame);

    PlayerPawn = Cast<APlayerPawn>(UGameplayStatics::GetPlayerPawn(this, 0));
    if (!PlayerPawn) return;

    ChangeShootLevel(true);

    PlayerPawn->PawnDamaged.AddDynamic(this, &AStreamArcadeGameModeBase::ExplodePawn);

    GetWorld()->GetTimerManager().SetTimer(IncreaseDifficultyTimer, this,
                                           &AStreamArcadeGameModeBase::IncreaseDifficulty,
IncreaseDifficultyPeriod,
                                           true);
}

void AStreamArcadeGameModeBase::ExplodePawn_Implementation()
{
    PlayerPawn->ExplodePawn();

    HealthsComponent->ChangeHealths(-1);

    ChangeShootLevel(false);

    if (!IsGameOver)
        GetWorld()->GetTimerManager().SetTimer(RecoverTimer, this,
&AStreamArcadeGameModeBase::RecoverPawn,
                                               PlayerRecoverTime, false);
}

void AStreamArcadeGameModeBase::RecoverPawn_Implementation()
{
    PlayerPawn->RecoverPawn();
}

void AStreamArcadeGameModeBase::EndGame()
{
    IsGameOver = true;
```

```
    EnemySpawnController->SetActive(false);

    GameOver.Broadcast();

    UGameplayStatics::GetPlayerPawn(this, 0)->Destroy();

    UE_LOG(LogTemp, Log, TEXT("GAME OVER!!!"));

    SetPause(UGameplayStatics::GetPlayerController(this, 0), nullptr);
}

void AStreamArcadeGameModeBase::IncreaseDifficulty()
{
    EnemySpawnController->ChangeStageTimeMultiplier = FMath::Max(EnemySpawnController-
>ChangeStageTimeMultiplier * 0.95,
                                                                 0.4);
    UE_LOG(LogTemp, Log, TEXT("Difficulty increased: %f"), EnemySpawnController-
>ChangeStageTimeMultiplier);
}

void AStreamArcadeGameModeBase::AddPoints(int Points)
{
    GamePoints += Points;
}

bool AStreamArcadeGameModeBase::ChangeShootLevel(bool Up)
{
    PlayerPawn = Cast<APlayerPawn>(UGameplayStatics::GetPlayerPawn(this, 0));
    if (!PlayerPawn) return false;

    int NewLevel = FMath::Clamp(CurrentShootLevel + (Up ? 1 : -1), 0, ShootInfoLevels.Num() - 1);

    if (NewLevel == CurrentShootLevel) return false;

    CurrentShootLevel = NewLevel;

    PlayerPawn->ShootComponent->ShootInfos = ShootInfoLevels[CurrentShootLevel].ShootInfos;
    PlayerPawn->ShootComponent->ShootPeriod = ShootInfoLevels[CurrentShootLevel].ShootPeriod;
    UE_LOG(LogTemp, Log, TEXT("ChnagedShootPeriod: %f"),
ShootInfoLevels[CurrentShootLevel].ShootPeriod);
    if (Up) PlayerPawn->ShootComponent->RestartShooting();

    return true;
}
```

StructsCollection.h

```
#pragma once

#include "CoreMinimal.h"

#include "Actors/Bonuses/Bonus.h"
#include "Actors/Projectiles/ShootProjectile.h"

#include "StructsCollection.generated.h"


class AEnemyPawn;

USTRUCT(BlueprintType)
struct FShootInfo
{
        GENERATED_BODY()

public:
        UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Shooting")
        TSubclassOf<AShootProjectile> ProjectileClass;

        UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Shooting")
        float Damage;

        UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Shooting")
        FVector Offset;

        UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Shooting")
        float Angle;
};


USTRUCT(BlueprintType)
```

```
struct FEnemySpawnInfo
{
	GENERATED_BODY()

public:
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Enemies")
	TSubclassOf<AEnemyPawn> EnemyClass;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Enemies")
	FTransform SpawnTransform;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Enemies")
	int NumOfEnemies;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Enemies")
	float SpawnDelay;
};


USTRUCT(BlueprintType)
struct FBonusChance
{
	GENERATED_BODY()

public:

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Bonus")
	TSubclassOf<ABonus> BonusClass;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Bonus")
	float Chance;

};


USTRUCT(BlueprintType)
struct FShootInfoLevel
{
	GENERATED_BODY()

public:
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Shooting")
	TArray<FShootInfo> ShootInfos;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Shooting")
	float ShootPeriod;
};
```

EnemyPawn.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Pawn.h"
#include "Components/BoxComponent.h"
#include "Components/ShootComponent.h"
#include "Components/HealthComponent.h"
#include "Actors/Bonuses/Bonus.h"
#include "Components/ArrowComponent.h"
#include "Sound/SoundBase.h"
#include "EnemyPawn.generated.h"


UCLASS()
class STREAMARCADE_API AEnemyPawn : public APawn
{
	GENERATED_BODY()

public:
	// Sets default values for this pawn's properties
	AEnemyPawn();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

	void SpawnBonuses();
```

```
        UFUNCTION()
        void KillPawn();

        UFUNCTION()
        void OnEnemyOverlap(AActor* OverlapedActor, AActor* OtherActor);

public:
        // Called every frame
        virtual void Tick(float DeltaTime) override;

        UFUNCTION(BlueprintCallable, Category="Pawn")
        void DestroyPawn();

        UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Pawn")
        UBoxComponent* PawnCollision;

        UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Pawn")
        UArrowComponent* ArrowComponent;

        UPROPERTY(VisibleDefaultsOnly, BlueprintReadOnly, Category = "Pawn")
        UStaticMeshComponent* PawnMesh;

        UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Shooting")
        UShootComponent* ShootComponent;

        UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Shooting")
        UHealthComponent* HealthComponent;

        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Pawn")
        int DestroyPoints;

        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Bonus")
        TArray<FBonusChance> PossibleBonuses;

        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Visual")
        UParticleSystem* DestroyParticle;

        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Audio")
        USoundBase* DestroySound;
};
```

EnemyPawn.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.

#include "EnemyPawn.h"

#include "StreamArcadeGameModeBase.h"

#include "Kismet/GameplayStatics.h"
#include "GameFramework/DamageType.h"
#include "Components/StaticMeshComponent.h"

#include "Engine/World.h"

// Sets default values
AEnemyPawn::AEnemyPawn()
{
    // Set this pawn to call Tick() every frame.  You can turn this off to improve performance if
you don't need it.
    PrimaryActorTick.bCanEverTick = true;

    PawnCollision = CreateDefaultSubobject<UBoxComponent>(TEXT("PawnCollision"));
    RootComponent = PawnCollision;
    PawnCollision->SetCollisionProfileName("Pawn");

    PawnMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("PawnMesh"));
    PawnMesh->SetupAttachment(PawnCollision, NAME_None);
    PawnMesh->SetCollisionEnabled(ECollisionEnabled::NoCollision);

    ArrowComponent = CreateDefaultSubobject<UArrowComponent>(TEXT("Arrow"));
    ArrowComponent->SetupAttachment(PawnCollision, NAME_None);

    ShootComponent = CreateDefaultSubobject<UShootComponent>(TEXT("ShootComponent"));
    HealthComponent = CreateDefaultSubobject<UHealthComponent>(TEXT("HealthComponent"));
}

// Called when the game starts or when spawned
void AEnemyPawn::BeginPlay()
{
```

```
    Super::BeginPlay();

    HealthComponent->OnHealthEnded.AddDynamic(this, &AEnemyPawn::KillPawn);
    OnActorBeginOverlap.AddDynamic(this, &AEnemyPawn::OnEnemyOverlap);
}

void AEnemyPawn::KillPawn()
{
    AStreamArcadeGameModeBase* Gamemode =
Cast<AStreamArcadeGameModeBase>(UGameplayStatics::GetGameMode(this));
    if (Gamemode) Gamemode->AddPoints(DestroyPoints);

    SpawnBonuses();

    DestroyPawn();
}

void AEnemyPawn::DestroyPawn()
{
    if (DestroyParticle)
        UGameplayStatics::SpawnEmitterAtLocation(GetWorld(), DestroyParticle, GetActorTransform(),
true);
    UGameplayStatics::SpawnSound2D(GetWorld(), DestroySound);

    Destroy();
}

void AEnemyPawn::OnEnemyOverlap(AActor* OverlapedActor, AActor* OtherActor)
{
    if (OtherActor != UGameplayStatics::GetPlayerPawn(this, 0)) return;

    const float AppliedDamage = UGameplayStatics::ApplyDamage(OtherActor, 100.f, GetController(),
this, UDamageType::StaticClass());

    if (AppliedDamage > 0.f) DestroyPawn();
}

void AEnemyPawn::SpawnBonuses()
{
    FRandomStream Random;
    Random.GenerateNewSeed();

    FActorSpawnParameters SpawnParameters;
    SpawnParameters.SpawnCollisionHandlingOverride =
ESpawnActorCollisionHandlingMethod::AlwaysSpawn;

    for (FBonusChance Bonus : PossibleBonuses) {
        float RandChance = Random.RandRange(0.f, 100.f);
        UE_LOG(LogTemp, Log, TEXT("Bonus: %s, Chance needed: %f, Chance random: %f"),
*Bonus.BonusClass->GetName(), Bonus.Chance, RandChance);
        if (RandChance < Bonus.Chance) {
            UE_LOG(LogTemp, Log, TEXT("Bonus spawned"));
            GetWorld()->SpawnActor<ABonus>(Bonus.BonusClass, GetActorLocation(), FRotator(0.f),
SpawnParameters);
            break;
        }
    }
}

