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

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-07

Дипломний проект

здобувача освіти денної форми навчання
РП.07.02.000.ДП

ВОСКОБОЙНИКА СТАНІСЛАВА ОЛЕКСАНДРОВИЧА

м. Одеса
2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

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

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

Група: **4РП-07**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity

Проектний матеріал складається з пояснювальної записки на 76 сторінках та графічного (презентаційного) матеріалу на 12 аркушах (слайдах)

Дипломник _____ (Воскобойник С.О.)

Керівник _____ (Джабраілов Д.В.)

Консультанти:

з економічного розділу _____ (Іванченков В.С.)

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

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «17» 06 2024 р. Протокол ЕК № 1

Оцінка ЕК 4 (добре) / 75 б.

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

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

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітньо-професійна програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"16" 01 2024 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Воскобойнику Станіславу Олександровичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity*

затверджена наказом по коледжу від "02" 11 2023 р. № 244-А2-ОД

2. Термін здачі закінченого проекту 10.06.2024

3. Вихідні данні до проекту *Використання програмного рушія Unity; Використання мови програмування C# та бібліотек Unity для розробки гри; Використання принципів модульності у проектуванні та розробці ігрових проектів; Мають бути реалізовані основні ігрові елементи скролл-шутеру; Гра повинна представляти гравцю можливість переходити з рівня на рівень, після зарахування необхідної кількості ігрового рахунку; Гра повинна мати мінімальний інтерфейс.*

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

Аналіз ігрового жанру 2D скролл-шутер; Аналітичний огляд програмних рушіїв з умовно безкоштовним доступом; Вибір інструментів розробки; Формування концепту ігрового процесу 2D-гри в жанрі скролл-шутер; Проектування основних елементів ігрового процесу та принципи їх роботи; Реалізація основних елементів ігрового процесу; Тестування працездатності ігрових елементів.

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

Особливості ігрового жанру 2D скролл-шутер; Особливості програмного рушія Unity та причини його вибору для розробки проекту; Особливості ігрових механік розроблюємого проекту; Огляд основних елементів ігрового процесу; Принцип роботи основних елементів ігрового процесу; Етапи реалізації основних елементів ігрового процесу; Хід тестування гри; Скріншоти готової гри.

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Джабраілов Д.В.*		
Економічний розділ	*Іванченков В.С.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 15.01.2024.

Керівник *Джабраілов Д.В.* _____
(підпис)

Завдання прийняв до виконання *Воскобойник С.О.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Вступ. Постановка мети та задач проектування*	*20.05.2024*	
2	*Аналіз ігрового жанру 2D скролл-шутер*	*23.05.2024*	
3	*Аналітичний огляд та вибір програмного рушія*	*25.05.2024*	
4	*Формування концепту ігрового процесу гри*	*28.05.2024*	
5	*Проектування основних елементів ігрового процесу*	*30.05.2024*	
6	*Проектування графічної частини гри*	*01.06.2024*	
7	*Реалізація графічної частини гри*	*03.06.2024*	
8	*Реалізація основних елементів ігрового процесу*	*05.06.2024*	
9	*Імплементація та відлагодження ігровіх елементів*	*07.06.2024*	
10	*Тестування працездатності елементів гри*	*09.06.2024*	
11	*Виправлення виявлених помилок*	*10.06.2024*	
12	*Аналіз результатів, підготовка слайдів презентації*	*11.06.2024*	
13	*Економічні розрахунки та питання з охорони праці*	*12.06.2024*	
14	*Підготовка графічної частини проекту*	*13.06.2024*	
15	*Підготовка проекту до захисту та тестування ПП*	*14.06.2024*	

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 07. 02 000. 00 ДП	Дипломний проект		
А4			РП 07. 02 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 07. 02 000. 00 ДП		
Розробив		Воскобойник С.О.		10.06	Розробка 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity	Літ.	Аркуш 4 / Аркушів 76
Перевірив		Джабраілов Д.В.		10.06		Н Д П	
Н. контр.		Петрашова В.І.		10.06		ВСП "ОТФК ОНТУ" гр.4РП-07	
Затверд.		Кривченко Ю.В.		10.06			

ЗМІСТ

ВСТУП

Індустрія комп'ютерних ігор є частиною світого IT-сектору. Для її підтримання у розробці ігор займаються такі спеціалісти як програмісти, керівники проектів, аніматори та інші. З кожнім роком кількість ігор, які виробляє цей сектор IT збільшується.

Це відбувається з одного боку через збільшення інвестицій у виробництво ігор, з іншого боку у зв'язку із спрощенням розробки проектів у простих жанрах, або із використанням класичних елементів ігрового процесу. Інвестиції дозволяють створювати більш технологічні проекти, які мають великий масштаб та складність архітектури. В той же час із розвитком технологій спрощується інструментарій для розробки, тому на ряду із програмними рушіями крупних розробників чи видавництв почали з'являтись безкоштовні або умовно безкоштовні рішення.

Через спрощення розробки кількість спеціалістів необхідних для створення гри різко зменшилась до однієї людини. Можна самостійно створювати гру з нуля, або використовувати вже заздалегідь створені ігрові обєкти із безкоштовними умовами розповсюдження, використовувати програмні модулі – чорні ящики.

Зважаючи на все вище зазначене, можна сказати, що ігрова індустрія є важливою та актуальною частиною IT-сектору. Для отримання навичок у розробці більш складних ігор, необхідно оволодіти основами. Тема мого дипломного проекту є «Розробка 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity», що дасть чудову змогу потренуватись у розробці ігор та на власному досвіді отримати уяву про складнощі в цьому процесі.

Гра виконується у класичному жанрі скролл-шутеру, що дасть змогу також переглянути існуючі варіанти розроблених ігор, отримати уяву про основні елементи ігрового жанру.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз ігрового жанру 2D скролл-шутер

1.1.1 Загальний аналіз жанру

Перед початком розробки теми дипломного проекту потрібно визначитись з основними етапами розробки комп'ютерних ігор. Таких етапів може бути багато, та вони можуть відрізнятись від одного проекту до іншого, але в загальному сенсі він складається із деяких складових.

Для початку, перед розробкою гри, якщо на неї є запит, виконується визначання з ігровим жанром. Такі жанри в ігровій індустрії представлені дуже широким переліком, яких не менше, а в деякому сенсі навіть більший ніж перелік кінематографічних жанрів. До ігрових жанрів відносять багато найменувать, але існують основні ігрові жанри, а також похідні, які можуть виділятись своєю особливістю, чи гібридні, які поєднують особливості декількох жанрів. Ігровий жанр визначається ігровими механіками, прийомами зовнішнього відображення, елементами ігрового процесу, розташування ігрової камери.

Звертаючись до теми мого дипломного проекту, можна побачити чіткий запит на жанр скролл-шутер. Це дає змогу відразу визначитись із тим, як буде виглядати гра, який в ній буде представлений ігровий процес, а також елементи та правила гри. Скролл-шутар – це гра в якому гравець переміщується або зверху-вниз, або знизу-верх, а також справа-наліво та зліва-направо. Під час переміщення по рівню, на зустріч гравцю відображаються елементи оточення або вороги, які необхідно знищувати, або уникати. Сам гравець отримує ушкодження, або знищується від торкання до ворога чи об'єкта на рівні. Також, негативні наслідки настають від ураження вогнем ворога. Можуть бути й інші варіанти впливу на гравця методами пасивних чинників, або умов гри.

Наступним етапом є визначення із технологіями виконання цифрової гри. У випадку із моєю темою дипломного проекту, чітко визначений ігровий програмний рушій, на якому буде виконуватись робота. Окрім визначення

платформи розробки у вигляді ігрового програмного рушія, потрібно визначитись і з іншими інструментами розробки. Такий вибір частково та декілька в більшій мірі залежить від вподобань розробника, а також від технічних вимог ігрового двигуна. Слід також розуміти, що у випадку роботи у команді, або в складі якоїсь команди, програмне забезпечення може бути продиктоване місцем праці, а не вподобанням розробника. Як вже було зазначено раніше, маючи чітко визначений ігровий програмний рушій, необхідно обирати такі програмні рішення та інструменти розробки, які відповідали б його потребам.

Ще один із етапів попередньої розробки лежить у визначенні основних ігрових елементів та механік, що будуть виконуватись під час розробки та будуть відігравати важливу роль у зацікавленні гравця до продукту. Частково їх перелік диктується ігровим жанром, втім часто розробники намагаються додати якусь несподіваний для жанру елемент, що буде «продавати» гру користувачу. Таким чином створювались комбіновані жанри ігор, коли посеред спортивної гри можуть з’являтись елементи шутеру, або стратегії.

Нарешті останній з етапів – способи реалізації проекту. Це в більшій мірі є визначення із архітектурою самої гри, її елементів, ігрових механік та елементів. Правильно спроектована архітектура набагато полегшує розробку проекту, його підтримку, а також продуктивність додатку. Це в тому числі формується платформою розробки та платформою виконання кінцевого продукту. Вимоги до архітектури гри можуть бути специфічними, у випадку роботи із портативними платформами, або досить вільними у разі, якщо гра буде виконуватись лише на комп’ютері.

Ігровий жанр скролл-шутерів має велику історичну ретроспективу, адже стояв у самому початку шляху формування ігрової індустрії. Оскільки основні ігрові елементи та механіки доволі прості з точки зору реалізації, цей жанр отримав велику розповсюдженість серед ігрових платформ, від ігрових автоматів, кишенькових пристроїв, до перших ігрових приставок для телевізорів, а також робочих комп’ютерів. Досить сильно для укорінення цього жанру в

масах зіграла його простота під час ігрового процесу, часто інтуїтивно зрозумілий інтерфейс та управління, які з перших хвилин гри випробують гравця та викликають у ньому азарт до гри.

Перші ігри такого жанру мали обмежений зміст ігрового процесу, а також прості правила «одного торкання». Це правило мало на увазі, що будь-яка взаємодія із ворогом корпусом гравця, або із кулею ворога призводила до програшу. Втім те ж саме діяло і у сторону ворога, коли той знищувався при контакті із кулею гравця. В подальшому було додана можливість додаткових «шансів», коли можна було помирати декілька разів перед повним перезапуском проходження, що у здебільшому продиктовано бажанням заробити більше грошей на токенах. В подальшому, жанр розвився разом із технологічними можливостями та отримував більш яскравий візуальний стиль, а також більшу кількість ігрових механік за рахунок можливостей фізичних носіїв зберігати більшу кількість інформації в роботі. Для визначення наповнення мого ігрового проекту, я переглянув декілька ігор цього жанру починаючи з ретроспективи до сьогодення.

1.1.2 Аналіз особливостей жанру гри Zero Wing

Я почав своє дослідження з гри Zero Wing яка є одною із класичних цифрових ігор жанру скролл-шутеру. Вона була створена у 1989 році в Японії та 1990 році в Північній Америці для ігрових автоматів. У подальшому отримала свою версію для ігрової приставки Sega Mega Drive. У 2022 роках офіційно вийшла як повноцінна гра для персональних комп’ютерів із переробкою графіки та деякими нововведеннями до ігрового процесу. В свої часи гра отримала велику популярність через свою різноманітність, візуальний стиль а також звукове супроводження. Ігровий процес є дуже простим – гравець безперервно рухається по рівню зліва-направо, а йому на зустріч рухаються вороги та «рельєф» оточення, реалізований принцип «одного торкання», тому частіше за все гравцю давалося 3 «життя», та додавались нові за ігровий рахунок, або находження його на рівні.

На відміну від багатьох інших ігор цього жанру тих часів, Zero Wing мав свій сюжет, а також різни додаткові ігрові механіки, як модифікація зброї та корабля гравця. Модифікація зброї та корабля гравця виконувалась за рахунок окремих об’єктів, які збирались на рівні під час ігрового процесу. Ці об’єкти могли випадати з ворогів, або вільно рухатись по рівню у напрямку лівого боку екрану. Додаткове життя гравець міг отримати за ігровий рахунок, або підібравши спеціальний об’єкт на рівні. Переглянути скріншот гри Zero Wing для Sega Mega Drive можна на рисунку 1.1:

Рисунок 1.1. Скріншот гри Zero Wing для Sega Mega Drive

1.1.3 Аналіз особливостей жанру гри Jets’n’Guns

Jets’n’Guns – ще одна гра в жанрі скролл-шутеру, була випущена у 2004 році для персональних комп’ютерів з різними операційними системами, і доступна для покупки в наш час у магазині Steam. У своєму часі стала дуже популярною, оскільки мала динамічний ігровий процес та жвавий музикальний супровід. Не дивлячись на те, що гравець міг переміщуватись по екрану, втім прогресія по рівню йшла із визначеною швидкістю, але можливість рухатись надавала гравцю можливість маневрувати під час ігрового процесу, уникати

ворогів, їх куль, збирати модифікації у складних місцях на рівні. На відміну від Zero Wing, в цій грі реалізовано декілька параметрів, які впливають на гру: міцність корпусу та перегрів зброї, а також вміст грошового рахунку, який можна використовувати для покращення корабля гравця між рівнями. Хаб між рівнями став особливістю цієї гри, який давав змогу відредагувати свій підхід до проходження гри. Додатково, гравець міг використовувати модифікації перед мочатком рівня, для того, щоби отримувати якісь бонуси під час проходження гри. Скріншот гри зображені на рисунку 1.2:

Рисунок 1.2. Скріншот гри Jets’n’Guns

1.1.4 Аналіз особливостей жанру гри Cuphead

Cuphead – також гра жанру скролл-шутер, яка вийшла в світі на всіх ігрових платформах у 2017 році та набула великої популярності за свій візуальний стиль та рівень складності ігрового процесу. Розробники зробили ставку на стиль класичних мультфільмів 40-50-х років, додавши до них яскравих кольорів. Cuphead так само як і попередні ігри має типовий ігровий процес під

час якого необхідно проходити через рівень зліва-направо, знищуючи ворогів, а також уникаючи торкання із небезпечним оточенням.

Одною із особливостей, що відрізняють гру від інших сучасних аналогів, відсутність отримання пошкоджень, а миттєва втрата життя. Через це та складні паттерни в поведінці противників, гра отримала популярність, як складне випробування для вибагливих гравців. Так само, як і в Jets'n'Guns гравцю доступні паузи між рівнями, на яких він може обирати наступний рівень, а також отримувати нові інструменти для проходження гри. Скріншот гри можна побачити на рисунку 1.3:

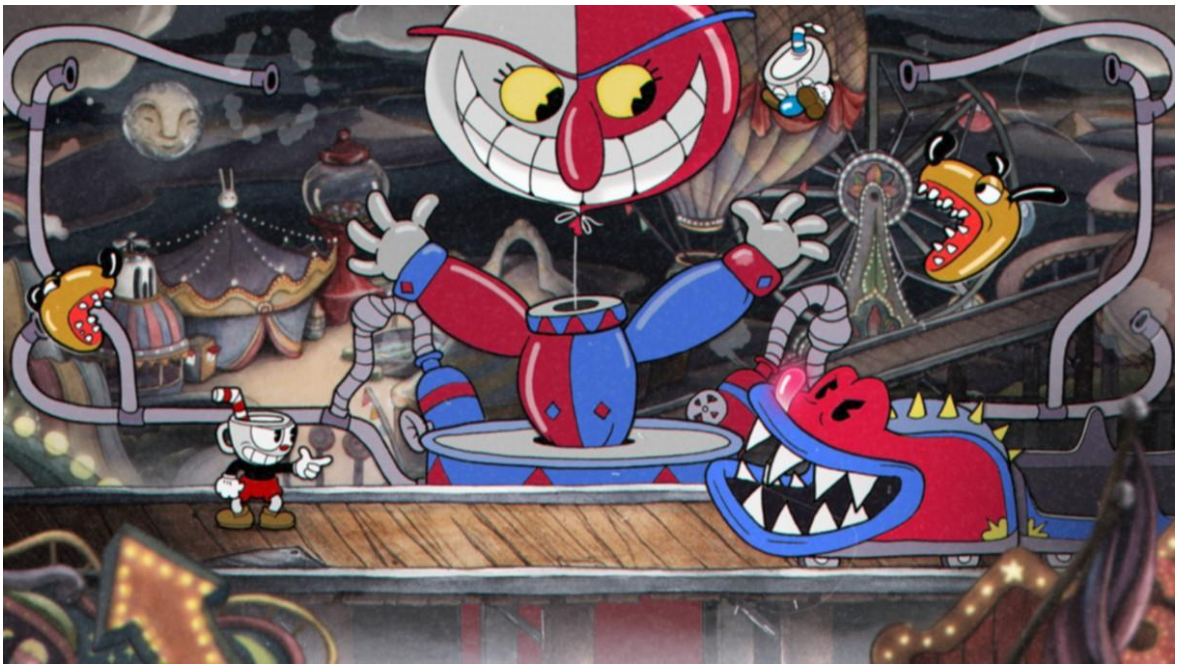


Рисунок 1.3. Скріншот гри Cuphead

1.1.5 Результати аналізу

Після аналізу існуючих аналогів у ігровій індустрії, можна зробити декілька висновків. В обраному жанрі є сталі шаблони по ігровому процесу, як автоматичний рух гравця по рівню, але можливість маневрувати на сцені. Впливання оточення на ігровий процес. У більш сучасних цифрових іграх гравцю доступні параметри, що відображають «здоров'я», або інших показників, а ігровий рахунок можна витратити на якісь модифікатори чи зброю. Виходячи з цього можна визначити, що необхідно реалізовувати від жанру у проекті моєї дипломної роботи.

1.2 Аналітичний огляд програмних рушіїв з умовно безкоштовним доступом

Буд-яка розробка ігрового проекту основана на виборі програмного забезпечення для вирішення питання технічної реалізації та супроводу. Вибір ігрового програмного рушія є дуже важливою частиною початку розробки, оскільки визначає важливі питання використання технологій розробки, принципів архітектури гри, способів реалізації механік та багато інших питань. Дуже часто ігрові програмні рушії диктують розробникам те, як буде працювати їх проект і чи буде він працювати зовсім. Наприклад, одною із головних причин провалу гри Anthem – це її технічна сторона, яка була дуже поганою через створення рольової гри із елементами кастомізації та менеджменту, відкритого світу та глибокої мультиплеєрної взаємодії, використовувався внутрішній ігровий програмний рушій компанії Electronic Arts, що початково створювався як ігровий програмний рушій для ігор жанру шутер, в яких не було наміру реалізовувати ресурси, глибоку кастомізацію та багаторівневу мультиплеєрну взаємодію. Через це розробникам доводилось витрачати багато часу та зусиль для того, щоби створювати окремий інструментарій для розробки рольових ігор, залучаючи для цього розробників з інших підрозділів компанії Electronic Arts.

