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

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Розробка програмного забезпечення»

Група: 4РП-06

Дипломний проект

здобувача освіти денної форми навчання
РП.06.10.000.ДП

ЄРЬОМЕНКО
АРТЕМА
КОСТЯНТИНОВИЧА

м. Одеса
2023 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

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

Освітня програма: «**Розробка програмного забезпечення»**

Група: **4РП-06**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) на тему:

Розробка комп'ютерної 2D-гри в жанрі вертикального скролл-шутеру за допомогою ігрового двигуна Unity.

Проектний матеріал складається з пояснювальної записки на 71 сторінках та графічного (презентаційного) матеріалу на 12 аркушах (слайдах).

Дипломник _____ (Єрьоменко А.К.)

Керівник _____ (Джабраілов Д.В.)

Консультанти:

з економічної частини _____ (Копайгородська Т.Г.)

з охорони праці _____ (Чорновол Н.І.)

з дотримання вимог ЄСКД _____ (Петрашова В.І.)

старший консультант _____ (Кунуп Т.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «22» 06 2023 р. Протокол ДКК № 1

Оцінка ДКК 5 (відмінно)

Секретар ДКК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

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

Спеціальність **121 «Інженерія програмного забезпечення»**

Освітня програма **«Розробка програмного забезпечення»**

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР _____

Беркань І.В.

"_____" _____ 2023 р.

ЗАВДАННЯ

на дипломний проект (роботу)

Здобувачеві (здобувачці) освіти *Єрьоменко Артему Костянтиновичу*

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Розробка комп'ютерної 2D-гри в жанрі вертикального скролл-шутеру за допомогою ігрового двигуна Unity*

затверджена наказом по коледжу від "17" жовтня 2023р. № 235-А2-ОД

2. Термін здачі закінченого проекту (роботи) 09.06.2023р.

3. Вихідні данні до проекту (роботи)

1. Використання ігрового двигуна Unity;

2. Використання принципів модульної розробки ігор;

3. Використання мови програмування C# та бібліотек Unity для розробки гри;

4. Реалізувати основних ігрових механік комп'ютерної 2D-гри в жанрі вертикального скролл-шутеру

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

Опис предметної області. Огляд існуючих рішень у ігровій індустрії. Формування загального концепту ігрового процесу. Огляд інструментів та програмних рішень. Проектування основних модулів гри, принцип їх роботи. Реалізація основних модулів гри. Тестування проекту на працездатність. Робота над Економічною частиною. Робота над Охороною праці.

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

Особливості ігрового жанру вертикального скролл-шутеру; Особливості ігрового двигуна Unity; Формування загального концепту ігрового процесу; Основні модулі гри; Механізм роботи основних модулів гри; Етапи реалізації основних модулів гри; Скріншоти розробленої гри.

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1.Технологічний розділ	*Джабраілов Д.В.*		
2.Екон. частина	*Копайгородська Т.Г.*		
3.Охорона праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		

7. Дата видачі завдання 24.04.2023 р.

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

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1.	*Вступ. Постановка мети та задач проектування*	*19.05.2023*	Виконав
2.	*Опис предметної галузі*	*20.05.2023*	Виконав
3.	*Огляд існуючих рішень у ігровій індустрії*	*21.05.2023*	Виконав
4.	*Загальних технічний опис вирішення основних задач*	*22.05.2023*	Виконав
	розробки гри		Виконав
5.	*Огляд інструментів та програмних рішень*	*23.05.2023*	Виконав
6.	*Проектування основних модулів гри, алгоритм їх*	*24.05.2023*	Виконав
	роботи		Виконав
7.	*Реалізація модулів вводу*	*25.05.2023*	Виконав
8.	*Реалізація модулів ігрової логіки*	*27.05.2023*	Виконав
9.	*Реалізація інтерфейсу*	*29.05.2023*	Виконав
10.	*Тестування проекту на працездатність*	*31.06.2023*	Виконав
11.	*Виконання Економічної частини*	*02.06.2023*	Виконав
12.	*Виконання Охорони праці*	*04.06.2023*	Виконав
13.	*Підготовка матеріалів до захисту*	*06.06.2023*	Виконав
14.	*Попередній малий захист*	*12.06.2023*	Виконав
15.	*Проведення захисту дипломного проекту*	*19.06.2023*	Виконав

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 06. 10 000. 00 ДП	Дипломний проект		
А4			РП 06. 10 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 06. 10 000. 00 ДП		
Розробив		Єрьоменко А.К.			Розробка комп'ютерної 2D-гри в жанрі вертикального скролл-шутеру за допомогою ігрового двигуна Unity	Літ.	Аркуш 4 / Аркушів 71
Перевірив		Джабраілов Д.В.				Н Д П	
Н. Контр.		Петрашова В.І.				ВСП "ОТФК ОНТУ" гр.4РП-06	
Затверд.		Кривченко Ю.В.					

ЗМІСТ

ВСТУП

В наш час ігрова індустрія займає велику частину IT-сектору. Кількість комп'ютерних ігор, що виходить з кожним роком збільшується, а разом із тим розвивається способи та методи розробки ігор. Ігрова сфера існує не тільки за рахунок великих та дорогих ігор, навпаки більшу частину обороту складають невеликі проекти, тому такі проекти із простих жанрів завжди будуть актуальними для користувачів.

Окрім того, ігрова індустрія сама в собі є актуальною, адже зараз знаходиться лише на початку свого становлення. Так само як і кіноіндустрія, ігрова сфера розпочиналась із малого та обмеженого. Перші ігри були створені на не придатних для цього пристроях, і так продовжувалось до тих пір поки не стали створювати ігрові автомати та ігрові консолі. В наш час цифрові ігри можна отримати майже на будь-який пристрій, в якого є вихід в інтернет.

Оскільки ігрова індустрія актуальна в наші дні – актуальним є і способи створення цифрових ігрових продуктів. Цей процес має певні складові, які необхідно виконувати для створення кінцевого продукту. Темою мого дипломного проекту є «Розробка комп'ютерної 2D-гри в жанрі вертикального скролл-шутеру за допомогою ігрового двигуна Unity» і саме процес створення такої гри висвітлює основні етапи розробки проектів такого типу.

Проте, створення гри в жанрі скролл-шутеру не є унікальною з точки зору процесу розробки, тому розробка подібного проекту дасть основу розуміння типових проблем розробки цифрових ігор взагалі. Реалізація окремих елементів гри дасть змогу краще розуміти етапи та процеси роботи у ігровому двигуні Unity, та покладе основу для розробки інших проектів за допомогою використання принципів модульної розробки.

Зазначене вище вказує й на технологічну актуальність виконання теми дипломного проекту, адже для реалізації проекту цифрової гри необхідно буде використовувати сучасні засоби розробки.

1 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

1.1 Опис предметної області

До предметної області теми дипломного проектування відносяться питання розробки цифрових комп'ютерних ігор. У окремих випадках цей процес може сильно відрізнятись, але в загальному сенсі він складається із деяких загальних етапів.

Перед початком розробки цифрової гри, якщо на неї є запит, виконується визначання з ігровим жанром. Такі жанри в ігровій індустрії представлені дуже широким переліком, яким не менше, а в деякому сенсі навіть більший ніж перелік кінематографічних жанрів. 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.

Якщо звернутись до теми дипломного проектування, то в нас чітко визначений 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 знизу-верх, або зверху-вниз, а також зліва-направо та справа-наліво. Під час переміщення по рівню, на зустріч гравцю відображаються вороги або елементи оточення, які необхідно знищити. Сам гравець отримує ушкодження, або знищується від торкання до ворога чи об'єкта на рівні. Також, негативні наслідки настають від ураження вогнем ворога.

Наступним етапом є визначення із технологіями виконання цифрової гри. У випадку із темою дипломного проектування, чітко визначений ігровий двигун, на якому буде виконуватись створення проекту гри. Окрім визначення двигуна, потрібно визначитись із іншими інструментами розробки. Такий вибір частково та декілька в більшій мірі залежить від вподобань розробника, а також від технічних вимог ігрового двигуна. Як вже було зазначено раніше, маючи чітко

визначений ігровий двигун, необхідно обирати такі програмні рішення та інструменти розробки, які відповідали б його потребам.

Ще один із етапів попередньої розробки полягає у визначенні ключових ігрових елементів та механік, які будуть виконані під час розробки та гратимуть ключову роль у формуванні зацікавленості гравця у продукті. Частково їх перелік диктується ігровим жанром, втім часто розробники намагаються додати якусь несподіваний для жанру елемент, що буде «продавати» гру користувачу.

Нарешті останній з етапів – способи реалізації проекту. Це в більшій мірі є визначення із архітектурою самої гри, її елементів, ігрових механік та елементів. Правильно збудована архітектура набагато полегшує розробку проекту, його підтримку, а також продуктивність додатку. В деякій мірі ці питання диктуються вимогами чи можливостями ігрового двигуна, який може бути як простим у цьому відношенні, так і потребувати визначених рішень архітектури та внутрішньоігрової логістики.

1.2 Огляд існуючих рішень у ігровій індустрії

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. На користь популярності також грала інтуїтивно зрозумілий ігровий процес, що з перших хвилин викликає у гравця почуття випробування.

Перші цифрові ігри такого плану мали доволі обмежену ігрову складову да дуже прості правила «одного торкання», коли будь-який контакт із противником, його кулею, або оточенням викликав поразку гравця. Те ж саме правило діяло і в іншу сторону противника. В подальшому було додана можливість додаткових «шансів», коли можна було помирати декілька разів перед повним перезапуском проходження. 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. Є доцільним розглянути існуючі аналоги таких ігор.

Zero Wing – є одною із класичних цифрових ігор жанру скролл-шутера. Була випущена 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. В свої часи гра отримала велику популярність через свою різноманітність, візуальний стиль а також звукове супроводження. Додатково до популярності на користь йшло старт саме на ігрових автоматах, які були дуже розповсюджені та популярні в ті часи. 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», да додавались нові за ігровий рахунок, або находження його на рівні. Також, дуже

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. Модифікації для зброї гравця отримувались на рівні, як окремі об'єкти, які рухались по ігровій сцені. Модифікації корпусу також розміщались на рівні. 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

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. Також представляє собою один із еталонних прикладів гри в жанрі скролл-шутерів. Тут гравцю необхідно послідовно проходити рівні від початку до кінця, маючи можливість рухатись по сцені, але прогресія рівня все одно йде зі своєю швидкістю. На відміну від Zero Wing, тут у гравця вже є декілька ігрових ресурсів, як міцність корпусу, перегрів зброї, а також замість ігрового рахунку – гроші. Також в цій грі реалізована активність між рівнями, на яких гравець може модифікувати свій корабель, включити модифікатори та отримати нове завдання. Скріншот гри зображені на рисунку 1.2:

Рисунок 1.2. Скріншот гри Jets’n’Guns

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:

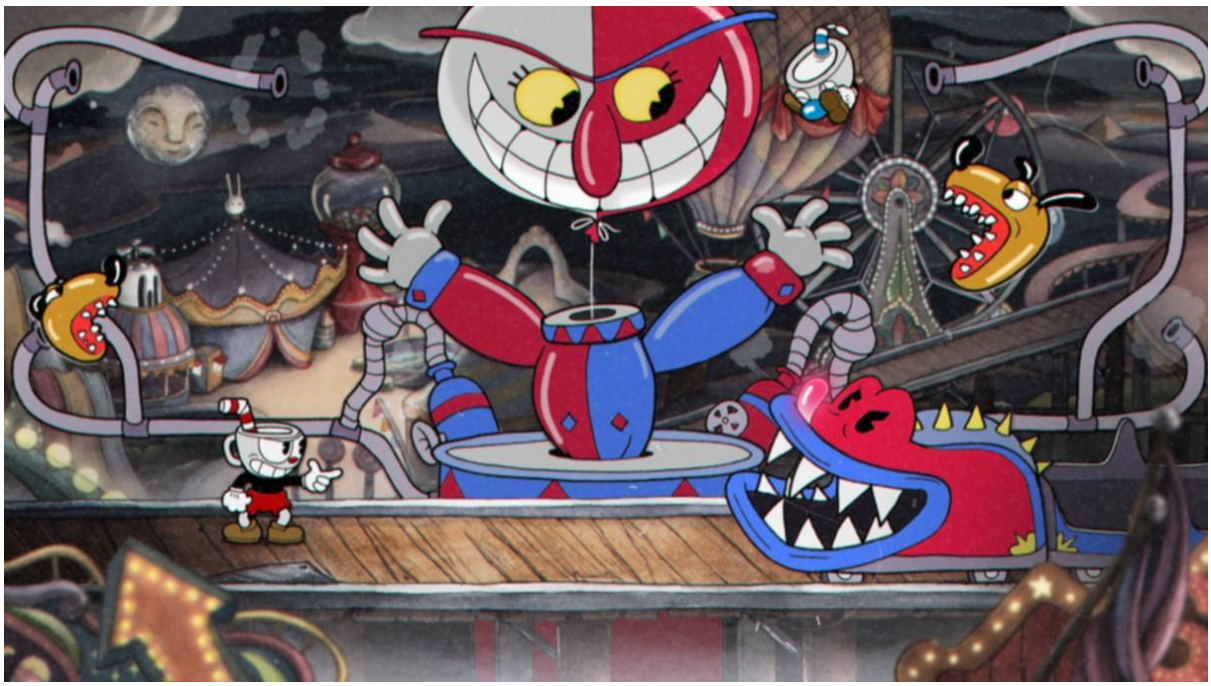


Рисунок 1.3. Скріншот гри Cuphead

Таким чином, розглянувши існуючі аналоги в 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.

1.3 Формування загального концепту ігрового процесу

Для виконання поставлених задач дипломного проектування, необхідно виконати повний цикл розробки проекту цифрової гри, який складається з декількох етапів. Для початку технічного опису, необхідно виконати постановку завдань, а для цього розібрати з яких елементів буде складатись гра.

Оскільки ігровий жанр визначений в темі дипломного проектування, наступним кроком буде виділення загального ігрового концепту. З жанрової точки зору, скролл-шутер, як вже було неодноразово показано, в першу чергу виражається через особливість проходження рівня – гравець може перемішуватись лише в межах сцени, але рух по рівню залишається непідконтрольним йому, чи іноді прискорюється/сповільнюється.

Окреме питання отримання пошкоджень гравцем. В цифрових іграх цього жанру є варіанти з принципом «одного торкання», та такі, де у гравця є значення здоров'я, або міцності корпусу. Обидва підходу мають свої нюанси, які впливають на ігровий дизайн. Наприклад, для ігор із «одним торканням» не потрібно розробляти та балансувати систему пошкоджень, а також розміщувати об'єкти для відновлення показників здоров'я чи міцності. З іншої сторони, для таких ігор є нагальним питанням розподіл «жетонів» для додаткового життя, а також загальної складності ігрового процесу.

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;
- На рівнях будуть з'являтись об'єкти для модифікації зброї гравця;
- 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 двигуна Unity, на якому буде виконуватись робота. Безпосередньо рівень буде реалізовуватись завдяки сцені, на якій будуть знаходитись гравець, його камера – вона буде визначати межі сцени ігрового процесу, по якій гравець зможе переміщатись – його кораблі/вороги та об'єкти

оточення. Оскільки дія буде проходити в космічному просторі, то об’єктами будуть астероїди, а ворогами космічні кораблі. 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. Вони будуть пов’язані із інтерфейсом, а також ігровим процесом для прямого та зворотного зв’язку.

