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

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-07

Дипломний проект

здобувача освіти денної форми навчання
РП.07.11.000.ДП

ЛАТИШЕВА
КИРИЛА
ОЛЕКСАНДРОВИЧА

м. Одеса
2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

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

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

Група: **4РП-07**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка 2D-гри у жанрі топ-даун шутер на програмному рушії Unity

Проектний матеріал складається з пояснювальної записки на 75 сторінках та графічного (презентаційного) матеріалу на 12 аркушах (слайдах)

Дипломник _____ (Латишев К.О.)

Керівник _____ (Джабраілов Д.В.)

Консультанти:

з економічного розділу _____ (Іванченков В.С.)

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

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Скорнякова О.В.)

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

Оцінка ЕК 3(задовільно)/70Б.

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

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

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітньо-професійна програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"16" 01 2024 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Латишеву Кирилу Олександровичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка 2D-гри у жанрі топ-даун шутер на програмному рушії Unity*

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

2. Термін здачі закінченого проекту 10.06.2024

3. Вихідні данні до проекту *Використання програмного рушія Unity; Використання мови програмування C# та бібліотек Unity для розробки гри; Використання принципів модульності у проектуванні та розробці ігрових проектів; Реалізація основних ігрових механік комп'ютерної 2D-гри в жанрі топ-доун шутер; Гра повинна мати основні признаки жанру 2D топ-даун шутер; Гра повинна мати інтерфейс; Гравець повинен мати змогу пересуватись по ігровому простору та взаємодіяти з неігровими персонажами.*

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

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

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

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

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

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

7. Дата видачі завдання 15.01.2024

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

Завдання прийняв до виконання *Латишев К.О.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

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

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 07. 11 000. 00 ДП	Дипломний проект		
А4			РП 07. 11 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 07. 11 000. 00 ДП		
Розробив		Латишев К.О.		10.06	Розробка 2D-гри у жанрі топ-даун шутер на програмному рушії Unity	Літ.	Аркуш 4 / Аркушів 75
Перевірив		Джабраілов Д.В.		10.06		Н Д П	
Н. контр.		Петрашова В.І.		10.06		ВСП "ОТФК ОНТУ" гр.4РП-07	
Затверд.		Кривченко Ю.В.		10.06			

ЗМІСТ

ВСТУП

Сфера IT має багато різних складових від високотехнологічного виробництва мікросхем та процесорів, до створення комп'ютерних ігор. Сектор ігрової індустрії нараховує багато мільярдів доларів обороту та працевлаштовує багату кількість IT-спеціалістів по всьому світу різного рівня компетенцій та спеціальностей.

Створення сучасної комп'ютерної гри є дуже складним та трудомістким процесом, який займає великий час циклу розробки, від дизайн-документу до релізної версії, інколи багато років. Окрім того, більшість сучасних ігрових проектів потребують післярелізної підтримки, тому великі компанії розробники завжди тримають частину команди для цих цілей.

Темою мого дипломного проекту є «Розробка 2D-гри у жанрі топ-даун шутер на програмному рушії Unity». Розробка цього проекту дасть змогу більше зрозуміти основні етапи розробки ігор. Не зважаючи на те, що 2D-гра в жанрі топ-даун шутер не така трудомістка, але для її розробки необхідно буде також розробляти дизайн-документ, проектувати роботу модулів гри, виконувати проектування взаємодії елементів, відладку проекту.

Розробка буде виконуватись на програмному рушії Unity, який дає змогу безкоштовно створювати ігрові додатки для різних ігрових платформ. Окрім того, проекти, створені на цьому програмному рушії дуже легко портувати між різними платформами, майже не вносячи в них змін.