Ігрові програмні рушії можна розділити на безкоштовні, умовно безкоштовні та платні. До останніх найчастіше відносяться ігрові програмні рушії створені всередині крупних ігрових компаній для внутрішніх проектів, найчастіше, ці компанії не діляться своїми рішеннями з конкурентами, тільки у випадках партнерства, або фінансових відносин. Безкоштовні ігрові програмні рушії найчастіше розповсюджуються у вигляді бібліотек із мінімалістичним інструментарієм для розробки, що викликає досить серйозні проблеми для розробки. Ідеальними у відношенні ціна-якість є ігрові програмні рушії, що розповсюджуються на умовно безплатній основі. Ці програмні рушії мають величезний інструментарій для розробки, а також не потребують плати за використання, при виконанні окремих умов.

Тема мого дипломного проекту «Розробка 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity». Не зважаючи на те, що в темі прописано обраний програмний рушій, важливим кроком у пояснювальній записці буде проілюструвати, чому мій вибір зупинився саме на ігровому програмному рушії Unity.Для цього я проведу аналіз його конкурентів.

1.2.1 Аналіз ігрового програмного рушія Unity

Unity — це популярний кросплатформний ігровий програмний рушій і середовище розробки, створений Unity Technologies. Він надає розробникам інструменти для створення 2D і 3D ігор, додатків віртуальної та доповненої реальності, моделювання, візуалізації тощо. Простота використання та доступність Unity роблять його привабливим вибором як для новачків, так і для досвідчених розробників. Він забезпечує зручний графічний інтерфейс, багато готових ресурсів і компонентів, а також використання потужної мови програмування C# для створення взаємодії та логіки гри. На рисунку 1.4 показано приклад роботи у редакторі ігрового програмного рушія Unity:

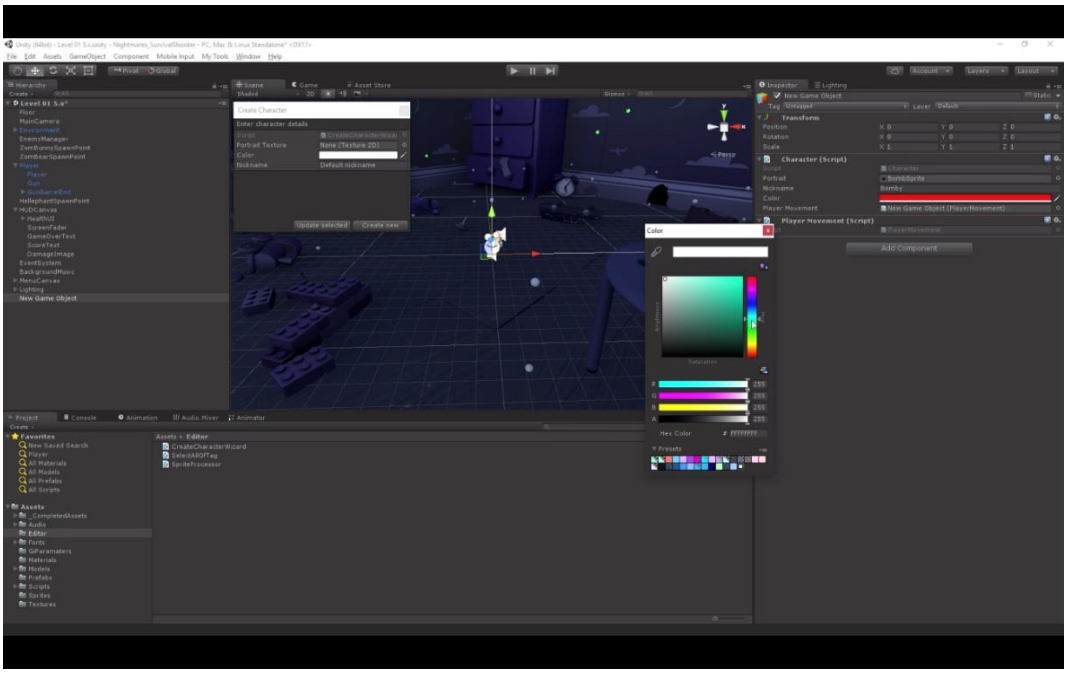


Рисунок 1.4. Приклад роботи у ігровому програмному рушії Unity

Однією з ключових особливостей Unity є його кросплатформенність. Він підтримує кілька платформ, включаючи ПК, консолі, мобільні пристрої, віртуальну реальність і доповнену реальність, що дозволяє розробникам

охоплювати більшу аудиторію та запускати свої продукти на кількох пристроях.

З точки зору ліцензування, механізм ігрового програмного рушія Unity надає повний доступ до своїх функцій безкоштовно для створення ігор будь-якого рівня. Купівля ліцензії потрібна лише тоді, коли сума, зароблена на одного користувача або його власну команду чи компанію, досягає двохсот тисяч доларів.

1.2.2 Аналіз ігрового програмного рушія CryEngine

Для того, щоби дати розуміння причини обрання ігрового рушія Unity, розглянемо його двох найближчих конкурентів. Почнемо з ігрового програмного рушія CryEngine який є потужним ігровим програмним рушієм, розробленим німецькою компанією Crytek. Сам ігровий програмний рушій відомий своїми вражаючими фізикою та графікою, а також широкими можливостями для створення відкритих багаторівневих та реалістичних ігрових світів. CryEngine був використаний у різних популярних іграх, таких як серія Crysis, Ryse: Son of Rome та Hunt: Showdown. Початково отримав відомість як ігровий програмний рушій, що відтворює настільки неймовірно реалістичну графіку, що навантажує найсильніші системи. Його правдоподібна симуляція фізики та руйнування довкілля також дуже довго була основною рисою цього ігрового програмного рушія. На рисунку 1.5 можна побачити приклад графіки, що відтворює програмний рушій CryEngine.

Рисунок 1.5. Приклад графіки, що відтворює програмний рушій CryEngine

З точки зору можливостей, то однією з ключових особливостей CryEngine є його підтримка технології трасування променів (ray tracing), яка дозволяє досягти фотореалістичних ефектів освітлення та відображення поверхонь на різних матеріалах, як метал або водна поверхня. Крім того, CryEngine має високий рівень масштабованості, що дозволяє створювати як невеликі інді-проекти, так і великі ААА-ігри. Програмний рушій також забезпечує розробників інструментами для створення інтерактивного та живого ігрового середовища, включаючи систему штучного інтелекту, керування анімацією персонажів, а також інструменти для створення звукової атмосфери, що дуже часто використовують для створення досить якісних ігор в жанрі хоррор. CryEngine також пропонує гнучку систему багатоплатформної розробки, що дозволяє випускати ігри на різних платформах, включаючи ПК, консолі та мобільні пристрої. На рисунку 1.6 можна побачити приклад роботи з вбудованим аніматором рушія CryEngine.

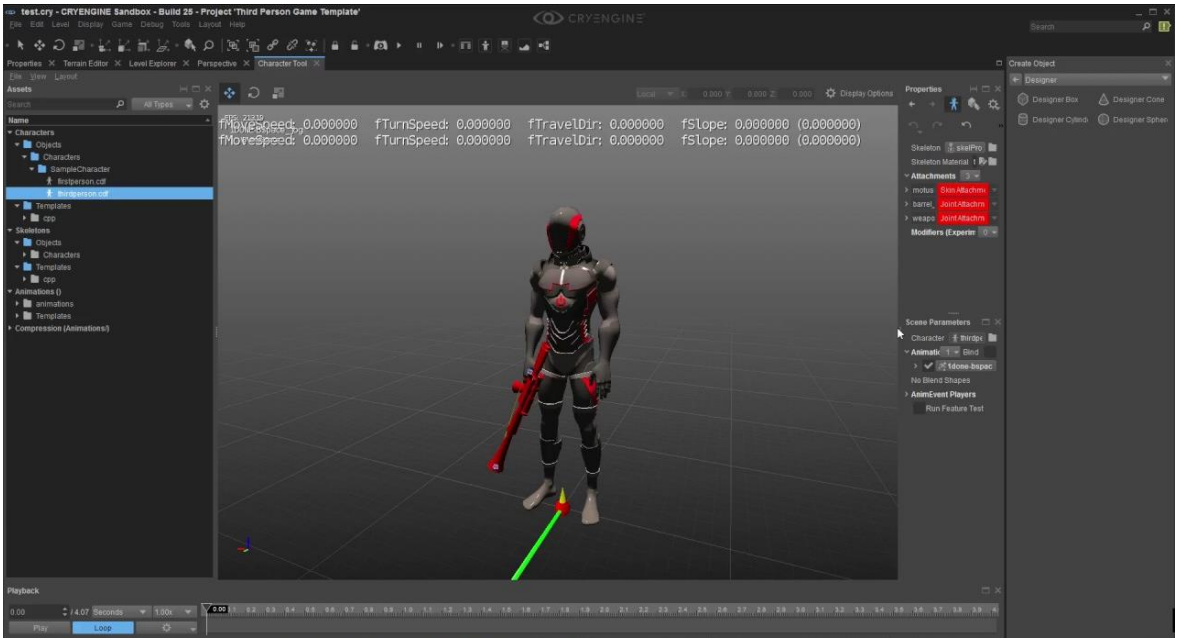


Рисунок 1.6. Приклад роботи з вбудованим аніматором рушія CryEngine

Загалом ігровий програмний рушій CryEngine є безкоштовним, втім система побудована таким чином, що використовувати індивідуальну підтримку, бібліотеку ассетів, можна лише з покупкою ліцензії. Окрім того, її все одно необхідно буде купувати, якщо кількість фінансового обороту користувача програмного рушія перевищить визначеного рівня.

1.2.3 Аналіз ігрового програмного рушія Unreal Engine

Другим відомим конкурентом Unity є ігровий програмний рушій Unreal Engine. Це потужний і широко використовуваний ігровий програмний рушій який був розроблений компанією Epic Games та вперше був використаний для створення гри Unreal Turnament звідки й взяв своє ім'я. Він надає розробникам інструменти для створення високоякісних ігор та візуалізацій, а також розробки віртуальної реальності (VR), доповненої реальності (AR) та багатьох інших інтерактивних проектів, різність та глибина яких залежить від вмінь розробника. Ігровий програмний рушій Unreal Engine пропонує широкий набір функцій, включаючи потужний рушій графіки, підтримку власного рушія фізіки, інструменти для роботи зі штучним інтелектом персонажів та середовища, глибоку анімацією персонажів, звуком та багатьом іншим. Він також включає графічний інтерфейс користувача, який спрощує процес розробки та створення контенту. На рисунку 1.7 можна побачити приклад роботи із анімацією у ігровому рушії Unreal Engine.

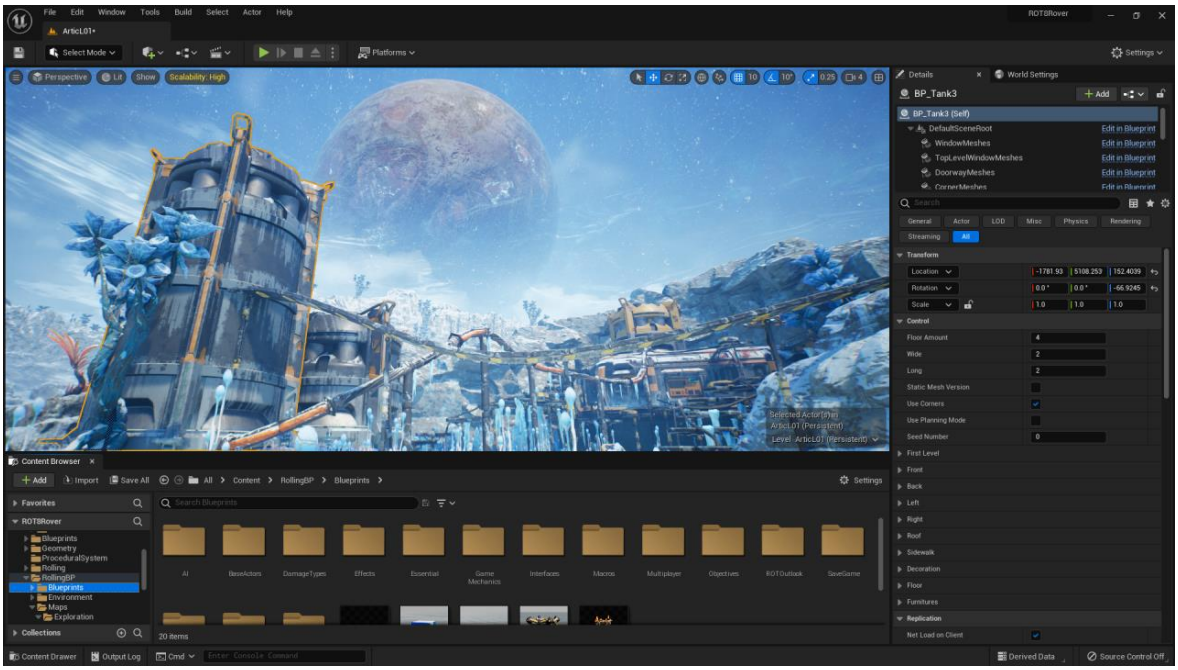


Рисунок 1.7. Приклад роботи із анімацією у ігровому рушії Unreal Engine

Однією з ключових особливостей Unreal Engine є його графічний рушій, який підтримує передові технології, такі як трасування променів (ray tracing), що дозволяє створювати неймовірно реалістичні графічні ефекти відображення та

бліків на водній, або металічних поверхнях. Це робить Unreal Engine популярним вибором для розробників ААА-ігор, а також для створення вражаючих візуалізацій та симуляцій в інших галузях. Також, цей програмний рушій використовують для створення фільмів за допомогою спеціального інструментарію, який за допомогою штучного інтелекту будує кадр навколо людини. Ігровий програмний рушій Unreak Engine лежить в основі багатомільярдної гри Fortnite, яка є лідером у жанрі королівської битви з елементами платформеру, крафтингу та пісочниці. На рисунку 1.8 можна побачити приклад графіки яку відтворює Unreal Engine:

Рисунок 1.8. Приклад графіки яку відтворює Unreal Engine

З точки зору ліцензування, то ігровий програмний рушій Unreal Engine є безкоштовним, проте не дає доступу до коду самого ігрового програмного рушія, деякі платформи розробки та можливості заблоковані, а також закритий доступ до виділеного середовища розробників та підтримки.

1.2.4 Результати аналізу та обґрунтування використання Unity

Проаналізувавши ігрові програмні рушії було вирішено використовувати рушій Unity. Рішення було зумовлено в першу чергу доцільністю використання можливостей ігрових рушіїв. Його конкуренти надаються передову графіку дуже високого рівня, яка не потрібна у проекті в 2D жанрі. Темою мого дипломного

проекту є «Розробка 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity», тому використовувати настільки потужні рушіх не має сенсу. Окрім того, для Unity існує багато навчального матеріалу та відкритий сайт із документацією, яку можна вивчити у разі виникнення проблем. Також ком'юніті цього ігрового програмного рушія надає допомогу всім бажаючим безкоштовно. Ще одним важливим плюсем у виборі цього ігрового програмного рушія є можливість використовувати AssetStore, в якому вільно розповсюджуються графічні, і не тільки, матеріали для розробників ігор. Окрім того, цей програмний рушій дозволяє вільно побудовувати ігрові проекти на будь-які доступні ігрові платформи. Через вказані причини, вибір програмного рушія був зроблений на користь Unity.

1.3 Вибір інструментів розробки

Наступним важливим кроком у підготовці до розробки ігрового проекту є обрання інструментарію. Програмне забезпечення для розробки має дуже велику роль у процесі роботи, адже кожне рішення має свої особливості та що не менш важливо, користувач може мати свої улюблені редактори, в яких має навички роботи та знає всі нюанси. Більшість крупних компаній розробників досить вільно відносяться до прикладного інструментарію розробки, наприклад, як редактори коду. У ході виконання моєї дипломної роботи, мені потрібно було реалізувати деяку графічну частину, тому до вибору графічного редактору потрібно було поставитись дуже відповідно.

Редактор коду – це спеціалізоване програмне забезпечення, яке використовується для створення, модифікації та організації вихідного коду програм. Такі редактори зазвичай мають інтуїтивно зрозумілий інтерфейс, підтримують підсвічування синтаксису, автоматичне доповнення коду, пошук та заміну тексту, а також інтеграцію з іншими інструментами розробки, включаючи системи контролю версій, дебагери та компілятори. Серед відомих редакторів коду можна виділити Visual Studio Code та Sublime Text.

У процесі вибору з численних редакторів коду, я віддав перевагу Microsoft Visual Studio. Цей вибір був зумовлений тим, що ігровий движок Unity працює

на мові програмування C#, що вимагає від середовища розробки не лише підтримки форматування коду, але й здатності взаємодіяти з іншими класами, методами, ігровими об'єктами та бібліотеками. Крім того, Microsoft Visual Studio дозволяє легко доступитися до проекту гри і є рекомендованим вибором серед розробників та підтримки Unity.

Що стосується роботи з графікою, то потрібен графічний редактор. Графічний редактор – це програмне забезпечення, призначене для створення, модифікації та оптимізації графічних зображень. Він надає користувачам інструменти для створення ілюстрацій, редагування фотографій та інших видів графіки, включаючи кисті, олівці, спреї, штампи, інструменти виділення та багато іншого, а також можливості для коригування кольору, розміру, прозорості та інших параметрів зображення. Серед популярних графічних редакторів можна назвати Adobe Photoshop, GIMP, CorelDRAW та Adobe Illustrator.

У контексті розробки цієї гри я вирішив використовувати Adobe Photoshop, оскільки у мене вже є досвід роботи з цим інструментом. На даному етапі розробки гра не вимагає створення складної графіки, а лише потребує розробки кількох спрайтів та редагування зображень, включаючи зміну пропорцій та розмірів.

Важливою частиною процесу розробки є також система контролю версій. Система контролю версій – це інструмент, який дозволяє розробникам відслідковувати зміни у вихідному коді та інших файлах проекту, порівнювати версії, відкочувати до попередніх станів та об'єднувати роботу різних учасників. Вона також забезпечує історію змін, що сприяє кращому розумінню процесу розробки та співпраці над проектом. До відомих систем контролю версій належать Git, Subversion та Mercurial.

У моєму випадку я використовую BitBucket та SourceTree. BitBucket – це веб-базована система контролю версій, яка є безкоштовною та інтегрована з багатьма іншими сервісами в рамках екосистеми Atlassian. SourceTree – це клієнтська програма, яка забезпечує графічний інтерфейс для роботи з системами контролю версій, що робить процес більш зручним і інтуїтивно

зрозумілим. Ці інструменти є незамінними для збереження результатів розробки в онлайн-середовищі.