// Called every frame
void AEnemyPawn::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    const float WorldMoveOffset = -200.f * DeltaTime;
    AddActorWorldOffset(FVector(WorldMoveOffset, 0.f, 0.f));
}
```

PlayerPawn.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"

#include "GameFramework/Pawn.h"
#include "Components/BoxComponent.h"
#include "Camera/CameraComponent.h"
#include "Components/ShootComponent.h"
```

```
#include "PlayerPawn.generated.h"

DECLARE_DYNAMIC_MULTICAST_DELEGATE(FPawnDamagedEvent);

UCLASS()
class STREAMARCADE_API APlayerPawn : public APawn
{
    GENERATED_BODY()

public:

    APlayerPawn();

    virtual void Tick(float DeltaTime) override;
    virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;

    UFUNCTION(BlueprintPure, BlueprintNativeEvent, Category = "Healths")
    bool CanBeDamaged();
    bool CanBeDamaged_Implementation();

    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "Healths")
    void ExplodePawn();
    void ExplodePawn_Implementation();

    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "Healths")
    void RecoverPawn();
    void RecoverPawn_Implementation();

    UFUNCTION(BlueprintCallable, Category = "ScreeenRes")
    static FVector2D GetGameViewportSize();

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Pawn")
    UBoxComponent* PawnCollision;

    UPROPERTY(VisibleDefaultsOnly, BlueprintReadOnly, Category="Pawn")
    UStaticMeshComponent* PawnMesh;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Pawn")
    UCameraComponent* PawnCamera;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Shooting")
    UShootComponent* ShootComponent;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Controls")
    float TouchMoveSensivity;

    UPROPERTY(BlueprintAssignable, Category = "Healths")
    FPawnDamagedEvent PawnDamaged;

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Pawn")
    UMaterialInterface* RecoverMaterial;

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Visual")
    UParticleSystem* DestroyParticle;

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "RotationAnimation")
    float maxRotationAngle = 40;

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "RotationAnimation")
    float targetInterp = 0;

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "RotationAnimation")
    float delayTimerInterp = 0.05;

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "RotationAnimation")
    float stepInterp = 0.05;

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;
    virtual float TakeDamage(float Damage, const FDamageEvent& DamageEvent, AController*
InstigatedBy,
                             AActor* DamageCauser) override;
    void OnTouchMove(ETouchIndex::Type FingerIndex, FVector Location);
    void OnTouchPress(ETouchIndex::Type FingerIndex, FVector Location);
    void OnTouchReleased(ETouchIndex::Type FingerIndex, FVector Location);
    virtual void PossessedBy(AController* NewController) override;
    void RotationAnimation(const FVector& NewLocation);
```

```
    void RotateBack();
    APlayerController* PlayerController;
    FVector2D MoveLimit;
    FTimerHandle rotateAnimTimer;

private:
    FVector2D TouchLocation;
    float fromInterp;
    double currentRotation = 0;
    UMaterialInterface* PawnMaterial;
};
```

PlayerPawn.cpp

```
#include "PlayerPawn.h"


#include "AnimationCompression.h"
#include "Components/StaticMeshComponent.h"
#include "Particles/ParticleSystemComponent.h"
#include "Components/InputComponent.h"
#include "GameFramework/PlayerController.h"
#include "Kismet/GameplayStatics.h"
#include "TimerManager.h"
#include "EngineMinimal.h"
#include "Engine/Engine.h"
#include "Math/UnrealMathUtility.h"
#include "Kismet/KismetMathLibrary.h"

APlayerPawn::APlayerPawn() : TouchMoveSensivity(1.f)
{
    //  MoveLimit(FVector2D(500.f, 600.f))
    PrimaryActorTick.bCanEverTick = true;

    PawnCollision = CreateDefaultSubobject<UBoxComponent>(TEXT("PawnCollision"));
    RootComponent = PawnCollision;
    PawnCollision->SetCollisionProfileName("Pawn");
    PawnCollision->SetCollisionResponseToChannel(ECC_PhysicsBody, ECR_Ignore);

    PawnMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("PawnMesh"));
    PawnMesh->SetupAttachment(PawnCollision, NAME_None);

    PawnCamera = CreateDefaultSubobject<UCameraComponent>(TEXT("PawnCamera"));

    ShootComponent = CreateDefaultSubobject<UShootComponent>(TEXT("ShootComponent"));
}

// Called when the game starts or when spawned
void APlayerPawn::BeginPlay()
{
    Super::BeginPlay();
    ShootComponent->StartShooting();
    PawnMaterial = PawnMesh->GetMaterial(0);
    MoveLimit = GetGameViewportSize();

    UE_LOG(LogTemp, Log, TEXT("x:  %f, y:  %f"), MoveLimit.X, MoveLimit.Y);
}

// Called every frame
void APlayerPawn::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
}

void APlayerPawn::PossessedBy(AController* NewController)
{
    PlayerController = Cast<APlayerController>(NewController);
}


bool APlayerPawn::CanBeDamaged_Implementation()
{
    return bCanBeDamaged;
}

void APlayerPawn::ExplodePawn_Implementation()
{
```

```
    SetActorEnableCollision(false);

    ShootComponent->StopShooting();

    PawnMesh->SetMaterial(0, RecoverMaterial);

    if (DestroyParticle)
        UGameplayStatics::SpawnEmitterAtLocation(GetWorld(), DestroyParticle, GetActorTransform(),
true);

    for (UActorComponent* Component : GetComponentsByClass(UParticleSystemComponent::StaticClass()))
    {
        Component->Deactivate();
    }
}

void APlayerPawn::RecoverPawn_Implementation()
{
    SetActorEnableCollision(true);

    ShootComponent->StartShooting();

    PawnMesh->SetMaterial(0, PawnMaterial);

    for (UActorComponent* Component : GetComponentsByClass(UParticleSystemComponent::StaticClass()))
    {
        Component->Activate(true);
    }
}

float APlayerPawn::TakeDamage(float Damage, const FDamageEvent& DamageEvent, AController*
InstigatedBy,
                              AActor* DamageCauser)
{
    if (!CanBeDamaged()) return 0.f;

    Super::TakeDamage(Damage, DamageEvent, InstigatedBy, DamageCauser);
    PawnDamaged.Broadcast();
    return Damage;
}

// Called to bind functionality to input
void APlayerPawn::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    InputComponent->BindTouch(IE_Pressed, this, &APlayerPawn::OnTouchPress);
    InputComponent->BindTouch(IE_Released, this, &APlayerPawn::OnTouchReleased);
    InputComponent->BindTouch(IE_Repeat, this, &APlayerPawn::OnTouchMove);
}

// Touch controls
void APlayerPawn::OnTouchMove(ETouchIndex::Type FingerIndex, FVector Location)
{
    if (FingerIndex != ETouchIndex::Touch1) return;
    GetWorld()->GetTimerManager().ClearTimer(rotateAnimTimer);
    FVector2D TouchDeltaMove = FVector2D(TouchLocation.X - Location.X, TouchLocation.Y -
Location.Y);

    TouchDeltaMove = TouchDeltaMove * TouchMoveSensivity;

    FVector NewLocation = GetActorLocation();
    NewLocation.X = FMath::Clamp(NewLocation.X + TouchDeltaMove.Y, -MoveLimit.Y, MoveLimit.Y);
    NewLocation.Y = FMath::Clamp(NewLocation.Y + TouchDeltaMove.X * -1.f, -MoveLimit.X,
MoveLimit.X);

    RotationAnimation(NewLocation);

    SetActorLocation(NewLocation);
    TouchLocation = FVector2D(Location.X, Location.Y);
}

void APlayerPawn::OnTouchPress(ETouchIndex::Type FingerIndex, FVector Location)
{
    TouchLocation = FVector2D(Location.X, Location.Y);
}

void APlayerPawn::OnTouchReleased(ETouchIndex::Type FingerIndex, FVector Location)
{
```

```
    fromInterp = currentRotation;
    GetWorld()->GetTimerManager().SetTimer(rotateAnimTimer, this, &APlayerPawn::RotateBack,
delayTimerInterp, true, 0);
}

void APlayerPawn::RotationAnimation(const FVector& NewLocation)
{
    const auto forwardVector = UKismetMathLibrary::GetDirectionUnitVector(NewLocation,
GetActorLocation());
    const auto forwardY = forwardVector.Y < 0 ? 1 : forwardVector.Y == 0 ? 0 : -1;

    currentRotation = PawnMesh->GetComponentRotation().Roll + 1.01 * forwardY;
    if (UKismetMathLibrary::Abs(currentRotation) <= maxRotationAngle)
    {
        PawnMesh->SetWorldRotation(FRotator(0, 0, currentRotation));
    }
}

void APlayerPawn::RotateBack()
{
    fromInterp = FMath::FInterpTo(fromInterp, targetInterp, 3, stepInterp);
    PawnMesh->SetWorldRotation(FRotator(0, 0, fromInterp));
    if (fromInterp == targetInterp) GetWorld()->GetTimerManager().ClearTimer(rotateAnimTimer);
}

FVector2D APlayerPawn::GetGameViewportSize()
{
    FVector2D Result = FVector2D(1, 1);
    if (GEngine && GEngine->GameViewport)
    {
        GEngine->GameViewport->GetViewportSize(Result);
    }
    return Result;
}
```

EnemySpawnController.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"

#include "Components/ActorComponent.h"
#include "Pawns/EnemyPawn.h"
#include "StructsCollection/StructsCollection.h"

#include "EnemySpawnController.generated.h"


UCLASS( ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class STREAMARCADE_API UEnemySpawnController : public UActorComponent
{
	GENERATED_BODY()
public:

	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Enemies")
	TArray<FEnemySpawnInfo> SpawnStages;

	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Enemies")
	float StageMinDelay;

	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Enemies")
	float StageMaxDelay;

	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Enemies")
	float ChangeStageTimeMultiplier;

protected:
	virtual void BeginPlay() override;
	virtual void Deactivate() override;
	void StartSpawnStage();
	void SpawnEnemy();

	FEnemySpawnInfo SpawnStage;
	int EnemiesSpawned;
	FTimerHandle ChangeStageTimer;
	FTimerHandle EnemySpawnTimer;
	FRandomStream Random;
};
```

EnemySpawnController.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.

#include "EnemySpawnController.h"
#include "Engine/World.h"
#include "TimerManager.h"

// Called when the game starts
void UEnemySpawnController::BeginPlay()
{
    Super::BeginPlay();
    Random.GenerateNewSeed();
    StartSpawnStage();
}

void UEnemySpawnController::Deactivate()
{
    Super::Deactivate();

    GetWorld()->GetTimerManager().ClearTimer(ChangeStageTimer);
    GetWorld()->GetTimerManager().ClearTimer(EnemySpawnTimer);
}

void UEnemySpawnController::StartSpawnStage()
{
    SpawnStage = SpawnStages[Random.RandRange(0, SpawnStages.Num() - 1)];

    EnemiesSpawned = 0;
    SpawnEnemy();

    const float ChangeStageTime = Random.RandRange(StageMinDelay, StageMaxDelay) *
ChangeStageTimeMultiplier;
    GetWorld()->GetTimerManager().SetTimer(ChangeStageTimer, this,
&UEnemySpawnController::StartSpawnStage,
                                           ChangeStageTime, false);
}

void UEnemySpawnController::SpawnEnemy()
{
    const FActorSpawnParameters SpawnParameters;
    GetWorld()->SpawnActor<AEnemyPawn>(SpawnStage.EnemyClass, SpawnStage.SpawnTransform,
SpawnParameters);

    EnemiesSpawned++;
    if (EnemiesSpawned < SpawnStage.NumOfEnemies)
    {
        GetWorld()->GetTimerManager().SetTimer(EnemySpawnTimer, this,
&UEnemySpawnController::SpawnEnemy,
                                               SpawnStage.SpawnDelay, false);
    }
}
```