В цілому, розробка такої гри дає основу для подальшого її розвитку шляхом додавання нових рівнів, зброї, ворогів, або ігрових механік. Імплементацію мережевого геймплею, монетизацію контенту.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз ігрового жанру топ-даун шутер

1.1.1 Загальні дані про ігровий жанр

Перед початком виконання проектування та реалізації гри дипломного проекту, необхідно провести аналіз обраного ігрового жанру. Це дасть змогу виробити необхідні кроки розробки, особливості реалізації розробляємого продукту, візуальні та ігрові нюанси майбутньої гри. Отже, що можна сказати про ігровий жанр топ-даун шутер?

Топ-даун шутер – це жанр відеоігор, в яких ігровий світ відображається зверху вниз, що дає гравцям огляд всієї карти. У таких іграх гравець керує персонажем, який часто озброєний зброєю, і прагне вижити або виконати ціль, стріляючи по ворогах та уникаючи їх атак.

Основні характеристики топ-даун шутерів:

- Перспектива польоту: Ігровий світ відображається з висоти, що дозволяє гравцям бачити всю карту та планувати свої дії, такі як ухилення від атак ворогів або вибір оптимального шляху;
- Стрілянина та бій: Основний акцент робиться на стрільбі та битвах із противниками. Гравці повинні використовувати зброю та тактичні прийоми, щоб перемогти ворогів та виконати завдання;
- Стратегія та тактика: Часто в таких іграх важлива не тільки майстерність у стрільбі, а й стратегічне мислення. Гравці повинні вирішувати, як найкраще використовувати довкілля, розподіляти ресурси та координувати дії з командою;
- Мультиплеєр: Багато топ-даун шутерів пропонують мультиплеєрні режими гри, де гравці можуть боротися один з одним або співпрацювати у командних боях;
- Прогресія та розвиток: У багатьох іграх цього жанру передбачена система прокачування персонажа або покращення зброї, що дозволяє гравцям ставати сильнішими та ефективнішими в міру проходження гри;

Топ-даун шутери можуть бути як одиночними, так і мультиплеєрними і пропонувати різноманітні сценарії та умови гри, що робить їх привабливими для широкого кола гравців.

1.1.2 Аналіз гри Robotron: 2084

Для виконання вдалого аналізу необхідно розглянути деякі ігри цього жанру, починаючи з основ до найвідоміших тайтлів сьогодення.

Перші ігри жанру топ-даун шутер з’явилися наприкінці 1970-х та на початку 1980-х років. Одним з найбільш відомих і ранніх прикладів є гра Robotron: 2084, випущена в 1982 році. У цій грі гравець управляє персонажем, який має боротися з хвилями роботів у ворожому світі. Robotron: 2084 була розроблена Євгеном Джаром і швидко стала популярною завдяки своїй динамічній ігровій механіці та швидкому темпу. Цей успіх став початком розвитку жанру топ-даун шутерів. На рисунку 1.1 можна побачити скріншоти гри Robotron: 2084:

Рисунок 1.1. Скріншоти гри Robotron: 2084

1.1.3 Аналіз гри Hotline Miami та Enter the Gungeon

Hotline Miami – це культова гра, випущена в 2012 році, яка поєднує в собі швидку та насильницьку гру з ретро-стилізованим візуальним стилем. Гравці керують антигероєм, який виконує завдання, вбиваючи супротивників у різних місцях. Скріншоти гри приведено на рисунку 1.2.

Рисунок 1.2. Скріншот гри Hotline Miami

Enter the Gungeon – це roguelike топ-даун шутер, випущений у 2016 році. Гравці досліджують підземелля, борючись із хвилями ворогів та збираючи зброю та предмети для покращення своїх шансів на виживання.

1.1.4 Аналіз гри Helldivers, Alien Swarm та Nuclear Throne

Helldivers – це мультиплеєрний кооперативний шутер, випущений у 2015 році. Гравці керують членами елітного загону Helldivers, які борються з ворожими силами у космічній боротьбі за свободу. Скріншоти гри приведено на рисунку 1.3.