Генерацією об’єктів для зміни показників ресурсів, модифікації зброї також буде займатись ігровий менеджер, що за допомогою коду буде генерувати об’єкти різних типів, в тому числі астероїди та ворогів. Із загальним концептом ігрового процесу можна ознайомитись на рисунку 1.4:

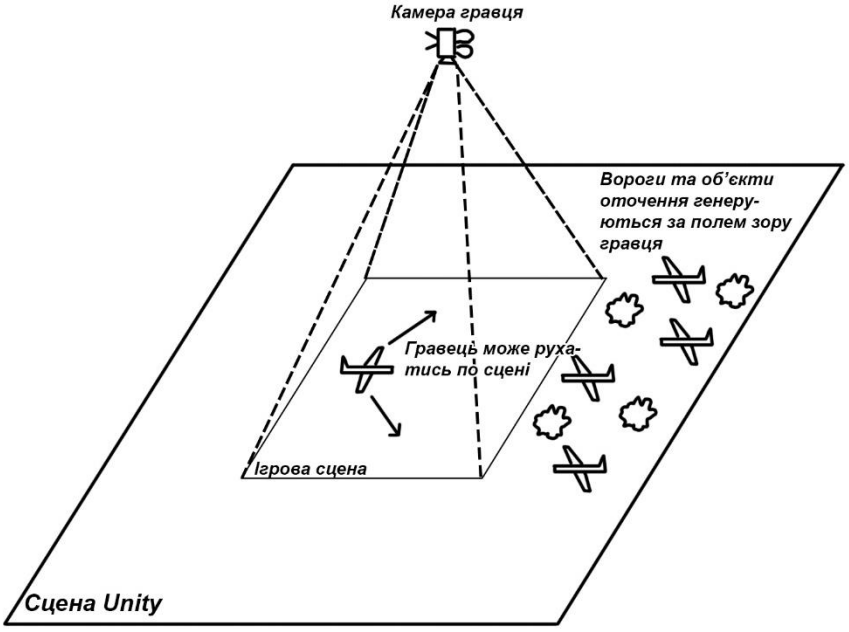


Рисунок 1.4. Загальний концепт ігрового процесу

Окремо слід розглянути питання ігрового інтерфейсу, який буде використовуватись гравцем. Всього він має відображати декілька показників: міцність корпусу, ємність щитів, енергія корабля, ігровий рахунок. Всі ці елементи мають змінюватись в реальному часі, в залежності від ігрової ситуації, а тому мають бути видними для гравця у любий момент гри. Тому ці елементи будуть відображатись на краях екрану у вигляді панелей, які будуть

заповнюватись в залежності від стану. Ігровий рахунок буде відображатись у текстовому виді. Із загальним концептом ігрового інтерфейсу гри можна ознайомитись на рисунку 1.5:

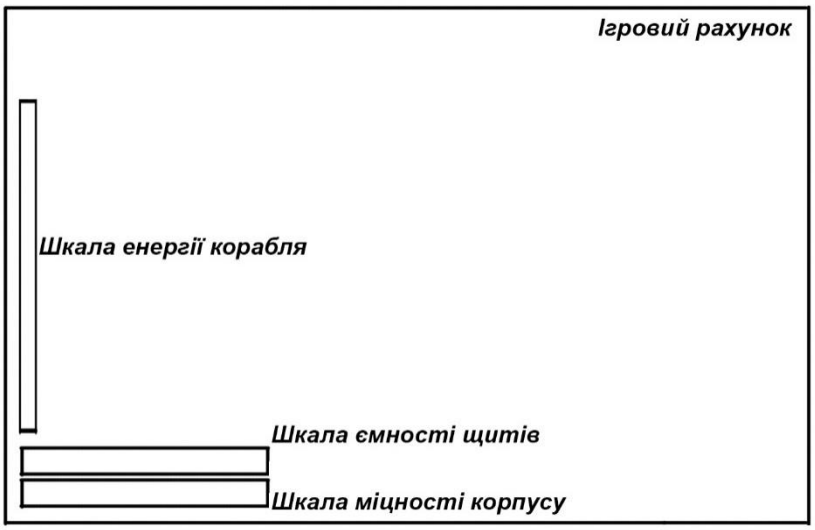


Рисунок 1.5. Загальний концепт ігрового інтерфейсу гри

1.4 Огляд інструментів та програмних рішень

Для виконання завдання дипломного проекту було використано ігровий двигун Unity, середа розробки програмних проектів Visual Studio 2017, а також растровий графічний редактор Adobe Photoshop CS6.

Unity є сучасним ігровим двигуном, який регулярно отримує оновлення функціоналу та можливостей, постійно розширяючись і розвиваючись. На рисунку 1.6 можна розглянути робочу середу ігрового двигуна Unity. Він є одним із найрозповсюджених ігрових двигунів, який частіше використовують для розробки ігор для смартфонів, або кишенькових пристроїв. Він має зручний та зрозумілий інтерфейс, який можна налаштувати під свої потреби в залежності від того, над яким проектом ви працюєте, або які обов’язки у розробці виконуєте.

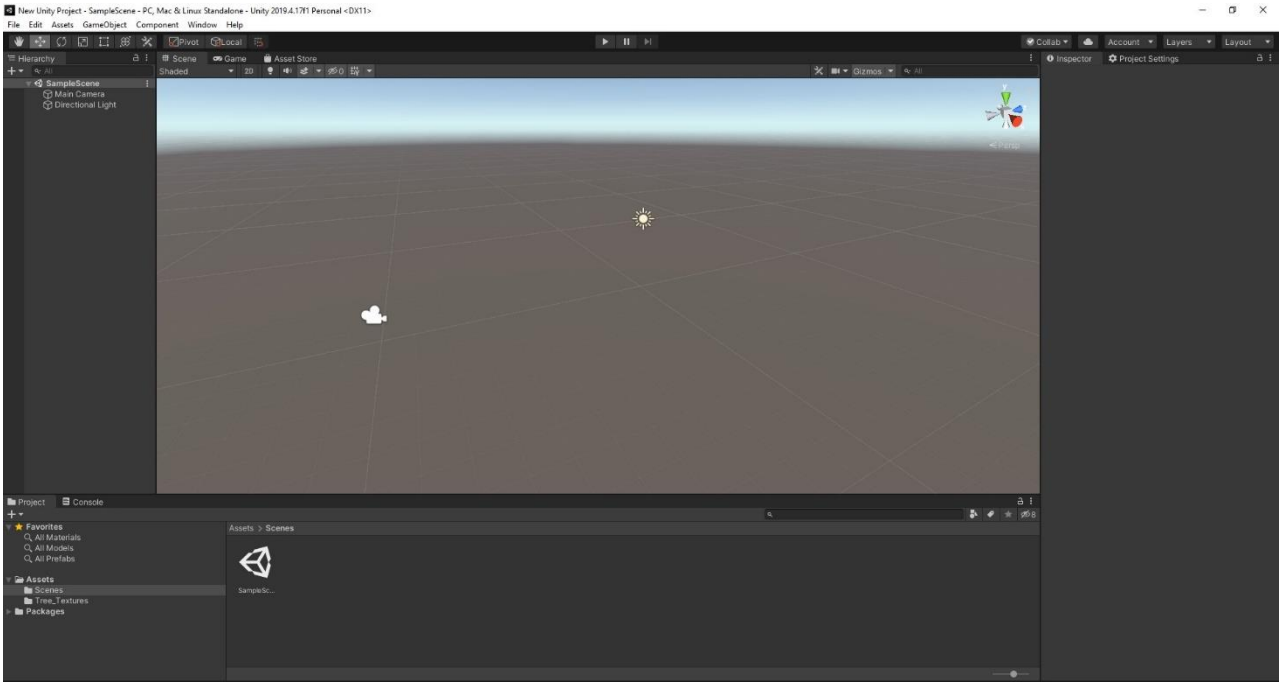


Рисунок 1.6. Робоча середа ігрового двигуна Unity

Цей ігровий двигун дає змогу працювати у зручному трьохвимірному просторі редактора сцени. Розмішуючи на ній об’єкти, редагуючи їх властивості, компоненти та можливості, в залежності від ваших намірів. Крім того, інтерфейс та можливості Unity також дозволяють створювати основні типи файлів необхідні для роботи над проектом і редагувати їх безпосередньо у редакторі.

Окрім того, середа розробки дає можливість працювати не лише із візуальною складовою ігрового проекту, а також дає можливість редагувати ігровий код за допомогою системи скриптів, які додаються до об’єктів да надають їм додаткового функціоналу. В поточній версії Unity використовується мова програмування C#, що дає змогу використовувати всю потужність цієї мови програмування. Окрім бібліотек та можливостей самого C#, Unity має велику кількість своїх бібліотек, з якими можна працювати у своєму проекті. А зрозуміла та обширна документація на середу розробки дає змогу швидко орієнтуватись у можливостях елементів цього ігрового двигуна.

Також одним із плюсів цього ігрового двигуна є умови ліцензування. Unity цілком безкоштовний для цілей навчання, а також персонального використання. Навіть якщо створити гру та розмістити її у магазинах, ліцензію треба буде платити лише в разі річного грошового прибутку гри більше ніж

100000 доларів. На даний час, ігровий двигун Unity використовували для створення великої кількості мобільних ігор, а також низки крупних ігрових проектів, таких як Escape from Tarkov (рисунок 1.7), Outer Wilds (рисунок 1.8), Cuphead (рисунок 1.9), Cult of the Lamb (рисунок 1.10).

Така кількість ігрових проектів створених за допомогою ігрового двигуна Unity показує його велику популярність та конкурентоспроможність серед інших аналогів.

Рисунок 1.7. Скріншот гри Escape from Tarkov

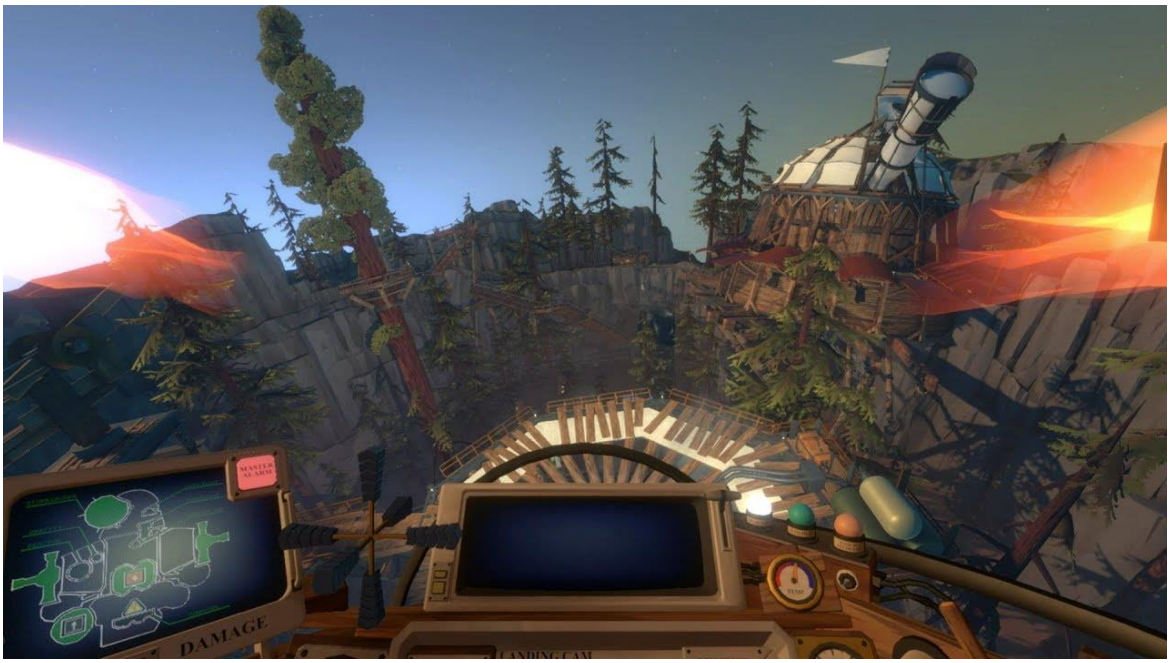


Рисунок 1.8. Скріншот гри Outer Wilds

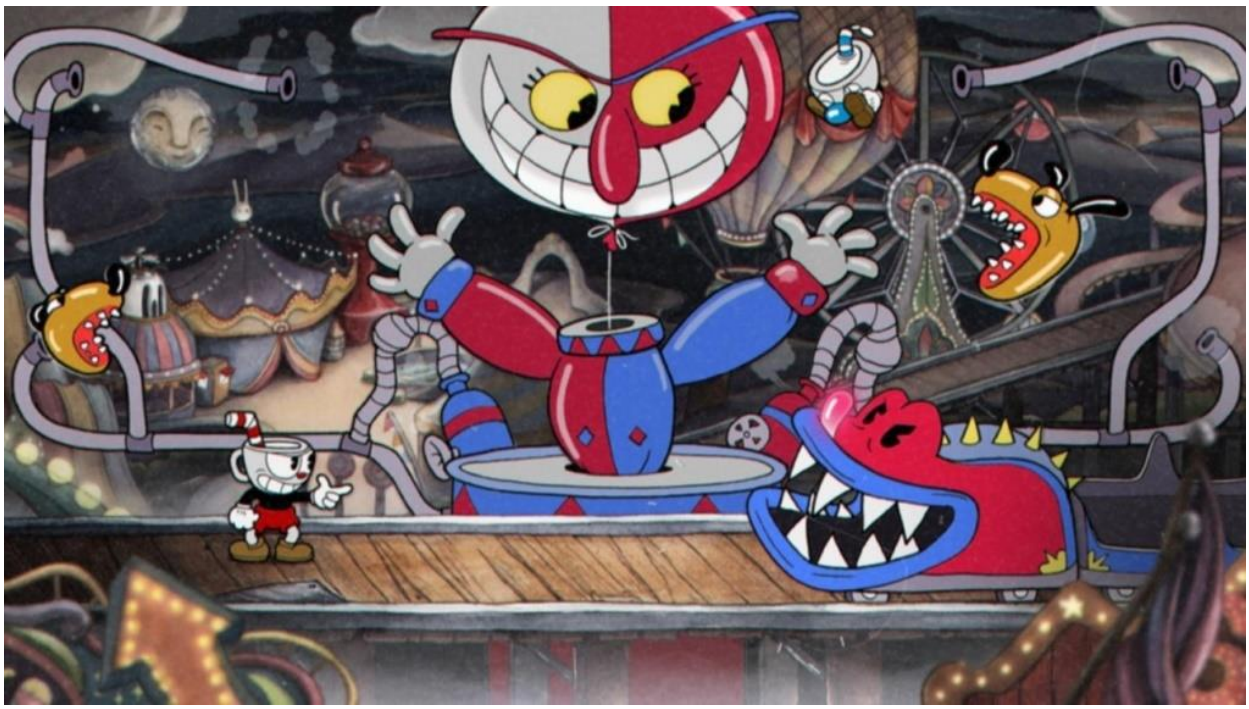


Рисунок 1.9. Скріншот гри Cuphead

Рисунок 1.10. Скріншот гри Cult of the Lamb

Для написання коду гри використовується Visual Studio 2017. Visual Studio є пакетом середи розробки програмних рішень від Microsoft. За допомогою цієї середи розробки можна створювати проекти із використанням різних мов програмування, баз даних, бібліотек, а також інших програмних рішень. Visual Studio представляє собою зручний редактор коду із можливістю

навігації по проекту, його бібліотекам та редагуванню посилань в проекті та між проектами.

Ліцензія за якою поширюється Visual Studio є дуже гнучкою. Для використання середи розробки в цілях освіти, науки, персонального та некомерційного використання, вона розповсюджується безплатно. Окрім того між безкоштовною та платною версією немає функціональних різниць.

Adobe Photoshop CS6 використовується для створення, або редагування растрових зображень. Кількість інструментів для редагування та створення зображень дуже велика та дозволяє виконувати будь-які роботи, від простих скетчів, до генерації частин зображення за допомогою нейромереж у пізніх версіях програмного забезпечення. В цій програмі можна не тільки працювати із зображеннями, а ще й додавати та редагувати тексти.

Це програмне забезпечення розповсюджується лише за ліцензією, але має безкоштовну пробну версію, яку й використано для створення спрайтів та зображень для проекту цифрової гри.

1.5 Проектування основних модулів гри, принцип їх роботи

Перед початком виконання реалізації проекту, необхідно виконати проектування основних модулів гри. Для цього спочатку треба розділити проект на модулі, згідно концепції модульної розробки цифрових ігрових проектів. Всього можна виділити наступні основні модулі проекту:

- Код ворожих об’єктів;
- Код гравця;
- Код інтерфейсу;
- Код ігрових систем.