GameHealthComponent.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"

#include "Components/ActorComponent.h"

#include "GameHealthComponent.generated.h"

DECLARE_DYNAMIC_MULTICAST_DELEGATE(FHealthsEndedEvent);
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FHealthsChangedEvent, int, ChangeValue);


UCLASS( ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class STREAMARCADE_API UGameHealthComponent : public UActorComponent
{
       GENERATED_BODY()

public:
       // Sets default values for this component's properties
       UGameHealthComponent();
```

```
        UFUNCTION(BlueprintCallable, Category = "Game Health")
    void ChangeHealths(int ByValue);

        UFUNCTION(BlueprintPure, Category = "Game Health")
    int GetHealths();

        UPROPERTY(BlueprintAssignable, Category = "Game Health")
        FHealthsEndedEvent HealthsEnded;

        UPROPERTY(BlueprintAssignable, Category = "Game Health")
        FHealthsChangedEvent HealthsChanged;

protected:
        virtual void BeginPlay() override;

        UPROPERTY(EditAnywhere, Category = "Game Health")
        int Healths;
};
```

GameHealthComponent.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.

#include "GameHealthComponent.h"

#include "GameFramework/Pawn.h"
#include "Kismet/GameplayStatics.h"

// Sets default values for this component's properties
UGameHealthComponent::UGameHealthComponent():
        Healths(3)
{
}

// Called when the game starts
void UGameHealthComponent::BeginPlay()
{
        Super::BeginPlay();

        APawn* PlayerPawn = UGameplayStatics::GetPlayerPawn(this, 0);

        if (!PlayerPawn) {
                UE_LOG(LogTemp, Error, TEXT("No playerPawn!!!"));
                return;
        }
}

void UGameHealthComponent::ChangeHealths(int ByValue)
{
        Healths += ByValue;

        HealthsChanged.Broadcast(ByValue);

        if (Healths <= 0) {
                HealthsEnded.Broadcast();
        }

        UE_LOG(LogTemp, Log, TEXT("Health changed: %i"), Healths);
}

int UGameHealthComponent::GetHealths() const
{
        return Healths;
}
```

HealthComponent.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"

#include "GameFramework/Controller.h"
#include "Components/ActorComponent.h"

#include "HealthComponent.generated.h"
```

```
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FHealthEndedEvent);

UCLASS(ClassGroup = (Custom), meta = (BlueprintSpawnableComponent))
class STREAMARCADE_API UHealthComponent : public UActorComponent
{
    GENERATED_BODY()

public:
    UHealthComponent();

    UFUNCTION(BlueprintCallable, Category = "Health")
        void ChangeHealth(float Value);

    UFUNCTION(BlueprintPure, Category = "Health")
        float GetHealth();

    UPROPERTY(BlueprintAssignable, Category = "Health")
    FHealthEndedEvent OnHealthEnded;

protected:
    virtual void BeginPlay() override;

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Health")
    float Health;

    UFUNCTION()
        void OnOwnerDamaged(AActor* DamagedActor, float Damage, const UDamageType* DamageType,
AController* Instigator, AActor* DamageCauser);
};
```

HealthComponent.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.

#include "HealthComponent.h"

#include "GameFramework/Actor.h"


// Sets default values for this component's properties
UHealthComponent::UHealthComponent():
       Health(100)
{
}

// Called when the game starts
void UHealthComponent::BeginPlay()
{
       Super::BeginPlay();

       GetOwner()->OnTakeAnyDamage.AddDynamic(this, &UHealthComponent::OnOwnerDamaged);
}

void UHealthComponent::OnOwnerDamaged(AActor * DamagedActor, float Damage, const UDamageType*
DamageType, AController * Instigator, AActor * DamageCauser)
{
       ChangeHealth(-Damage);
}

void UHealthComponent::ChangeHealth(float Value)
{
       Health += Value;

       if (Health <= 0.f) {
              GetOwner()->OnTakeAnyDamage.RemoveDynamic(this, &UHealthComponent::OnOwnerDamaged);
              OnHealthEnded.Broadcast();
       }
}

float UHealthComponent::GetHealth() const
{
       return Health;
}
```

ShootComponent.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once
```

```
#include "CoreMinimal.h"

#include "StructsCollection/StructsCollection.h"
#include "Actors/Projectiles/ShootProjectile.h"

#include "Components/ActorComponent.h"

#include "ShootComponent.generated.h"

UCLASS( ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class STREAMARCADE_API UShootComponent : public UActorComponent
{
	GENERATED_BODY()

public:
	UShootComponent();

	UFUNCTION(BlueprintCallable, Category = "Shooting")
    void StartShooting();

	UFUNCTION(BlueprintCallable, Category = "Shooting")
    void StopShooting();

	UFUNCTION(BlueprintCallable, Category = "Shooting")
    void RestartShooting();

	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Shooting")
	float ShootPeriod;

	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Shooting")
	TArray<FShootInfo> ShootInfos;

protected:
	virtual void BeginPlay() override;
	void Shoot();
	FTimerHandle ShootingTimer;

};
```

ShootComponent.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.

#include "ShootComponent.h"

#include "Engine/World.h"
#include "TimerManager.h"

// Sets default values for this component's properties
UShootComponent::UShootComponent() : ShootPeriod(1.f)
{
}

void UShootComponent::BeginPlay()
{
    Super::BeginPlay();

    StartShooting();
}

void UShootComponent::Shoot()
{
    for (FShootInfo ShootInfo : ShootInfos)
    {
        FActorSpawnParameters SpawnParameters;
        SpawnParameters.Owner = GetOwner();
        SpawnParameters.SpawnCollisionHandlingOverride =
ESpawnActorCollisionHandlingMethod::AlwaysSpawn;

        FVector SpawnLocation =
            GetOwner()->GetActorLocation()
            +
            GetOwner()->GetActorRotation().RotateVector(ShootInfo.Offset);

        FRotator SpawnRotation = GetOwner()->GetActorRotation();
        SpawnRotation.Add(0.f, ShootInfo.Angle, 0.f);

        AShootProjectile* Projectile = GetWorld()-
>SpawnActor<AShootProjectile>(ShootInfo.ProjectileClass, SpawnLocation, SpawnRotation,
SpawnParameters);
```

```
        if (Projectile) Projectile->Damage = ShootInfo.Damage;
    }
}

void UShootComponent::StartShooting()
{
    GetWorld()->GetTimerManager().SetTimer(ShootingTimer, this, &UShootComponent::Shoot,
ShootPeriod, true, ShootPeriod);
}

void UShootComponent::StopShooting()
{
    GetWorld()->GetTimerManager().ClearTimer(ShootingTimer);
}

void UShootComponent::RestartShooting()
{
    StopShooting();
    StartShooting();
}
```

PlaygroundBorder.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"

#include "GameFramework/Actor.h"

#include "PlaygroundBorder.generated.h"

UCLASS()
class STREAMARCADE_API APlaygroundBorder : public AActor
{
    GENERATED_BODY()

public:
    APlaygroundBorder();

    virtual void NotifyActorEndOverlap(AActor* OtherActor) override;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
    class UBoxComponent* Trigger;
};
```

PlaygroundBorder.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.

#include "PlaygroundBorder.h"
#include "Pawns/PlayerPawn.h"

#include "Components/BoxComponent.h"

// Sets default values
APlaygroundBorder::APlaygroundBorder()
{
       Trigger = CreateDefaultSubobject<UBoxComponent>(TEXT("Trigger"));
       SetRootComponent(Trigger);

       Trigger->SetCollisionProfileName("OverlapAll");
}

void APlaygroundBorder::NotifyActorEndOverlap(AActor * OtherActor)
{
       Super::NotifyActorEndOverlap(OtherActor);

       if (!OtherActor) return;
       if (Cast<APlayerPawn>(OtherActor)) return;

       OtherActor->Destroy();
}
```

ShootProjectile.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once
```

```
#include "CoreMinimal.h"

#include "GameFramework/Actor.h"
#include "Components/SphereComponent.h"

#include "ShootProjectile.generated.h"

UCLASS()
class STREAMARCADE_API AShootProjectile : public AActor
{
    GENERATED_BODY()

public:
    AShootProjectile();
    virtual void Tick(float DeltaTime) override;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Shooting")
    USphereComponent* Collision;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Shooting")
    UStaticMeshComponent* Mesh;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Shooting")
    UParticleSystemComponent* Particle;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Shooting")
    float ProjectileSpeed;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Shooting")
    float Damage;

protected:
    virtual void BeginPlay() override;