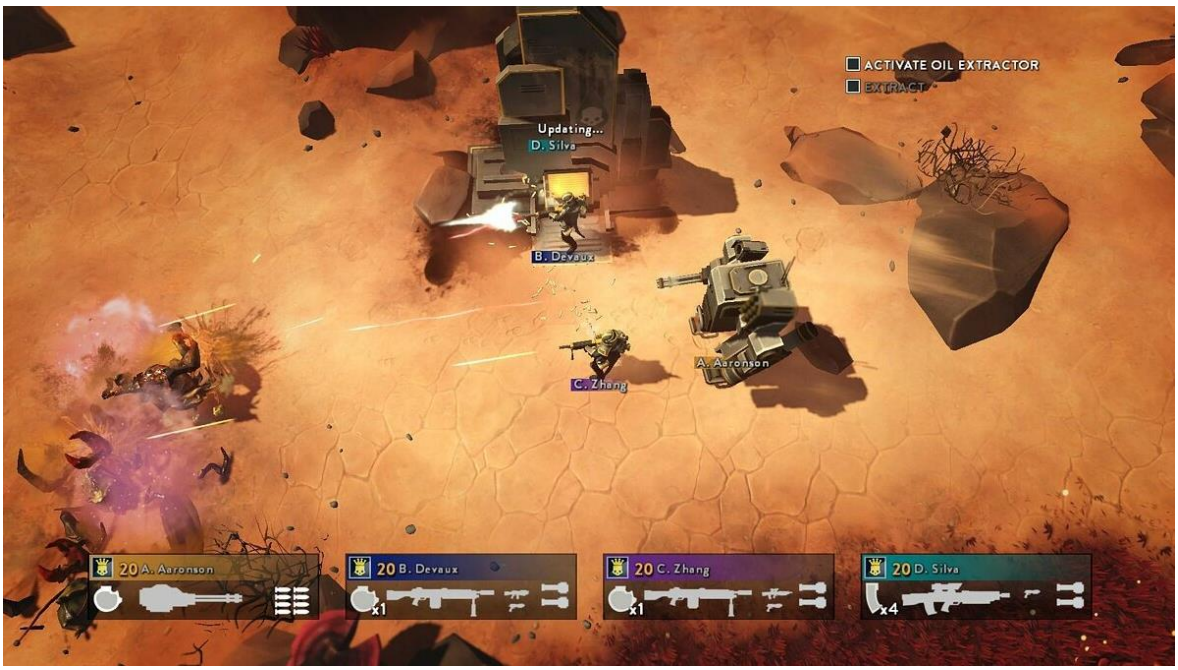


Рисунок 1.3. Скріншот гри Helldivers

Alien Swarm – це безкоштовна кооперативна гра, випущена в 2010 Valve Corporation. Гравці управляють членами загону космічних морпіхів, борючись із інопланетними істотами на зараженому космічному кораблі. Скріншоти гри приведено на рисунку 1.4.

Рисунок 1.4. Скріншот гри Alien Swarm

Nuclear Throne – це roguelike топ-даун шутер, випущений у 2015 році. Гравці досліджують небезпечні мутантні землі, борючись із ворогами та покращуючи свого персонажа у міру проходження.

1.1.5 Висновки з аналізу

Це лише кілька прикладів, і жанр топ-даун шутерів має багато різних ігор із різними стилями та механіками гри.

Як можна бачити з розглянутих прикладів ігор, в основному проекти в жанрі топ-даун шутер напряму слідують своїй назві. В основному камера розміщується зверху, або дещо збоку, утворюючи ізометрію. Втім, можна побачити, що графічна частина проектів буває різною, як простою 2D-графікою, так і 3D-моделями.

Окрім того, кажучи про інтерфейс, також можна побачити, що сталого стандарту не має. В грі Hotline Miami інтерфейсу немає зовсім, а у Helldivers, навпаки, є розгорнутий інтерфейс, з відмітками над персонажами, завданнями та іншим.