1.4 Формування концепту ігрового процесу 2D-гри в жанрі скролл-шутер

Для реалізації задачі дипломного проектування потрібно виконати основні елементи циклу розробки будь-якої гри, який також складається з формування ігрового концепту. Ігровим концептом можна назвати формування переліку потреб, які сформовані на етапі проробки ідей гри. А взагалі концепт-документ гри – це документ, який містить основні ідеї, концепції, механіки геймплею, сюжетні елементи та інші ключові аспекти, пов’язані з розробкою гри. Цей документ зазвичай є основою для її подальшого розвитку. У концепт-документі гри частіше за все розглядають такі питання:

- Опис гри: Загальний опис ігрового світу, сюжету, атмосфери та основної мети гри;
- Механіка геймплею: Опис ігрових механік, механіки управління, фізики ігрового світу, системи прогресії, системи бою та інших ігрових аспектів;
- Сюжет та персонажі: Опис сюжету гри, персонажів, їх характеристик, мотивацій та відносин між ними;
- Графіка та аудіо: Концепції та ідеї з візуального стилю, арт-дизайну, анімації та звукового оформлення гри;
- Світогляд і фон: Додаткові деталі про світ гри, його історію, культуру, технології та інші аспекти, які можуть впливати на ігровий досвід.
- Технічні вимоги та можливості: Обговорення технічних аспектів розробки гри, включаючи платформи, програмні рушії, інструменти та інше.

Як вже зрозуміло з опису, такий документ допомагає розробникам чітко визначити напрям роботи та розділити гру на чіткі зони відповідальності між командою, та частини розробки гри, яку можна потім розподіляти між командою

у часі та етапах розробки. Завдяки цьому, коли є концепт документ, уникається розповсюджена проблема розробників, коли проект набуває ігрових елементів, або механік, які не передбачені концепт документом та призводять до створення хаосу у розробці.

Темою мого дипломного проекту є «Розробка 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity» з якої чітко видно, що потрібно розробляти гру в жанрі скролл-шутер із горизонтальною орієнтацією та елементами ігрового процесу, який притаманний цьому жанру. Також ми визначились із ігровим програмним рушієм, який також може створювати свої вимоги до розробляємого продукту. Виходячи з вищезазначеного я можемо розпочати розробку концепту моєї гри.

Отже розглядаючи гри з жанрової точки зору, потрібно мати на увазі її особливості. Такі, наприклад як проходження рівнів – об’єкт гравця може переміщуватись по рівню досить вільно, але тільки в межах визначених камерою. При тому в цілому рух по тлу рівня залишається непідконтрольним для гравця. Проте, цей рух по рівню можна прискорювати, або навпаки сповільнювати.

Також відмічаю питання отримання пошкодженням гравцем. В класичних скрол-шутерах є два варіанти механіки отримання ушкоджень. Перший є класичним – «одне торкання», коли гравець не має права на «помилку» та знищується кожний раз коли торкається корабля ворога, або його кулі, чи елементів оточення, які також можуть бути на рівні. Другий варіант – це реалізація системи ушкоджень, яка складається з параметрів «здоров’я» корабля гравця, або його щита. Таким чином, кожний раз коли гравець взаємодіє із оточенням, кораблем противника або його кулею, гравець втрачає частину здоров’я корабля або його щита. Слід також відмітити, що існує механіка «жетонів», коли гравець будучі знищеним може знову почати гру з місця, де отримав поразку заплативши один жетон, або раніше це називали життям. Втім цю механіку можна використовувати для обох варіантів реалізації отримання пошкоджень від оточення.

Ще одним основним елементом гри, який пов’язаний із ігровим жанром є прогресія по рівням. В одних іграх гравець перемішується від одного рівня до іншого послідовно та безперервно. В інших дається змога обирати наступний рівень та модифікувати свої ігрові параметри, зброю, персонаж/корабель.

Розглянувши ці основні жанрові питання, можна сформувати із них наступні положення, щодо ігрового концепту майбутньої гри:

- Гра буде мати постійну прогресію між рівнями, без пауз між рівнями;
- Гра буде реалізовувати систему ігрових ресурсів гравця, як міцність корпусу, броня, або набої;
- Гравець не буде мати можливість пришвидшувати, або сповільнювати прогресію по рівню;
- На рівнях будуть з’являтися об’єкти для підвищення ігрових ресурсів;
- Гравцю необхідно буде протистояти двом типам ворогів – оточенню та кораблі-вороги.

Визначивши основні елементи ігрового концепту, є можливість розглянути технічну сторони реалізації цих ігрових концептів.

Розділ гри на рівні та сцену необхідно реалізовувати виходячи с технічних особливостей ігрового програмного рушія Unity, на якому буде виконуватись робота. Сам рівень буде виконуватись завдяки сцені, на якій знаходиться гравець та його камера – вона буде визначати межі сцени ігрового процесу, по якій гравець зможе переміщатись та огороджувати простір руху корабля гравця у сцені ігрового програмного рушія Unity та його кораблі/вороги та об’єкти оточення. Оскільки дія буде проходити в космічному просторі, як класичному проявленню ігрового простору скролл-шутерів, то об’єктами оточення будуть астероїди, а ворогами космічні кораблі, які будуть мати інший дизайн за дизайн гравця. Самі рівні, технічно не будуть змінюватись. Змінюватись буде ігрова обстановка, яка буде диктуватись спеціальним менеджером.

Система ігрових ресурсів також буде реалізовуватись за допомогою програмних рішень та реалізації об’єкту гравця зі своїми параметрами. Ці ігрові

параметри реалізовуватимуться таким чином, щоби вільно взаємодіяти із інтерфейсом гравця, передаючи у нього інформацію, як в одну сторону, так і в іншу.

Для того щоби генерувати нові об’єкти, ворогів, предмети для підвіщення ресурсів, необхідно реалізувати ігровий менеджер, що програмним шляхом буде відтворювати копії генеруємих ігрових об’єктів. Із загальним концептом ігрового процесу можна ознайомитись на рисунку 1.9, який схематично зображує розміщення елементів ігрового програмного рушія у просторі редактору рушія.

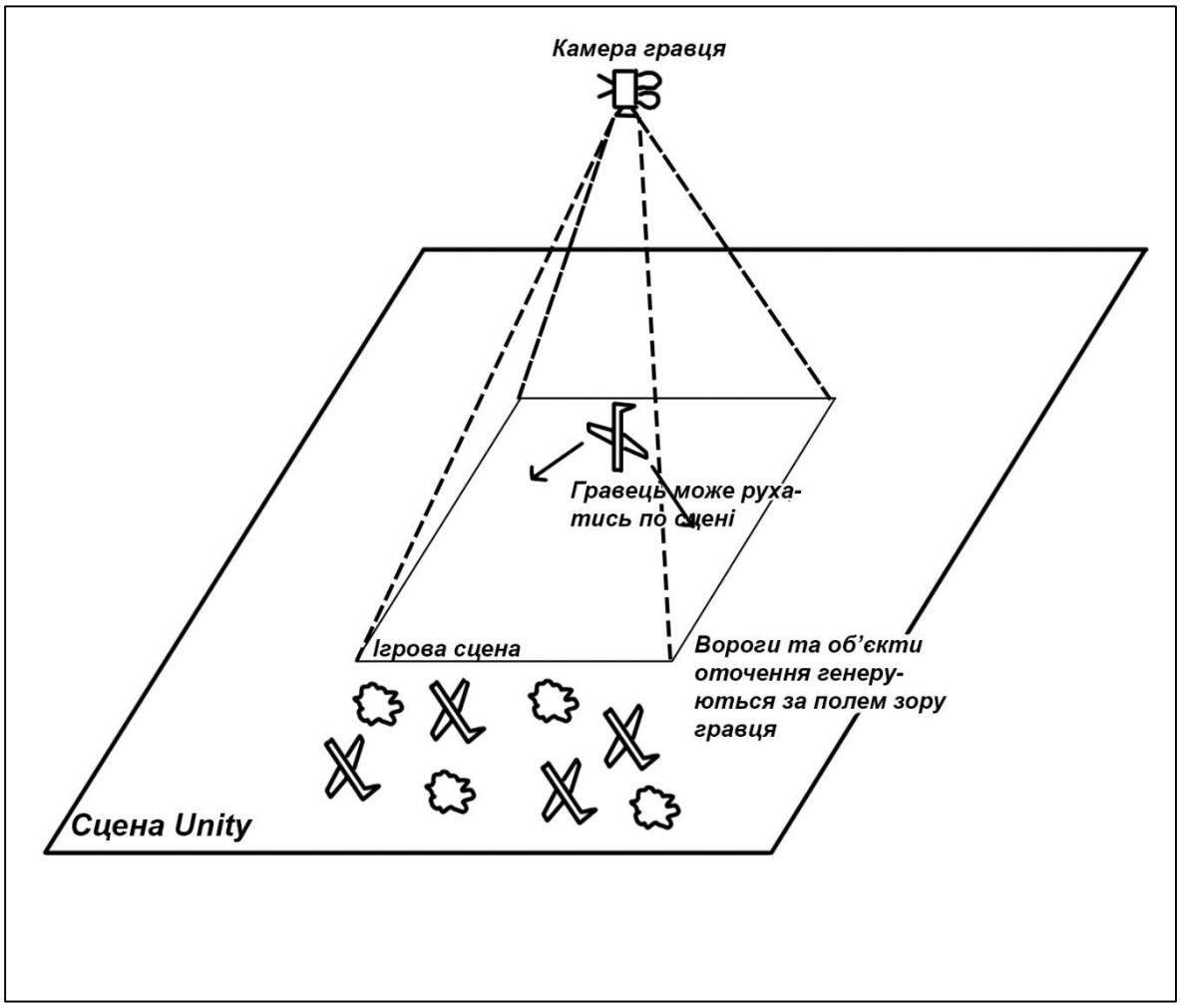


Рисунок 1.9. Загальний концепт ігрового процесу

Окремо слід розглянути питання ігрового інтерфейсу, який потрібно буде реалізувати на етапі розробки. Всього він має відображати декілька показників: міцність корпусу, щити та ігровий рахунок. Кожний з цих показників має змінюватись у реальному часі, при отримані ушкоджень, або зарахуванні

ігрового рахунку. Для того, щоби бути замітними вони будуть реалізовані у виді панелей на краях екрану, які будуть заповнюватись в залежності від стану. Ігровий рахунок буде відображатись у текстовому виді та буде змінюватись у ході ігрового процесу за знищення ігрових об’єктів, або кораблів-ворогів на рівні. Із загальним концептом ігрового інтерфейсу гри можна ознайомитись на рисунку 1.10:

Рисунок 1.10. Загальний концепт ігрового інтерфейсу гри

Виконавши формування ігрового концепту, або концепт-документу гри, можна переходити до етапу проектування елементів гри. Завдяки виділеним особливостям проекту, висунувши потреби до елементів та механік до реалізації, можна буде більш конкретно виконувати роботи з проектування не допускаючи помилок та не витрачаючи часу на опрацювання моментів у роботі та взаємодії всіх елементів проекту.

1.5 Проектування основних елементів ігрового процесу та принципи їх роботи

Перед початком реалізації безпосередньо ігрових механік логіки та елементів, необхідно виконати їх проектування, проектування скриптів та

модулів гри. Спочатку я виконав розподілення гри на модулі, виходячи з тієї концепції, яка була розроблена раніше. Всього можна виділити наступні модулі у грі:

- Код ворожих об’єктів;
- Код оточення;
- Код генераторів;
- Код гравця;
- Код інтерфейсу.

Кожний з перерахованих модулів буде мати в собі перелік того коду, якій потрібно реалізувати. Схематичне зображення цих модулів та коду що до них належіть зображено на рисунку 1.11:

Рисунок 1.11. Схема основних модулів гри та коду, що належить до них

Слід зазначити, що концепція модулів дає змогу додавати елементи, без виконання додаткових робіт з налаштування проекту. Наприклад, якщо буде необхідність додати новий ворожий об’єкт, то це можна буде зробити досить швидко, виконавши розділення такого об’єкта на складові. Це ж відноситься і

для додавання буд-якого елементу гри не залежно від його складності, потрібно лише спроектувати його таким чином, щоби він або відносився до вже існуючих модулів, або налаштувати із ними зв’язок, якщо такий взагалі потрібен будет. Схему змісту складових типового ворожого об’єкту можна побачити на рисунку 1.12:

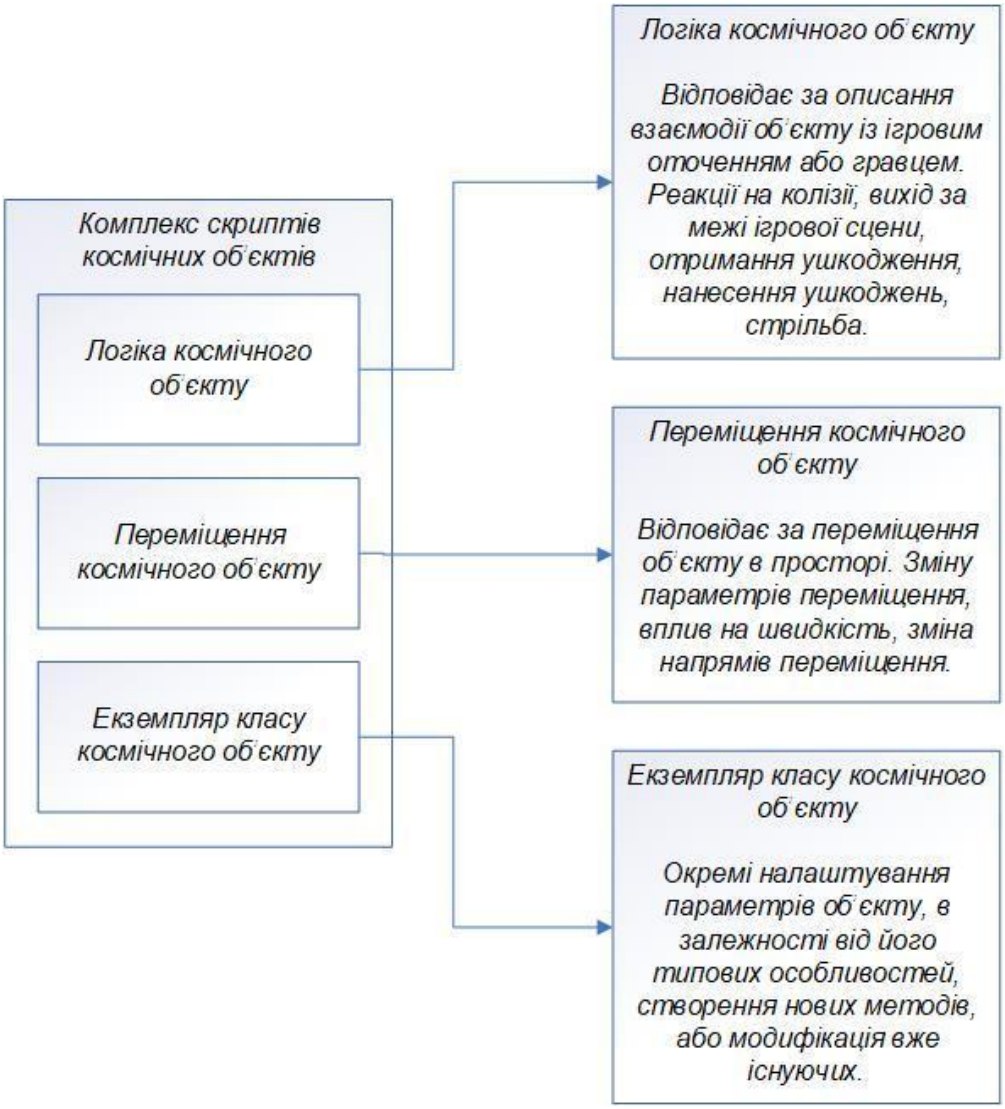


Рисунок 1.12. Схема змісту складових типового ворожого об’єкту

Як можна бачити з рисунку 1.12, космічні об’єкти мають логічну структуру із коду. Ця структура приймає на себе роль за реалізацію механік переміщення об’єкту, використання ним своєї логіки, та, у разі такої потреби, створення додаткових параметрів для об’єктів, або функціональних методів. Виконання останнього пункту стає можливим лише через використання загального класу космічних об’єктів, який є основою для будь-якого ігрового

об’єкту на сцені, якщо такий приймає участь у ігровому процесі. Схему структури загального класу космічного об’єкту можна побачити на рисунку 1.13.

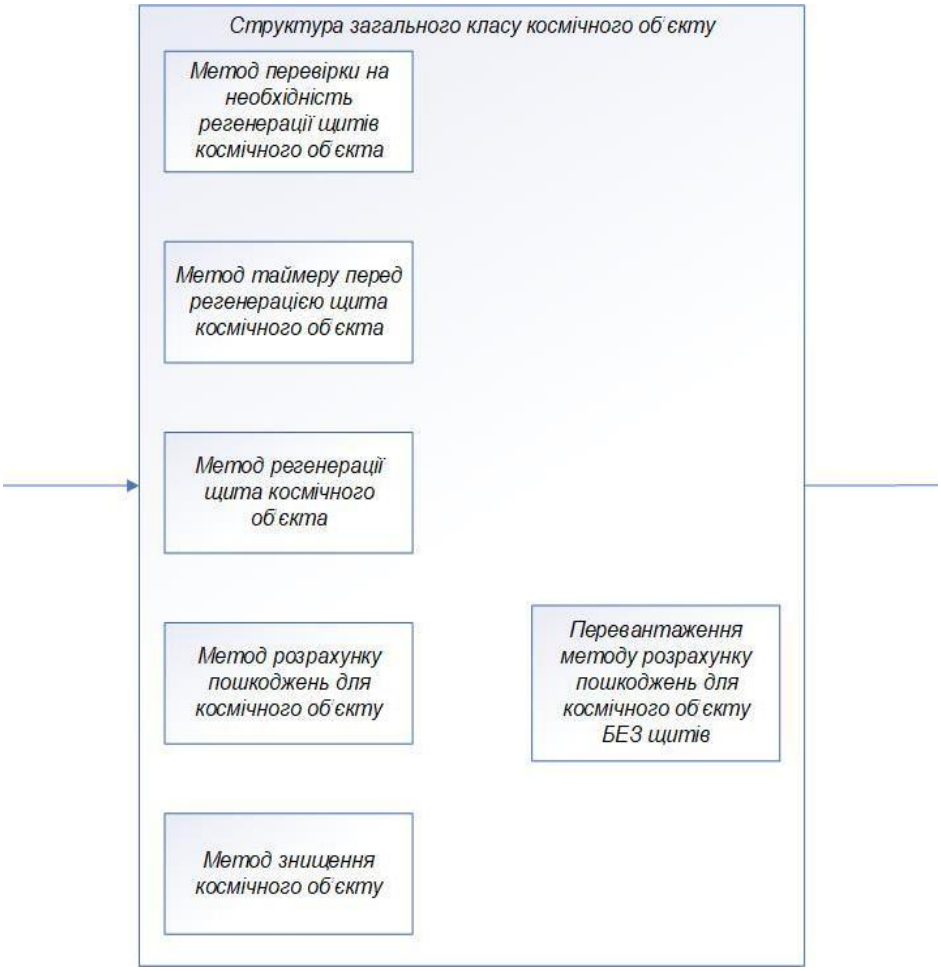


Рисунок 1.13. Схема структури загального класу космічного об’єкту

Загальний принцип взаємодії між модулями гри можна описати як наскрізну взаємодію між собою, передача даних від одного модулю до іншого. Можна виділити окремі модулі, які виділяються, а саме генератори. Вони існують поза ігровим процесом, постійно генеруючи ігрові об’єкти, з якими можна взаємодіяти – ворогів, або предмети для ремонту корабля, отримання щитів. Саме завдяки такій архітектурі передачі даних, можна гарантувати безпечний, а також зрозумілий принцип передачі даних. Крім того, із такою архітектурою додання новим об’єктів чи модулів до ігрового проекту може проходити значно швидше. Переглянути схему передачі даних між модулями та об’єктами гри можна на рисунку 1.14:

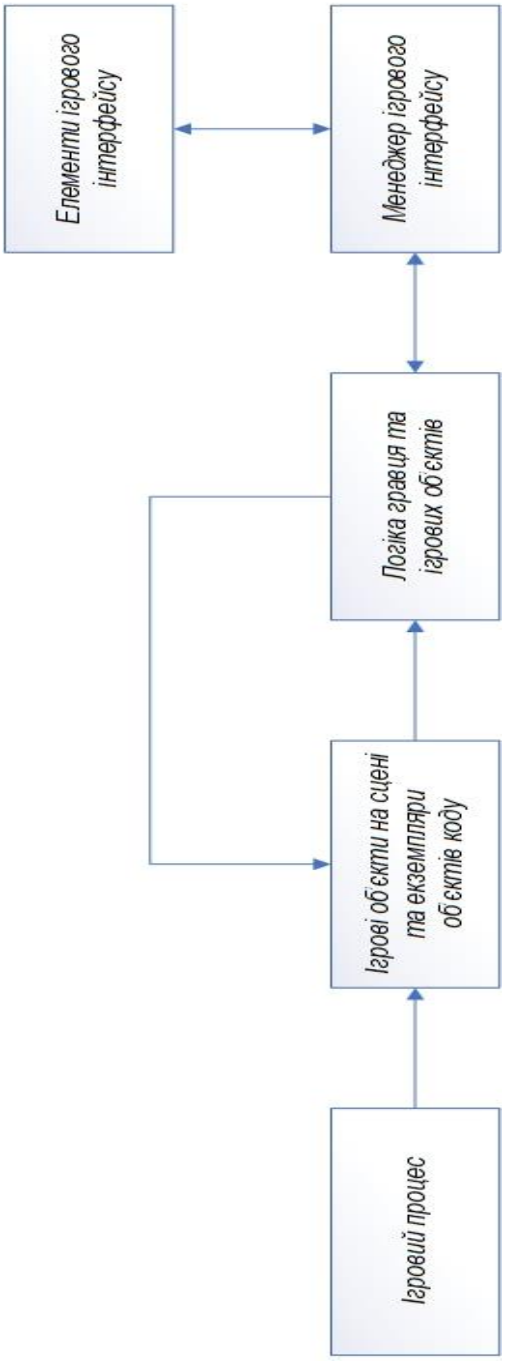


Рисунок 1.14. Схема передачі даних між модулями та об’єктами гри

Потрібно окремо зупинитись на Менеджері ігрового інтерфейсу. Його роботу можна було б організувати методом передачі даних через одного посередника, який би керував всіма процесами у грі. Але це дещо ускладнює проект, оскільки створює нехай більш надійну, але складну структуру. Через це я вирішив передавати дані напряму до ігрового інтерфейсу.

Його роль складається у передачі даних від модулів гри до ігрового інтерфейсу, а також, в разі необхідності, передачу даних від ігрового інтерфейсу до цих самих модулів гри та далі до цільових об’єктів. Оскільки кількість

вірогідних елементів інтерфейсу не така велика, можна створити посилання від відповідних елементів параметрів гравця до елементів інтерфейсу та навпаки.

1.6 Реалізація основних елементів ігрового процесу

Для успішного виконання теми мого дипломного проекту, потрібно створити гру на обраному ігровому програмному рушії. Для створення проекту використовується відповідний редактор та програма UnityHub, яка поєднує в собі не тільки інструменти створення ігрових проектів, але й концентрує в удобному одному місці їх перелік. На рисунку 1.15 можна побачити вікно створення проекту у UnityHub. Як можна побачити тут є деякі шаблони для створення проектів, які завантажують початкові ігрові об’єкти та налаштування сцени редактору.

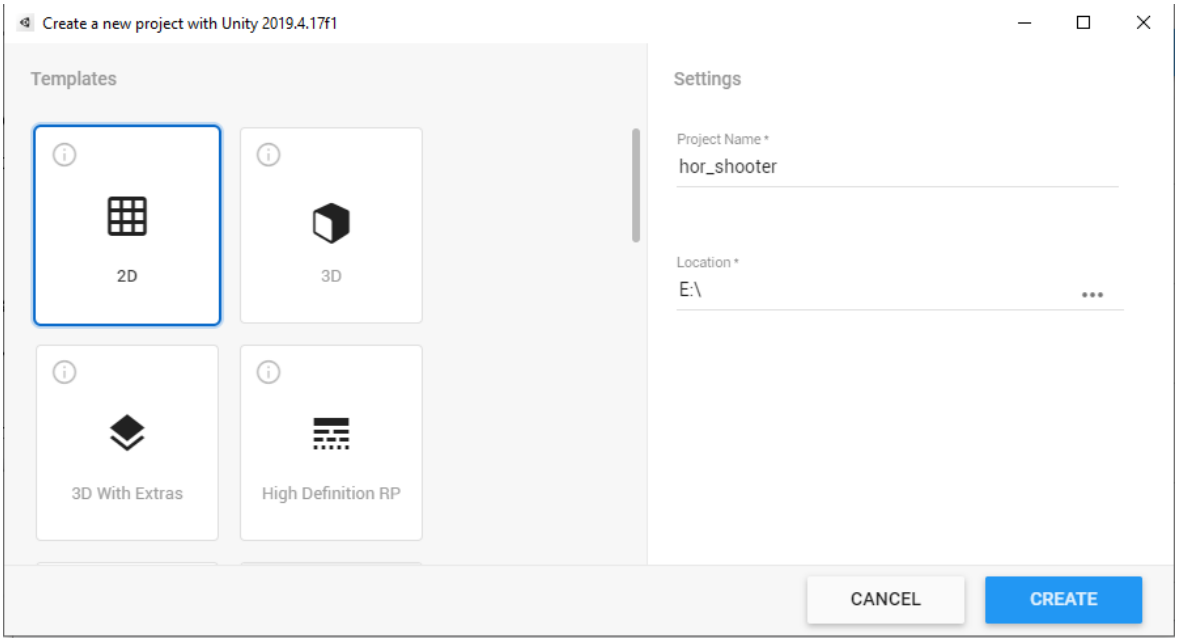


Рисунок 1.15. Вікно створення проекту у UnityHub

Після натискання на кнопку New проект буде створено автоматично, разом з усіма необхідними файлами, директоріями та папками. 2D-проект використано не тільки через тему, але й тому, що такі проекти потребують менше ресурсів для розробки, а також більш прості у розумінні для гравця.

Отже виконавши створення нового проекту буде автоматично згенерована пуста сцена проекту, із стандартним набором об’єктів – камери гравця та джерела світла, що можна побачити на рисунку 1.16. Ці об’єкти вже

дають можливість запустити проект, але гра покаже лише пустий простій, адже «дивитись» на гру ми будемо через камеру, а джерело світла не має «тіла».

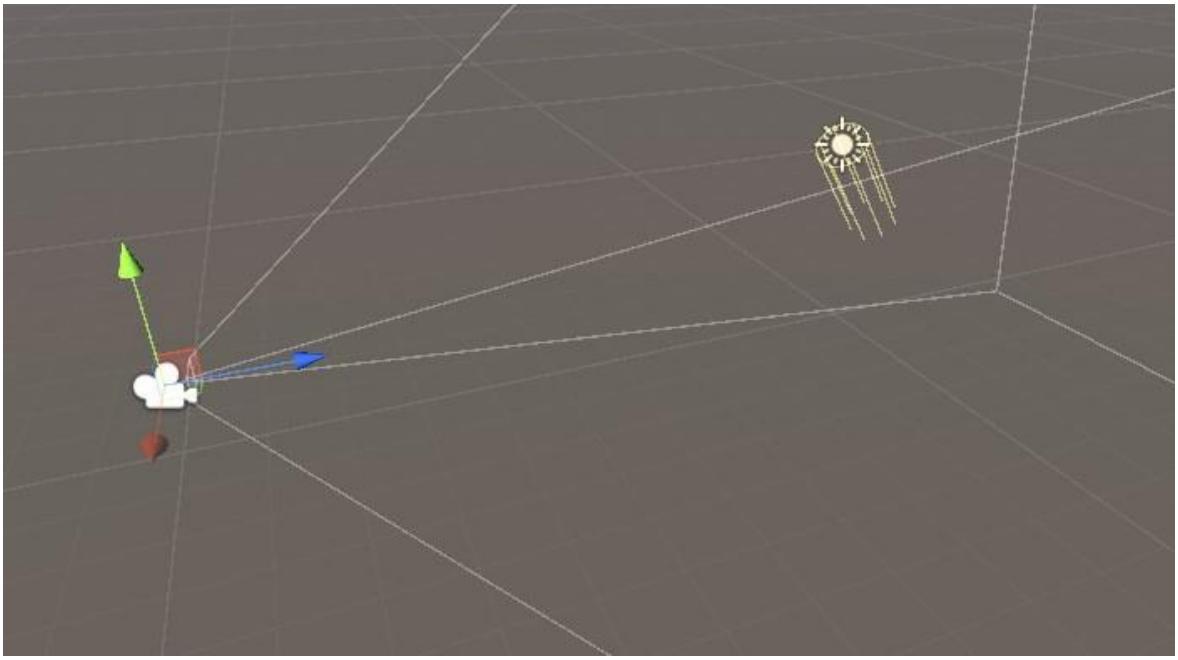


Рисунок 1.16. Стартові об’єкти в новому проекті Unity

Наступним кроком, є розміщення ігрових об’єктів. Для початку потрібно реалізувати задній фон гри. Оскільки гра виконується у космічному просторі, я використав відкриті для вільного користування фото зоряного неба. Після деяких змін у редакторі графіки зображення можна використовувати для гри. Щоби створити більшу динаміку кадра, я буду використовувати два таких зображення зоряного неба та рухати їх на фоні із невеликою швидкістю, що буде створювати почуття руху у космічному просторі. Такий рух буде створюватись за рахунок написання коду, який буде переміщувати ігрові об’єкти спрайтів зоряного фону у постійному напрямку та швидкості. Безперервність переміщення реалізовується за рахунок переміщення зображень фону на попереднє місце старту після надходження до краю екрану гравця. На рисунку 1.17 можна побачити ігрову сцену із розміщеним зображенням зоряного неба:

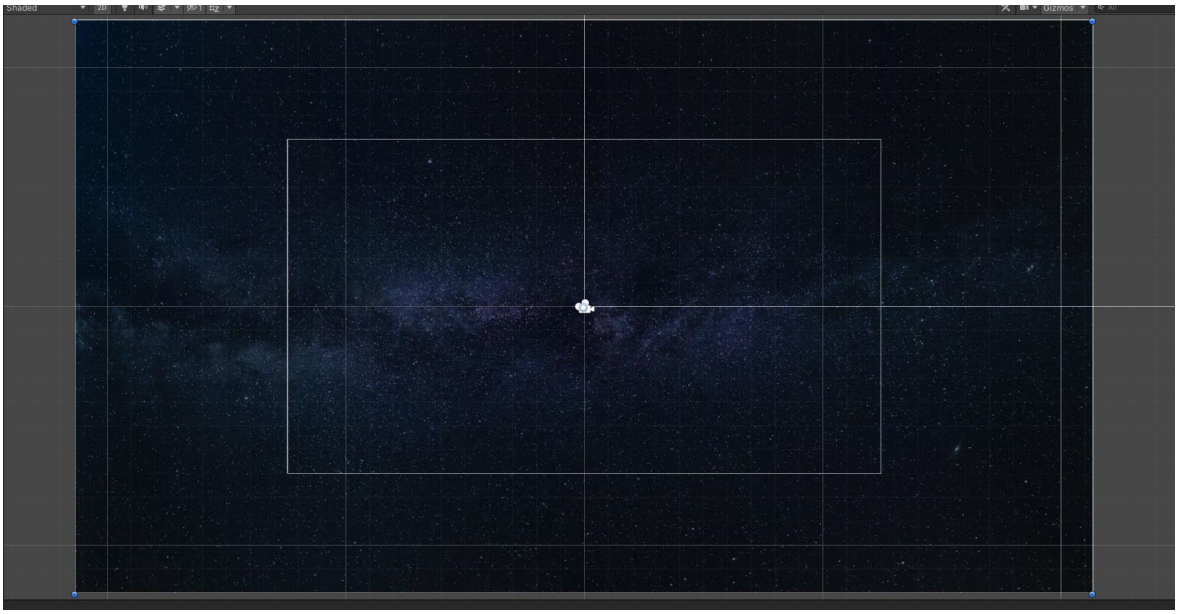


Рисунок 1.17. Ігрова сцена із розміщеним зображенням зоряного неба

Далі потрібно розмістити ігрові об’єкти гравця та ворогів. Для початку реалізуємо кораблі ворогів, а до астероїдів ми використаємо шаблон коду корабля ворога, лише дещо його змінимо. Для створення спрайтів кораблів я також використав зображення що знаходяться у вільному доступі. Після їх редагування та налаштування орієнтації я розмістив їх на сцені. Також було створено спрайти для лазерів гравця та ворога. На рисунку 1.18 можна побачити ігрові об’єкти гравця та ворога.

Рисунок 1.18. Ігрові об’єкти гравця та ворога на ігровій сцені

Після розміщення основних об’єктів на сцені, потрібно також створити ігровий інтерфейс. Це робиться через створення «полотна». Полотно є новою

системою інтерфейсів для Unity, яка дає змогу працювати із ним, як з ігровим об’єктом на сцені. Для створення полотна Canvas потрібно створити його через інтерфейс ієрархії ігрових об’єктів, або через кнопку створити у верхньому меню редактору. Далі, в разі якщо потрібно додати новий елемент інтерфейсу, його потрібно створювати, як нащадок об’єкта полотна. Цікаве те, що полотно ніби є на сцені, але його не можна побачити через екран просмотру сцени, він буде видимий лише для камери. Кожний новий елемент буде з’являтись на сцені, після чого із ним можна буде працювати, переміщуючи по полотну, деформуючи, розгортаючи у необхідну сторону. Після створення полотна інтерфейсу та розміщення на ньому відповідних елементів було створено Ігровий інтерфейс, що можна побачити на рисунку 1.19:

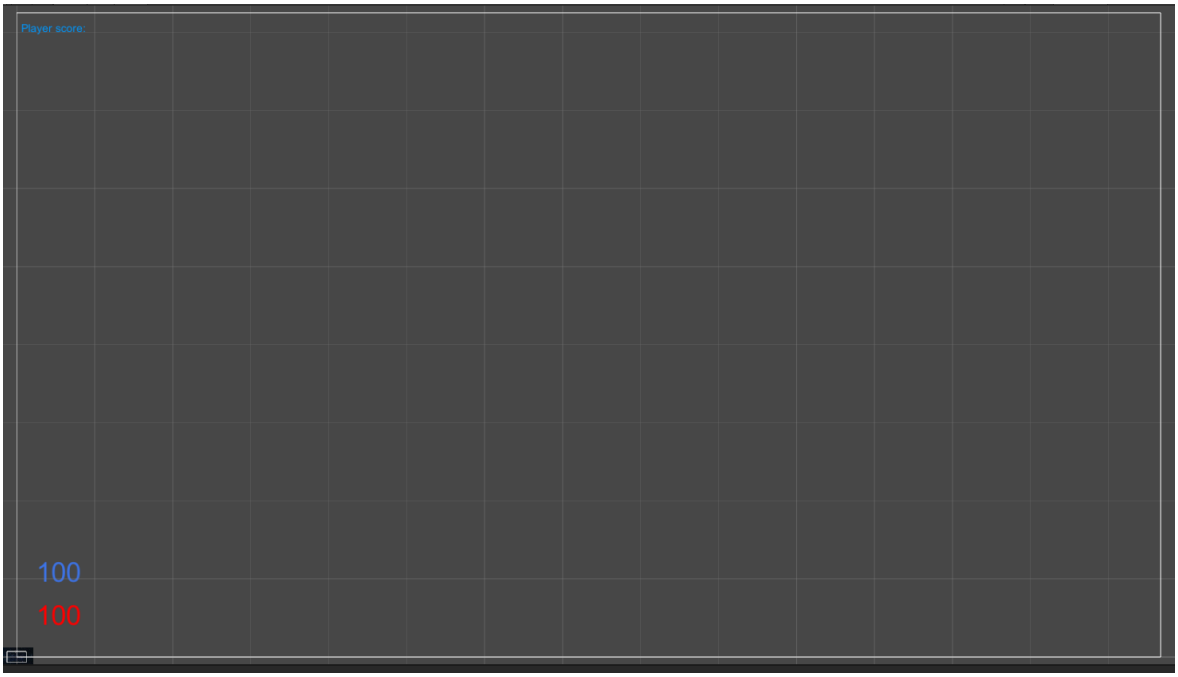


Рисунок 1.19. Ігровий інтерфейс гри

Як було зазначено в концепт-документі інтерфейс реалізовано із використанням текстових полів. Усього на інтерфейсі розміщено три елементи, кожний з котрих буде відповідати за параметри здоров’я, щитів та ігрового рахунку. Хоча це не було прописано у концепт-документі, але максимальної кількості здоров’я та щитів немає, тому гравець може накопичувати ці значення по ходу ігрового процесу.

Окремо слід визначити порожній ігровий об’єкт з іменем GameManager. В

ньому будуть знаходитись файли коду для керування ігровим процесом на сцені. Такий підхід потрібно використовувати через те, що написаний у Unity код працює лише тоді, коли він доданий до якогось ігрового об’єкту. Якщо уважно придивитись до імені початкового файлу класу скриптів, то можна побачити там MonoBehaviour, що можна перевести як самоповедінка. Тобто мається на увазі, що кожний такий скріп буде відповідати за поведінку об’єкта на сцені всеціло, або лише частково. Тому такі скрипти, які виконують керуючу, або допоміжну роль та мають на увазі повний цикл життя від початку гри до її кінця, мають бути додані до подібних пустих ігрових об’єктів. Такі об’єкти не мають фізичного тіла або інших компонентів, які могли б зробити їх повноцінними акторами ігрової сцени.