Кожний із таких змістовних модулів має перелік коду, який належить до модулю. Схему основних модулів та коду, що належить до нього можна побачити на рисунку 1.11:

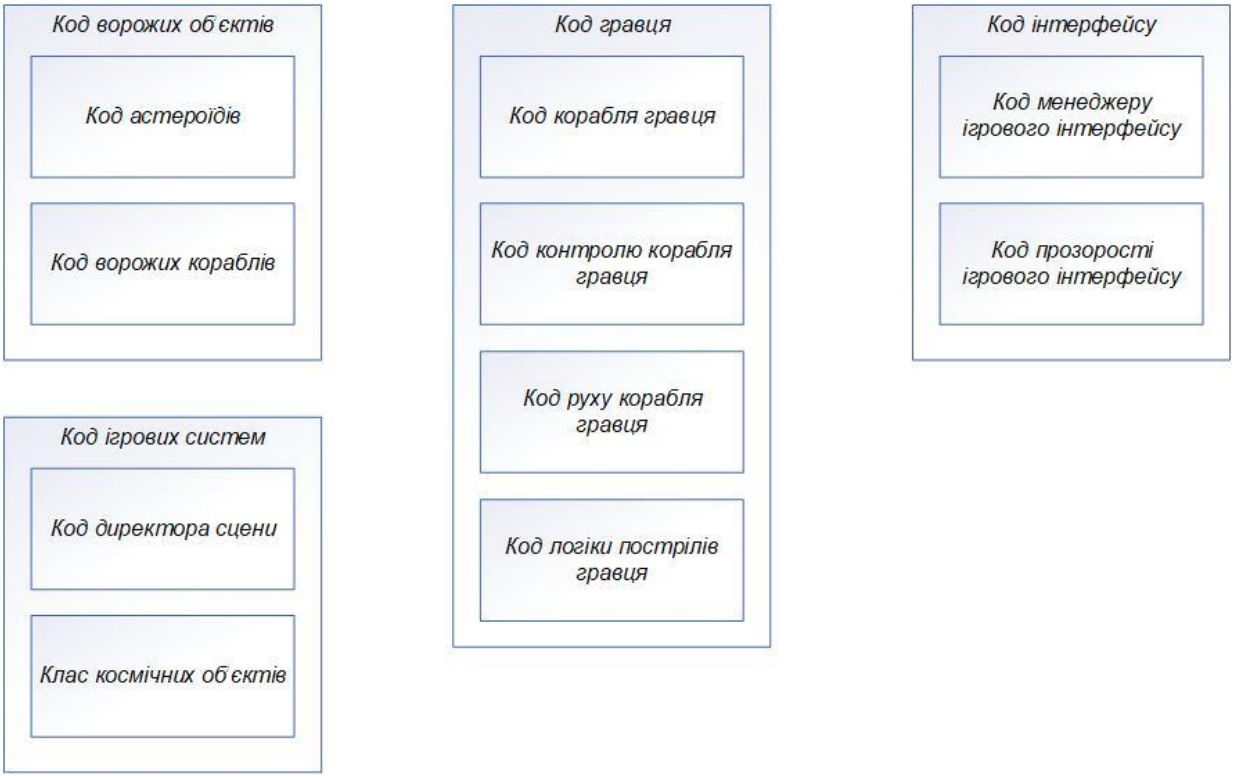


Рисунок 1.11. Схема основних модулів гри та коду, що належить до них

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. Схему змісту складових типового ворожого об’єкту можна побачити на рисунку 1.12.

Як можна бачити з рисунку 1.12, об’єкт має логічну структуру із коду, який відповідає за його переміщення, ігрову логіку такого об’єкту, а також, за потреби, створення додаткових параметрів, методів, або функцій об’єкту. Останній пункт є можливим через створення загального класу космічних об’єктів, який є основою для будь-якого об’єкту на сцені, що має безпосередню та активну участь у ігровому процесі.

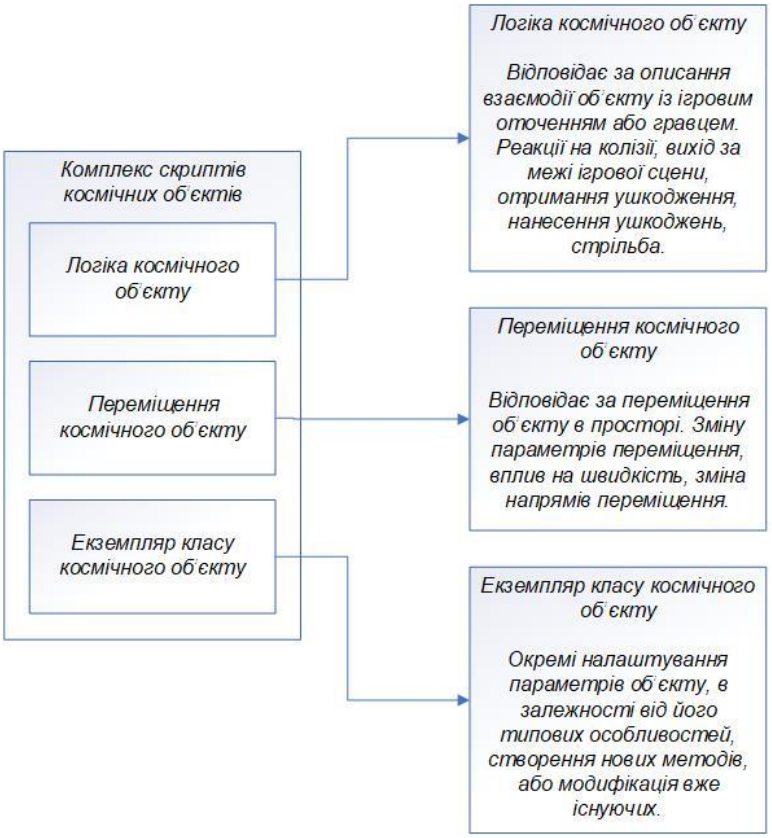


Рисунок 1.12. Схема змісту складових типового ворожого об’єкту

Схему структури загального класу космічного об’єкту можна побачити на рисунку 1.13.

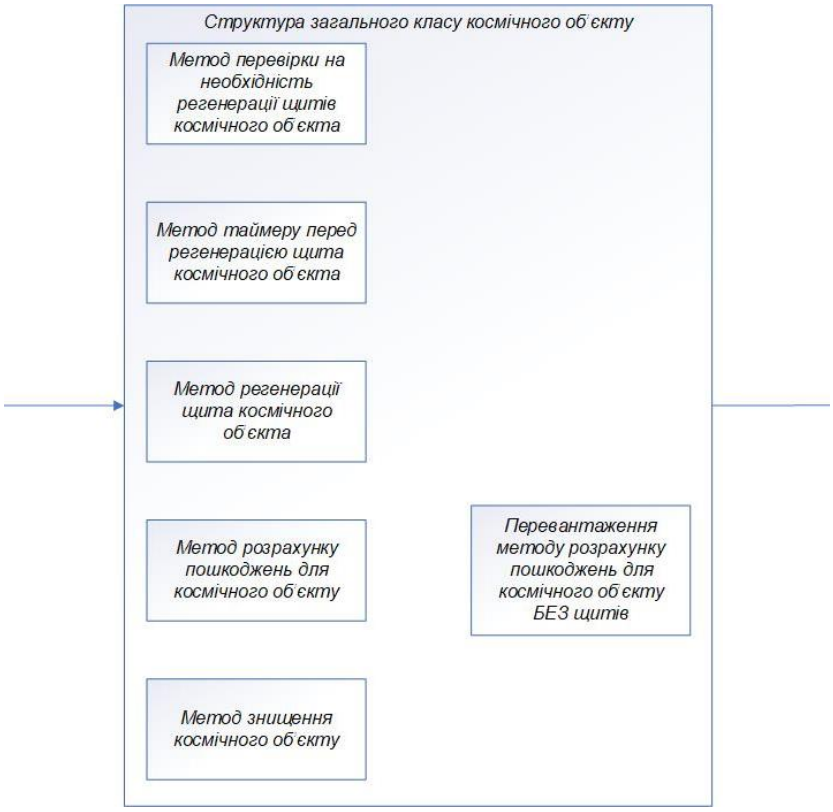


Рисунок 1.13. Схема структури загального класу космічного об’єкту

Загальний принцип взаємодії між модулями гри описується, як наскрізна передача даних від одного модулю до іншого. Єдиною особливістю є «Директор сцени», який являє собою зв'язуючим ланцюгом між ігровим процесом за інтерфейсом. Це зроблено через бажання виключити передачу неправильних даних між об'єктами, а також більш централізовану роботу із модулями. 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, адже робота по інтеграції буде зводитись лише до безпосередньо додавання нового елементу або об'єкту, та включення його в перелік об'єктів «Директора сцени». 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

Окремо має сенс розглянути роботи «Директора сцени». Це особливий об'єкт, який займається передачею даних між ігровим інтерфейсом – через «Менеджер ігрового інтерфейсу» – та ігровим процесом. Окрім того, «Директор сцени» виконує роль зв'язуючого між об'єктами ігрового процесу, та об'єктами. Наприклад, саме цей об'єкт керує кількістю та типом ворожих об'єктів на сцені, їх поведінкою а також просуванням самого ігрового процесу.

Є також подібний об'єкт до «Директора сцени» - «Менеджер ігрового інтерфейсу». Його роль складається у передачі даних від «Директора сцени» до ігрового інтерфейсу, а також, в разі необхідності, передачу даних від ігрового інтерфейсу до «Директора сцени» та далі до цільових об'єктів. Такий прошарок зроблено також навмисно для того, щоби уникнути множення зв'язків між

об’єктами та ігровим інтерфейсом. Також, не є доцільним створювати посилання на елементи інтерфейсу від кожного об’єкту на сцені, що потенційно може посприяти на погіршення продуктивності гри.

1.6 Реалізація основних модулів гри

Для виконання завдання дипломного проектування необхідно реалізувати ряд основних модулів, зо були визначені у попередніх розділах. Не дивлячись на те, що проект заявлений як 2D-гра, проект буде створено на базі пресету 3D-проекту Unity (рисунок 1.15). Є декілька причин для цього.

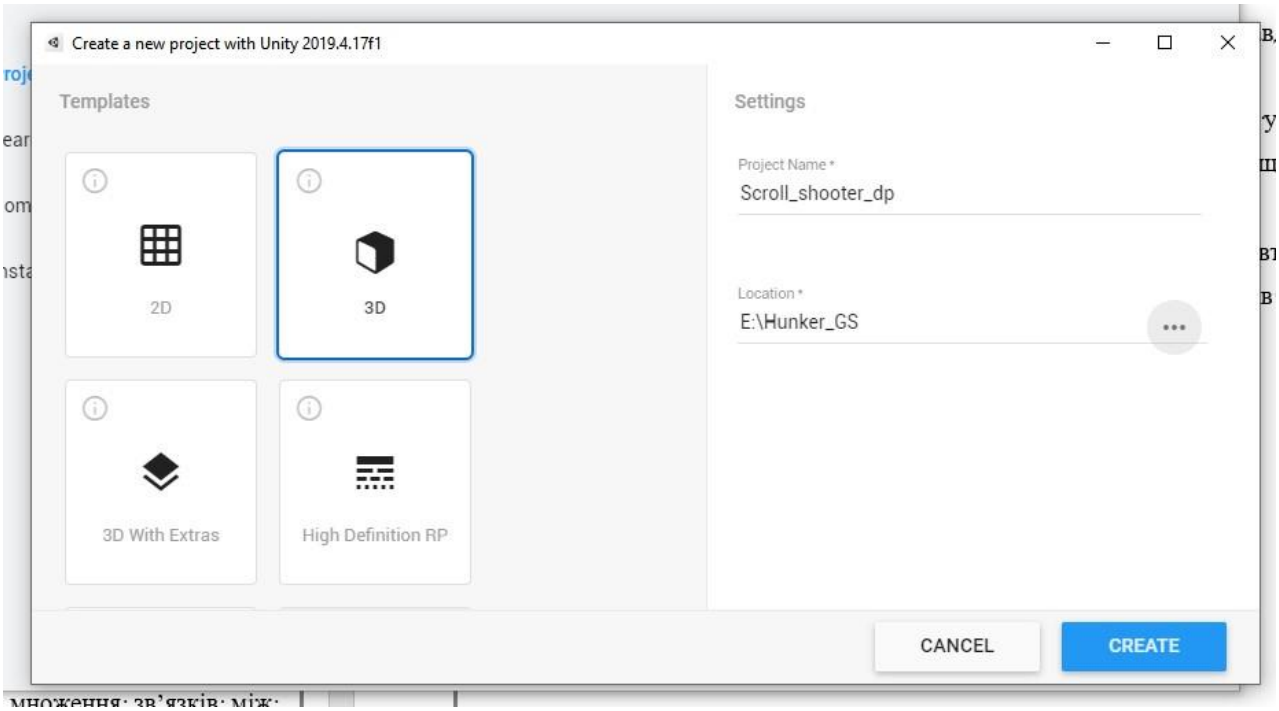


Рисунок 1.15. Створення проекту скролл-шутера у Unity

Головною причиною є можливість використовувати 3D-моделі для об’єктів гравця, оточення або моделей ворогів. Виконання проекту у трьох вимірах дасть можливість використовувати реалістичну модель фізичної взаємодії між об’єктами в грі. Якщо проект був би виконаний із використанням тільки спрайтових зображень, то було б складно реалізувати правдоподібну картину взаємодії об’єктів один із одним.

Також, використання 3D-шаблону дає змогу обирати варіанти управління гравцем, а також переміщення об’єктів в просторі, ніж якщо проект реалізовувався на базі 2D-шаблону.

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».

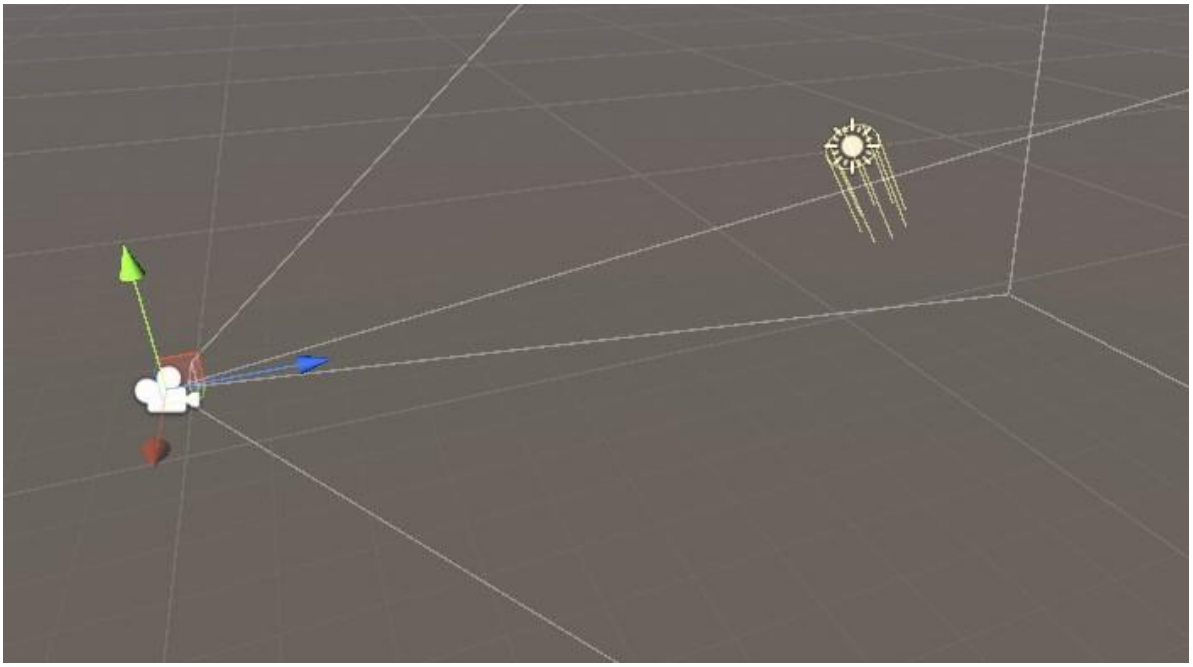


Рисунок 1.16. Стартові об’єкти в новому проекті Unity

Наступним кроком є створення у проекті ігрових об’єктів для реалізації функціоналу основних модулів гри. Для виконання цієї мети, буде достатньо розмістити на сцені об’єкт, який буде належати гравцю, а також створити об’єкт, що буде грати роль астероїда. На рисунку 1.17 можна побачити об’єкт гравця, створений для проекту, а також та рисунку 1.18 можна побачити об’єкт астероїда, також створений для проекту:

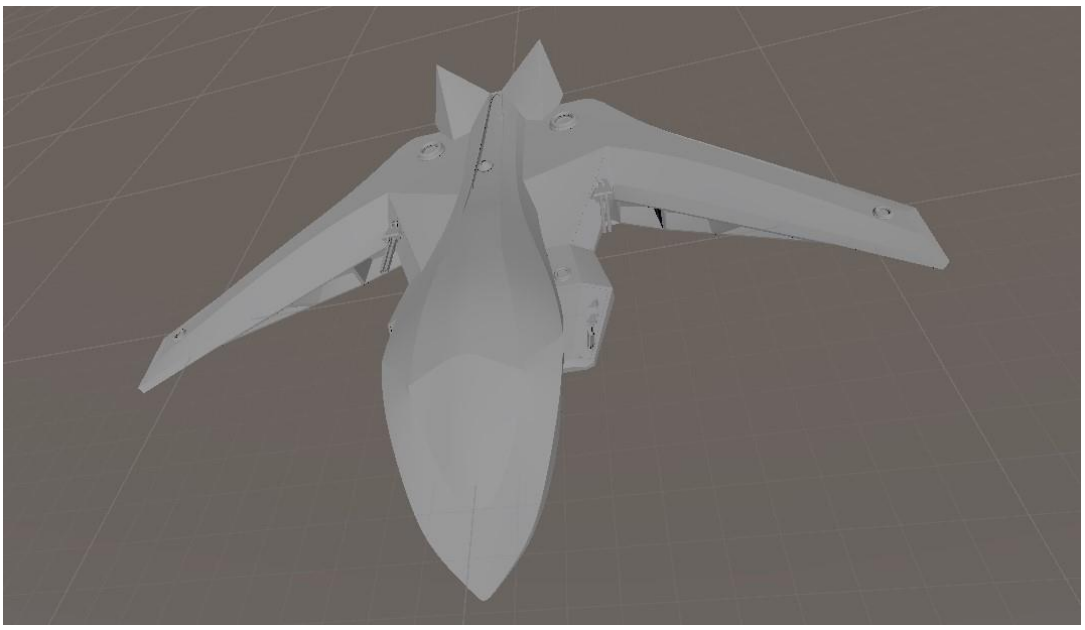


Рисунок 1.17. Об’єкт гравця створений для проекту гри

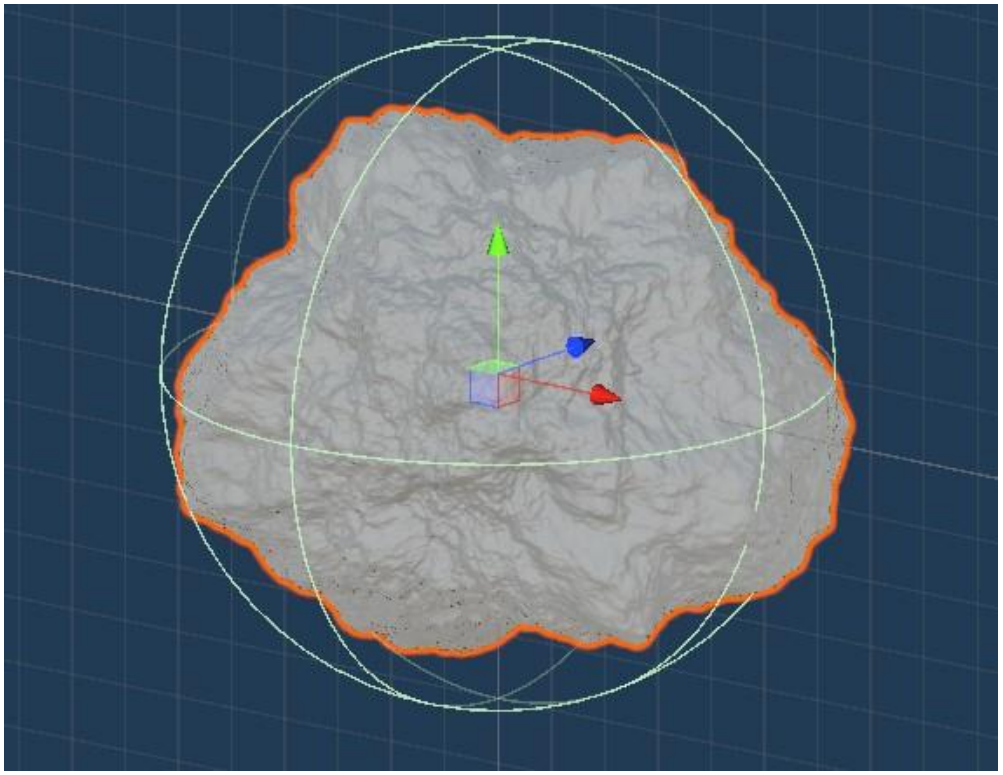


Рисунок 1.18. Об’єкт астероїду створеного для проекту гри

Цих об’єктів буде достатньо для реалізації основних модулів для ігрового процесу, а також написання для нього коду скриптів. Якщо об’єкт гравця використовується, просто як новий об’єкт на сцені, то логіка використання моделі астероїда інакша, про що буде написано нижче у відповідній частині роботи. Розміщення нових об’єктів на сцені в Unity відбувається через панель

Hierarchy, яка зображена на рисунку 1.19. Через цю панель можна додавати на сцену різні шаблони об’єктів, від примітивів до складних об’єктів оточення, як відео програвач.

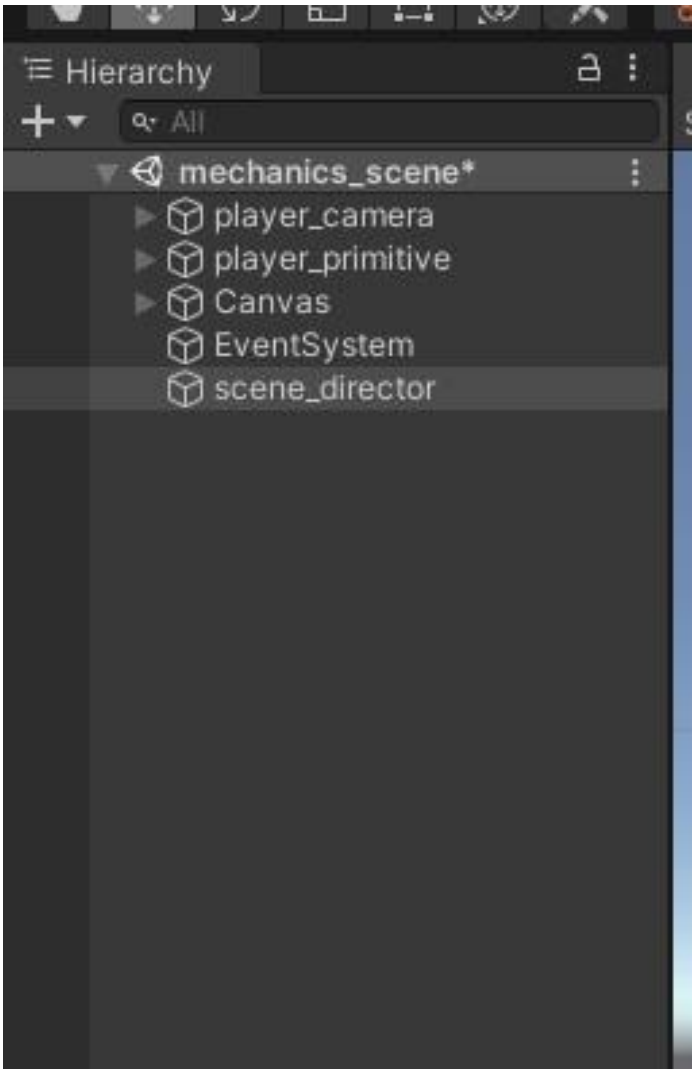


Рисунок 1.19. Зовнішній вигляд панелі Hierarchy ігрового двигуна Unity

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. Процес досить простий: необхідно створити полотно інтерфейсу у переліку об’єктів сцени, після чого поступово додавати нові елементи саме в це полотно. 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 1.20:

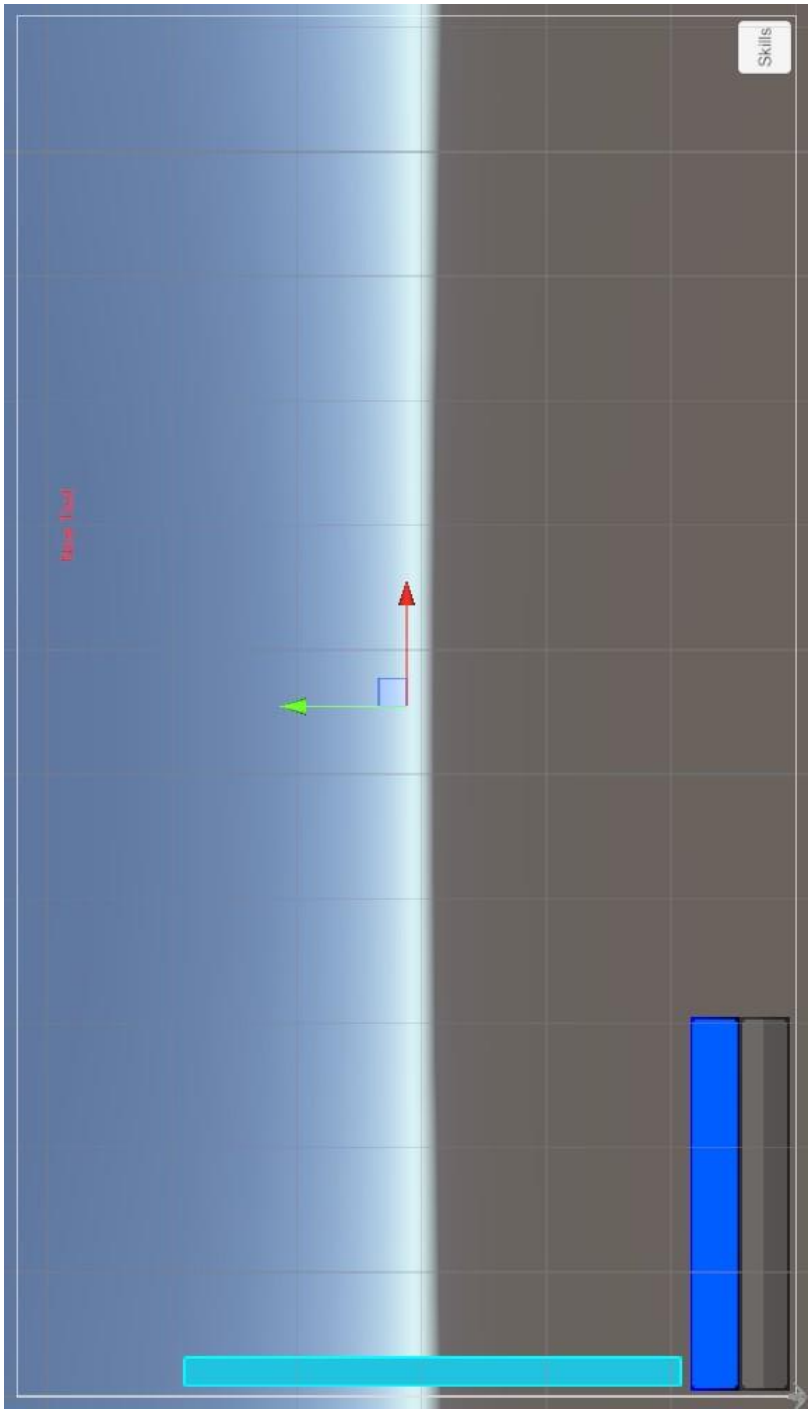


Рисунок 1.20. Ігровий інтерфейс гри

Панелі для відображення стану міцності корпусу, щитів та енергії корабля були створені за одним принципом, комбінування елементів інтерфейсу, який складається із склайдеру та двох зображень – перше відповідає за безпосередньо полосу заповнення, а друге за рамку панелі. Оскільки панель виконана за допомогою слайдеру, у майбутньому дуже просто можна змінювати значення об’єму щитів, міцності корпусу чи енергії гравця, що дає змогу

створювати модифікації, або бонуси на рівні. Також, зміна показника параметру виконується зміною показника слайдеру, значення якого є процентною шириною зображення панелі. Приклад побудови панелі можна побачити на рисунку 1.21, а також на рисунку 1.22 можна побачити приклад роботи панелі:

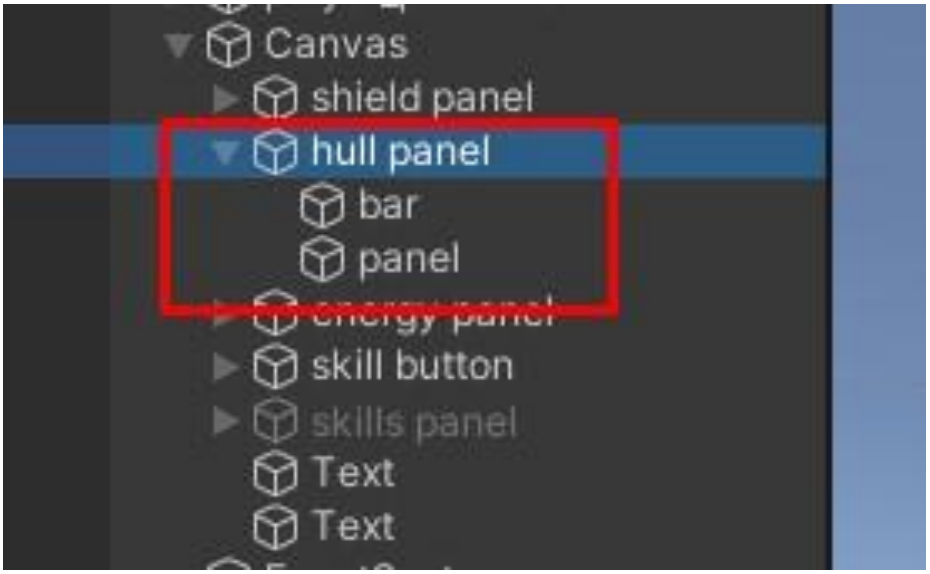


Рисунок 1.21. Структура панелі міцності корпусу гравця

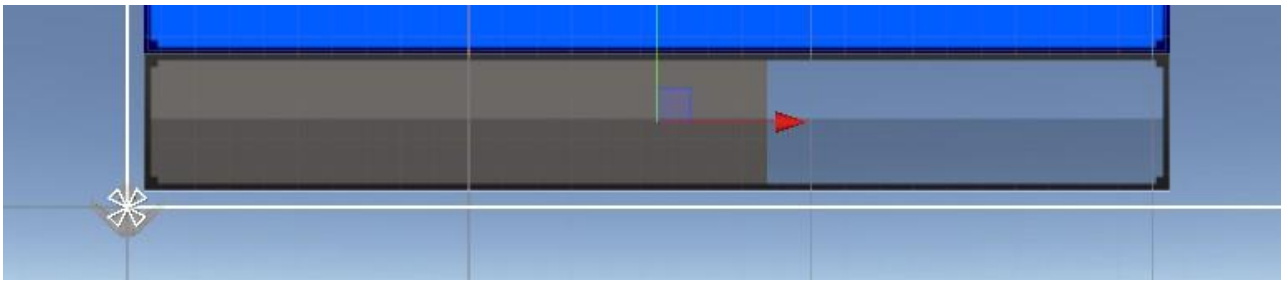


Рисунок 1.22. Приклад роботи панелі міцності корпусу гравця

Окремо слід відзначити створення пустого ігрового об’єкти із довільним розміщенням. Цей ігровий об’єкт буду вміщати в собі скрипт «Директора сцени». Це зроблено в першу чергу через те, що скрипти, написані для Unity працюють лише тоді, коли додані до ігрового об’єкту на сцені. Можна було б додати цей скрипт до будь-якого об’єкту, але слід пам’ятати, що якщо об’єкт буде знищено, то скрипт завершить свою роботу. Тому було створено пустий об’єкт, до нього доданий скрипт. Таким чином можна буде дуже швидко отримувати до нього доступ під час редагування сцени, або під час відладки роботи гри. Тепер можна запустити проект та побачити зображення схоже на ігровий процес. Переглянути результат можна на рисунку 1.23. Втім, на даний

час всі не має можливості керувати процесом.

Рисунок 1.23. Гра після додавання об’єкту гравця та ігрового інтерфейсу

Коли із розміщенням всіх необхідних для роботи елементів завершено, можна переходити до написання коду основних модулів гри. Для цього спершу необхідно створити головний клас космічних об’єктів.

Цей клас створюється із ім’ям ship_parent.cs. В основу цього класу покладені основні методи для взаємодії та функціонування космічного об’єкту, як отримання параметрів об’єкту, за запуск основних методів по обробці отриманих пошкоджень. Оскільки цей клас не буде використовуватись напряму, в основу його коду лежить ідея про зовнішнє використання, тому всі методи написані максимально уніфіковано, щоби при зверненні до них не було проблем із параметрами методів. Нижче представлено код перевантаження одного з методів, що відповідає за розрахунку пошкоджень:

```
public virtual void damage_calculate(float damage, ref float hull, out bool
is_hull_break)
{
hull -= damage;
if(hull <= 0)
{
hull = 0;
is_hull_break = true;
}
else
{
is_hull_break = false;
}
}
```

Як можна бачити із коду методу, він для роботи використовує як параметри референційні змінні, що дає змогу звертатись до цього із середини класів нащадків, використовуючи ці методи, як вмонтовані.

Наступник кроком є робота над модулем гравця. Він складається із:

- Код корабля гравця;
- Код контролю корабля гравця;
- Код руху корабля гравця;
- Код логіки пострілів корабля гравця.

Кожний з цих пунктів буде представляти собою окремий скрипт, що буде виконувати визначені дії.