З точки зору ігрового процесу, майже всі топ-даун шутери зосереджені на знищені ворогів на локаціях, які представляють собою плоскі рівні. На них є вороги, які реагують на гравця якимось чином, наприклад атакують гравця на відстані, або наближаються для рукопашної атаки.

Також, однією з особливостей даного ігрового жанру є елементи, які можна підбирати з землі. Наприклад зброю, або медпакети чи набої. Це дає змогу ігровому дизайнеру побудувати ігровий процес таким чином, щоби контролювати напругу для гравця, або підтримати його в разі, якщо поблизу буде головний ворог рівня. Таким чином, за допомогою різних предметів на локації можна впливати на ігровий процес.

1.2 Аналітичний огляд програмних рушіїв з умовно безкоштовним доступом

Для успішного виконання завдання дипломного проектування необхідно визначитись з ігровим рушієм. Не зважаючи на те, що в темі дипломного проекту вже зазначено рушій Unity, але потрібно пояснити, чому вибір пав саме на нього. Огляд буде проводитись з основних рушіїв, які використовуються в ігровій індустрії.

1.2.1 Огляд ігрового програмного рушія Unreal Engine

Першим розглянутим ігровим програмним двигуном є Unreal Engine, створений компанією Epic Games. Цей потужний і широко відомий ігровий програмний рушій надає розробникам інструменти для створення високоякісних ігор, візуалізацій, а також проектів віртуальної та доповненої реальності. Він має широкий набір функцій, включаючи потужний графічний двигун, підтримку фізичного моделювання, інструменти для роботи зі штучним інтелектом, анімацією персонажів, звуком та багатьом іншим. Unreal Engine також має графічний інтерфейс користувача, який спрощує процес розробки і створення контенту. На рисунку 1.5 представлений приклад графіки у цьому ігровому рушії.