    UFUNCTION()
    void OnProjectileOverlap(UPrimitiveComponent* OpelappedComp, AActor* OtherActor,
UPrimitiveComponent* OtherComp,
                             int32 BodyIndex, bool Sweep, const FHitResult& Hit);
};
```

ShootProjectile.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.

#include "ShootProjectile.h"

#include "Kismet/GameplayStatics.h"
#include "GameFramework/Pawn.h"
#include "GameFramework/DamageType.h"
#include "Components/BoxComponent.h"
#include "Components/StaticMeshComponent.h"
#include "Particles/ParticleSystemComponent.h"
//#include "Pawns/EnemyPawn.h"

// Sets default values
AShootProjectile::AShootProjectile()
    :
    ProjectileSpeed(1000.f)
{
    // Set this actor to call Tick() every frame.  You can turn this off to improve performance if
you don't need it.
    PrimaryActorTick.bCanEverTick = true;

    Collision = CreateDefaultSubobject<USphereComponent>(TEXT("ProjectileCollision"));
    RootComponent = Collision;
    Collision->SetCollisionEnabled(ECollisionEnabled::NoCollision);

    Mesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Mesh"));
    Mesh->SetupAttachment(Collision);
    Mesh->SetCollisionProfileName("NoCollision");

    Particle = CreateDefaultSubobject<UParticleSystemComponent>(TEXT("Particle"));
    Particle->SetupAttachment(Collision);
}

// Called when the game starts or when spawned
void AShootProjectile::BeginPlay()
{
    Super::BeginPlay();
```

```
    if (GetOwner())
    {
        UBoxComponent* OwnerCollision = GetOwner()->FindComponentByClass<UBoxComponent>();
        Collision->IgnoreComponentWhenMoving(OwnerCollision, true);
        OwnerCollision->IgnoreComponentWhenMoving(Collision, true);

        Collision->SetCollisionEnabled(ECollisionEnabled::QueryOnly);
    }

    Collision->OnComponentBeginOverlap.AddDynamic(this, &AShootProjectile::OnProjectileOverlap);
}

void AShootProjectile::OnProjectileOverlap(UPrimitiveComponent* OpelappedComp, AActor* OtherActor,
                                           UPrimitiveComponent* OtherComp, int32 BodyIndex, bool
Sweep,
                                           const FHitResult& Hit)
{
    APawn* OtherPawn = Cast<APawn>(OtherActor);
    if (!OtherActor || !OtherPawn) return; // If no overlapped actor or it is not a pawn

    if (!GetOwner()) return;
    APawn* PawnOwner = Cast<APawn>(GetOwner());
    if (!PawnOwner) return;
    AController* DamageInstigator = PawnOwner->GetController();

    if (!PawnOwner->GetController() && !OtherPawn->GetController()) return;

    UGameplayStatics::ApplyDamage(OtherActor, Damage, DamageInstigator, this,
UDamageType::StaticClass());

    Destroy();
}

void AShootProjectile::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    AddActorLocalOffset(FVector(ProjectileSpeed * DeltaTime, 0.f, 0.f));
}
```

PawnShield.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"

#include "GameFramework/Actor.h"

#include "PawnShield.generated.h"

class APlayerPawn;

UCLASS()
class STREAMARCADE_API APawnShield : public AActor
{
    GENERATED_BODY()
public:
    APawnShield();

    UFUNCTION(BlueprintCallable, Category = "Shield")
    void ActivateShield(APlayerPawn* PlayerPawn);

    UFUNCTION(BlueprintCallable, Category = "Shield")
    void DeactivateShield();

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Shield")
    float ShieldTime;

protected:
    FTimerHandle ShieldTimer;
    class APlayerPawn* ShildForPawn;
};
```

PawnShield.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.

#include "PawnShield.h"
#include "Pawns/PlayerPawn.h"
```

```
#include "Engine/World.h"
#include "TimerManager.h"

// Sets default values
APawnShield::APawnShield():
    ShieldTime(5.f)
{
}

void APawnShield::ActivateShield(APlayerPawn* PlayerPawn)
{
    if (!PlayerPawn)
    {
        Destroy();
        return;
    }
    ShildForPawn = PlayerPawn;
    PlayerPawn->bCanBeDamaged = false;

    FAttachmentTransformRules AttachRules = FAttachmentTransformRules(
        EAttachmentRule::SnapToTarget,
        EAttachmentRule::SnapToTarget,
        EAttachmentRule::KeepWorld,
        false
    );
    AttachToActor(PlayerPawn, AttachRules);
    GetWorld()->GetTimerManager().SetTimer(ShieldTimer, this, &APawnShield::DeactivateShield,
ShieldTime, false);
}

void APawnShield::DeactivateShield()
{
    if (!ShildForPawn) return;
    ShildForPawn->bCanBeDamaged = true;
    Destroy();
}
```

Bonus.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"

#include "GameFramework/Actor.h"

#include "Bonus.generated.h"

UCLASS(Blueprintable)
class STREAMARCADE_API ABonus : public AActor
{
    GENERATED_BODY()

public:
    // Sets default values for this actor's properties
    ABonus();

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Shooting")
    class USphereComponent* Collision;

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Visual")
    UParticleSystem* CollectParticle;

protected:
    virtual void NotifyActorBeginOverlap(AActor* OtherActor) override;
    virtual void Tick(float DeltaTime) override;

    UFUNCTION(BlueprintNativeEvent)
    void BonusCollected();
    virtual void BonusCollected_Implementation();
};
```

Bonus.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.

#include "Bonus.h"
```

```
#include "Pawns/PlayerPawn.h"

#include "Components/SphereComponent.h"
#include "Components/StaticMeshComponent.h"
#include "Kismet/GameplayStatics.h"

// Sets default values
ABonus::ABonus()
{
    PrimaryActorTick.bCanEverTick = true;

    Collision = CreateDefaultSubobject<USphereComponent>(TEXT("BonusCollision"));
    RootComponent = Collision;

    Collision->SetCollisionObjectType(ECC_WorldDynamic);
    Collision->SetSphereRadius(50);
}

void ABonus::NotifyActorBeginOverlap(AActor* OtherActor)
{
    Super::NotifyActorEndOverlap(OtherActor);

    UE_LOG(LogTemp, Log, TEXT("Bonus overlap"));
    if (!OtherActor) return;
    if (!Cast<APlayerPawn>(OtherActor)) return;

    UE_LOG(LogTemp, Log, TEXT("Bonus CHAR overlap"));
    BonusCollected();
}

void ABonus::BonusCollected_Implementation()
{
    if (CollectParticle)
        UGameplayStatics::SpawnEmitterAtLocation(GetWorld(), CollectParticle, GetActorTransform(),
true);
    Destroy();
}

// Called every frame
void ABonus::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    float WorldMoveOffset = -200.f * DeltaTime;
    AddActorWorldOffset(FVector(WorldMoveOffset, 0.f, 0.f));
}
```

BonusPoint.h

```
#pragma once

#include "CoreMinimal.h"

#include "Actors/Bonuses/Bonus.h"

#include "BonusPoints.generated.h"


UCLASS()
class STREAMARCADE_API ABonusPoints : public ABonus
{
    GENERATED_BODY()

public:
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Bonus")
    int Points;

protected:
    virtual void BonusCollected_Implementation() override;
};
```

BonusPoint.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.

#include "BonusPoints.h"

#include "Kismet/GameplayStatics.h"
#include "StreamArcadeGameModeBase.h"
```

```
void ABonusPoints::BonusCollected_Implementation()
{
	AStreamArcadeGameModeBase* Gamemode =
Cast<AStreamArcadeGameModeBase>(UGameplayStatics::GetGameMode(this));
	if (Gamemode) Gamemode->AddPoints(Points);

	Super::BonusCollected_Implementation();
}
```

BonusShield.h

```
#pragma once

#include "CoreMinimal.h"

#include "Actors/Bonuses/Bonus.h"

#include "BonusShield.generated.h"

class APawnShield;

UCLASS()
class STREAMARCADE_API ABonusShield : public ABonus
{
    GENERATED_BODY()

public:
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Shield")
    TSubclassOf<APawnShield> ShieldClass;

protected:
    virtual void BonusCollected_Implementation() override;
};
```

BonusShield.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.

#include "BonusShield.h"
#include "Kismet/GameplayStatics.h"
#include "Pawns/PlayerPawn.h"
#include "Engine/World.h"
#include "Actors/Other/PawnShield.h"

void ABonusShield::BonusCollected_Implementation()
{
	APawn* Pawn = UGameplayStatics::GetPlayerPawn(this, 0);
	if (!Pawn) return;
	APlayerPawn* PlayerPawn = Cast<APlayerPawn>(Pawn);

	if (!PlayerPawn || !PlayerPawn->bCanBeDamaged) return;

	FActorSpawnParameters SpawnParams;
	SpawnParams.Owner = PlayerPawn;
	SpawnParams.SpawnCollisionHandlingOverride =
ESpawnActorCollisionHandlingMethod::AlwaysSpawn;

	APawnShield* Shield = GetWorld()->SpawnActor<APawnShield>(ShieldClass, SpawnParams);

	if(Shield) Shield->ActivateShield(PlayerPawn);

	Super::BonusCollected_Implementation();
}
```

BonusLevelUp.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"

#include "Actors/Bonuses/Bonus.h"

#include "BonusShootLevelUp.generated.h"


UCLASS()
class STREAMARCADE_API ABonusShootLevelUp : public ABonus
{
	GENERATED_BODY()
```

```
protected:

	virtual void BonusCollected_Implementation() override;
};
```

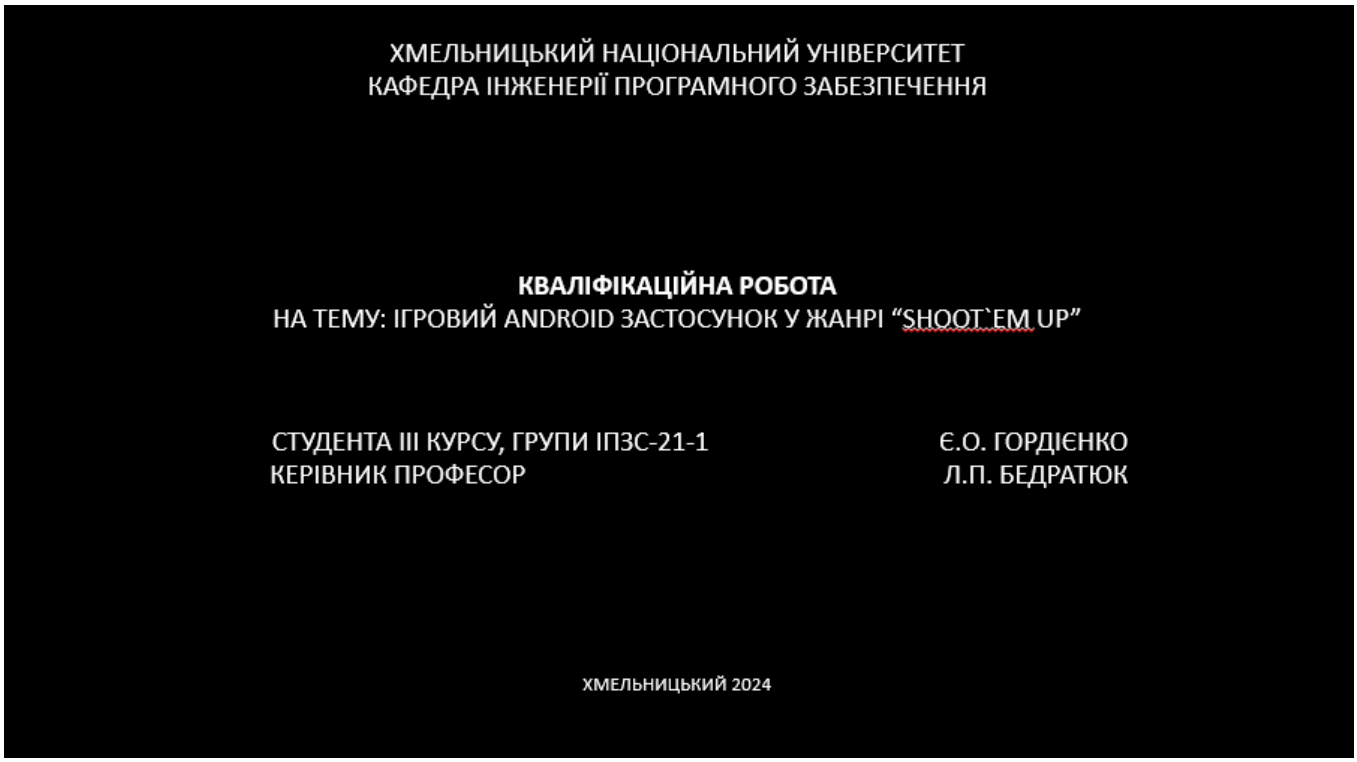
BonusLevelUp.cpp

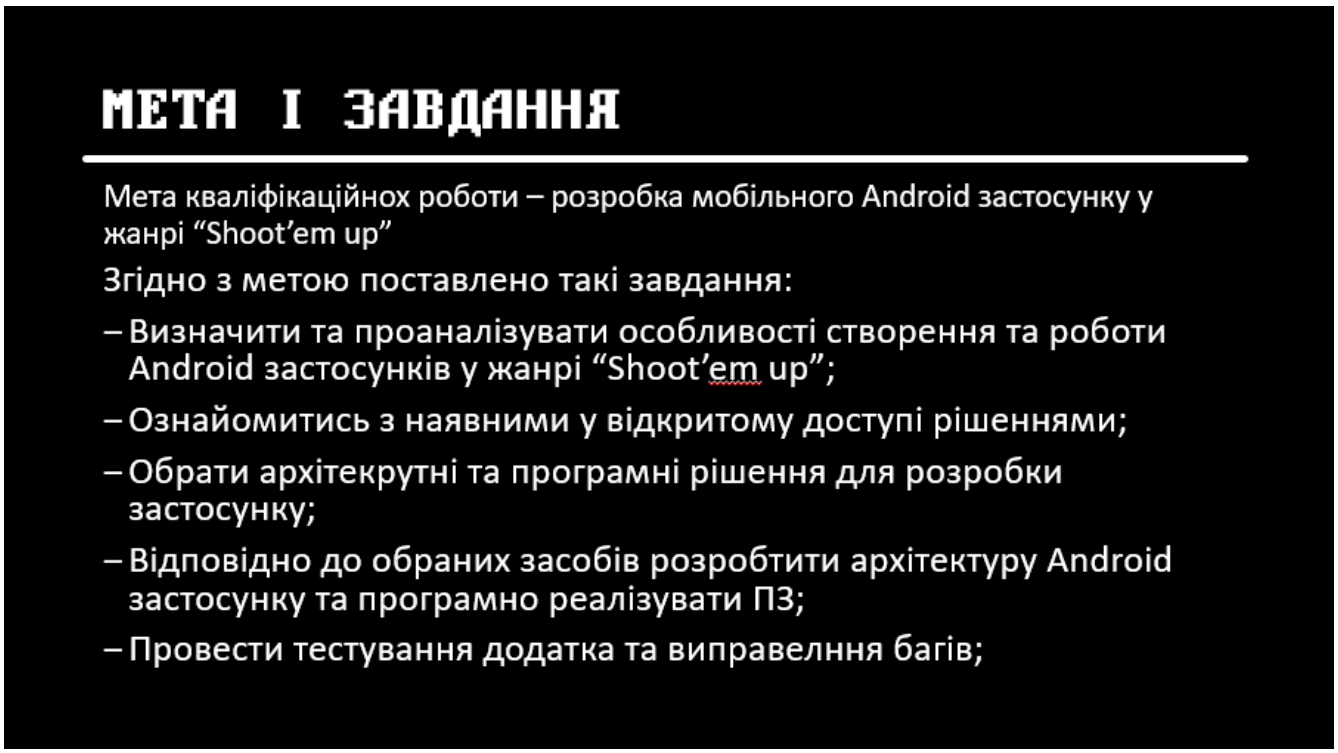
```
// Fill out your copyright notice in the Description page of Project Settings.

#include "BonusShootLevelUp.h"

#include "Kismet/GameplayStatics.h"

#include "StreamArcadeGameModeBase.h"