Також, окрім гравця, ігрового менеджеру, двох ігрових об’єктів фону, а також Canvas-у слід додати ще ігрові об’єкти для створення границь ігрового простору зліва, зверху та знизу, для того, щоби гравець не вилітав за границі ігрового кадру. Повний перелік створених ігрових об’єктів можна побачити на рисунку 1.20:

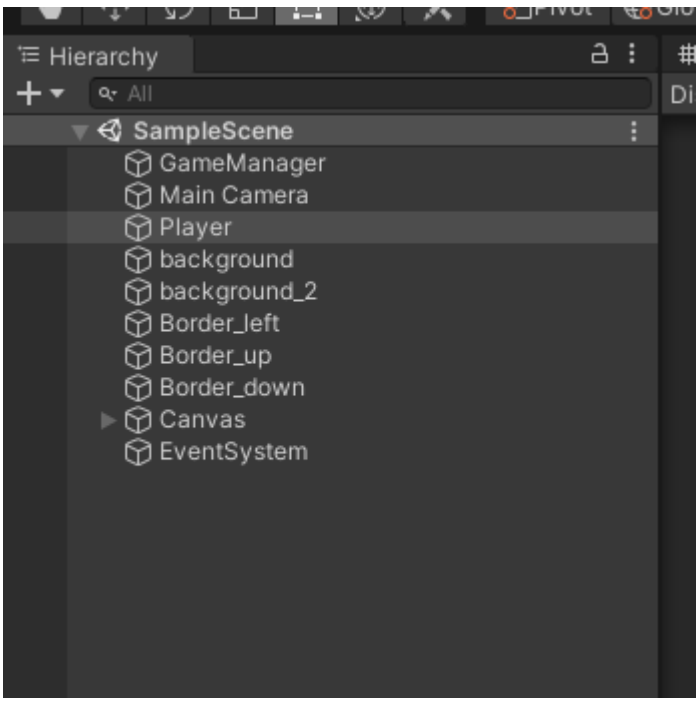


Рисунок 1.20. Перелік створених ігрових об’єктів для гри

На даному етапі вже можна запустити сцену та переглянути отриманий результат. Необхідно це зробити для того, щоби переглянути чи правильно

виставлені всі ігрові об’єкти які було створено для гри. Перевірити положення камери, її дистанцію та орієнтацію у просторі.Це можливо завдяки функціоналу виконання сцен у будь-який час розробки прямо з редактору ігрового програмного рушія Unity. При натисканні на відповідну кнопку сцена запускається та відтворює свій зміст, в залежності від того, які властивості у об’єктів та які скрипти до них додані. Кожний елемент буде виконуватись відповідно до свого наповнення компонентами та коду. На рисунку 1.21 можна побачити поточний результат виконання гри:

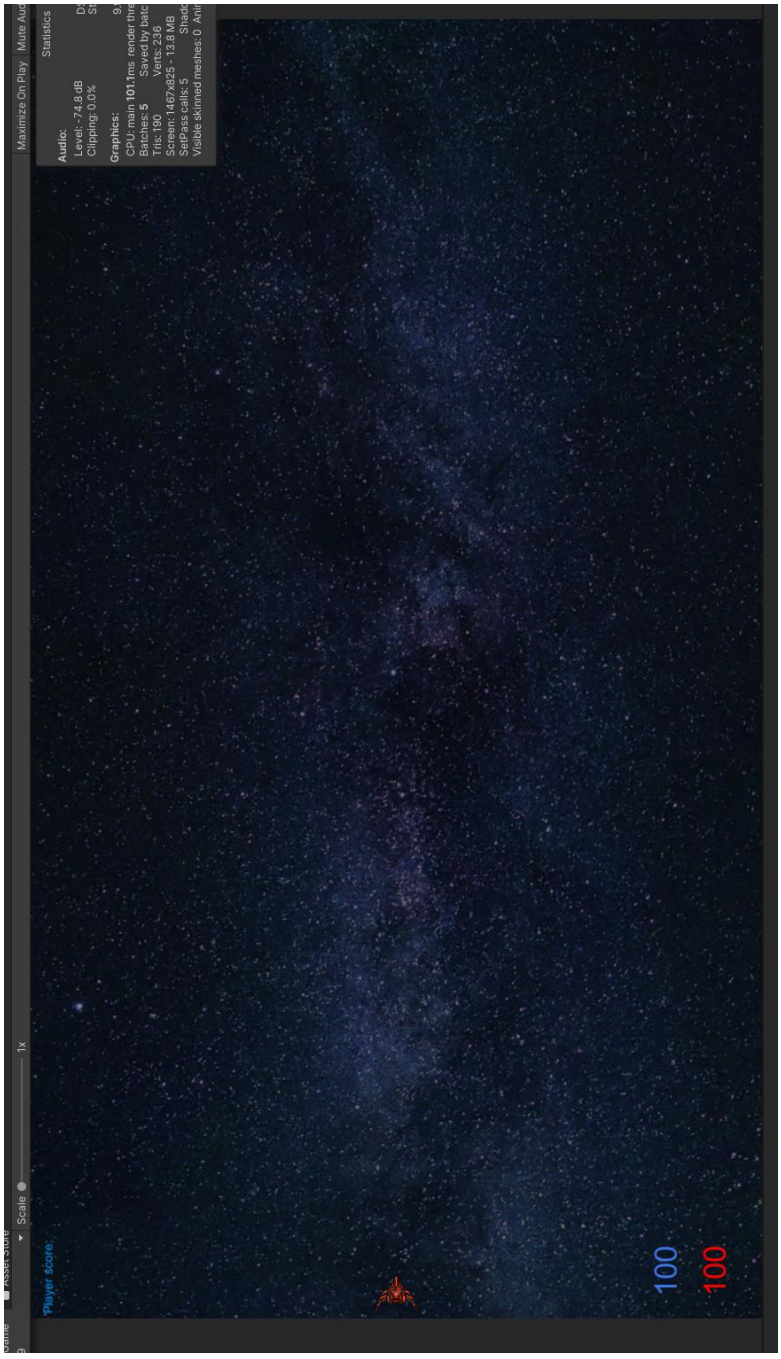


Рисунок 1.20. Загальний вигляд гри без управляючих скриптів.

Підготовка ігрових об’єктів закінчена тепер потрібно перейти до реалізації програмного коду гри. В першу чергу потрібно написати код для зоряного фону, адже зараз він не рухається, а просто статично розміщений на позаду гравця. У ассетах створюємо відповідну папку та скрипт.

Він буде працювати дуже просто. Скрипт буде отримувати два файли зоряного фону, та послідовно переміщувати їх вліво, симулюючи рух корабля у просторі. Для цього буде достатньо знати координати границі ігрового полю та виконувати переміщення зображень – синхронно – кожний ігровий кадр. Нижче зображено фрагмент коду, який виконує переміщення зображень на фоні:

```
[SerializeField] GameObject background_1;
[SerializeField] GameObject background_2;
Vector3 bg1_coord;
Vector3 bg2_coord;

void Start()
{
	bg1_coord = new Vector3(8.8f, 0f, 0f);
	bg2_coord = new Vector3(51.4f, 0f, 0f);
}

void FixedUpdate()
{
	background_1.transform.position = bg1_coord;
	background_2.transform.position = bg2_coord;
	bg1_coord.x -= 0.01f;
	bg2_coord.x -= 0.01f; ;
	if (bg1_coord.x <= -33.8f)
		bg1_coord.x = 51.4f;
	if (bg2_coord.x <= -33.8f)
		bg2_coord.x = 51.4f;
}
```

Розглянемо цей код більш детально, та конкретно деякі команди та прийоми, які будуть і далі використовуватись. По-перше, це так звані серіалізовані поля, які дають змогу робити змінні із закритим доступом для програми, видимими для розробника у панелі компонентів. Це дуже зручно, коли потрібно отримати доступ до якогось елементу гри, але програмний доступ відкрити не можна, або це буде дуже складно.

По-друге, типи змін, які можуть бути не зовсім типами, а варіантами об’єктів або конструктів, які отримують посилання на щось, але вони не являють собою саме чимось. Наприклад створення змінної із «типом» GameObject не створює ігровий об’єкт, а лише створює змінну, з іменем background_1, яка буде мати посилання на той ігровий об’єкт що розробник приєднає до цього поля.

По-третє, потрібно розуміти, що всі об’єкти на сцені мають свої координати. І навіть не дивлячись на те, що гра створена у 2D-просторі, але об’єкти все одно мають координату Z, хоча вона ні на що не впливає, окрім порядку відрисовки на екрані. Тому для кожного зображення зоряного фону було створено свою змінну типу Vector3 для збереження та зміни координат.

Також потрібно відмітити такі методи як Start() та Update(). Обидва методи є стандартними методами для скриптів Unity, тому створюються кожний раз разом із скриптом. Перший метод виконується в той момент, коли створюється ігровий об’єкт і скрипт починає працювати. Це не самий початок існування сцени, а лише початок існування об’єкту. Другий метод відтворює свій зміст кожний кадр, який може відтворити робоча машина. Якщо буде відтворено 50000 кадрів, то зміст цього методу буде виконано 50000 разів. У цього метода є його заміна FixedUpdate() який намагається виконатись фіксовану кількість разів – 60. Ці два методи потрібно використовувати уважно, оскільки вони впливають на ігровий процес корінним чином. Наприклад рух об’єктів у Update() буде виконуватись значно швидше, ніж у FixedUpdate() та об’єкт буде проходити далі за секунду часу. На рисунку 1.21 зображено різницю між цими методами.

Ми можемо побачити з цього коду, що у методі Start() було встановлено початкові координати для наших фонів. Після цього, саме у FixedUpdate() я виконую переміщення фону для того, щоби він рухався більш плавно. Тому, спочатку я передаю до кожного ігрового об’єкту фону координату з векторів позицій. Потів змінюю координату X цих векторів на -0.1f та проводжу перевірку того, чи дійшли об’єкти фону до визначених координати X краю кадру. Після чого, у разі досягнення такого краю координати встановлюються на початкові координати по X-у.

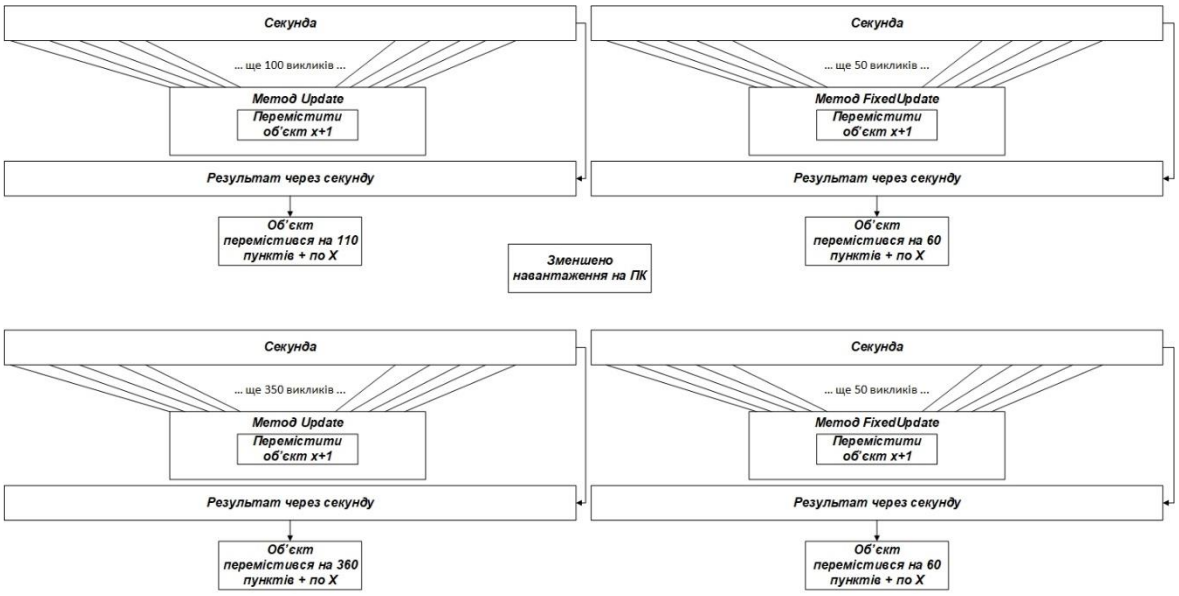


Рисунок 1.21. Схематичне зображення різниці у роботі методів Update() та FixedUpdate()

Наступним ігровим елементом для реалізації є переміщення гравця. Для цього використовую об’єкт для зчитування натискання на кнопки як Input. Він постійно «слухає» статус всіх кнопок управління, які тільки може зчитувати ігровий програмний рушій Unity. І який би не був їх статус, завжди можна звернутись до цього об’єкту та отримати значення натискання на кнопку, тригер, клавішу або стік. Для управління переміщення корабля я буду використовувати метод GetAxis. Цей метод зчитує так звані осі управління, які задаються у файлі параметрів проекту. Кожна з таких осей має своє ім’я, тому щоби вказати яку саме ось ми хочемо зчитати, потрібно вказати її ім’я у форматі змінної строкового типу string. Цей метод завжди буде повертати значення, не зважаючи на те, чи була натиснута кнопка, чи ні, відрізняється лише значення, яке цей метод поверне. Якщо гравець не використовує ось до якої буде виконано звернення, то метод поверне значення «0». Також, для реалізації руху я буду використовувати компонент твердого тіла RigidBody, який додаю до корабля гравця. У цього компоненту є властивість velocity, яка виконує рух твердого тіла у просторі у визначеному векторі руху. Нижче приведено код, який реалізує рух гравця:

```
void Update()
{
```

```
        horizontal_input = Input.GetAxis("Horizontal");
        vertical_input = Input.GetAxis("Vertical");
    }
    private void FixedUpdate()
    {
        _player_rigidbody.velocity = new Vector3(horizontal_input *
    player_speed, vertical_input * player_speed, 0f);
    }
```

Тут я використовую два Updat-а. Один для оперативного зчитування на натискання клавіш управління, а другий для руху корабля гравця за допомогою твердого тіла. У методі Update() можна побачити використання об’єкту Input, який отримує значення сили натискання на осі управління, що потім передаються до вектору руху твердого тіла.

Також, для повної реалізації функціоналу гравця потрібно створити код для параметрів гравця. Він буде зберігати значення рівня ушкоджень корабля, щитів та ігрового рахунку гравця. Це будуть прості цільночисельні змінні. Для подальшої реалізації цього коду також потрібно створити код для Менеджера інтерфейсу, оскільки саме код параметрів гравця звертається до інтерфейсу. Цей код буде отримувати посилання на текстові поля, та змінювати їх зміст у разі виклику відповідного методу. Нижче приведено код цього скрипта:

```
[SerializeField] Text player_hp_text;
[SerializeField] Text player_shield_text;
[SerializeField] Text player_score_text;

public void change_player_hp(int value)
{
    player_hp_text.text = "Ship hull: " + value;
}

public void change_player_shield(int value)
{
    player_shield_text.text = "Ship shield: " + value;
}

public void change_player_score(int value)
{
    player_score_text.text = "Player score: " + value;
}
```

Як можна побачити для кожного елемента інтерфейсу є метод для зміни

його змісту. Повертаючись до коду параметрів гравця, можна повноцінно створити методи для отримання пошкоджень, підвіщення значення здоров'я корабля, а також підвищення ігрового рахунку. Всі ці методи будуть звертатись до Менеджеру Інтерфейсу, оскільки вони змінюють значення ігрових ресурсів гравця. Нижче приведено код цих методів:

```
public void take_damage(int damage_value)
{
        player_hp -= damage_value;
        _ui_manager.change_player_hp(player_hp);
        if (player_hp <= 0)
        {
                Destroy(this.gameObject);
        }
}

public void take_hp_point(int hp_point_value)
{
        player_hp += hp_point_value;
        _ui_manager.change_player_hp(player_hp);
}

public void increase_player_score(int value)
{
        player_score += value;
        _ui_manager.change_player_score(player_score);
}
```

Для закінчення реалізації коду гравця, потрібно дати йому можливість робити постріли. Це відразу зачіпає два скрипта – логіка лазеру гравця та управління гравця. Управління буде змінено, оскільки потрібно буде реалізувати натискання на клавішу пострілу. Почну з реалізації коду логіки лазеру гравця. Для лазеру потрібно створити лише змінну швидкості лазеру та ігровий об'єкт лазеру. Цей ігровий об'єкт лазеру потрібно зробити префабом. Префаб – це ігровий об'єкт, який знаходиться в переліку ассетів проекту. За потреби такий префаб може буте створений безліч разів і кожний такий екземпляр буде існувати незалежно один від одного. Додавши до ігрового об'єкту лазеру компонент колайдеру, я переключив цей компонент у режим тригеру. Сам по собі колайдер працює, як фізична границя для об'єкту в якої є тверде тіло, але

якщо переключити режим у тригер, тоді цей компонент починає зчитувати контакт із другими колайдерами та викликати відповідні події у коді свого ігрового об’єкту. На рисунку 1.22 зображено типовий зміст префабу ігрового об’єкту лазеру як для гравця, так і для ворога. Єдине, що їх відрізняє – колір лазеру.

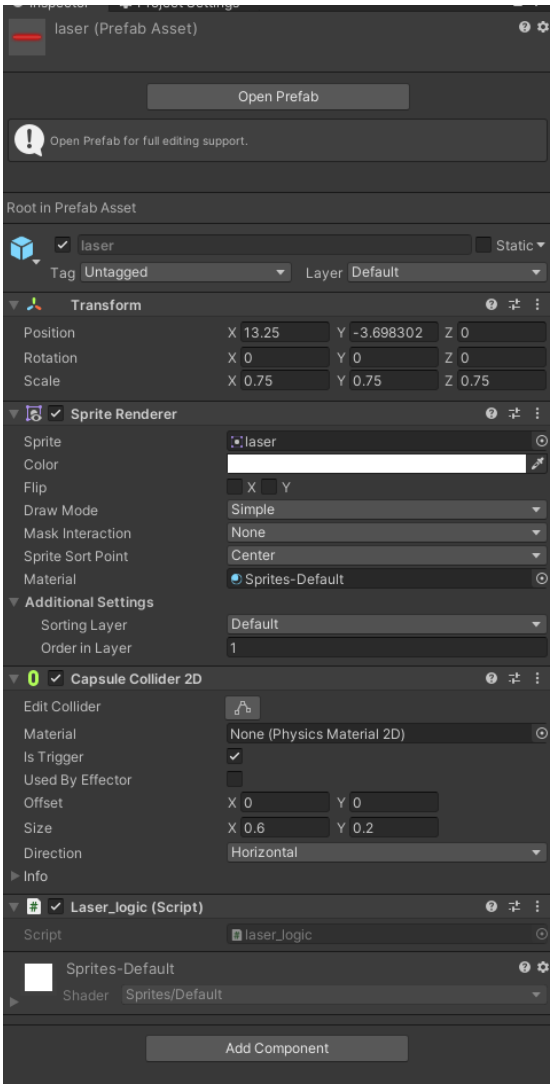


Рисунок 1.22. Типовий зміст префабу ігрового об’єкту лазеру для гравця та ворогів

З точки зору коду логіка лазеру виконується дуже просто. Визначивши швидкість для лазеру, єдине, що потрібно зробити це реалізувати рух лазеру у ігровій сцені строго вправо – для гравця, або вліво – для ворога. В кінці цього руху, у випадку коли координата X позиції лазеру буде більшою, або меншою за 16, знищити ігровий об’єкт лазеру. 16 та -16 це координати зліва та справа від ігрового кадру. Це потрібно робити, щоби створені при пострілах лазери не

перевантажували систему гравця та автоматично самознищувались, коли виходять за границі ігрового кадру. Також необхідно написати код події тригеру при контакті колайдеру лазера з іншим колайдером. В середині цієї події потрібно спробувати отримати компонент ворожої логіки у об'єкту з яким був контакт. Якщо це вдалось зробити, тоді нанести пошкодження для цього екземпляру ворогу. Нижче приведено код логіки лазеру гравця:

```
float laser_speed = 0.2f;

private void FixedUpdate()
{
    transform.Translate(laser_speed, 0f, 0f, Space.Self);
    if(transform.position.x > 16f)
    {
        Destroy(this.gameObject);
    }
}

private void OnTriggerEnter2D(Collider2D collision)
{
    enemy_logic _enemy_logic;
    _enemy_logic = collision.GetComponent<enemy_logic>();
    if(_enemy_logic)
    {
        _enemy_logic.take_damage(45);
        Destroy(this.gameObject);
    }
}
```

Для коду управління гравцем у частині, де я отримую натискання на клавіші управління по осям вертикалі та горизонталі, також написав код для отримання затискання клавіші Space на клавіатурі, після чого викликається код генерації лазеру. Для того, щоби лазери не генерувались постійним потоком та створювалось відчуття скорострільності, я використав сопроцесори у методі генерації ігрового об'єкту лазеру. Нижче приведено цей код, який викликається при натисканні на Space:

```
private IEnumerator laser_shoot(){
    if(can_shoot){
        can_shoot = false;
        Vector3 laser_ini_position = new Vector3(transform.position.x +
```

```
            0.8f, transform.position.y, transform.position.z);
                Quaternion laser_rotation_ini = new Quaternion(0f, 0f, 0f, 0f);
                laser_go = Instantiate(laser_prefab, laser_ini_position,
            laser_rotation_ini) as GameObject;
                yield return new WaitForSeconds(0.2f);
                can_shoot = true;
            }
        }
```

Методи сопросецiв, або IEnumerator працюють з командою yield return та імітують паралельне виконання коду, хоча це не зовсім так. Принци роботи цього методу полягає в тому, що я перевіряю, чи можу виконувати постріл, якщо так, то на весь час виконання цього методу, перемикаю бульову зміну можливості ведення пострілу у значення false. Після цього, я отримую позицію для лазеру, яка дорівнює позиції гравця, але трохи правіше, по координаті X. Визначаю кут повороту лазеру. Наступним кроком є використання команди Instantiate, яка створює ігровий об’єкт на сцені у заданих координатах із заданим кутом повороту. І в саме цей момент я викликаю команду yield return та даю команду чекати 0.2 секунди і тільки після того, як цей час пройде перемикаю бульове значення можливості вести постріл у значення true.