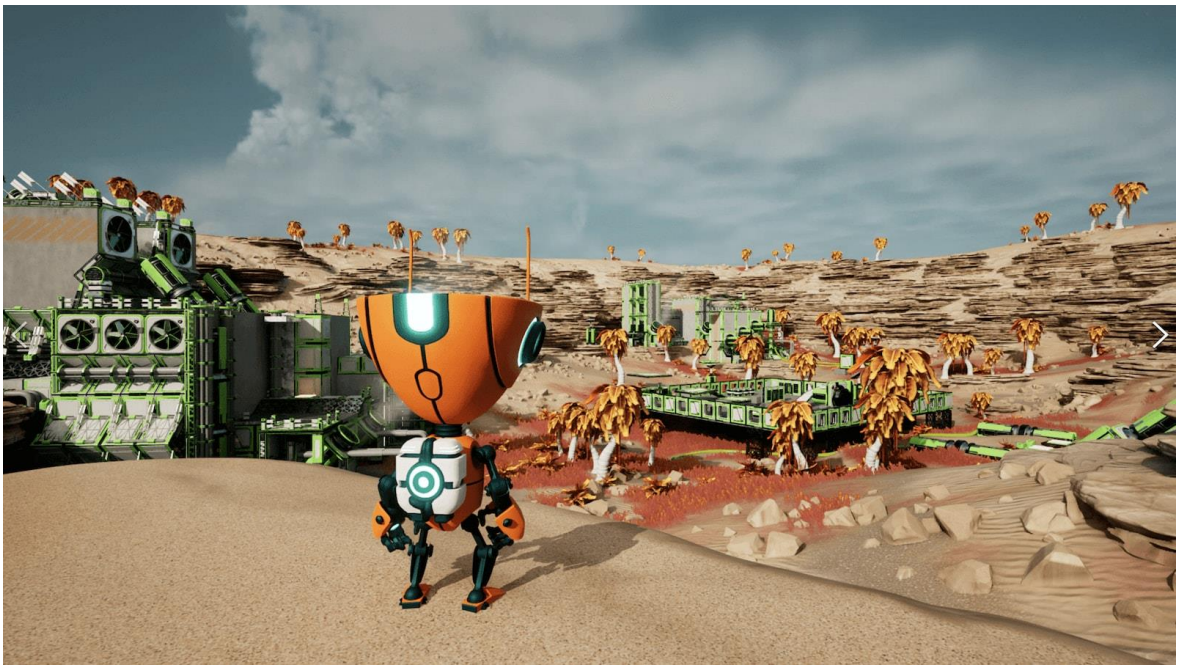


Рисунок 1.5. Приклад графіки програмного рушія Unreal Engine

Одна з основних переваг Unreal Engine полягає в його потужному графічному програмному рушії, який підтримує передові технології, включаючи трасування променів. Це дозволяє створювати неймовірно реалістичні графічні ефекти. Завдяки цим можливостям, Unreal Engine стає популярним вибором для розробників ААА-ігор та створення вражаючих візуалізацій та симуляцій в інших галузях.

У контексті ліцензування, Unreal Engine пропонує безкоштовну версію, однак з деякими обмеженнями: код ігрового рушія залишається закритим, доступ до певних платформ розробки обмежений, а також заблокований доступ до окремого середовища розробників та підтримки.

1.2.2 Огляд ігрового програмного рушія CryEngine

Ще один програмний рушій для аналізу – CryEngine. Це потужний рушій, який був створений компанією CryTek. Вона займає нішу компаній, які впроваджують рішення із високою якістю комп’ютерної графіки. Ігровий програмний рушій CryEngine був створений як провідний фізичний та графічний рушій, та був використаний у різних ігрових проектах як Crysis, Ryse: Son of Rome та Hunt: Showdown. Одною із основних рис ігрового двигуна була можливість легко створювати руйнування оточення. Рисунок 1.6 показує якість графіки у ігровому рушії CryEngine.

Рисунок 1.6. Якість графіки ігрового програмного рушія CryEngine

Однією із ключових можливостей, на ряду з іншими ігровими рушіями, CryEngine дає змогу симулювати трасування променів, що створює фотореалістичне освітлення сцен та відображення на різних поверхнях. Також цей програмний рушій має високу ступінь масштабованності, яка дозволяє розробляти проекти від крупних AAA-проектів, до маленьких інді-розробок. Серед інструментів CryEngine є такі, що забезпечують створення інтерективного оточення, живого ігрового середовища, системи керування анімацією, системи штучного інтелекту для ботів, інструменти для моделювання реалістичного звукового оточення. Окремо слід зазначити можливість портування проектів на різні платформи.

CryEngine в цілому є безкоштовним ігровим рушієм, але в нього існують умови для ліцензування, які включають у себе умови обов’язкової покупки ліцензії у разі підвищення грошового обороту. Сама ж ліцензія відкриває доступ до додаткових ассетів ігрового рушія, а також офіціальної підтримки від розробників ігрового рушія. Також ліцензія відкриває доступ до закритих форумів розробників. Одним із головних плюсів є можливість роботи разом із розробниками ігрового програмного рушія над спеціальним інструментарієм безпосередньо для вашого ігрового проекту. На рисунку 1.7 можна побачити роботу із mesh-ем за допомогою інструментарію CryEngine.