void ABonusShootLevelUp::BonusCollected_Implementation()
{
	AStreamArcadeGameModeBase* Gamemode =
Cast<AStreamArcadeGameModeBase>(UGameplayStatics::GetGameMode(this));
	if (!Gamemode) return;
	Gamemode->ChangeShootLevel(true);
	Super::BonusCollected_Implementation();
}
```

ДОДАТОК В

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ





ОГЛЯД РІШЕНЬ

- Space invaders
- Radiant

Space invaders

Radiant

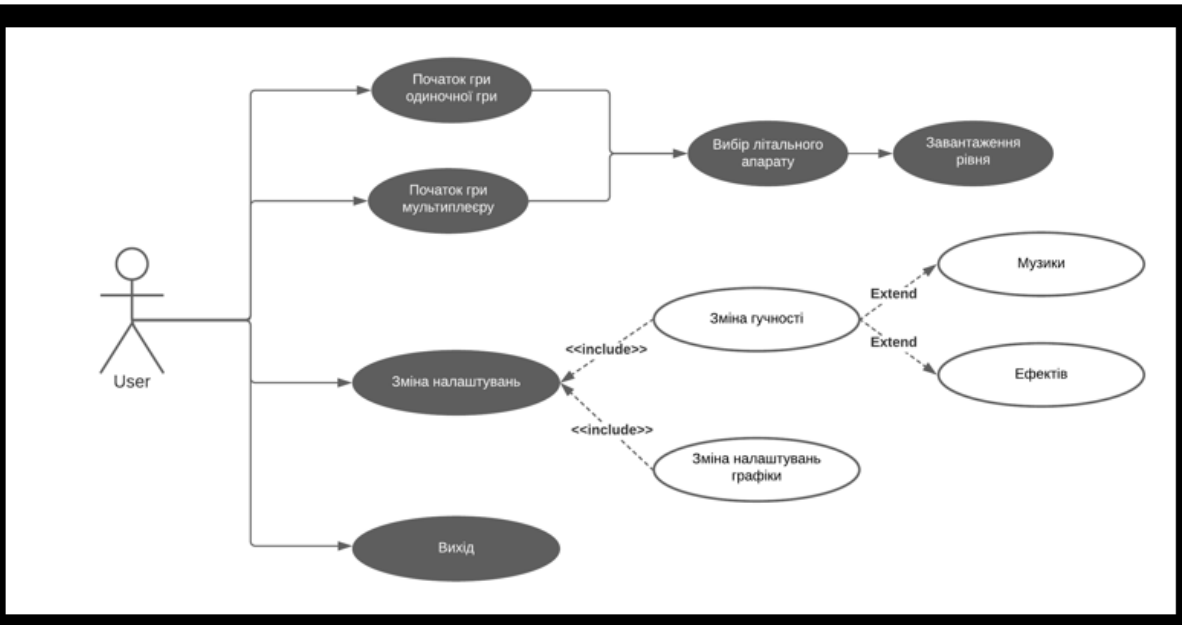
Порівняльна таблиця

	Space invaders	Radiant
Змінний рівень складності	НІ	ТАК
Наявність різних моделей літаків з різними характеристиками	НІ	ТАК
Зручність та зрозумілість інтерфейсу	Через вік гри інтерфейс її може здатися не зрозумілим та застарілим	Сучасніша гра, має портовану версію на Android, в якій адаптовано інтерфейс для даної платформи
Мультиплеєр	НІ	НІ
Графіка	2D, проста, піксельна	2D, яскрава, стильна

Перелік вимог

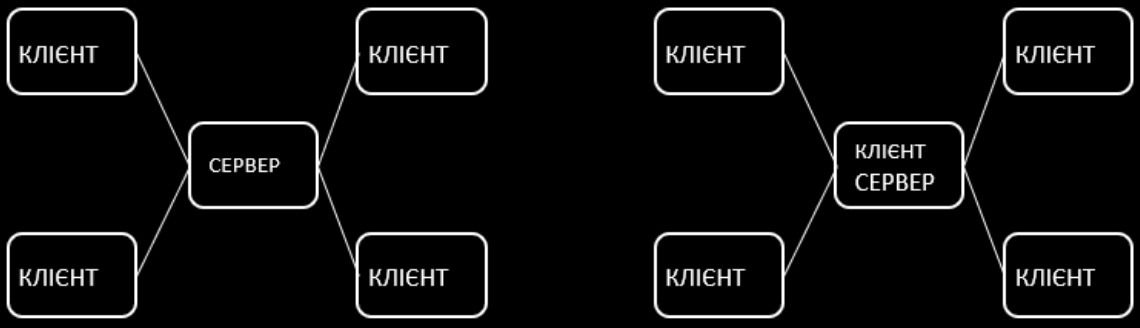
- Android додаток
- жанр Shoot`em up
- нескінченна кількість ворогів
- різні види літальних апаратів
- мультиплеєр
- адаптивний UI

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



МУЛЬТИПЛЕЕР

ТИПИ СЕРВЕРІВ У UNREAL ENGINE 5

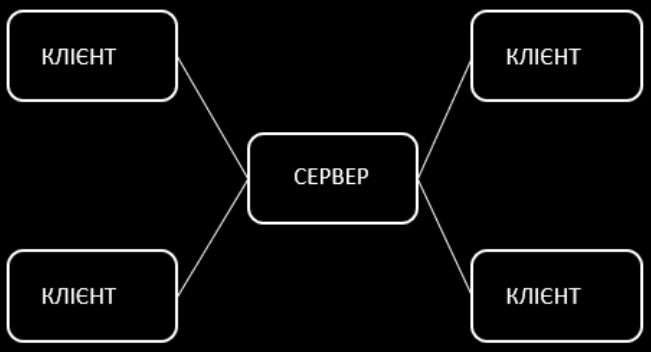


DEDICATED

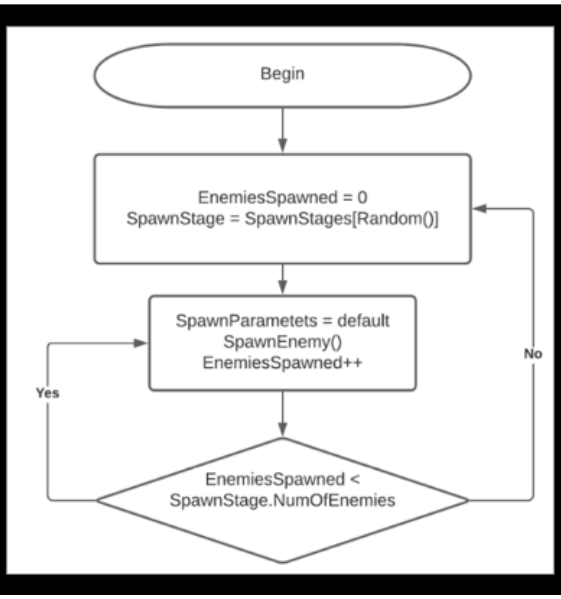
LISTEN

МОДЕЛЬ РОБОТИ МУЛЬТИПЛЕЄРУ

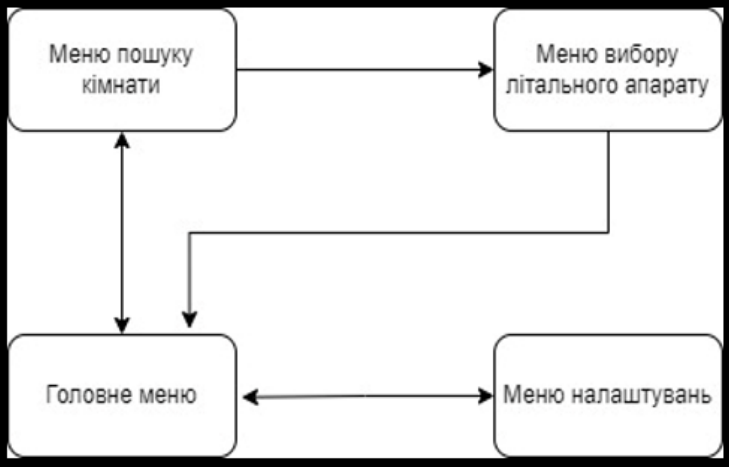
- Принцип клієнт-сервер
- Клієнти підключаються до сессії
- Сервер є головний



РЕАЛІЗАЦІЯ НЕСКІНЧЕННОЇ ГРИ



СТРУКТУРА ІНТЕРФЕЙСУ



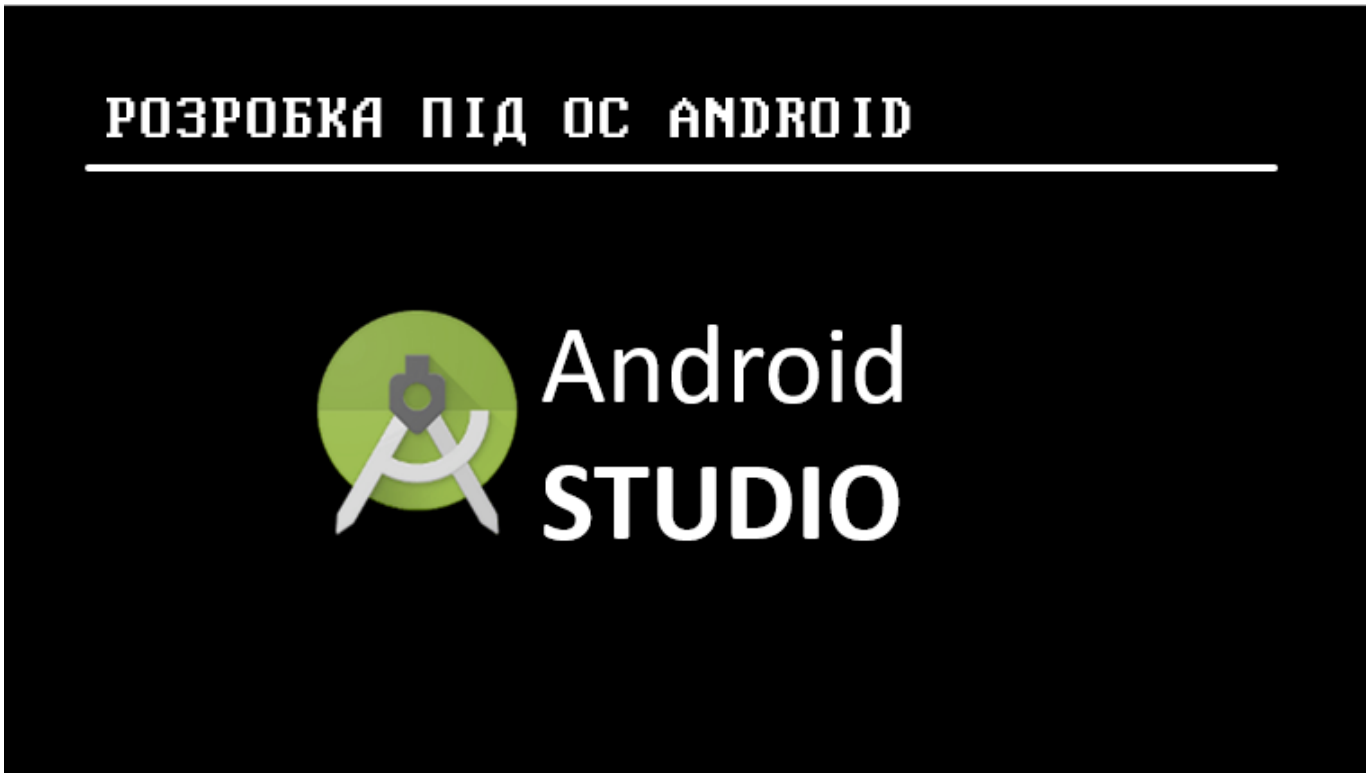
Використані технології

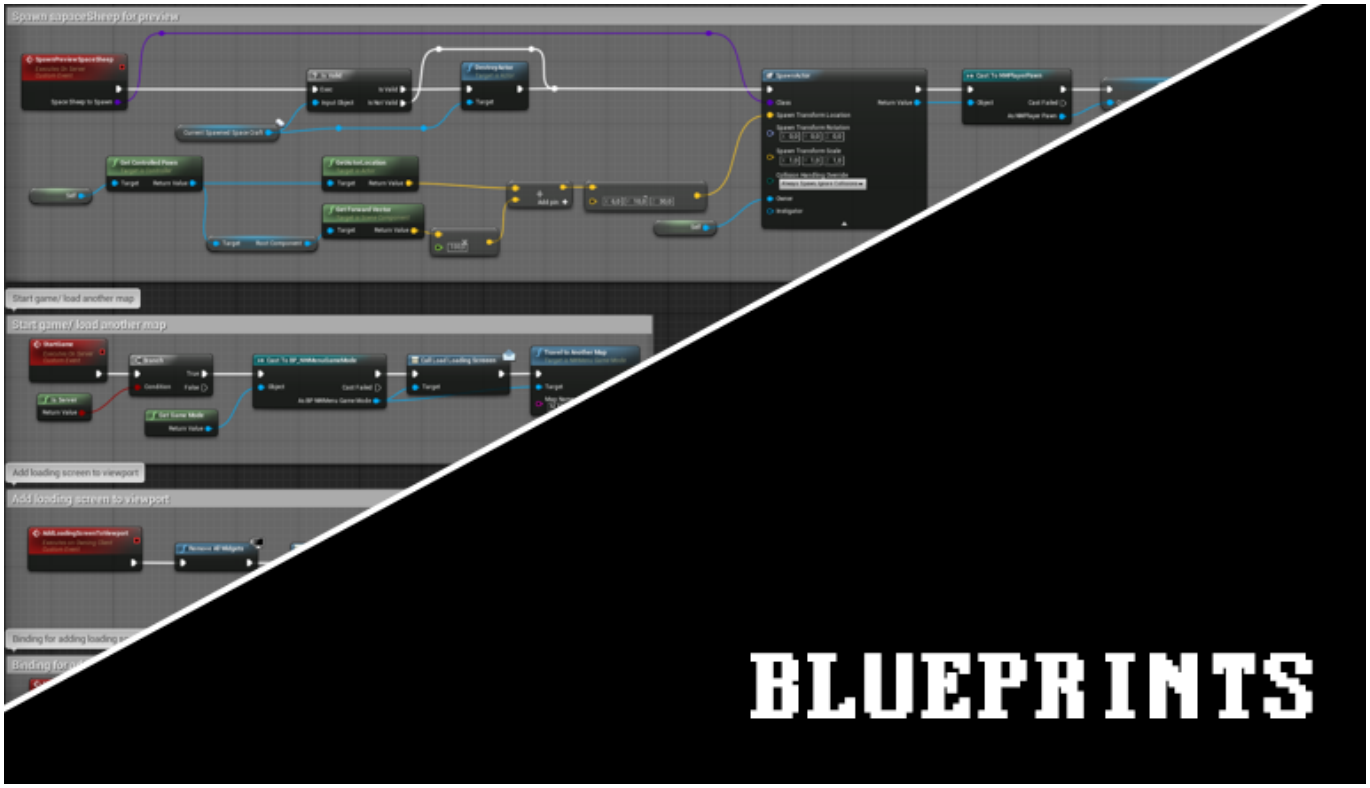
Unreal engine 5 і його плагіни:

- AutoSettings
- StarSphere

AutoSettings

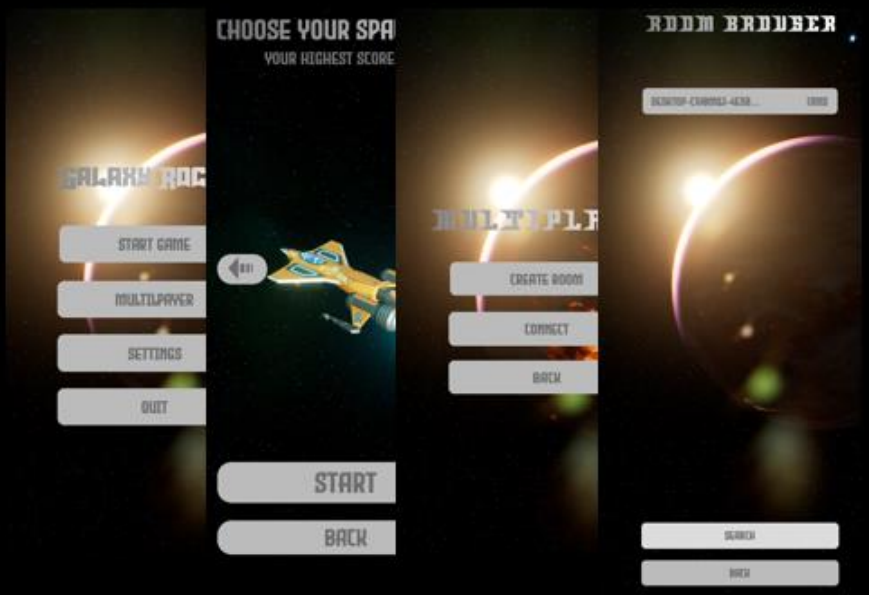
StarSphere


РОЗРОБКА ПІД ОС ANDROID
Android
STUDIO


BLUEPRINTS

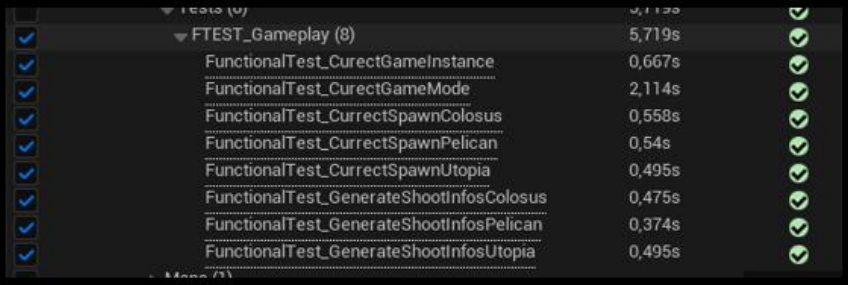
ІНТЕРФЕЙС

- Головне меню
- Меню вибору літального апарату
- Меню мультиплеєру
- Меню вибору кімнати для приєднання



ТЕСТУВАННЯ

Для перевірки якості програмного продукту використано модульне тестування (unit test)



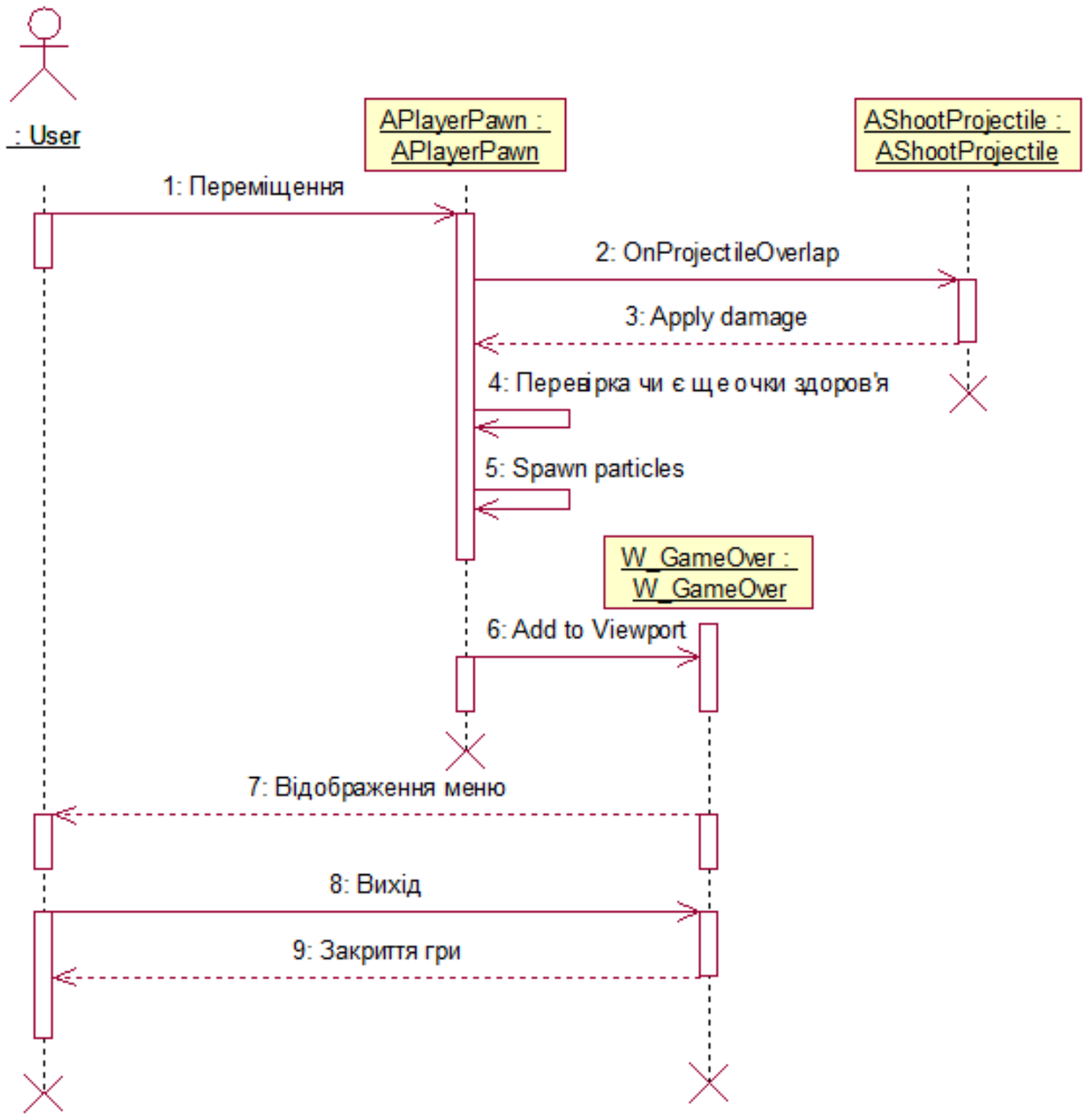
ВИСНОВКИ

При виконанні кваліфікаційнох роботи на тему “Ігровий Android застосунок у жанрі “Shoot em’up”” було розроблено архітектуру додатку після чого спроектовано та реалізовано функціонал Android додатку.

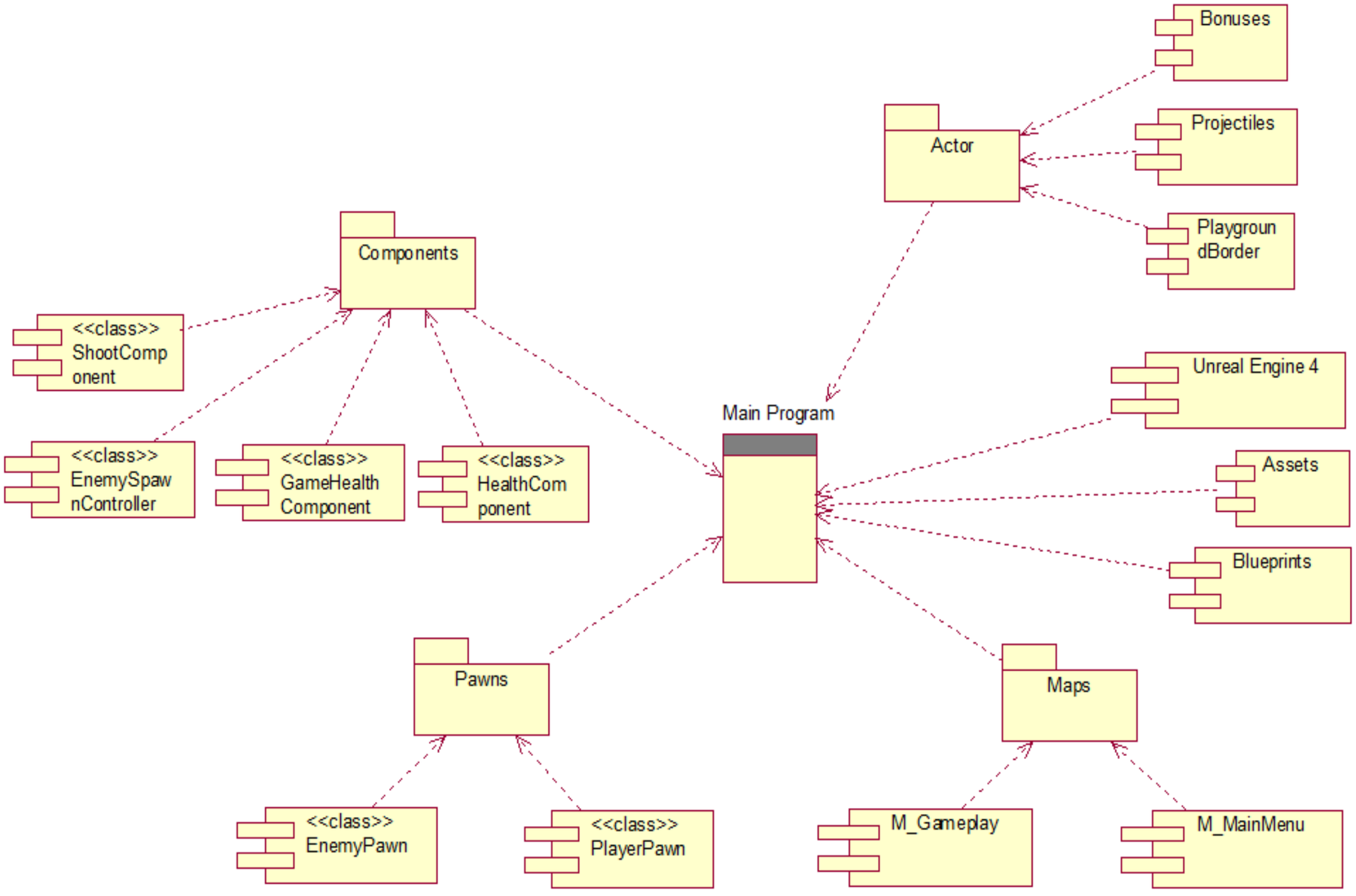
- Ознайомлення з наявними у відкритому доступі рішеннями.
- Визначені та проаналізовані особливості створення та роботи Android застосунків у жанрі “Shoot em up”
- Обрані архітекрутні та програмні рішення для розробки застосунку
- Проведено тестування додатка та виправелені наявні баги

Дякую за увагу

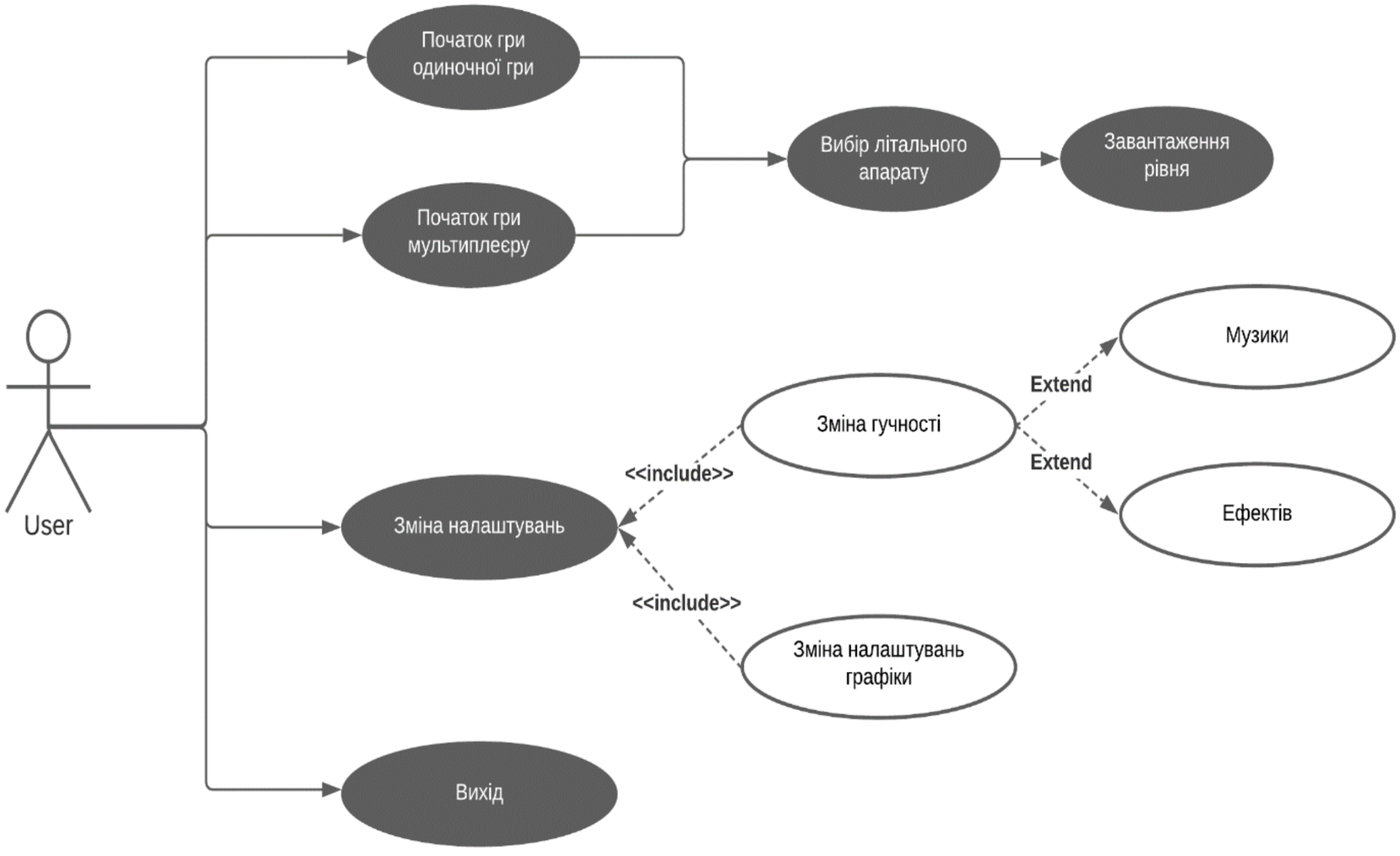
ГРАФІЧНА ЧАСТИНА



					КвРІПЗ. 2101099.01.02.Е8		
Зм	Арк.	№ докум.	Підпис	Дата			
Виконав		Гордієнко Є.О.			Діаграма послідовності	Літ.	Арк. 1 / Акрушів 3
Керівник.		Бедратюк Л.П.					
.						ХНУ, ІПЗс-21-1	
Н. контр.		Праворська Н. В.					
Зав. каф.		Бедратюк Л.П.					



					КвРІПЗ. 2101099.01.02.Е8	
Зм	Арк.	№ докум.	Підпис	Дата		
Виконав		Гордієнко Є.О.			Діаграма компонентів	Літ. / Арк. 2 / Акрушів 3
Керівник.		Бедратюк Л.П.				
.						ХНУ, ІПЗс-21-1
Н. контр.		Праворська Н. В.				
Зав. каф.		Бедратюк Л.П.				



					КвРІПЗ. 2101099.01.02.Е8	
Зм	Арк.	№ докум.	Підпис	Дат		