Наступним кодом для реалізації буде код генерації ворогів та об’єктів для підбору. Ці скрипти дуже схожі, єдине, що їх відрізняє це об’єкти для генерації та алгоритм частоти генерації. Для генерації ворогів спочатку створюються префаби ворогів. Далі в самому скрипті генератора я створюю змінну, яка зберігає кількість ворогів для генерації, а також булеву змінну яка регулює, чи може генератор створювати ворогів. Кожний ігровий кадр викликається сопроцес генерації ворогів. Він перевіряє, чи може цей генератор створювати ворогів. Якщо так, то булева змінна перемикається у стан false, після чого викликається цикл for що створює кораблі ворогів у випадкових позиціях за межами ігрового кадру у випадкових позиціях по координаті Y. Після того, як цей цикл виконує свій зміст, сопроцес чекає 1.5 секунди та виставляє булеву зміну у true та генерує випадкове значення для змінної кількості кораблів для генерації від 1 до 6. Нижче приведено фрагмент коду генератору ворогів:

```
protected IEnumerator ship_spawner()
```

```
{
        if(can_spawn)
        {
                can_spawn = false;
                for(int i=0; i<enemy_ship_spawn_count; ++i)
                {
                        Vector3 new_enemy_spawn_pos = new Vector3(13.25f,
                Random.Range(-6.4f, 6.4f), 0);
                        enemy_ship_go = Instantiate(enemy_prefab,
                new_enemy_spawn_pos, enemy_prefab.transform.rotation) as
                GameObject;
                }
                yield return new WaitForSeconds(1.5f);
                can_spawn = true;
                enemy_ship_spawn_count = Random.Range(1, 6);
        }
}
```

Код генератору об’єктів такий самий, як і генератор ворогів, лише змінюється випадкова кількість об’єктів для генерації від 0 до 2 та самі об’єкти для генерації.

Наступним кроком потрібно реалізувати код логіки ворогів. Наприклад кораблів ворогів. У здебільшому цей код є компіляцією коду управління гравця та його коду параметрів, але в одному місці. Рух ворога запрограмований у методі FixedUpdate та виконується постійно на задану величину швидкості корабля ворога. Якщо позиція ворога по координаті X менша за -13, то ігровий об’єкт ворога знищується.

Також у даному коді логіки ворога написано код для отримання пошкоджень від атак лазером гравця. Цей метод віднімає кількість здоров’я корабля ворогу. Якщо кількість цього параметру дорівнює 0, або менше, то знаходиться ігровий об’єкт з іменем Player, з цього об’єкту отримується доступ до його компоненту параметрів гравця та додаються 100 очок до ігрового рахунку.

Постріли ворогом реалізовані у тому ж вигляді, як і постріли гравця, лише різниця в тому, що ворог веде постріли постійно, без натискання кнопки Space, тобто сопроцес пострілу викликається відразу після руху ворогу. В самому сопроцесі вказано час затримки 4 секунди, для того, щоби гравець міг

маневрувати між всіма пострілами ворогів.

Нарешті, останній код для реалізації, це код поведінки предметів для підбору. Цей код буде розглянуто на прикладі блоку для здоров'я гравця. Весь код зводиться до постійного переміщення, як ворог, вліво по координаті X. Якщо тригер ігрового об'єкту для підбору вступає у контакт із компонентом колайдера гравця, то отримується посилання на компонент параметрів гравця і в цей компонент передається зміна кількості здоров'я гравця на 50 пунктів. Після цього ігровий об'єкт для підбору самознищюється. Код представлено нижче:

```
private void FixedUpdate(){
        transform.Translate(-0.015f, 0f, 0f, Space.Self);
}

private void OnTriggerEnter2D(Collider2D collision){
        player_parameters _player_parameters;
        _player_parameters = collision.GetComponent<player_parameters>();
        if (_player_parameters)
        {
                _player_parameters.take_hp_point(50);
                Destroy(this.gameObject);
        }
}
```

Після реалізації всіх основних кодів та спрайтів гри, проект отримав свою кінцеву структуру, яку можна побачити на рисунку 1.23:

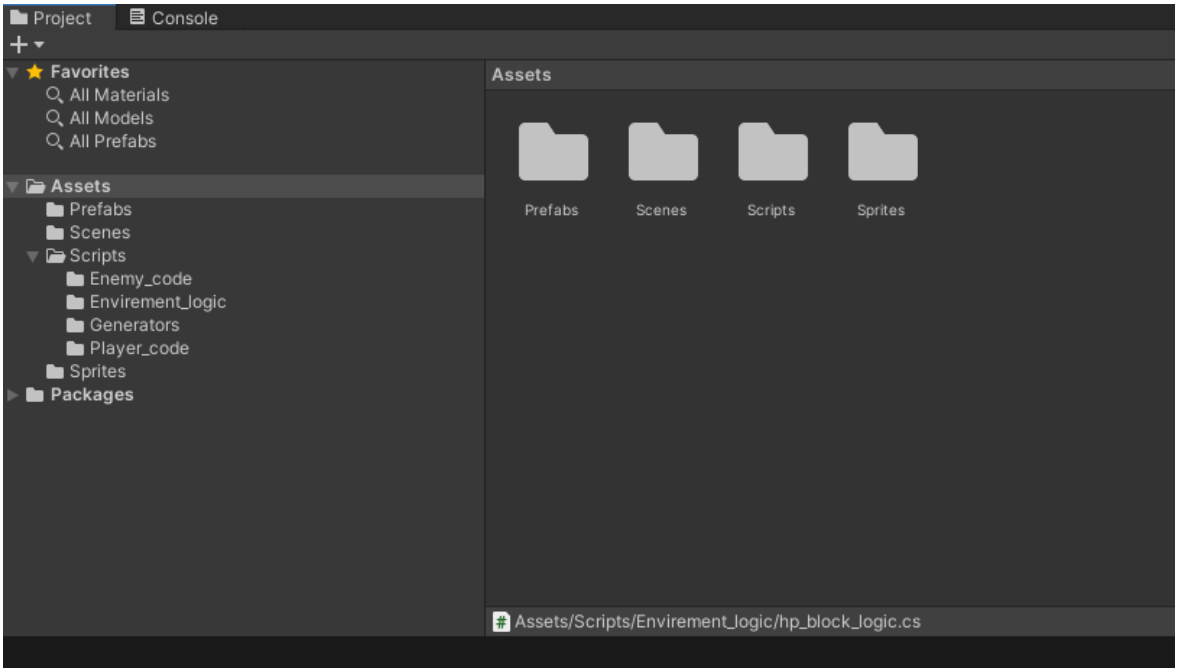


Рисунок 1.23. Кінцева структура проекту гри

Як можна бачити всі основні файли, з якими проводилась робота розміщені у підкаталогах. Також до складу проекту входять файли ігрового програмного рушія Unity. Всі файли проекту було розподіллено по підкаталогам розділу проекту Assets, розміщення котрих можна побачити на рисунку 1.24.

Дана реалізація проекту дає змогу зіграти у горизонтальний скролл-шутер, переміщуватись по ігровому простору, взаємодіяти з ворогами, підбирати бонуси та отримувати ігровий рахунок за знищення об’єктів ворогів та астероїдів.

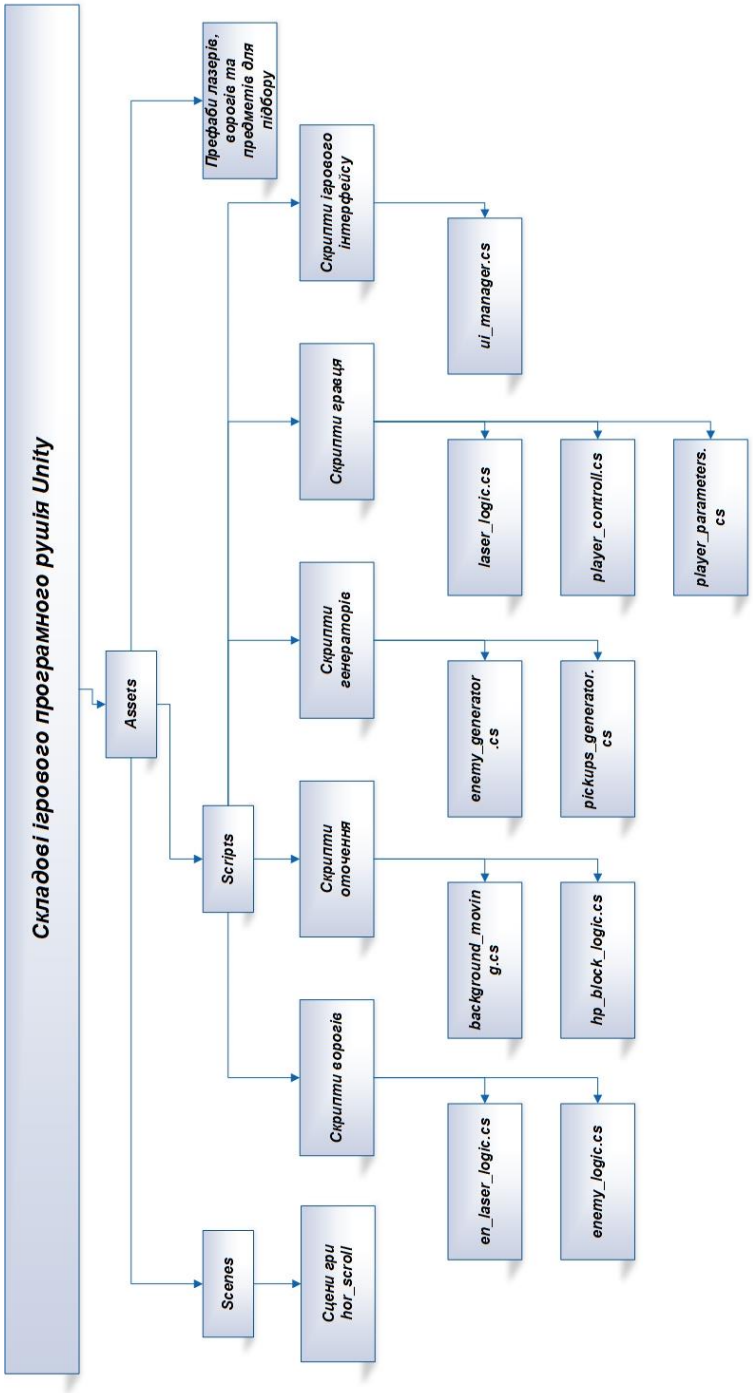


Рисунок 1.24. Розміщення файлів проекту по підкаталогам розділу Assets

1.7 Тестування працездатності ігрових елементів

Тестування та відладка є важливою частиною розробки будь-якого програмного продукту. Для ігрових проектів виконання тестування є ще більш важливим процесом, оскільки він дозволяє знаходити помилки наявність котрих не є очевидною. Треба розуміти, що у ігрових проектах помилки можуть призводити до візуальних або внутрішніх проблем. Перші знаходяться дуже легко, оскільки візуальні артефакти, або помилки у розмішені об’єктів, дуже швидко знаходяться під час виконання ігрового процесу. Інша річ програмні помилки, які неможливо візуально оприділити під час виконання гри. Іноді такі помилки призводять до часткового порушення ігрового процесу, а в більш рідких випадках до повної неможливості пройти гру, адже можуть зачепляти механіки, які пов'язані з переходом до іншого рівня, або запуском ключового скрипту у сюжетній послідовності.

У випадку з реалізацією ігор на ігровому програмному рушії Unity, цей процес виконується безпосередньо в самому редакторі. Для тестування та відладки гри під час розробки, потрібно встановлювати ігрові об’єкти та сцени у такий стан, який потрібно протестувати та натиснути кнопку «Play». Таким же чином виконується тестування і відладка після закінчення розробки, достатньо виставити гру у її початковий стан и натиснути ту ж саму кнопку. На рисунку 1.25 можна побачити процес тестування гри.

Як і було сказано, основною метою тестування було проведення перевірки готовності поточного «білду» проекту. Він дає змогу отримати ігровий досвід горизонтального скролл-шутеру. Було перевірено взаємодію гравця із ворогами, із їх пострілами, справність отримання пошкоджень. Взаємодію гравця із об’єктами, які підбираються, а також роботу інтерфейсу. Помилок у роботі основних ігрових механік виявлено не було. Було проведено балансування ігрового процесу у сенсі частоти пострілів гравця та ворогів, швидкості переміщення об’єктів по сцені.

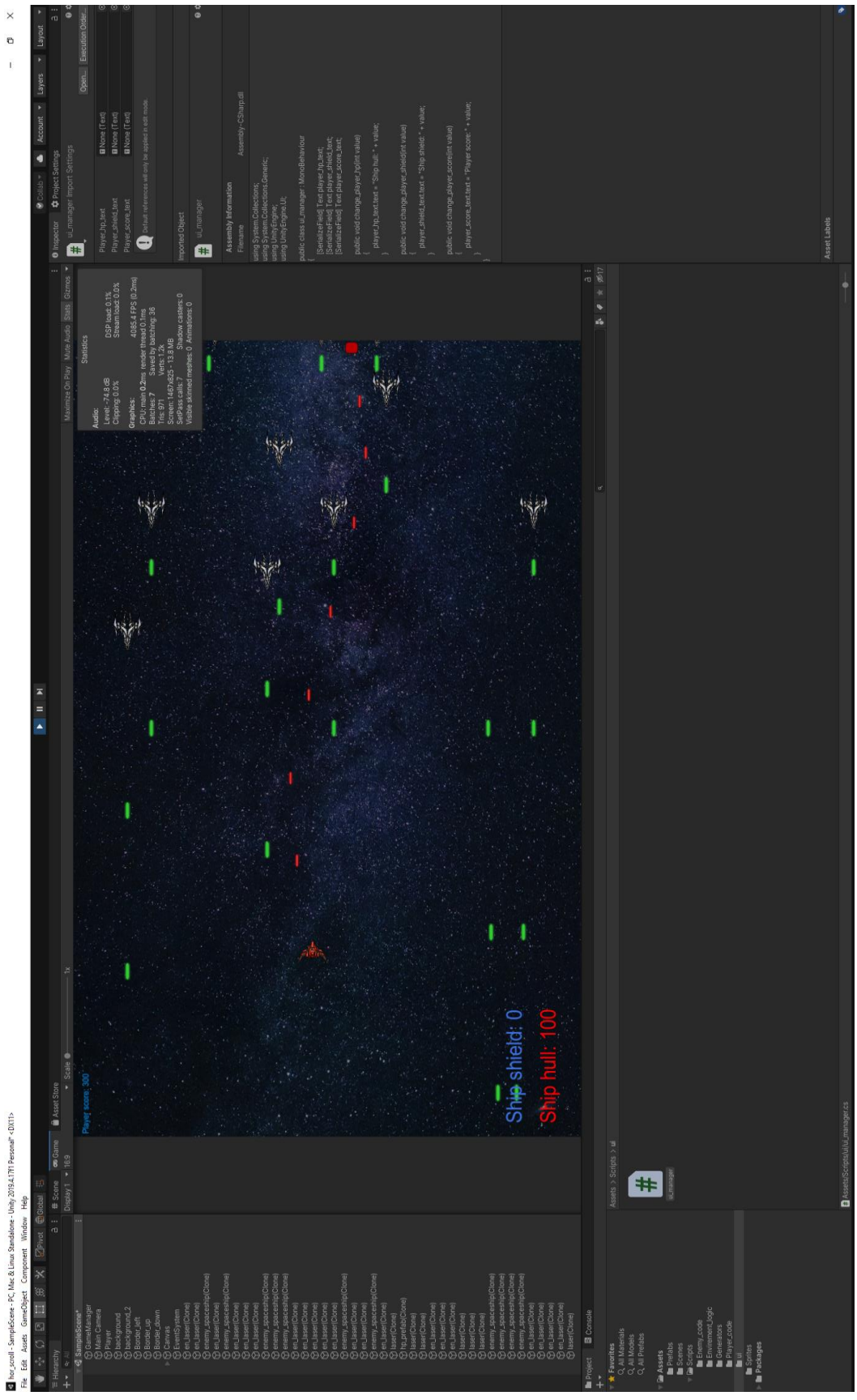


Рисунок 1.25. Процес тестування гри

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

В даному дипломному проекті розроблено та реалізовано 2D-гру у жанрі горизонтального скролл-шутеру на програмному рушії Unity. Ефективність кожного програмного продукту визначається його якістю та ефективністю процесу розробки. Якість ПП визначається наступними складовими: з точки зору користувача; з позиції використання ресурсів; виконання вимог до програмного забезпечення.