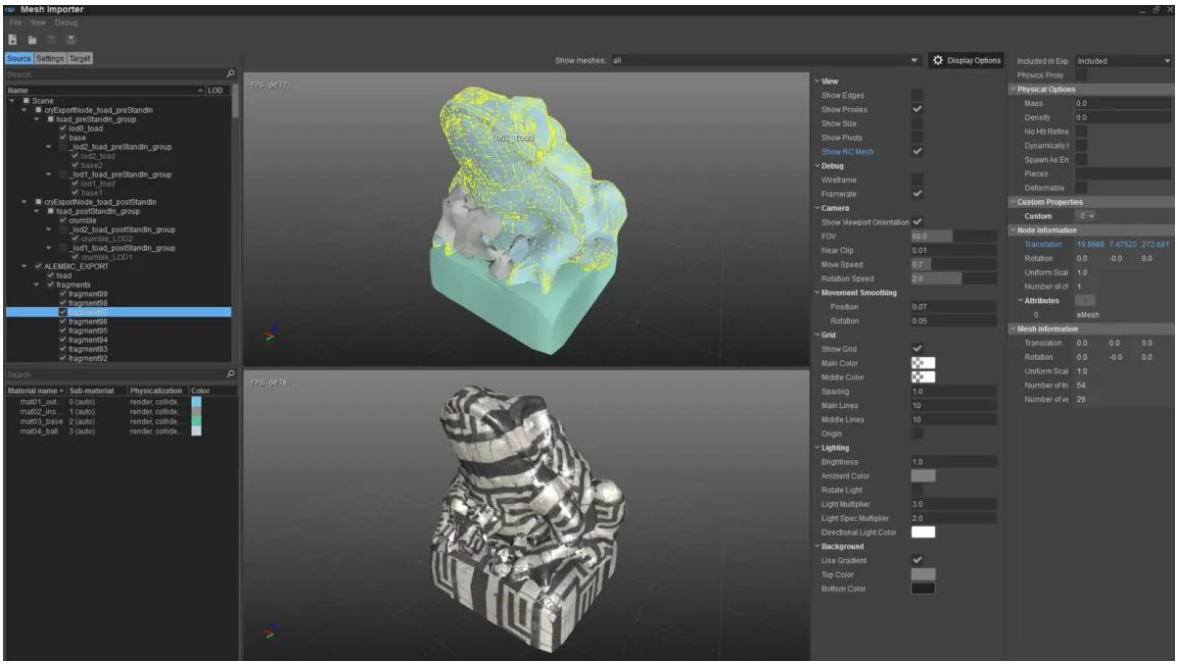


Рисунок 1.7. Робота із Mesh-ем за допомогою інструментарію CryEngine

1.2.3 Огляд ігрового програмного рушія Unity

Ігровий програмний рушій Unity – це популярний багатоплатформне рішення та середовище розробки, створений компанією Unity Technologies ще у далекому 2010 році. Він надає розробникам інструменти для створення 2D та 3D ігор, додатків віртуальної та доповненої реальності, симуляцій, візуалізацій та багато іншого. Unity відрізняється простотою використання та доступністю, що робить його привабливим вибором для новачків у розробці ігор, а також для досвідчених розробників. Ця важлива риса зробила цей ігровий рушій найрозповсюдженим рішенням для освіти, а також стартапів в ігровій індустрії. Він надає зручний графічний інтерфейс, безліч готових ресурсів та компонентів, а також потужну мову програмування C# для створення інтерактивності та логіки ігор.

Окрім того, основою цього програмного рушія є його ком’юніті, яке створює якісь рішення, які розповсюджуються через вбудований до рушія AssetStore. Він дозволяє створювати ігрові ассети та ділитись ними з іншими розробниками, як на безкоштовній так и платній основі. Для використання ассетів не потрібно нічого окрім реєстрації на ресурсі та додавання ассету до кошика ассетів. На рисунку 1.8 можна побачити приклад роботи у ігровому програмному рушії Unity.