Основним скриптом буде «Код корабля гравця». Він реалізований у скрипті із назвою Ship_player.cs. Як було написано вище, кожний об’єкт на сцені, що буде взаємодіяти із гравцем матиме в своїй основі свій клас, що буде нащадком головного класу космічних об’єктів ship_parent.cs. Проте, на відміну від батьківського класу, у новому будуть додані параметри гравця, як міцність корпусу, ємність щитів, запас енергії та інші параметри, що сприяють на ігровий процес – загалом 13 параметрів. Ці параметри не є доступними зовні, але виведені до панелі середи розробки завдяки команді [SerializeField] під час їх створення. Переглянути ці параметри на панелі Inspector двигуна Unity можна на рисунку 1.24:

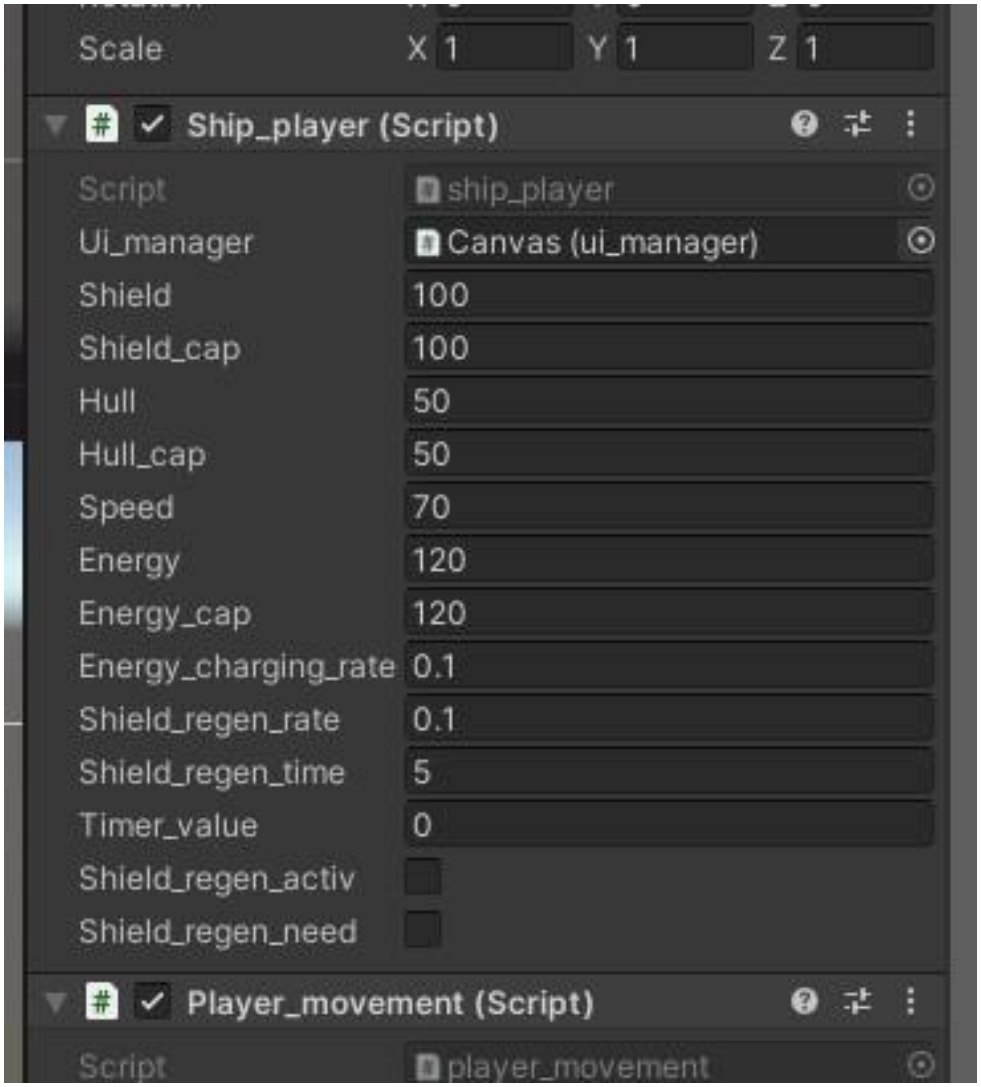


Рисунок 1.24. Параметри корабля гравця на панелі Inspector

Також, цей клас відрізняється від інших тим, що має ініціалізувати ігровий інтерфейс під час старту сцени, блокування чи розблокування керуванням корабля під час паузи.

Серед інших методів, що були додані саме до цього класу є:

- Метод ремонту корпусу гравця;
- Метод використання енергії;
- Метод відновлення енергії;
- Передачі швидкості гравця;
- Перемикання доступу до логіки корабля гравця;
- Метод для отримання пошкоджень.

Слід лише відмітити останній метод. Він не переписує існуючий метод, перевантаження якого було приведено вище, а викликає процес розрахунку. Оскільки заздалегідь немає можливості знати, які параметри є у космічного об’єкта, тому їх передачу має виконувати сам об’єкт, звертаючись до свого методу отримання пошкоджень, що в свою чергу звертається до вмонтованого методу розрахунку пошкоджень.

Виконання коду контролю корабля гравця та його переміщення зроблені в окремих скриптах player_control.cs та player_movement.cs відповідно. Це зроблено через намагання розділити управління кораблем в цілому від інших процесів. Таким чином буде значно простіше змінювати умови керування, якщо така потреба з’явиться. Скрипт переміщення викликається із скрипту player_control.cs, та виконує фізично коректну зміну координат положення об’єкта гравця у просторі. Також виконується перевірка виходу гравця за межі ігрової сцени. Нижче представлений приклад коду переміщення гравця в просторі ігрової сцени:

```
void FixedUpdate(){
if (can_move){
if (transform.position.x >= 205.0f){
if (Input.GetAxis("Horizontal") > 0)
keyboard_force_x = 0.0f;
```

```
else if (Input.GetAxis("Horizontal") < 0)
keyboard_force_x = Input.GetAxis("Horizontal");}
else if (transform.position.x <= -205.0f){
if (Input.GetAxis("Horizontal") > 0)
keyboard_force_x = Input.GetAxis("Horizontal");
else if (Input.GetAxis("Horizontal") < 0)
keyboard_force_x = 0.0f;}
else
keyboard_force_x = Input.GetAxis("Horizontal");
if (transform.position.y >= 105.0f){
if (Input.GetAxis("Vertical") > 0)
keyboard_force_y = 0.0f;
else if (Input.GetAxis("Vertical") < 0)
keyboard_force_y = Input.GetAxis("Vertical");}
else if (transform.position.y <= -105.0f){
if (Input.GetAxis("Vertical") > 0)
keyboard_force_y = Input.GetAxis("Vertical");
else if (Input.GetAxis("Vertical") < 0)
keyboard_force_y = 0.0f;}
else
keyboard_force_y = Input.GetAxis("Vertical");
keyboard_force = new Vector3(keyboard_force_x, keyboard_force_y, 0.0f);
keyboard_force = keyboard_force.normalized * player_speed;
player_rigidbody.velocity = keyboard_force;
moving_rotation();}
else
player_rigidbody.velocity = Vector3.zero;
}
```

Спочатку даний код перевіряє, чи може гравець виконувати переміщення по сцені завдяки змінній бульовій can_move. Ця змінна перемикається, якщо

гравець був знищений, або було відкрита панель меню. Якщо значення цієї змінної буде true, тоді починається перевірки, чи вийшов гравець за межі полів ігрової сцени. Межи визначаються світовими координатами. Якщо гравець знаходиться в межах тих координат, то зчитуються натискання на кнопки управління. Після чого дані зберігаються та в кінці перевірок передаються на обробку до змінної keyboard_force. Вона перетворює отримані дані в вектори, нормалізує їх, після чого придає силу поштовху об’єкту гравця у потрібному напрямку. Також, окремо виконується метод крену корабля гравця у напрямку руху для візуальної віддачі ігрового процесу.

Також, згідно схеми проекту, реалізовано код скрипта для логіки пострілів корабля гравця player_bullet_logic.cs. Цей скрип головним чином викликається із скрипту player_control.cs та створює фізичну кулю, яка рухається перпендикулярно кораблю верх по екрану. Можна було і не створювати фізичну кулю, а розраховувати враження цілі за допомогою променів, але вибір пав на фізичну кулю через можливість у подальшому створювати різні варіанти видів зброї та унікальну взаємодію пострілів із оточенням. Також, цей скрипт виконує переміщення фізичного об’єкту кулі у просторі. Таким чином кожна куля яка згенерована пострілом гравця має своє унікальне фізичне переміщення. Нижче приведений код, який реалізовує переміщення кулі в просторі:

```
private void FixedUpdate()
{
bullet_force = bullet_force.normalized * bullet_speed;
bullet_rigid_body.velocity = new Vector3(bullet_force.x, bullet_force.y, bullet_force.z);
if (transform.position.y > 115.0f)
{
Destroy(this.gameObject);
}
}
```

Наступний модуль має на увазі створення коду для ворожих об’єктів. На даному етапі розробки буде достатньо створити астероїди, що будуть слугувати першими ворогами. Вони у своїй архітектурі коду схожі із комплексом кодів для гравця, окрім відсутності коду для управління астероїдами.

Так само створюється екземпляр класу космічного об’єкту від батьківського ship_parent.cs – ship_asteroid.cs. В цьому об’єкті додаються свої параметри та методи, схожі на ті, що були у гравця. Чуттєвою різницею є відсутність параметру щитів, тому код для отримання ушкоджень працює інакше і звертається до перевантаження методу розрахунку пошкоджень.

Код для переміщення працює за тим же принципом, як і у гравця, виконуючи переміщення космічного об’єкту за фізичними законами у просторі, а не просто поступово змінюючи його координати.

Код скрипта логіки в більшій мірі перевіряє вихід космічного об’єкта за межі ігрової сцени, його знищення, та нанесення пошкоджень при контакту із кораблем гравця.

Наступній модуль для реалізації – Модуль інтерфейсу. Він в своїй проектній архітектурі має всього два скрипти. Менеджер ігрового інтерфейсу, а також код що забезпечує прозорість елементів інтерфейсу.

ui_transperency.cs виконує задачу створення ефекту прозорості на елементах інтерфейсу у разі, якщо гравець заведе свій корабель за панелі показників міцності корпусу, або ємності щитів. Принцип роботи такого скрипту дуже простий. Він отримує посилання до елементів інтерфейсу, та кожний кадр перевіряє положення гравця у ігровому просторі. Якщо координати гравця попадуть будуть збігатись із координатами панелей, то скрипт зменшує показники властивості alpha зображень елементів інтерфейсу.

Менеджер ігрового інтерфейсу реалізований в коді ui_manager.cs. Головна задача цього коду передавати за потребою актуальні дані до інтерфейсу, а також встановлювати нові граничні значення показників міцності корпусу, ємності щита та запасів енергії. Ці процеси виконуються за допомогою відповідних методів, як на прикладі нижче:

```
public void set_new_shield_bar_cap(float new_value)
{
shield_bar.maxValue = new_value;
shield_bar.value = shield_bar.maxValue;
}
```

В даному випадку метод встановлює нове граничне значення для панелі ємності щитів. Це стає можливим через зв’язок скрипта Менеджера ігрового інтерфейсу з елементами ігрового інтерфейсу да допомогою посилань у панелі Inspector.

Останнім модулем для реалізації є Модуль ігрових систем. В нього входить код Директору сцени, та батьківський клас космічних об’єктів. Останній був реалізований раніше. Код реалізації Директора сцени написаний в скрипті scene_director.cs. Головна мета мета роботи цього коду генерувати ігровий процес, а також бути посередником між інтерфейсом та ігровими елементами.

Цікавою особливістю є реалізація генерації «випадкових» ігрових ситуацій у Директорі сцени. Саме цей код генерує ту кількість астероїдів, яку задає розробник. Окрім того, значення кількості астероїдів може задаватись динамічним чином. Самі астероїди генеруються не однакові. На початку реалізації проекту, модель астероїдів не була розміщена на ігровій сцені. Замість цього був створений префаб астероїду. 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. Самі префаби астероїдів, які вже мають додані скрипти для астероїдів, при генерації на екрані створюються із випадковими параметрами розмірів, швидкості та запасу міцності «корпусу». Завдяки цьому можна генерувати різні ситуації на ігровій сцені, як показано на рисунку 1.25 та рисунку 1.26.

Рисунок 1.25. Приклад генерації астероїдів на початку рівня №1

Така система генерації випадкових значень, дає можливість створювати унікальні ситуації, що будуть відрізняти ігровий процес від однієї гри до іншої. Окрім того, архітектура реалізованого коду дає можливість у подальшому впровадити «срежисовані» сцени, коли значення будуть видаватись не випадково, а заздалегідь визначені, наприклад при досягненні якоїсь кількості ігрового рахунку.

1.26. Приклад генерації астероїдів на початку рівня №2

Проект отримав свою кінцеву структуру для даної ітерації розробки. Структура проекту генерується ігровим двигуном на початку роботи автоматично. Переглянути кінцеву структуру проекту можна на рисунку 1.27:

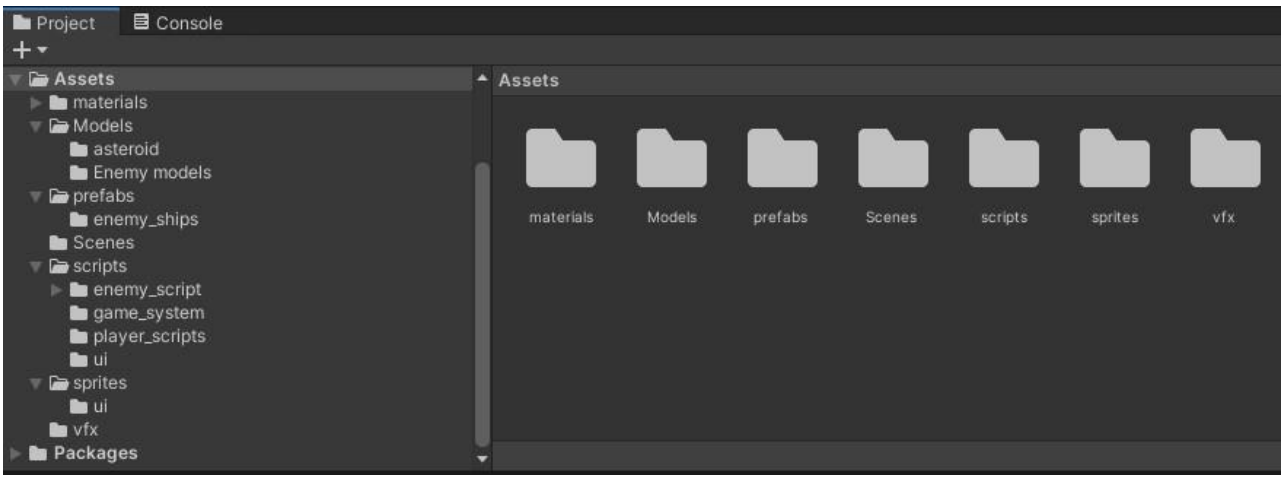


Рисунок 1.27. Структура проекта гри в Unity

Є папки, які належать двигунові і працювати із ними частіше за все не доводиться, тільки у випадках внесення змін у тонкі механізми роботи ПЗ.

Основі ж файли та папки проекту знаходяться у каталозі Assets, зміст якого можна бачити на рисунку 1.27. Файли проекту було структуровано за призначення та вмістом. Так, всі сцени знаходяться в папці Scenes, а скрипти для гри в каталозі scripts, в середині якого також створені свої каталоги для кожного модуля гри та її підмодулів. Також, в проекті були виконані роботи по створенню таких файлів як матеріали, моделі, спрайти та визуальні ефекти. Всі ці файли розміщуються у відповідних каталогах.

Використання таких файлів виконується не за посиланням в коді, а за допомогою посилать у панелі Inspector середи розробки ігрового двигуна. Тобто для використання матеріалу, не потрібно посилатись на нього, а достатньо перетягнути матеріал у видповідний пункт компоненту Material того ігрового об’єкту матеріал якого необхідно змінити. Такий самий принцип роботи із файлами використовується і для всіх інших робочих файлів.