Проведемо розрахунки визначення трудомісткості розробки даного програмного продукту.

2.2 Розрахунок ціни програмного продукту нормативним методом

2.2.1 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку.

У таблиці 2.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт.

Таблиця 2.1. Каталог аналогів

Найменування ПП	Обсяг функції ПП – V_o, усл. машинних командах.
1. ПП СУБД	2500 – 9800
2. Комплексні системи ведення БД	950 – 7430
3. ПП введення інформації	1060 – 5750
4. ПП оптимізації розрахунків	1300 – 4200
5. ПП автоматизації засобів по каталогу	680 – 7000
6. ПП автоматизованих розрахунків	1300 – 8600
7. ПП загальної математики і ПП імітаційного моделювання	7800 – 8800
8. ПП організації обчислювального процесу	13000 – 10200

Для нашого варіанта виділено сірим кольором.

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

Таблиця 2.2. Трудомісткість

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
1.00	229
2.00	244
3.00	262
4.00	283

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): Тар = 229 х 0,8 = 183,2 (люд/годин).

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

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

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

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

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 2.2.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.3.);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.4.).

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

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

Для нашого варіанта виділено сірим кольором.

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

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

Для нашого варіанта виділено сірим кольором.

Тому що розробка системи є ПП, що має аналоги програмних продуктів, то код ступеня новизни для мого ПП – В, а значення коефіцієнта $К_н$=0,7. По таблиці 2.5, знаючи код ступеня новизни, тепер можна визначити значення питомих коефіцієнтів трудомісткості:L_1=0,12;L_2=0,11;L_3=0,61;

Таблиця 2.5. Значення коефіцієнта ступеня використання в розробці типових програм

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

Для нашого варіанта виділено сірим кольором.

У розробленому програмному продукті використовується від 40 до 60 відсотків існуючих функцій, це значить, що $К_т$=0,7.

Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$Т_{тз}=Та*L_1*К_н=183,2 *0,12*0,7= 15,39 \text{ (люд/годин)}$$

Трудомісткість розробки технічного проекту

$$Т_{тп}=Та*L_2*К_н =183.2 *0,11*0,7 = 17,42 \text{ (люд/годин)}$$

Трудомісткість розробки робочого проекту

$$Т_{рп}=Та*L_3*К_н*К_т=183,2 *0,61*0,7*0,7=54,76 \text{ (люд/годин)}$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: - технічне завдання $N_{тз}$=2 (стр), - розробка ТП $N_{тп}$=28(стр), - розробка робочого проекту $N_{рп}$=37 (стр), - пояснювальна записка відповідно $N_{пз}$=30 (стр). Розрахунок зведений у таблицю 2.6

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{Pтз}$=15,39	$T_{кк}$=0,7*$N_{тз}$= 0,7*2=1,4	$T_{нк}$=0,15*$N_{тз}$=0,15*2=0,30
2.Розробка ТП	$T_{Pтп}$=14,12	$T_{кк}$=0,7*$N_{тп}$=0,7*28=19,6	$T_{нк}$=0,15*$N_{тп}$=0,15*28=4,2
3.Розробка РП	$T_{Pрп}$=54,76	$T_{кк}$=0,7*$N_{рп}$=0,7*37=25,9	$T_{нк}$=0,15*$N_{рп}$=0,15*37=5,55
4.Розробка ПЗ	$T_{пз}$=1,5**$N_{пз}$= 1,5*30 =45	$T_{кк}$=0,7*$N_{тз}$=0,7*30=21	$T_{нк}$=0,15*$N_{пз}$=0,15*30 =4,5
Усього, в т.ч.:	231,56		
- на розробку	$\Sigma T_{р}$=149,11		
- контроль керівника		$\Sigma T_{кк}$=67,8	
- нормоконтроль			$\Sigma T_{нк}$=14,55

2.2.2 Розрахунок ціни програмного продукту

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПП. Розрахунок основної заробітної плати виконавців приведений у таблиці 2.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2024» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2024 року - 8000 гривень; мінімальну погодинну тарифну ставку – 48.10 грн.

Таблиця 2.7. Розрахунок основної заробітної плати виконавців

Найменування робіт	Трудоміст. робіт, години	Погодинна тар. ставка, грн.	Розрахунок, грн.
1.Розробка ПП	149,11	48.10	5384,36
2.Контроль керівника	67,8	48.10	2583,18
3.Нормоконт-роль	14,55	48.10	554,36
Усього	-	-	ΣЗо= 8521,90

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в таблицю 2.8

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПЗ

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	100	1.5	150,0
Папір	Лист А1	4	15,0	60,0
Разом	-	-	-	$В_{м\ і}$ =210,0
Транспортно – заготівельні Витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$= 0,1х210=21,0
Усього				$В_м$=$В_{мі}$+$В_{тр\_з=}$ 231.0

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

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	231.0	$В_м$ (див. табл. 2.7)
2. Основна заробітна плата	8521,90	$З_о$ (див. табл. 2.6)
3.Додаткова заробітна плата	852,19	$Зд = 0{,}1 \times Зо$ =8521,90х0,1
4.Відрахування до єдиного фонду соціального внеску	2062,30	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$ =0,22х (8521,90+852,19)
5. Накладні витрати	3408,76	$Внак. = 0{,}4 \times Зо$ =0,4 х 8521,90
6. Повна собівартість	15076,15	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.= 231.0+8521,90+852,19+2062,30+ 3408,76

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

$$П=(С_п*Р)/100 \qquad (2.4)$$

Де р – плановий рівень рентабельності (10-20%).

П= (15076,15*10)/100 = 1507,61грн.

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Вступ

Безпечні умови праці – не тільки запорука комфортного існування працівників у межах підприємства, а в першу чергу – їх здоров’я та працездатності, а відтак і прибутковості підприємства. Безпека праці на підприємстві може бути на належному рівні тільки тоді, коли всебічно виконуються вимоги трудового законодавства, державних стандартів України, норм і правил, розроблених для збереження здоров'я працюючих.

В розділі охорона праці дипломного проекту розглядаються питання умов праці програміста, для безпечної роботи працівника при розробки 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity

3.2 Аналіз небезпечних і шкідливих факторів, що впливають на програміста при розробці програмного комплексу

Небезпечним називається фактор, вплив якого на працюючу людину в певних умовах може привести до виробничої травми або іншому раптовому різкому погіршенню здоров’я. Якщо ж виробничий чинник приведе до захворювання або зниження працездатності, то його вважають шкідливим. Залежно від рівня й тривалості впливу, шкідливий чинник може стати небезпечним.

В процесі роботи на користувачів ПК можуть мати вплив наступні небезпечні та шкідливі фактори:

- Невідповідність параметрів мікроклімату нормам;
- Недостатній рівень освітленості;
- Ураження електрострумом;
- Статична електрика;
- Порушення організації робочого місця тощо.

3.3 Гігієнічні вимоги до виробничого середовища

У відповідності з Правилами охорони праці під час експлуатації ОТ на робочому місці користувача ПК повинні бути створенні умови для високопродуктивної праці. Розглянемо ці умови.

3.3.1 Вимоги до приміщення

Для приміщень, які призначені для роботи з ВДТ, доцільно обрати орієнтацію вікон на північ або на північний схід. На вікнах повинні бути жалюзі, що регулюються, або штори, що дають можливість їх повністю закривати. Приміщення відповідно до ДБН В.2.5-28-2018 «Природне і штучне освітлення» повинні мати природне та штучне освітлення. з При приміщеннях ВДТ мають бути обладнані побутові приміщення для відпочинку, психологічного розвантаження тощо.

Площа на одне робоче місце для користувачів повинна складати не менше 6 кв.м, а об’єм – не менше 20,0 куб.м. Стіни пофарбовані матовою фарбою, у відповідності з санітарними вимогами.

3.3.2 Освітлення

Для освітлення приміщення, у якому працює користувач ПК, використовується змішане освітлення, тобто сполучення природного й штучного освітлення. Для загального освітлення приміщення використовуються газорозрядні лампи типу ЛД. Норма для необхідної освітленості робочого місця становить 300-500 лк.

3.3.3 Шум

При розумовій праці, яка вимагає зосередженості припустимий рівень шуму становить 50дБ. Для зменшення шуму й вібрації в приміщенні устаткування, апарати й прилади встановлюють на спеціальні прокладки, що амортизують. Якщо стіни в приміщенні є джерелами шумоутворення, вони повинні бути облицьовані звуковбирним матеріалом.

3.3.4 Мікроклімат

Порушення відповідності ц параметрів мікроклімату впливають на працездатність працівників, їх реакцій, збільшення кількості помилок. Тому в

приміщенні повинні бути установлені оптимальні параметри мікроклімату: температура повітря 22-25 ^{0}С, вологість повітря – 40-60%, швидкість пуху повітря – 0,1-0,2 м/с. Для цього приміщення має бути оснащене системами опалення й кондиціювання, що забезпечують постійне й рівномірне нагрівання, циркуляцію й очищення повітря від пилу й шкідливих речовин.

3.3.5 Електробезпека

Проходячи через організм людини електричний струм робить термічну, електролітичну і біологічну дію.

Для попередження поразок електричним струмом необхідно:

- У повному обсязі виконувати правила провадження робіт і правил технічної експлуатації;
- Виключати можливість доступу працівника до частин устаткування, що працює під небезпечною напругою, неізольованим частинам, призначеним для роботи при малій напрузі й не підключеним до захисного заземлення;
- Застосовувати ізоляцію, що служить для захисту від поразки електричним струмом.

Для попередження поразок електричним струмом необхідно:

- У повному обсязі виконувати правила провадження робіт і правил технічної експлуатації;
- Виключати можливість доступу працівника до частин устаткування, що працює під небезпечною напругою, неізольованим частинам, призначеним для роботи при малій напрузі й не підключеним до захисного заземлення;
- Застосовувати ізоляцію, що служить для захисту від поразки електричним струмом.

Заземлені конструкції, що знаходяться в приміщеннях, де розміщені робочі місця операторів (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном) мають бути надійно захищені діелектричними щитками або сітками з метою недопущення потрапляння працівника під напругу.

3.4 Вимоги до організації робочого місця працівника

Робочі місця повинні бути розташовані так, щоб у поле зору працюючого не попадали поверхні, що мають властивість віддзеркалювання, вікна освітлювальні прилади. Відеотермінали повинні встановлюватися під кутом 90-100 градусів від вікон, так, щоб світло падало з боку. Робочі місця з ВДТ доцільно розміщати в глибині приміщення. Розташування відео термінала, при якому працюючий звернений обличчям або спиною до вікон, неприпустимо при будь-якому способі реалізації загального висвітлення, як прямим, так і відбитим світлом.

Робочий стіл повинен регулюватися по висоті в границях 680-800 мм, а ширина – забезпечувати можливість виконання операцій в зоні досяжності моторного поля Рекомендовані розміри столу: висота 725 мм, ширина 600-1400 мм, глибина 800-1000 мм. Робочий стілець повинен бути оснащений підйомно-поворотним пристроєм для регулювання висоти сидіння і спинки, а також кута її нахилу. Регулювання кожного параметра повинне вироблятися легко, бути незалежним і надійно фіксуватися.

Розташування екрана ВДТ має забезпечувати зручність зорового спостереження у вертикальній площині під кутом $+30^0$ до нормальної лінії погляду працюючого.

Клавіатуру слід розташовувати на поверхні столу на відстані 100…300 мм від краю, звернутого до працюючого.

Організація робочого місця користувача комп'ютера повинна забезпечувати відповідність усіх елементів робочого місця та їх взаємного розташування ергономічним вимогам рисунок 3.1:

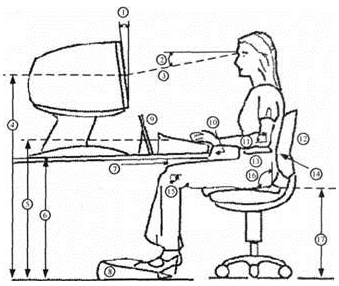


Рис.3.1. Робоче місце і робоча поза користувача комп'ютера

1 — кут екрана; 2 — кут огляду (зору); 3 — відстань огляду; 4 — висота середини екрана; 5 — висота клавіатури; 6 — висота столу; 7 — відстань колін від столу; 8 — підставка для ніг; 9 — підставка для документів; 10 — положення рук; 11 — кут ліктів; 12 — спинка крісла; 13 — підлокітник; 14 — опора для попереку; 15 — кут колін; 16 — кут спинки крісла; 17 — висота сидіння

3.5 Пожежна безпека

Пожежна безпека приміщень, що мають електричні мережі, регламентується ГОСТ 12.1.033-81, ГОСТ 12.1.004-85. Робота оператора ЕОМ повинна вестися в приміщенні, що відповідає категорії Д пожежної безпеки.

Пожежна безпека забезпечується:

-системою запобігання пожежі;

-системою протипожежного захисту;

- організаційно-технічними заходами.

Протипожежний захист приміщення забезпечується застосуванням установки автоматичної пожежної сигналізації, наявністю засобів пожежогасіння, організацією своєчасної евакуації людей.

Для ліквідації невеликих осередків пожеж, а також для гасіння пожеж у початковій стадії їх розвитку силами персоналу об’єктів, застосовуються первинні засобі пожежогасіння. Це вогнегасники (вуглекислотні та порошкові), пожежний інвентар (покривала з негорючого полотна, ящики з піском, бочки з водою), пожежний інвентар. Засоби зображені на рисунку 3.2:

Рис.3.2. Первинні засоби пожежогасіння

ВИСНОВКИ

Протягом виконання проектування дипломного проекту, було виконано дослідження вже існуючих ігрових рішень у жанрі скролл-шутеру. Аналіз отриманих даних дав змогу виділити ігрові елементи, елементи інтерфейсу та ігрового процесу для цього жанру.

Опираючись на отримані дані, було розроблено концепцію ігрового проекту, його стиль, та основу процесу гри. Створено базу матеріалів, а саме спрайтів фону, гравця, ворогів та інше. Частина графічного матеріалу було створено власноруч – інша частина була взята з відкритих баз спрайтів.

Розробка була поділена на етапи проектування гри та реалізації. Під час проектування, було проведено аналіз концепції гри та виділено основні ігрові модулі, які дали змогу створити алгоритми взаємодії між ними та ігровими об’єктами безпосередньо під час ігрового процесу. Проектування значно спростило процес розробки.

Реалізація алгоритмів роботи та взаємодії модулів гри між собою була виконана із дотриманням принципів ООП та модульної розробки ігор. Для написання коду використовувалась середа розробки Microsoft Visual Studio, на мові програмування C#.

Останньою ітерацією розробки було виконання відладки проекту, пошуку помилок та недоліків у коді, що могло призводити до критичних відмов у виконанні гри. Розігруючи різні ігрові ситуації було виправлено декілька помилок.

Як результат роботи, було отримано гру в жанрі горизонтального скролл-шутеру із дотриманням його основних рис. Ігровий процес виконано таким чином, щоби гравець з будь-яким досвідом міг відразу долучитись до гри.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Гусєв Б.С. Комп’ютерна схемотехніка [навчальний посібник] // – К.: НУБіП України, 2022. – 264с.
2. Матвієнко М.П., Розен В. П. Комп’ютерна схемотехніка: навчальний посібник. – К.: Видавництво Ліра-К, 2020. – 192 с.
3. Дейбук В.Г. Віртуальний електронний практикум: Навчальний посібник – Чернівці: Чернівецький нац. ун-т, 2021. – 188 с.
4. Трофименко О.Г. С++. Алгоритмізація та програмування: підручник / О.Г. Трофименко, Ю.В. Прокоп, Н.І. Логінова, О.В. Задерейко. 2-ге вид. перероб. і доповн. – Одеса : Фенікс, 2019. – 477 с.
5. Азаров О. Д., Гарнага В. А., Клятченко Я. М., Тарасенко В. П. Комп’ютерна схемотехніка: підручник. – Вінниця: ВНТУ, 2018. – 230 с.
6. Архангельский А.Я. Программирование в C++Builder, 7-е изд. – М.: ООО Бином-Пресс, 2010 г. – 1230с., ил.
7. Нікулін В.С. Перетворювальні пристрої, ведені мережею: Конспект лекцій. –Харків: УкрДАЗТ, 2008. – Ч.4. – 85 с.
8. Мосіюк О.О., Федорчук А.Л. Операційні системи та системне програмування: навчально-методичний посібник. Житомир: Вид-во ЖДУ ім. Івана Франка, 2022. – 76 с.
9. Stroustrup B. A Tour of C++ (Second Edition). – Addison-Wesley, 2018. – 240 p. – ISBN 978-0-13-499783-4.
10. Каганюк О.К. Комп'ютерна схемотехніка: Навчальний посібник. – Луцьк: РРВ Луцького НТУ, 2016. – 236 с.
11. Бібліотека MSDN [Електронний ресурс]. – Режим доступу: URL http://msdn.microsoft.com/ru-ru/library/default.aspx.