Виконав		Гордієнко Є.О.			Діаграма варіантів використання	Літ. / Арк. 3 / Акрушів 3
Керівник		Бедратюк Л.П.				
.						ХНУ, ІПЗс-21-1
Н. контр.		Праворська Н. В.				
Зав. каф.		Бедратюк Л.П.				

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Горідєнка Є.О.
Прізвище, ініціали

факультет ІТ, 3 курс, група ІПЗс-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті», згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

09.06.2024
дата

підпис

Mon Jun 10 07:14:49 EEST 2024, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

ID: 129204 Назва: БКР_Ігровий_Android_застосунок_у_жанрі_«Shoot'em_up» Додано в БД: 2024-06-10 Автора: Гордієнко Євгеній Керівники: Бедратюк Леонід, доктор фіз.-мат.наук, професор Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	61393	538	1771 (3%)	24 (4%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

ID перевірки:
1016334306

Дата перевірки:
08.06.2024 08:59:59 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
08.06.2024 09:40:12 EEST

ID користувача:
100012953

Назва документа: **БКР_Ігровий_Android_застосунок_у_жанрі_«Shoot'em_up»_Гордієнко_Є**

Кількість сторінок: **73** Кількість слів: **9805** Кількість символів: **78055** Розмір файлу: **4.99 MB** ID файлу: **1016134765**

6.04%
Схожість

Найбільша схожість: **0.76%** з Інтернет-джерелом (https://essuir.sumdu.edu.ua/bitstream-download/123456789/85764/1/N

4.91% Джерела з Інтернету	314	Сторінка 75
3.15% Джерела з Бібліотеки	102	Сторінка 77

1.55% Цитат

Цитати	11	Сторінка 78

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Замінені символи	1

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Гордієнко Євгеній Олегович

Тема Ігровий Android-застосунок у жанрі «Shoot 'em up»

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 108

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі Гордієнка Євгенія Олеговича розглянуто створення мобільного Android-застосунку в жанрі «Shoot 'em up» з використанням ігрового рушія Unreal Engine та мови програмування C++. Проект включає розробку як однокористувацького, так і багатокористувацького режиму гри. В роботі проведено аналіз предметної області, визначено вимоги до програмного забезпечення, спроектовано структуру та інтерфейс застосунку, реалізовано і протестовано ігрові механіки.

2. Висновок про відповідність роботи поставленому завданню Робота повністю відповідає поставленому завданню. Виконані всі етапи, включаючи дослідження предметної області, проектування, реалізацію та тестування ігрового застосунку.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи
- Дослідження предметної області: виконано детальний аналіз жанру «Shoot 'em up», визначено основні особливості та вимоги до гри.
- Проектування програмного забезпечення: спроектовано архітектуру додатку, розроблено UML-діаграми, обгрунтовано вибір технологій (Unreal Engine, C++).
- Програмна реалізація: створено основні ігрові механіки, графічний інтерфейс, реалізовано мультиплеєр.
- Тестування системи: проведено тестування функціоналу, описано тест-кейси.

Ступінь використання останніх досягнень науки і техніки та передових методів роботи:
- У роботі використано сучасні методи розробки програмного забезпечення, такі як об'єктно-орієнтоване програмування та використання ігрового рушія Unreal Engine.