Схематичне зображення файлової структури проекту можна переглянути на рисунку 1.28:

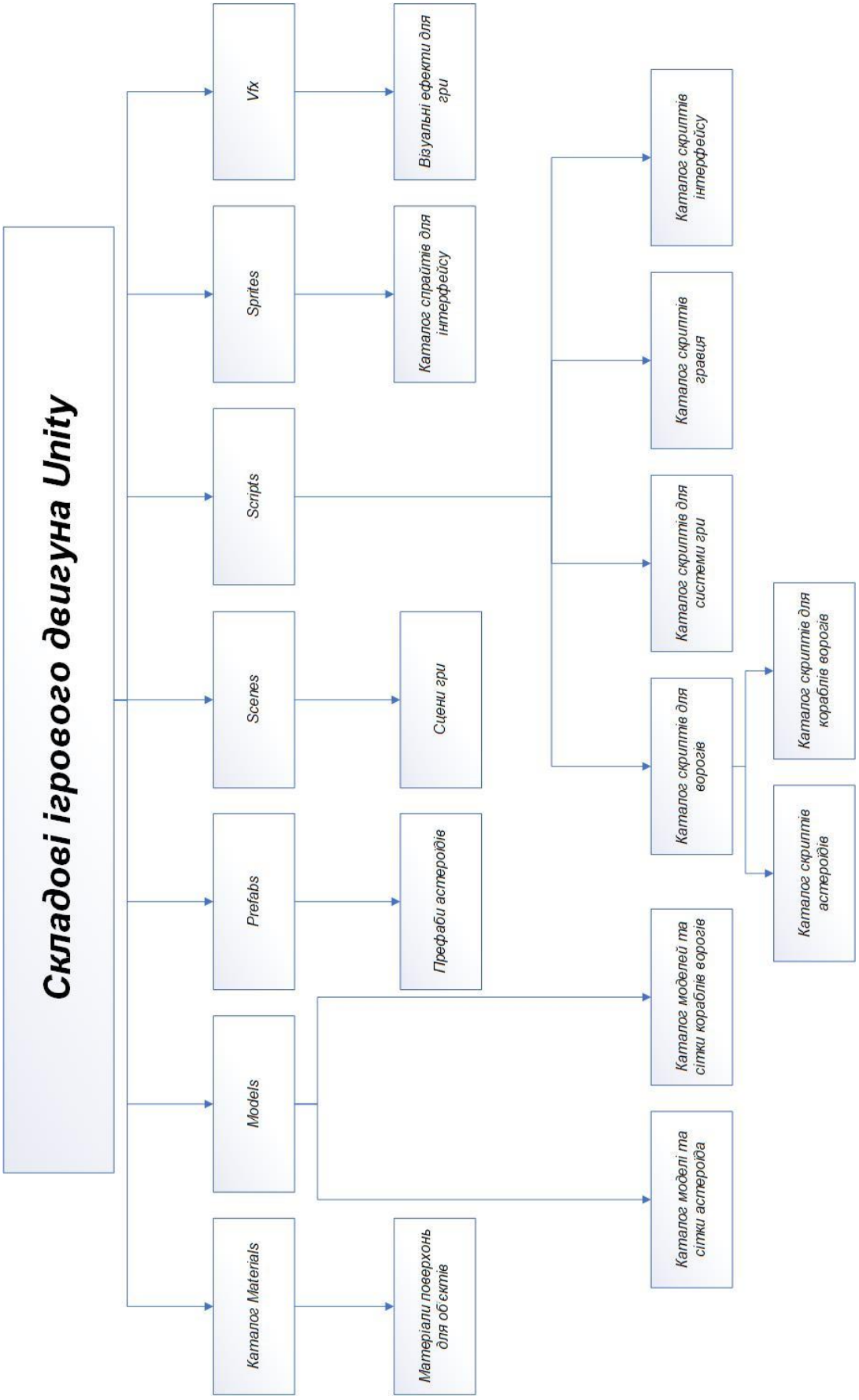


Рисунок 1.28. Схематичне зображення

1.7 Тестування проекту на працездатність

Завдяки можливостям ігрового двигуна Unity тестування гри можна проводити безпосередньо всередині самої середи розробки. Для цього достатньо розмістити ігрові об’єкти на свої місця на сцені, виставити всі необхідні ігрові параметри. Після виконання перших налаштувань, необхідно натиснути на кнопку «Play». Після чого гра запуститься у середі розробці, як зображено на рисунку 1.29:

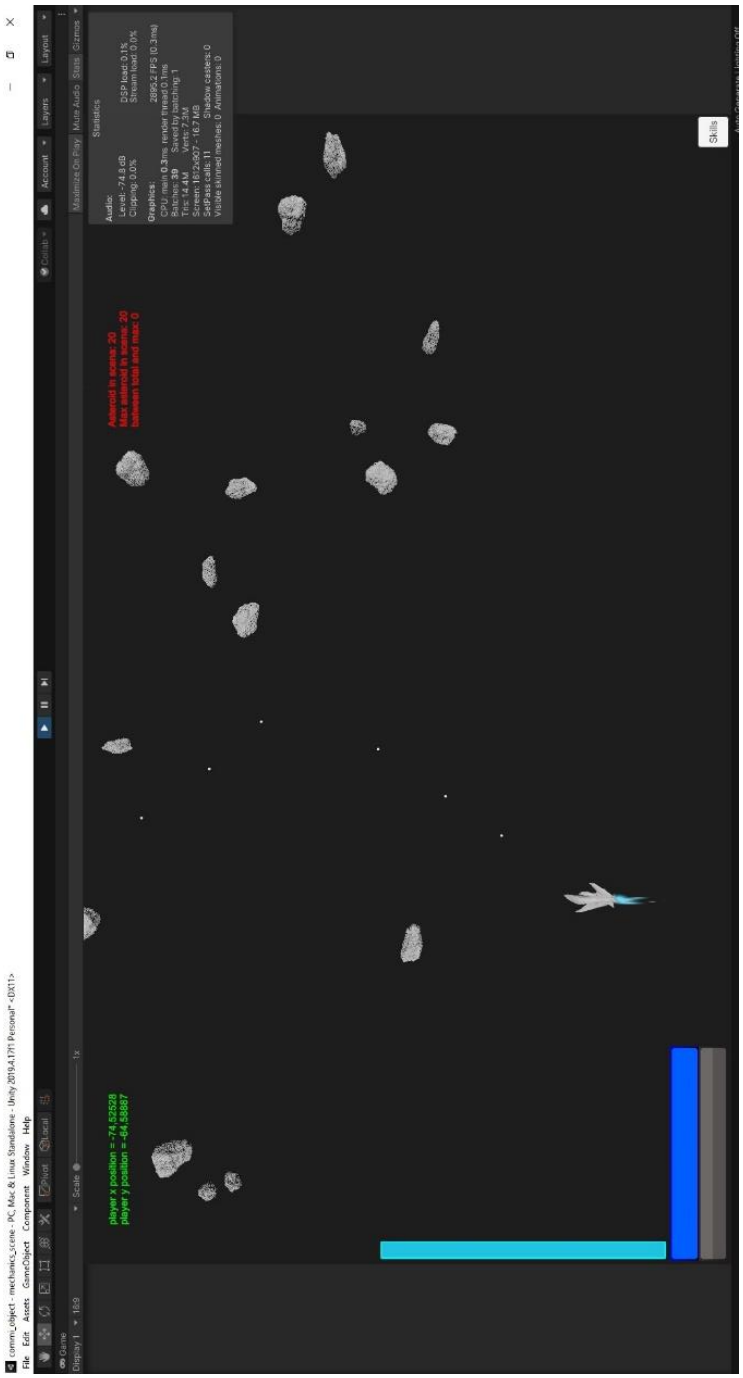


Рисунок 1.29. Приклад відладки гри у середі розробки

Основною метою тестування є перевірка справності керування кораблем гравця, та реакції середовища на його дії. Причини для такого тестування лежать у фізичній системі переміщення корабля гравця та астероїдів і необхідно переконатись, що їх взаємодія не приводить до некоректних ситуацій. Тестування показало справну роботу фізики, астероїди коректним чином реагують на зіткнення з гравцем, наносить йому пошкодження, а також, якщо не руйнуються при зіткненні, змінюють напрямок руху.

Наступним етапом тестування є перевірка реєстрації пошкодження астероїдами від пострілів гравця. Для цього достатньо пограти деякий час та переконатись, що астероїди отримують пошкодження та знищуються. Таким самим чином, можна перевірити реєстрацію пошкодження астероїдами гравця, достатньо зіткнутись із ними.

Додатково, через тестування пошкодження корабля гравця астероїдом, також перевіряється робота елементів інтерфейсу, які коректним чином відображають отримані пошкодження.

Таким чином, виконавши тестування, можна переконатись, що гра коректно виконує основні вимоги сформовані під час проектування проекту. Подальші модифікації модулів гри можуть розширити ігровий процес.

2 ЕКОНОМІЧНА ЧАСТИНА

2.1 Резюме

В даному дипломному проекті розроблено та реалізовано початкову версію 2D-гри в жанрі скролл-шутер із використанням в ній основних ігрових механік, притаманних даному ігровому жанру. В повній мірі було виконано роботу над програмною частиною інтерфейса та його взаємодією із ігровим процесом. Використання принципів модульної розробки дало можливість і далі модифікувати гру, додаючи нові ігрові елементі, механіки або режими.

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. Оцінка якості програмного продукту включає визначення трудомісткості і вартості його створення.

2.2 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку. Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначається обсяг програмних засобів, у тисячах умовних машинних команд програми аналога.

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

Таблиця 2.1. Каталог аналогів

Найменування ПП	Обсяг функції ПП – V_o, усл. машинних командах.
1. ПП автоматизації засобів по каталогу	680 – 7000
2. ПП автоматизованих розрахунків	1300 – 8600
3. ПП імітаційного моделювання	7800 – 8800

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

Таблиця.2.2. Таблиця трудомісткості

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

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

Трудомісткість програмного продукту визначається 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:

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

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

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

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (Таблиця 2.3); Кн – поправочний коефіцієнт, що враховує ступінь новизни (Таблиця 2.4); Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (Таблиця 2.5).

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

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

Для нашого варіанта виділено сірим кольором.

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

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

Для нашого варіанта виділено сірим кольором.

Таблиця 2.5. Значення коефіцієнта ступеня використання в розробці типових програм

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

Для нашого варіанта виділено сірим кольором.

Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$Т_{тз}=Та*L_1*К_н=160,3 *0,12*0,8= 15,3$(люд/годин)

Трудомісткість розробки технічного проекту

$Т_{тп}=Та*L_2*К_н =160,3 *0,11*0,8 = 14,1$ (люд/годин)

Трудомісткість розробки робочого проекту

$Т_{рп}=Та*L_3*К_н*К_т=160,3 *0,61*0,8*0,8= 62,5$ (люд/годин)

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}=2$ (стр), розробка ТП $N_{тп}=$ 15 (стр), розробка

робочого проекту $N_{рп}$=15 (стр), пояснювальна записка відповідно $N_{пз}$=46 (стр). Розрахунок зведений у таблицю 2.6.

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{Ртз}$=15,38	$T_{кк}$=0,7*$N_{тз}$= 0,7*2=1,4	$T_{нк}$=0,15*$N_{тз}$=0,15*2=0,30
2.Розробка ТП	$T_{Ртп}$=14,11	$T_{кк}$=0,7*$N_{тп}$=0,7*15= 10,5	$T_{нк}$=0,15*$N_{тп}$=0,15*15=2,25
3.Розробка РП	$T_{Ррп}$= 62,58	$T_{кк}$=0,7*$N_{рп}$=0,7*15=10,5	$T_{нк}$=0,15*$N_{рп}$=0,15*15=2,25
4.Розробка ПЗ	$T_{пз}$=1,5**$N_{пз}$= 1,5*46 =69	$T_{кк}$=0,7*$N_{тз}$=0,7*46 =32,2	$T_{нк}$=0,15*$N_{пз}$=0,15*46 =6,9
Усього, в т.ч.:	227,37		
- на розробку	$\Sigma T_р$=161,07		
- контроль керівника		$\Sigma T_{кк}$=54,6	
- нормоконтроль			$\Sigma T_{нк}$=11,7

2.3 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 6.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2023» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2023 року - 6700 гривень; мінімальну погодинну тарифну ставку – 40.46 грн.

Таблиця 2.7. Розрахунок основної заробітної плати виконавців.

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн.	Розрахунок, грн.
1.Розробка ПП	161,07	45,00	7248,15
2.Контроль керівника	54,6	65,10	3554,46
3.Нормоконтроль	11,7	65,10	761,67
Усього	-	-	ΣЗо= 11564,28

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

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПП

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	80	3.0	240,0
Разом	-	-	-	$В_{м\ і}$ =240,0
Транспортно – заготівельні Витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$= 0,1*240=24,00
Усього				$В_м=В_{мі}+В_{тр\_з=}$ 264.00

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	264.00	$В_м$ (див. табл. 2.8)
2. Основна заробітна плата	11564,28	$З_о$ (див. табл. 2.7.)
3.Додаткова заробітна плата	1156,43	$Зд = 0{,}1 \times Зо$ = 11564,28*0,1
4.Відрахування до єдиного фонду соціального внеску	2798,56	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$ =0,22* (11564,28+1156,43)
5. Накладні витрати	4625,71	$Внак. = 0{,}4 \times Зо$ =0,4 * 9084,49
6. Повна собівартість	20408,98	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.= 264.00+11564,28+1156,43+2798,56+ 4625,71

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

П=($С_п$*Р)/100=(20408,98*10)/100=2040,89грн

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

Оптова ціна (кошторисна вартість) визначається по формулі:

$Ц_о$ =$С_п$+П = 20408,98+2040,89= 22449,88 грн;

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$Ц_р$=$Ц_о$+ПДВ = 22449,88 +22449,88 *0.2 = 26939,85 грн;

3 ОХОРОНА ПРАЦІ

Згідно з ч. 1 ст. 13 Закону України «Про охорону праці» роботодавець зобов'язаний створити на робочому місці в кожному структурному підрозділі умови праці відповідно до нормативно-правових актів, а також забезпечити додержання вимог законодавства щодо прав працівників у галузі охорони праці.

Робота з комп'ютером характеризується значною розумовою напругою і нервово-емоційним навантаженням операторів, високою напруженістю зорової роботи і достатньо великим навантаженням на м'язи рук при роботі з клавіатурою ЕОМ. Велике значення має раціональна конструкція і розташовує елементів робочого місця, що важливе для підтримки оптимальної робочої пози людини-оператора. В процесі роботи з комп'ютером необхідно дотримувати правильний режим праці і відпочинку.

Дотримання норм охорони праці є спільним завданням як роботодавця, так і працівника. У вирішенні питань з охорони праці можна звернутися до законодавства України з охорони праці.

Тема дипломного проекту - Розробка комп'ютерної 2D-гри в жанрі вертикального скролл-шутеру за допомогою ігрового двигуна Unity, тому даний розділ присвячено вивченню оптимальних умов праці програміста та обов`язків з охорони праці.

3.1 Аналіз небезпечних і шкідливих факторів

На робочому місці розробника програмного забезпечення: підвищений рівень отримуваного електромагнітного випромінювання, статична електрика, високий рівень шуму, несприятливі умови мікроклімату, підвищена напруга на зір та мозок тощо.

Під час робочого процесу програміст піддається впливу великої кількості шкідливих та небезпечних факторів, а саме: шуми, вібрації, інфрачервоне випромінювання, електромагнітне випромінювання, електричний струм, емоційне та нервове навантаження, сидяче положення тіла протягом дового часу. Тому дуже

важливо забезпечити правильний нормований графік та організувати робочий процес так, щоб мінімізувати вплив усіх перелічених раніше небезпечних та шкідливих факторів.

3.2 Гігієнічні вимоги до виробничого середовища

Вимоги, що пред'являються до умов праці на виробництві, визначаються необхідністю забезпечення таких умов праці на робочому місці, при яких виключено несприятливий вплив на працездатність і здоров'я працюючих і можуть бути забезпечені оптимальні границі поділу і кооперації праці, а в кінцевому підсумку підвищення ефективності та якості праці.

3.2.1 Вимоги до приміщення

Об’ємно-планувальні рішення будівель та приміщень для роботи з ВДТ мають відповідати вимогам ДСанПІН 3.3.2.007-98. Розміщення робочих місць з ВДТ ЕОМ і ПЕОМ у підвальних приміщеннях, на цокольних поверхах заборонено. Площа на одне робоче місце становить не менше 6,0 м^2, а об’єм – не менше ніж 20,0 м^3. У приміщеннях слід щоденно робити вологе прибирання. Вони повинні бути оснащені аптечками першої медичної допомоги. При приміщеннях мають бути обладнанні побутові приміщення для відпочинку.

Для стін і робочих поверхонь використовують мало насичені (основні) кольори, для невеликих помешкань або ділянок, що рідко потрапляють у поле зору працюючих, а також для створення контрастності – кольори середньої насиченості (допоміжні), для маленьких по площі поверхонь – насичені (акценти) – як функціональне фарбування. Поверхні устаткування в приміщеннях повинні бути матовими або напівматовими для виключення випадку відблисків світла в очі працюючого, а стіни бути пофарбованими фарбами пастельних тонів.

3.2.2 Освітлення

Відповідність характеристик систем освітлення нормативним вимогам гарантує не тільки збереження здоров'я, а й високі продуктивність і якість праці. На підприємствах використовується природне і штучне освітлення. Приміщення

для роботи з ВДТ повинні мати природне та штучне освітлення, відповідно до ДБН В.2.5-28:2018 « Природне і штучне освітлення».

Для штучного освітлення у приміщенні використовуються люмінесцентні лампи типу ЛБ, які в порівнянні з лампами розжарювання мають ряд істотних переваг: за спектральним складом світла вони близькі до природного світла, мають підвищену світлову віддачу (у 2-5 разів вищу, ніж у ламп розжарювання); мають триваліший термін служби – до 10 тис годин.

3.2.3 Шум

Рівні шуму та вібрації на робочих місцях осіб, що працюють з ПК, визначаються відповідно до ДСанПіН 3.3.2.007-98.

Для забезпечення дотримання допустимих рівнів шуму на робочих місцях застосовуються засоби звукопоглинання, вибір яких обґрунтовується спеціальними інженерно-акустичними розрахунками (п. 3.3.3 ДСанПіН 3.3.2.007-98).

3.2.4 Мікроклімат

У виробничих приміщеннях на робочих місцях мають забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря – ДСН 3.3.6.042-99 «і норми мікроклімату виробничих приміщень».