ДОДАТОК А. Лістинг коду основних модулів гри мовою C#

Скрипт en_laser_logic.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class en_laser_logic : MonoBehaviour
{
    float laser_speed = -0.05f;
    private void FixedUpdate()
    {
        transform.Translate(laser_speed, 0f, 0f, Space.Self);
        if (transform.position.x < -13f)
        {
            Destroy(this.gameObject);
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        player_parameters _player_parameters;
        _player_parameters = collision.GetComponent<player_parameters>();
        if (_player_parameters)
        {
            player_parameters.take_damage(15);
            Destroy(this.gameObject);
        }
    }
}
```

Скрипт enemy_logic.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class enemy_logic : MonoBehaviour
{
    [SerializeField] GameObject enemy_laser_prefab;
    GameObject enemy_laser_go;
    int enemy_ship_hp = 100;
    float enemy_ship_speed = -0.025f;
```

```
    bool can_shoot = true;
    private void FixedUpdate()
    {
        transform.Translate(enemy_ship_speed, 0f, 0f, Space.World);
        StartCoroutine(enemy_shoot());
        if (transform.position.x < -13f)
        {
            Destroy(this.gameObject);
        }
    }

    public void take_damage(int damage_value)
    {
        enemy_ship_hp -= damage_value;
        if (enemy_ship_hp <= 0)
        {
            GameObject _player_go = GameObject.Find("Player");
            player_parameters _player_parameters =
_player_go.GetComponent<player_parameters>();
            _player_parameters.increase_player_score(100);
            Destroy(this.gameObject);
        }
    }

    private IEnumerator enemy_shoot()
    {
        if (can_shoot)
        {
            can_shoot = false;
            Vector3 laser_ini_position = new Vector3(transform.position.x - 0.8f,
transform.position.y, transform.position.z);
            Quaternion laser_rotation_ini = new Quaternion(0f, 0f, 0f, 0f);
            enemy_laser_go = Instantiate(enemy_laser_prefab, laser_ini_position,
laser_rotation_ini) as GameObject;
            yield return new WaitForSeconds(4f);
            can_shoot = true;
        }
    }
}
```

Скрипт background_moving.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class background_moving : MonoBehaviour
{
    [SerializeField] GameObject background_1;
    [SerializeField] GameObject background_2;
    Vector3 bg1_coord;
    Vector3 bg2_coord;

    void Start()
    {
        bg1_coord = new Vector3(8.8f, 0f, 0f);
        bg2_coord = new Vector3(51.4f, 0f, 0f);
    }

    void FixedUpdate()
    {
        background_1.transform.position = bg1_coord;
        background_2.transform.position = bg2_coord;
        bg1_coord.x -= 0.01f;
        bg2_coord.x -= 0.01f; ;
        if (bg1_coord.x <= -33.8f)
            bg1_coord.x = 51.4f;
        if (bg2_coord.x <= -33.8f)
            bg2_coord.x = 51.4f;
    }
}
```

Скрипт hp_block_logic.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class hp_block_logic : MonoBehaviour
{
    private void FixedUpdate()
    {
        transform.Translate(-0.015f, 0f, 0f, Space.Self);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        player_parameters _player_parameters;
        _player_parameters = collision.GetComponent<player_parameters>();
        if (_player_parameters)
        {
```

```
            _player_parameters.take_hp_point(50);
            Destroy(this.gameObject);
        }
    }
}
```

Скрипт enemy_generator.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class enemy_generator : MonoBehaviour
{
    [SerializeField] GameObject enemy_prefab;
    GameObject enemy_ship_go;
    int enemy_ship_spawn_count = 0;
    bool can_spawn = true;

    void Start()
    {
        enemy_ship_spawn_count = Random.Range(1, 6);
    }

    private void FixedUpdate()
    {
        StartCoroutine(ship_spawner());
    }

    protected IEnumerator ship_spawner()
    {
        if(can_spawn)
        {
            can_spawn = false;
            for(int i=0; i<enemy_ship_spawn_count; ++i)
            {
                Vector3 new_enemy_spawn_pos = new Vector3(13.25f, Random.Range(-6.4f, 6.4f), 0);
                enemy_ship_go = Instantiate(enemy_prefab, new_enemy_spawn_pos, enemy_prefab.transform.rotation) as GameObject;
            }
            yield return new WaitForSeconds(1.5f);
            can_spawn = true;
            enemy_ship_spawn_count = Random.Range(1, 6);
        }
```

Скрипт pickups_generator.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class pickups_generator : MonoBehaviour
{
    [SerializeField] GameObject hp_block_prefab;
    GameObject hp_block_go;
    int hp_block_count = 0;
    bool can_spawn_pickups = true;

    void Start()
    {
        hp_block_count = Random.Range(0, 2);
    }

    private void FixedUpdate()
    {
        StartCoroutine(pickups_spawner());
    }

    protected IEnumerator pickups_spawner()
    {
        if (can_spawn_pickups)
        {
            can_spawn_pickups = false;
            for (int i = 0; i < hp_block_count; ++i)
            {
                Vector3 new_pickips_spawn_pos = new Vector3(13.25f, Random.Range(-6.4f, 6.4f), 0);
                hp_block_go = Instantiate(hp_block_prefab, new_pickips_spawn_pos, hp_block_prefab.transform.rotation) as GameObject;
            }
            yield return new WaitForSeconds(4f);
            can_spawn_pickups = true;
            hp_block_count = Random.Range(0, 2);
        }
    }
}
```

Скрипт laser_logic.cs

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;

public class laser_logic : MonoBehaviour
{
    float laser_speed = 0.2f;

    private void FixedUpdate()
    {
        transform.Translate(laser_speed, 0f, 0f, Space.Self);
        if(transform.position.x > 16f)
        {
            Destroy(this.gameObject);
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        enemy_logic _enemy_logic;
        _enemy_logic = collision.GetComponent<enemy_logic>();
        if(_enemy_logic)
        {
            _enemy_logic.take_damage(45);
            Destroy(this.gameObject);
        }
    }
}
```

Скрипт player_controll.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class player_controll : MonoBehaviour
{
    [SerializeField] GameObject laser_prefab;
    GameObject laser_go;
    Rigidbody2D _player_rigidbody;
    float horizontal_input;
    float vertical_input;
    float player_speed = 3f;
    bool can_shoot = true;

    void Start()
    {
```

```
        _player_rigidbody = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        horizontal_input = Input.GetAxis("Horizontal");
        vertical_input = Input.GetAxis("Vertical");
        if(Input.GetKey(KeyCode.Space))
        {
            StartCoroutine(laser_shoot());
        }
    }

    private void FixedUpdate()
    {
        _player_rigidbody.velocity = new Vector3(horizontal_input * player_speed,
vertical_input * player_speed, 0f);
    }

    private IEnumerator laser_shoot()
    {
        if(can_shoot)
        {
            can_shoot = false;
            Vector3 laser_ini_position = new Vector3(transform.position.x + 0.8f,
transform.position.y, transform.position.z);
            Quaternion laser_rotation_ini = new Quaternion(0f, 0f, 0f, 0f);
            laser_go = Instantiate(laser_prefab, laser_ini_position,
laser_rotation_ini) as GameObject;

            yield return new WaitForSeconds(0.2f);
            can_shoot = true;
        }
    }
}
```

Скрипт player_parameters.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class player_parameters : MonoBehaviour
{
```

```
    [SerializeField] ui_manager _ui_manager;
    int player_hp = 100;
    int player_shield = 0;
    int player_score = 0;

    private void Start()
    {
        _ui_manager.change_player_hp(player_hp);
        _ui_manager.change_player_shield(player_shield);
        _ui_manager.change_player_score(player_score);
    }

    public void take_damage(int damage_value)
    {
        player_hp -= damage_value;
        _ui_manager.change_player_hp(player_hp);
        if (player_hp <= 0)
        {
            Destroy(this.gameObject);
        }
    }

    public void take_hp_point(int hp_point_value)
    {
        player_hp += hp_point_value;
        _ui_manager.change_player_hp(player_hp);
    }

    public void increase_player_score(int value)
    {
        player_score += value;
        _ui_manager.change_player_score(player_score);
    }
}
```

Скрипт ui_manager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ui_manager : MonoBehaviour
{
    [SerializeField] Text player_hp_text;
```

```
    [SerializeField] Text player_shield_text;
    [SerializeField] Text player_score_text;

    public void change_player_hp(int value)
    {
        player_hp_text.text = "Ship hull: " + value;
    }

    public void change_player_shield(int value)
    {
        player_shield_text.text = "Ship shield: " + value;
    }

    public void change_player_score(int value)
    {
        player_score_text.text = "Player score: " + value;
    }
}
```

ДОДАТОК Б. Слайди мультимедійної презентації

Розробка 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity

ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТА ГРУПИ 4РП-07: ВОСКОБОЙНИКА СТАНІСЛАВА ОЛЕКСАНДРОВИЧА

КЕРІВНИК: ДЖАБРАІЛОВ Д.В.

Особливості ігро|вого жанру 2D скролл-шутер

Скролл-шутар – це гра в якому гравець переміщується або зверху-вниз, або знизу-верх, а також справа-наліво та зліва-направо. Під час переміщення по рівню, на зустріч гравцю відображаються елементи оточення або вороги, які необхідно знищувати, або уникати.

Основними характеристиками жанру є:

- Динамічний ігровий процес;
- Яскравий геймплей;
- Тісна взаємодія із оточенням.

Особливості програмного рушія Unity та причини його вибору для розробки проекту

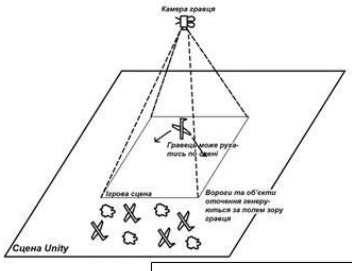
Ігровий двигун Unity є одним із популярніших ігрових двигунів у світі. За його допомогою створюють як мобільні, так і комп'ютерні ігри. Цей двигун досі активно розвивається!

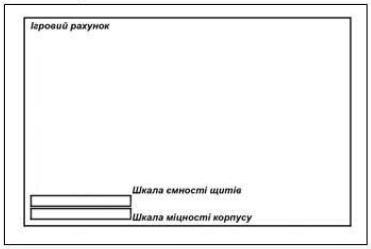
- Має зручну візуальну середу розробки;
- Має можливість міжплатформенної розробки ігор;
- Підтримує модульну систему компонентів;
- Використання мови програмування C# та його можливостей;
- Велике ком'юніті розробників;
- Зрозуміла та вичерпна документація, безліч курсів у інтернеті та літературі;
- Гнучка система ліцензування ігрового двигуна;
- Безплатні та платні ассети для проектів будь-якого рівня.

Особливості ігрових механік розроблюємого проекту

Можна виділити такі особливості розробляємого проекту:

- Гравець має свої параметри – міцність корпусу, щити;
- Ці параметри можуть накопичуватись;
- Можна збирати спеціальні об'єкти на рівні;
- Гра генерує різні типи ворогів та предметів;
- Гравець може вільно переміщуватись по рівню.





Огляд основних елементів ігрового процесу

Основним елементом ігрового процесу є переміщення по ігровій зоні, яка має нескінченний об’єм до тих пір, поки гравець не назбирає необхідну кількість ігрового рахунку.

Гравець не обмежений у переміщені в рамках ігрової зони.

Кожний знищений об’єкт додає до ігрового рахунку очки.

Параметри гравця можуть бути змінені бонусними предметами.

Принцип роботи основних елементів ігрового процесу

Для реалізації гри в жанрі скролл-шутер, достатньо реалізувати 4 типові модуля:

- Код ворожих об’єктів;
- Код гравця;
- Код інтерфейсу;
- Код оточення;
- Код генераторів.

Ці модулі мають свої окремі підмодулі, які можна за необхідністю додавати до проекту, щоби реалізовувати нові ігрові механіки.

Принцип роботи основних елементів ігрового процесу

Модуль ворожих об’єктів забезпечує комплекси коду для реалізації ігрових об’єктів, що будуть відігравати ролі ворогів.

В поточній версії проекту, реалізовані вороги у виді астероїдів та кораблів, що будуть генеруватись на рівні та рухатись на гравця.

Кожний з об’єктів є типовим, тому додавання нових ворогів не займає багато часу з точки зору коду та ігрового рушія.

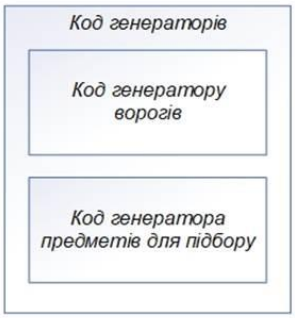
Принцип роботи основних елементів ігрового процесу

Модуль коду генераторів складається з скриптів:

- Генератора ворогів;
- Генератора Предметів для підбору.

Ці скрипти слідкують за кількістю ворогів у черзі на генерацію, створюють їх, задають напрямок руху.

У випадку генератора предметів для підбору, він робить те ж саме, але генерує об’єкти для поповнення параметрів гравця.

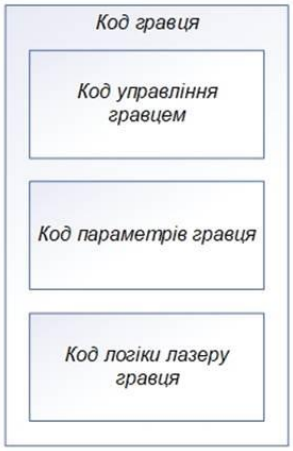


Принцип роботи основних елементів ігрового процесу

Модуль гравця є схожим до Модуля ворожих об’єктів. Так само він реалізує основний код для ігрового процесу гравця на ігровій сцені.

Його відрізняє відсутність своєї логіки. Замість цього тут присутній код, що реалізує управління об’єктом гравця.

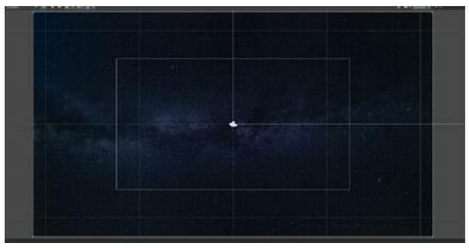
Також тут присутній код логіки лазерів для можливості у подальшому реалізовувати різні варіанти зброї для гравця

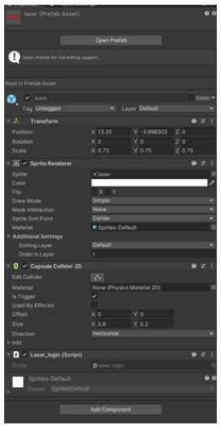


Етапи реалізації основних елементів ігрового процесу

Реалізація основних елементів гри складалась з:

- Реалізації ігрового фону для рівня;
- Створення префабів ворогів;
- Створення об’єкту гравця;
- Створення об’єктів для підбору та їх префабів;
- Написання коду для гри.



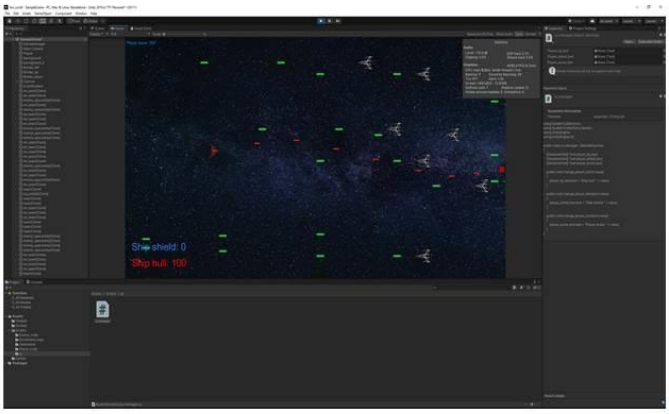


Хід тестування гри

Тестування гри проводилось з допомогою вбудованого у ігровий програмний рушій Unity інструмент.

Метою тестування було:

- Визначити коректність управління кораблем гравця;
- Визначити коректність роботи логіки пострілів;
- Визначити коректність роботи ворогів;
- Визначити коректність роботи логіки підбору предметів.



Скріншоти готової гри

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Воскобойника Станіслава Олександровича

(прізвище, ім`я та по батькові)

Спеціальність: 121 "Інженерія програмного забезпечення"

Освітня програма: «Розробка програмного забезпечення»

Тема дипломного проекту: Розробка 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка містить 76 сторінок. У пояснювальній записці виконано опис предметної області, способи створення гри в жанрі горизонтального скролл-шутеру. Проведено проектування та реалізація елементів гри. Графічна частина складається з 12 слайдів мультимедійної презентації, які передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.

б) самостійність роботи над проектом: Протягом всього строку дипломного проектування та переддипломної практики здобувач освіти Воскобойник С.О. поступово та послідовно виконував всі етапи розробки. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.

в) теоретична підготовка випускника (випускниці): Здобувач освіти Воскобойник С.О. під час роботи над дипломним проектом вивчив достатню кількість літературних джерел та матеріалів за даною тематикою.

Вважаю, що теоретична підготовка дипломника добра і він готовий до

г) вміння розв'язувати виробничі та конструкторські питання

Під час дипломного проектування здобувач освіти Воскобойник С.О. мав змогу самостійно приймати рішення з реалізації елементів і модулів гри, та показав вміння організовано працювати над поставленим завданням, складати схеми та проводити розробку коду за допомогою актуальних для теми комп'ютерних програмних засобів.

Оцінка розрахункової частини Добре

Оцінка графічної частини Добре

Загальна оцінка Добре

Прізвище, ім'я, по батькові керівника дипломного проекту

Джабраілов Дмитро Володимирович

Місце роботи і посада керівника дипломного проекту

ВСП "Одеський технічний фаховий коледж ОНТУ", викладач комісії комп'ютерних технологій та програмної інженерії

Підпис

«10» 06 2024 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Воскобойника Станіслава Олександровича

(прізвище, ім`я та по батькові)

Спеціальність 121 " Інженерія програмного забезпечення "

Освітня програма «Розробка програмного забезпечення»

Керівник дипломного проекту (роботи) Джабраілов Дмитро Володимирович

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) Розробка 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity

Обсяг розрахунково-пояснювальної записки 76 сторінок

Обсяг графічної (презентаційної) частини 12 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлений на рецензію дипломний проект повністю відповідає меті проектування та технічному завданню. Тематика дипломного проекту є актуальною для своєї галузі та присвячена питанням створення ігрових продуктів в цілому та розробці ігор на ігровому програмному рушії Unity, в цілому.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Дипломний проект складається зі вступу, трьох розділів, висновків, переліку використаних джерел. У технологічному розділі розглянуті питання проблематики розробки гри на ігровому програмному рушії Unity, сформовано вимоги до гри згідно до теми дипломного проекту та завданню, виконано проектування основних аспектів розробляємої гри. За допомогою відповідного програмного забезпечення реалізовані всі намічені зміни до ігрового процесу

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана акуратно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено

г) перелік позитивних якостей дипломного проекту (роботи)

1. Детально описано процес виконання розробки гри;
2. Виконано проектування елементів гри із поясненнями на схемах та за допомогою коду;
3. Розроблено функціонуючу гру за допомогою відповідних інструментів розробки.

д) основні недоліки дипломного проекту (роботи)

1. Гра має частковий ігровий функціонал з точки зору переходу до нових рівнів;
2. Реалізовано мало ворогів та ігрових механік на рівнях.
3. Робота містить помилки у тексті, на деяких скріншотах неможливо розібрати вміст.

Оцінка розрахункової частини Добре

Оцінка графічної частини Добре

Загальна оцінка Добре

Прізвище, ім'я, по батькові рецензента к.т.н. Кірєєв Ігор Анатолійович

Місце роботи і посада рецензента Державний університет інтелектуальних технологій і зв'язку, доцент каф. інформаційної безпеки та передачі даних

Підпис:

ПІДПИС ПОСВІДЧУЮ
НАЧАЛЬНИК ВІДДІЛУ
КАДРІВ ДУІТЗ

ВІДДІЛ КАДРІВ

«14» 06 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

ID перевірки:
1016337247

Дата перевірки:
09.06.2024 09:50:33 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2024 11:11:57 EEST

ID користувача:
100011688

Назва документа: 4РП-07_Воскобойник

Кількість сторінок: 44 Кількість слів: 8999 Кількість символів: 62236 Розмір файлу: 2.50 MB ID файлу: 1016138217

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

6%
Схожість

Найбільша схожість: 3.77% з Інтернет-джерелом (https://card-file.ontu.edu.ua/server/api/core/bitstreams/6c95086b-bff...

6% Джерела з Інтернету | 112 ..... Сторінка 46

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Підозріле форматування | 8 сторінок

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНГО ДИПЛОМНОГО ПРОЕКТА
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Воскобойник Станіслав Олександрович,
здобувач освіти гр. 4РП-07, та

Джабраілов Дмитро Володимирович,
керівник дипломного проекту,

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

«Розробка 2D-гри у жанрі горизонтального скролл-шутеру на програмному рушії Unity» (автор роботи – Воскобойник С.О., керівник роботи – Джабраілов Д.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

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

Виконавець _____ / Воскобойник С.О. /

Керівник _____ / Джабраілов Д.В./

«10» 06 2024 р.