- Застосовано передові практики проектування інтерфейсу користувача та розробки мультиплеєрних систем.
- Для забезпечення якості програмного продукту використано сучасні інструменти для тестування, включаючи автоматизовані модульні тести.
- Використання плагінів AutoSettings та StarSphere демонструє високу ступінь інтеграції з передовими технологіями.

4. Позитивні сторони роботи
- Використання сучасних інструментів та технологій для розробки (Unreal Engine, C++).
- Впровадження мультиплеєрної функціональності.
- Детальне проектування інтерфейсу користувача.
- Комплексне тестування системи.

5. Негативні сторони роботи - Деякі аспекти роботи, такі як управління ресурсами та оптимізація продуктивності, могли бути розглянуті більш детально.
- Обмежена кількість тестових сценаріїв для мультиплеєрного режиму.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення та пояснювальна записка виконані на високому рівні, містять усі необхідні схеми, діаграми та ілюстрації.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота Гордієнка Євгенія Олеговича є завершеним проектом, який демонструє високий рівень знань та навичок у галузі інженерії програмного забезпечення. Робота виконана відповідно до поставлених завдань та вимог, містить теоретичні та практичні результати, які можуть бути використані в подальшій розробці ігор.

8. Інші зауваження Варто було б додати більше тестових сценаріїв для різних ситуацій у мультиплеєрному режимі, що дозволило б більш повно оцінити стабільність та функціональність гри.

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота заслуговує на оцінку добре

РЕЦЕНЗЕНТ Говорущенко Тетяна Олександрівна, доктор технічних наук, зав кафедри комп'ютерної інженерії та інформаційних систем (КПС) ХНУ

"12" 06 2024 р.
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів:

Назва кваліфікаційної роботи: «Ігровий Android-застосунок у жанрі «Shoot`em up»
Автор: Гордієнко Євгеній Олегович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системою перевірки на плагіат Unicheck виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів, у рамках основних написів, у назвах публікацій переліку джерел посилання;

2) в якості запозичень системою Unicheck було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) запозичення, виявлені в тексті роботи, є фрагментарними.

Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 1.0%. Обсяг запозичень, визначений системою Unicheck виявлення збігів ідентичності/схожості, складає 6.04% і адресується до 314 джерел з Інтернету і 102 джерела з бібліотеки, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 08.06.2024 р.

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Леонід БЕДРАТЮК

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Гордієнко Євгеній Олегович,

студент III курсу спеціальності 121 – Інженерія програмного забезпечення, група ІПЗс-21-1

здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомився (-лась) з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

06 03 2024 р.

Підпис

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗс-21-1
Годлєвська Є. О.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: Ігровий Android-застосунок у жанрі „Shoot 'em up"

(керівник роботи – Бедратюк Леонід Петрович)
Прізвище, ім'я, по батькові

02.01.2024
Дата

Підпис студента