Параметри мікроклімату	значення параметрі	
	Взимку	влітку
Температура, C^0	22-24	23-25
Відносна вологість, %	40-60	40-60
Швидкість руху повітря, м/с	0,1	0,1-0,2

Нормалізація параметрів мікроклімату у виробничих приміщеннях здійснюється за допомогою систем опалення. Кількість теплоти, що генерується системою опалення, має відповідати втрат теплоти в приміщенні (через будівельні конструкції, на нагрів повітря в приміщенні, технологічні тепловтрати, нагрів надходять матеріалів і напівфабрикатів. Механічна вентиляція забезпечується

вентиляторами, що забирають повітря зовні і направляє його до будь-якого робочого місця. або устаткування, а також видаляють забруднене повітря

3.3 Вимоги до організації робочого місця працівника

До самостійної роботи на комп'ютерах допускаються особи, які пройшли медичний огляд, навчання по професії, вступний інструктаж з охорони праці та первинний інструктаж з охорони праці на робочому місці. В подальшому вони проходять повторні інструктажі з охорони праці на робочому місці один раз на півріччя, періодичні медичні огляди один раз на два роки

Робочі місця повинні бути розташовані так, щоб у поле зору працюючого не попадали поверхні, що мають властивість віддзеркалювання, вікна освітлювальні прилади. 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. На рисунку 3.1 можна побачити схему регулювання робочого стола програміста:

Рисунок 3.1. Схема регулювання робочого стола програміста.

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. 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.

3.4 Пожежна безпека

Під пожежною безпекою розуміють систему державних і суспільних заходів, спрямованих на охорону від вогню людей і власності. 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 (негорючі речовини й матеріали в холодному стані).

Всі приміщення повинні бути забезпечені первинними засобами пожежогасіння: пожежним водопостачанням (пожежні крани ПК), пожежні щити з набором пожежного інструменту, вуглекислотними або порошковими вогнегасниками. У випадку виникнення пожежі необхідно відключити електроживлення, викликати по телефону 101 пожежну команду, евакуювати людей із приміщення відповідно до плану евакуації і приступити до ліквідації пожежі.

ВИСНОВКИ

Метою виконання дипломного проекту було визначено розробку комп’ютерної 2D-гри в жанрі вертикального скролл-шутеру за допомогою ігрового двигуна Unity. Виконання циклу розробки, тестування кінцевого проекту.

Для виконання цієї задачі було досліджено питання існуючих аналогів цифрових ігор в ігровій індустрії. Визначені основні риси характерні до обраного жанру, а також направлення розробки.

Ігровий двигун Unity дає змогу в повній мірі виконати основні ігрові елементи та механізми, пов’язані з ігровим процесом. Код для гри виконувався через середу розробки Visual Studio 2017 із використанням вбудованого репозиторію.

Процес розробки цифрової гри почався із визначення основних ігрових механік взаємодії гравця із ігровим процесом. Дані ігрові механіки були ретельно продумані, для них були створені функційні блок-схеми. Наступник кроком було виконано постановка завдань для роботи, який поділив проект на етапи розробки. Подальший процес розробки складався із виконання намічених завдань, згідно етапів.

В результаті було розроблено та реалізовану початкову версію 2D-гри в жанрі скролл-шутер із використанням в ній основних ігрових механік, притаманних даному ігровому жанру. В повній мірі було виконано роботу над програмною частиною інтерфейса та його взаємодією із ігровим процесом.

Використання принципів модульної розробки дало можливість і далі модифікувати гру, додаючи нові ігрові елементі, механіки, або режими. Створені програмні модулі можна у подальшому використовувати для розробки цифрових ігрових проектів не тільки в жанрі скролл-шутер, але й в інших суміжних жанрах, а деякі модулі створено універсальним чином, для можливості імплементувати їх в буд-який ігровий проект із найменшою кількістю внесених змін.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ерік Фрімен, Елізабет Робсон, Берт Бейтс, Кеті Сієрра, Книга Head First. Патерни проектування, Київ, Фабула, 672 ст.
2. Коноваленко І.В., Програмування мовою C# 6.0, Тернопіль, ТНТУ, 2016 р., 227 ст.
3. Фленов М., Біблія C# 3-е видання., Київ, Фабула, 2016 р. 546 ст.
4. Алан Торн, Мистецтво створення сценаріїв в Unity, Київ, видавництво ДМК, 362 ст.
5. Дікінсон К. Оптимізація ігор у Unity 5. Поради і методики, Оптимізація додатків охоплюючи всі аспекти роботи у ігровому движку Unity 3D., Київ, Фабула, 2017 р., 306 ст.
6. Andrew R., Dave M. GAME Architecture and Design – A new Edition, Shelter Island, New York: Manning Publications. 1035 p.
7. Goldstone W. Unity Game Development Essentials, New York, PACKT, 266 p.
8. Hocking J. Unity in Action: Multiplatform Game Development in C# with Unity 5. Shelter Island, New York: Manning Publications. 352 p.
9. Unity User Manual: [Website]. URL: https://docs.unity3d.com/Manual/index.html (viewed on: 21.05.2023).
10. Unity: [Website]. URL: https://unity.com/en-us (viewed on: 21.05.2023).
11. C# docs: [Website]. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (viewed on: 21.05.2023).

ДОДАТОК А

Лістинг коду основних модулів гри

Скрипт ship_parent.cs

```
public class ship_parent : MonoBehaviour{
public virtual void shield_regen_checker(ui_manager _ui_manager, float _shield_regen_time, float _shield_regen_rate, float _shield_cap, ref float _shield, ref float _timer_value, ref bool _shield_regen_need, ref bool _shield_regen_activ){
if(_shield_regen_need){
shield_regen_time_counter(_shield_regen_time, ref _timer_value, ref _shield_regen_activ);
shield_regen(_ui_manager, _shield_regen_rate, _shield_cap, ref _shield, ref _shield_regen_need, ref _shield_regen_activ);}}

public virtual void shield_regen_time_counter(float _shield_regen_time, ref float _timer_value, ref bool _shield_regen_activ){
if (_timer_value < _shield_regen_time){
_timer_value += Time.deltaTime;
if (_timer_value >= _shield_regen_time){
_shield_regen_activ = true;}}}

public virtual void shield_regen(ui_manager _ui_manager, float _shield_regen_rate, float _shield_cap, ref float _shield, ref bool _shield_regen_need, ref bool _shield_regen_activ){
if(_shield_regen_activ){
_shield += _shield_regen_rate;
if (_shield > _shield_cap){
_shield = _shield_cap;
_shield_regen_need = false;
_shield_regen_activ = false;}
_ui_manager.set_shield_bar_value(_shield);}}

public virtual void damage_calculate(ui_manager _ui_manager, float _damage, ref float _shield, ref float _hull, out bool _shield_regen_need, out bool _shield_regen_activ, out float _timer_value){
if(_shield > 0){
_shield -= _damage;
if(_shield < 0){
_hull += _shield;
_shield = 0;}}
else if (_shield == 0){
_hull -= _damage;
if(_hull < 0){
```

```
_hull = 0;
hull_break();}}

_shield_regen_need = true;
_shield_regen_activ = false;
_timer_value = 0;
_ui_manager.set_shield_bar_value(_shield);
_ui_manager.set_hull_bar_value(_hull);}

public virtual void damage_calculate(float damage, ref float hull, out bool
is_hull_break){
hull -= damage;
if(hull <= 0){
hull = 0;
is_hull_break = true;}
else{
is_hull_break = false;}}

public virtual void hull_break(){
Destroy(this.gameObject);}}
```

Скрипт scene_director.cs

```
public class scene_director : MonoBehaviour{
[SerializeField] Text debug_text;
[SerializeField] GameObject asteroid_prefab;
[SerializeField] int max_asteroid_on_level = 1;
int total_active_asteroids = 0;

void Start(){
for (int i = 0; i < max_asteroid_on_level; ++i){
instantiate_asteroid();}}

private void Update(){
debug_text.text = "Asteroid in scena: " + total_active_asteroids +
"\nMax asteroid in scena: " + max_asteroid_on_level +
"\nbatween total and max: " + (max_asteroid_on_level - total_active_asteroids);}

public void decrease_active_asteroids(){
--total_active_asteroids;
if (total_active_asteroids < 0){
total_active_asteroids = 0;}}

public void instantiate_asteroid(){
Instantiate(asteroid_prefab, new Vector3(Random.Range(-205f, 205f),
```

```
Random.Range(115, 130), 0f), Quaternion.identity);
++total_active_asteroids;}}
```

Скрипт ship_player.cs

```
public class ship_player : ship_parent{
[SerializeField] ui_manager _ui_manager
[SerializeField] float shield = 0;
[SerializeField] float shield_cap = 0;
[SerializeField] float hull = 0;
[SerializeField] float hull_cap = 0;
[SerializeField] float speed = 0;
[SerializeField] float energy = 0;
[SerializeField] float energy_cap = 0;
[SerializeField] float energy_charging_rate = 0;
[SerializeField] float shield_regen_rate = 0;
[SerializeField] float shield_regen_time = 0;
[SerializeField] float timer_value = 0;
[SerializeField] bool shield_regen_activ = false;
[SerializeField] bool shield_regen_need = false;
bool can_player_logic = true;

private void Start(){
_ui_manager.set_new_shield_bar_cap(shield_cap);
_ui_manager.set_new_hull_bar_cap(hull_cap);
_ui_manager.set_new_energy_bar_cap(energy_cap); }

private void FixedUpdate(){
if(can_player_logic)    {
shield_regen_checker(_ui_manager, shield_regen_time, shield_regen_rate, shield_cap, ref shield, ref timer_value, ref shield_regen_need, ref shield_regen_activ);
energy_charging();
if (Input.GetKeyDown(KeyCode.T)){
damage_calculate(_ui_manager, 10f, ref shield, ref hull, out shield_regen_need, out shield_regen_activ, out timer_value);}
if (Input.GetKeyDown(KeyCode.Y)){
energy_using(10f);}}}

void hull_fix(float fixing_value){
hull += fixing_value;
if (hull > hull_cap){
hull = hull_cap;}}

void energy_using(float value){
energy -= value;
```

```
if(energy <0){
energy = 0;}}

void energy_charging(){
if(energy < energy_cap){
energy += energy_charging_rate;}
_ui_manager.set_energy_bar_value(energy);}

public float get_player_speed(){
return speed;}

public void switch_player_logic(){
can_player_logic = !can_player_logic;}

public void take_damage(float value){
damage_calculate(_ui_manager, value, ref shield, ref hull, out shield_regen_need, out
shield_regen_activ, out timer_value);}}
```

Скрипт player_movement.cs

```
public class player_movement : MonoBehaviour{
ship_player _ship_player;
Rigidbody player_rigidbody;
Vector3 keyboard_force;
Quaternion start_position;
float player_speed;
float keyboard_force_x = 0f;
float keyboard_force_y = 0f;
bool can_move = true;

void Start(){
_ship_player = GetComponent<ship_player>();
player_rigidbody = GetComponent<Rigidbody>();
player_speed = _ship_player.get_player_speed();
start_position = transform.rotation;}
void FixedUpdate(){
if (can_move){
if (transform.position.x >= 205.0f){
if (Input.GetAxis("Horizontal") > 0)
keyboard_force_x = 0.0f;
else if (Input.GetAxis("Horizontal") < 0)
keyboard_force_x = Input.GetAxis("Horizontal");}
else if (transform.position.x <= -205.0f){
if (Input.GetAxis("Horizontal") > 0)
keyboard_force_x = Input.GetAxis("Horizontal");
```

```
else if (Input.GetAxis("Horizontal") < 0)
keyboard_force_x = 0.0f;}
else
keyboard_force_x = Input.GetAxis("Horizontal");
if (transform.position.y >= 105.0f){
if (Input.GetAxis("Vertical") > 0)
keyboard_force_y = 0.0f;
else if (Input.GetAxis("Vertical") < 0)
keyboard_force_y = Input.GetAxis("Vertical");}
else if (transform.position.y <= -105.0f){
if (Input.GetAxis("Vertical") > 0)
keyboard_force_y = Input.GetAxis("Vertical");
else if (Input.GetAxis("Vertical") < 0)
keyboard_force_y = 0.0f;}
else
keyboard_force_y = Input.GetAxis("Vertical");
keyboard_force = new Vector3(keyboard_force_x, keyboard_force_y, 0.0f);
keyboard_force = keyboard_force.normalized * player_speed;
player_rigidbody.velocity = keyboard_force;
moving_rotation();
else    {
player_rigidbody.velocity = Vector3.zero;}}

private void Update(){
transform.position = new Vector3(transform.position.x, transform.position.y, 0f);}

void moving_rotation(){
if (player_rigidbody.velocity.x > 0 && transform.rotation.y > -0.35){
transform.Rotate(Vector3.down, 4);}
else if (player_rigidbody.velocity.x < 0 && transform.rotation.y < 0.35){
transform.Rotate(Vector3.up, 4);}
else if (player_rigidbody.velocity.x == 0 && transform.rotation.y != 0){
transform.rotation = Quaternion.Lerp(transform.rotation, start_position, 1);}}
public void set_player_speed(float value){
player_speed = value;}
public void switch_can_move(){
can_move = !can_move;}}
```

Скрипт player_control.cs

```
public class player_control : MonoBehaviour{
player_movement _player_movement;
ship_player _ship_player;
[SerializeField] ui_manager _ui_manager;
[SerializeField] GameObject bullet_prefab;
```

```
bool can_fire = true;

private void Start(){
_player_movement = this.GetComponent<player_movement>();
_ship_player = this.GetComponent<ship_player>();}
void Update()   {
if((Input.GetKeyDown(KeyCode.Space))&&(can_fire)){
Instantiate(bullet_prefab, new Vector3(transform.position.x+2.7f,
transform.position.y+3f, transform.position.z), Quaternion.identity);}
if(Input.GetKeyDown(KeyCode.C{
_ui_manager.toggle_skill_panel();
_ship_player.switch_player_logic();
_player_movement.switch_can_move();
switch_can_fire();}}
public void switch_can_fire(){
can_fire = !can_fire;}}
```

Скрипт player_bullet_logic.cs

```
public class player_bullet_logic : MonoBehaviour{
ship_player _ship_player;
Rigidbody bullet_rigid_body;
public float bullet_speed = 100f;
Vector3 bullet_force;

void Start(){
bullet_rigid_body = GetComponent<Rigidbody>();
bullet_force = new Vector3(0f, 1f, 0f);}
private void FixedUpdate(){
bullet_force = bullet_force.normalized * bullet_speed;
bullet_rigid_body.velocity = new Vector3(bullet_force.x, bullet_force.y,
bullet_force.z);
if (transform.position.y > 115.0f){
Destroy(this.gameObject);}}

private void OnCollisionEnter(Collision collision){
if(collision.gameObject.GetComponent<ship_asteroid>() != null){
Destroy(this.gameObject);}}}
```

Скрипт ship_asteroid.cs

```
public class ship_asteroid : ship_parent{
scene_director _scene_director;
[SerializeField] float hull = 100f;
[SerializeField] float speed = 2.5f;
bool is_hull_break = false;
```

```
private void Start(){
_scene_director = GameObject.FindGameObjectWithTag("scene_director").GetComponent<scene_director>();}

public float get_asteroid_speed(){
return speed;}

public void deal_damage_to_asteroid(float value){
damage_calculate(value, ref hull, out is_hull_break);
if (is_hull_break){
hull_break();}}

public new void hull_break(){
_scene_director.decrease_active_asteroids();
_scene_director.instantiate_asteroid();
Destroy(gameObject);}}
```

Скрипт asteroid_movement.cs

```
public class asteroid_movement : MonoBehaviour{
ship_asteroid _ship_asteroid;
Rigidbody asteroid_rigidbody;
[SerializeField] float as_rand_speed_range = 2.5f;
[SerializeField] float as_rand_dir_x_range = 2.5f;
[SerializeField] float as_rand_dir_z_range = 2.5f;
[SerializeField] float as_rand_rot_x_range = 2.5f;
[SerializeField] float as_rand_rot_y_range = 2.5f;
[SerializeField] float as_rand_rot_z_range = 2.5f;
Vector3 asteroid_rotation_force = Vector3.zero;
float asteroid_speed = 0f;

void Start(){
_ship_asteroid = this.GetComponent<ship_asteroid>();
asteroid_rigidbody = this.GetComponent<Rigidbody>();
asteroid_speed = Random.Range(0, as_rand_speed_range);
asteroid_rotation_force = new Vector3(Random.Range(-as_rand_rot_x_range, as_rand_rot_x_range),
Random.Range(-as_rand_rot_y_range, as_rand_rot_y_range),
Random.Range(-as_rand_rot_z_range, as_rand_rot_z_range));
asteroid_rigidbody.AddForce(new Vector3(Random.Range(-as_rand_dir_x_range, as_rand_dir_x_range),
-asteroid_speed,
Random.Range(-as_rand_dir_z_range, as_rand_dir_z_range)),
```

```
ForceMode.Impulse);
Vector3 scale_random = new Vector3(Random.Range(3, 8), Random.Range(3, 8), Random.Range(3, 8));
transform.localScale = scale_random;}
private void FixedUpdate(){
this.gameObject.transform.Rotate(asteroid_rotation_force);}}
```

Скрипт asteroid_logic.cs

```
public class asteroid_logic : MonoBehaviour{
scene_director _scene_director;
ship_asteroid _ship_asteroid;
Vector3 asteroid_position;

private void Awake(){
_scene_director = GameObject.FindGameObjectWithTag("scene_director").GetComponent<scene_director>();
ship_asteroid = GetComponent<ship_asteroid>();}

private void FixedUpdate(){
asteroid_position = this.transform.position;
if (asteroid_position.x < -250){
_ship_asteroid.hull_break();}
if (asteroid_position.x > 250){
_ship_asteroid.hull_break();}
if (asteroid_position.y < -125){
_ship_asteroid.hull_break();}
if (asteroid_position.y > 140){
_ship_asteroid.hull_break();}
if (asteroid_position.z < -215){
_ship_asteroid.hull_break();}
if (asteroid_position.z > 215){
_ship_asteroid.hull_break();}}

private void OnCollisionEnter(Collision collision){
ship_player _ship_player;
_ship_player = collision.gameObject.GetComponent<ship_player>();
if(_ship_player != null){
_ship_player.take_damage(10f);
_ship_asteroid.deal_damage_to_asteroid(45f);}
else{
_ship_asteroid.deal_damage_to_asteroid(35f);}}}
```

Скрипт ui_manager.cs

```
public class ui_manager : MonoBehaviour{
[SerializeField] GameObject skill_panel;
[SerializeField] Slider shield_bar;
[SerializeField] Slider hull_bar;
[SerializeField] Slider energy_bar;

public void toggle_skill_panel(){
if(skill_panel.activeSelf){
skill_panel.SetActive(false);}
else{
skill_panel.SetActive(true);}}

public void set_new_shield_bar_cap(float new_value){
shield_bar.maxValue = new_value;
shield_bar.value = shield_bar.maxValue;}

public void set_shield_bar_value(float value){
shield_bar.value = value;}

public void set_new_hull_bar_cap(float new_value){
hull_bar.maxValue = new_value;
hull_bar.value = hull_bar.maxValue;}

public void set_hull_bar_value(float value){
hull_bar.value = value;}

public void set_new_energy_bar_cap(float new_value){
energy_bar.maxValue = new_value;
energy_bar.value = energy_bar.maxValue;}

public void set_energy_bar_value(float value){
energy_bar.value = value;}}
```

Скрипт ui_transperency.cs

```
public class ui_transperency : MonoBehaviour{
[SerializeField] GameObject player_gameobject;
[SerializeField] CanvasGroup energy_bar;
[SerializeField] CanvasGroup shield_bar;
[SerializeField] CanvasGroup hull_bar;
[SerializeField] CanvasGroup skill_button;
[SerializeField] Text pl_pos_text;
Vector3 pl_pos;
bool energy_transper_status = false;
```

```
bool shield_hull_transper_status = false;
bool skill_transper_status = false;

public void Update(){
if(player_gameobject != null)
pl_pos = player_gameobject.transform.position;
pl_pos_text.text = "player x position = " + pl_pos.x +"\nplayer y position = " +
pl_pos.y;
if((pl_pos.x < -180) && (pl_pos.y < 75)){
transper_energy();
energy_transper_status = true;}
if ((pl_pos.x < -80) && (pl_pos.y < -70)){
transper_shield_hull();
shield_hull_transper_status = true;}
if ((pl_pos.x > 175) && (pl_pos.y < -85)){
transper_skill_button();
skill_transper_status = true;}
if(energy_transper_status){
if((pl_pos.x > -180) || (pl_pos.y > 75)){
detransper_energy();
energy_transper_status = false;}}
if(shield_hull_transper_status){
if ((pl_pos.x > -80) || (pl_pos.y > -70)){
detransper_shield_hull();
shield_hull_transper_status = false;}}
if(skill_transper_status){
if ((pl_pos.x < 175) || (pl_pos.y > -85)){
detransper_skill_button();
skill_transper_status = false;}}}

public void transper_energy(){
energy_bar.alpha = 0.01f;}

public void detransper_energy(){
energy_bar.alpha = 1f;}

public void transper_shield_hull(){
shield_bar.alpha = 0.01f;
hull_bar.alpha = 0.01f;}

public void detransper_shield_hull(){
shield_bar.alpha = 1f;
hull_bar.alpha = 1f;}
```

```
public void transper_skill_button(){
skill_button.alpha = 0.01f;}
public void detransper_skill_button(){
skill_button.alpha = 1f;}
```

ДОДАТОК Б

Слайди мультимедійної презентації

Розробка комп'ютерної 2D-гри в жанрі вертикального скролл-шутеру за допомогою ігрового двигуна Unity

ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТА ГРУПИ 4РП-06: ЄРЬОМЕНКО АРТЕМА КОСТЯНТИНОВИЧА

КЕРІВНИК: ДЖАБРАІЛОВ Д.В.

Особливості ігрового жанру вертикального скролл-шутеру

Ігровий жанр *скролл-шутеру* має велику історію та є одним із основних перших жанрів в ігровій індустрії.

До особлививостей ігрового жанру відносять:

- Велику інтенсивність ігрового процесу;
- Гнучкі вимоги до навичок гравця;
- Гнучкі візуальні можливості;
- Ігри цього жанру реалізуються, як в 2D так і 3D;
- Відносно вільна реалізація орієнтації камери;
- Ілюзія переміщення гравця у просторі зліва-направо, з право наліво;
- Концепція «Одного торкання»;
- Простота в управлінні.

Особливості ігрового двигуна Unity

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!

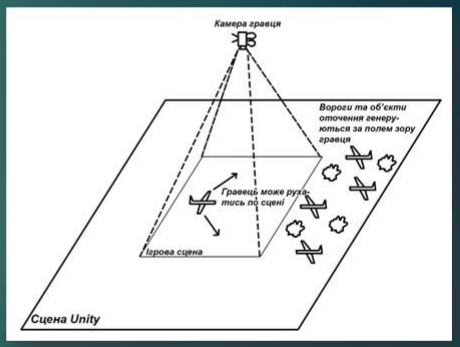
- Має зручну візуальну середу розробки;
- Має можливість міжплатформенної розробки ігор;
- Підтримує модульну систему компонентів;
- Використання мови програмування C# та його можливостей;
- Велике ком'юніті розробників;
- Зрозуміла та вичерпна документація, безліч курсів у інтернеті та літературі;
-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

Формування загального концепту ігрового процесу

Для створення гри необхідно виконати ряд сталих дій:

- Визначити ігровий концепт;
- Поставити завдання для роботи;
- Формалізувати концепт ігрового процесу;
- Спроектувати основні модулі;
- Імплементувати спроектовані модулі.

Імплементація моделей буде виконуватись за допомогою вбудованих функцій у Unity та написання коду скриптів, що будуть реалізувати ігрові механіки.

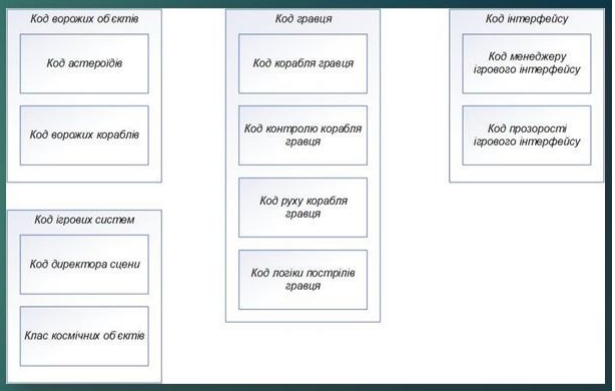


Основні модулі гри

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

- Модуль ворожих об'єктів;
- Модуль ігрових систем;
- Модуль гравця;
- Модуль інтерфейсу.

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

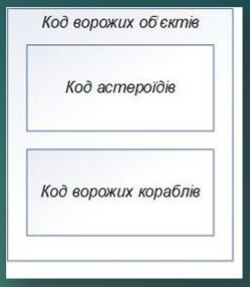


Основні модулі гри

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

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

Кожний підмодуль є типовим та має типовий код із Логікою об'єкту, Переміщенням об'єкту, Класом об'єкту.



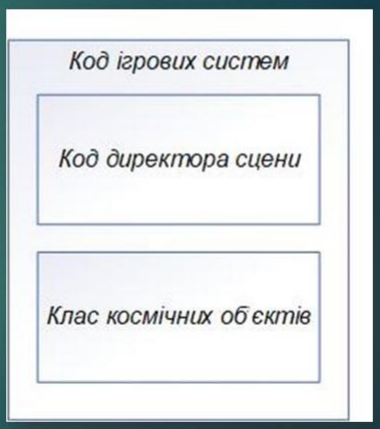
Основні модулі гри

Модуль ігрових систем забезпечує роботу основних ігрових систем. Такими є:

- Клас космічного об'єкту;
- Код «Директора сцени».

Код космічного об'єкту є батьківським класом для всіх космічних об'єктів та забезпечує основні методи для їх роботи.

Код «Директора сцени» реалізує спеціальний скрипт, який буде керувати ігровим процесом та обміном даних між об'єктами на сцені.

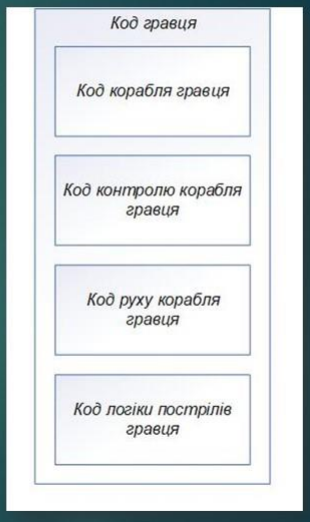


Основні модулі гри

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 керування об'єктом гравця.

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

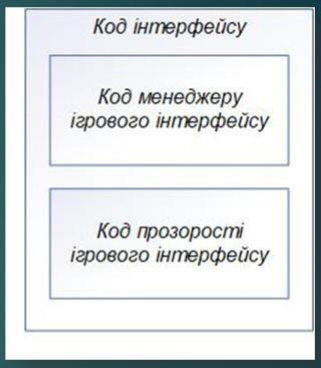


Основні модулі гри

Модуль інтерфейсу реалізовує код, що буде взаємодіяти із інтерфейсом.

Основним скриптом є «Менеджер ігрового інтерфейсу», який виконує передачу даних до інтерфейсу гравця, та зворотню взаємодію.

Окремо є код, що змінює ступін прозорості інтерфейсу за необхідності.

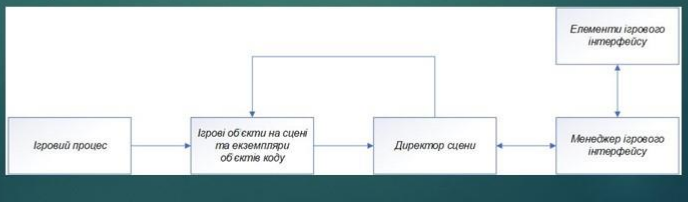


Механізм роботи основних модулів гри

- Модулі гри взаємодіють один між одним за допомогою зв'язків-посилань між їх скриптами. В основному, всі скрипти модулів відсилаються до «Директора сцени», що керує не тільки генерацією об'єктів на рівні, а також обміном даних.

- В окремих випадках дані передаються між скриптами модулів напряму. Скрипти закріплюються за ігровими об'єктами, отримають посилання на інші скрипти та відразу починають працювати зі старту сцени.

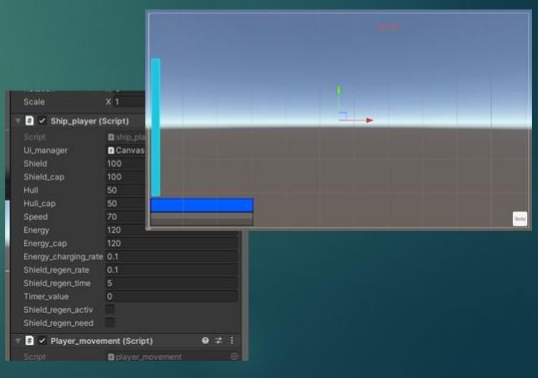
- Схема знизу демонструє принцип обміну даних між об'єктами на рівні та модулями. Кожний кадр гри, за потреби дані будуть пересилатись до інших скриптів, або в разі спрацювання якоїсь події.



Етапи реалізації основних модулів гри

Для реалізації основних модулів гри було виконані наступні етапи:

- Проектування модулів;
- Створення об'єктів та робочих файлів для них;
- Написання скриптів модулів;
- Розміщення об'єктів на рівні та додання до них відповідних скриптів;
- Створення інтерфейсу та імплементація до нього відповідних скриптів;
- Відладка та тестування модулів та гри.



Скріншоти розробленої гри

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Єрьоменко Артема Костянтиновича

(прізвище, ім`я та по батькові)

Спеціальність: 121 "Інженерія програмного забезпечення"

Освітня програма: «Розробка програмного забезпечення»

Тема дипломного проекту: Розробка комп'ютерної 2D-гри в жанрі вертикального скролл-шутеру за допомогою ігрового двигуна 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 71 сторінок. У пояснювальній записці виконано опис предметної області, проектування основних програмних модулів скролл-шутеру. Проведено розробка всіх основних елементів 2D-гри в жанрі вертикального скролл-шутеру. 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 Єрьоменко А.К. 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 Єрьоменко А.К. 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 захисту дипломного проекту

г) вміння розв'язувати виробничі та конструкторські питання

Під час дипломного проектування здобувач освіти Єрьоменко А.К. мав змогу самостійно приймати рішення з проектування та реалізації модулів гри та її елементів, показав вміння організовано працювати над поставленим завданням, складати схеми та проводити розробку коду за допомогою Visual Studio C# та ігрового двигуна Unity.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Добре

Загальна оцінка Відмінно

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

«09» 06 2023 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Єрьоменко Артема Костянтиновича

(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 комп'ютерної 2D-гри в жанрі вертикального скролл-шутеру за допомогою ігрового двигуна Unity*

Обсяг розрахунково-пояснювальної записки 71 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 присвячена питанням розробки комп'ютерних ігор на ігровому двигуні Unity.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Дипломний проект складається зі вступу, трьох розділів, висновків, переліку використаних джерел. У технологічному розділі виконано огляд проблематики питання, сформовано завдання до проекту та вірно підібрано необхідний інструментарій для розробки. Виконано проектування гри, її основних програмних та логічних частин. За проектом реалізовані всі намічені елементи 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. Аналіз та постановка завдання, формування концепції ігрового процесу;
2. Якісне проектування основних елементів гри та її логіки;
3. Реалізація повного ігрового функціоналу;
4. Закладена основа для подальшого розвитку ігрового процесу.

д) основні недоліки дипломного проекту (роботи)

1. Не вистачає різноманітності ворогів;
2. Немає візуальних ефектів.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Добре

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові рецензента Царьов Роман Юрійович

Місце роботи і посада рецензента Державний університет інтелектуальних технологій і зв'язку, старший викладач кафедри комп'ютерної інженерії та інформаційних систем

Підпис:

«_____» _____ 2023 р.

ПІДПИС ПОСВІДЧУЮ
НАЧАЛЬНИК ВІДДІЛУ
КАДРІВ ДУІТЗ

НАЦІОНАЛЬНА АКАДЕМІЯ УКРАЇНИ * МІНІСТЕРСТВО ОСВІТИ І НАУКИ * ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ ТА ЗВ'ЯЗКУ * 43997335 * ВІДДІЛ КАДРІВ

UNICHECK

Ім'я користувача:
Наталія Вікторівна Копусь

ID перевірки:
1015551980

Дата перевірки:
11.06.2023 23:13:41 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
11.06.2023 23:15:11 EEST

ID користувача:
100011688

Назва документа: 4РП-06 Єрьоменко А.К.

Кількість сторінок: 48 Кількість слів: 7858 Кількість символів: 56684 Розмір файлу: 2.45 MB ID файлу: 1015204220

9.43%
Схожість

Найбільша схожість: 1.41% з Інтернет-джерелом (https://otherreferats.allbest.ru/life/00092019_0.html)

9.43% Джерела з Інтернету	668	..... Сторінка 50

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Замінені символи	24

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

Ми, що нижче підписалися,

Єрьоменко Артем Костянтинович,
здобувач освіти гр. 4РП-06, та

Джабраілов Дмитро Володимирович,
керівник дипломного проекту,

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

«Розробка комп'ютерної 2D-гри в жанрі вертикального скролл-шутеру за допомогою ігрового двигуна Unity» (автор роботи – Єрьоменко А.К., керівник роботи – Джабраілов Д.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2023 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 _____ / Єрьоменко А.К. /

Керівник _____ / Джабраілов Д.В./

«09» 06 2023 р.