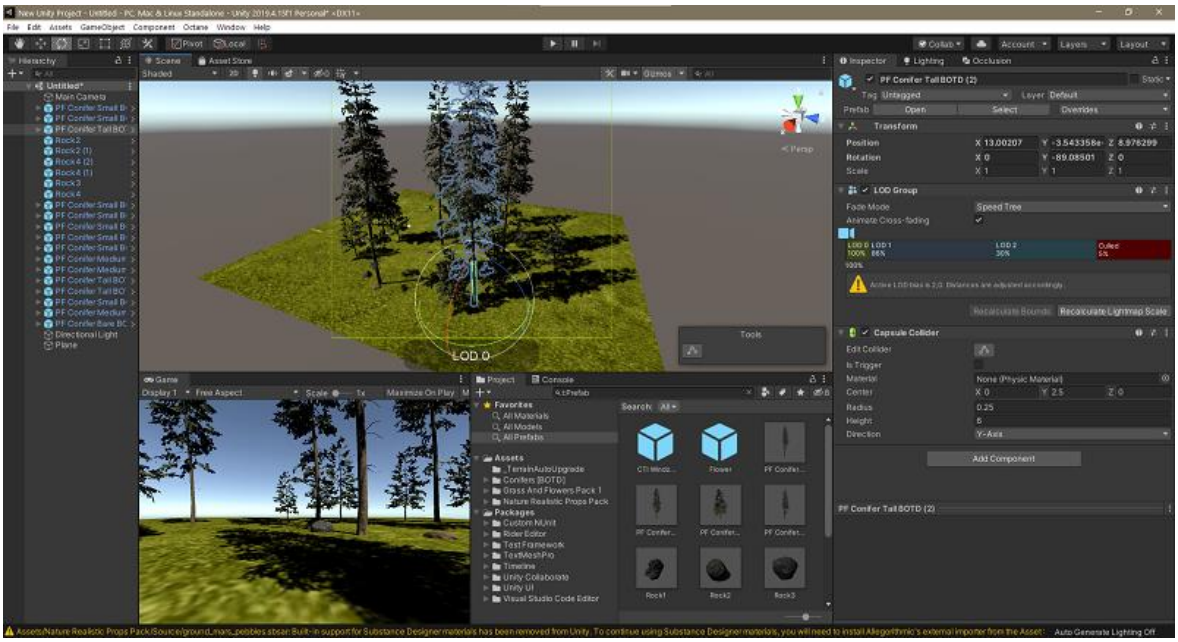


Рисунок 1.8. Приклад роботи у ігровому програмному рушії Unity

Однією з ключових особливостей Unity є багатоплатформність. Він підтримує різні платформи, включаючи ПК, консолі, мобільні пристрої, віртуальну реальність та доповнену реальність, що дозволяє розробникам досягати більшої аудиторії та випускати свої продукти на різних пристроях.

З точки зору ліцензування, то ігровий програмний рушій надає повний доступ до свого функціоналу безкоштовно, для створення ігор будь-якого рівня. Необхідність покупати ліцензію з'являється лише якщо сума зароблених грошей на користувача дійде до рівня у двісті тисяч доларів.

1.2.4 Обґрунтування обрання ігрового програмного рушія Unity

Аналіз трьох основних програмних рушіїв дав можливість обрати з них найоптимальніший варіант, яким став ігровий програмний рушій Unity. Таке рішення в першу чергу зумовлений доцільністю використання можливостей платформ. Кожний ігровий рушій використовує певну величину ресурсів ігрової середи, тому важливу роль відіграє те, який саме ігровий рушій використати. В нашому випадку немає завдання створювати реалістичну графіку, тому вибір пав на ігровий програмний рушій Unity. Також, використання цього рушія зумовлено великою кількістю навчального матеріалу та легкістю використання інструментарію. Не останню роль на вибір зіграв відкритий форум розробників, де можна звернутись за допомогою до професіональних розробників ігрових

проектів, а також мультимедійних роликів. Відкрита документація, доступна на багатьох мовах світу допоможе знайти відповіді на питання щодо роботи тих чи інших можливостей ігрового програмного рушія.

1.3 Вибір інструментів розробки

Після аналізу доступних програмних рушіїв, наступник кроком є обрання інструментів розробки. Перш за все необхідно обрати редактор коду. Редактор коду – це спеціалізоване програмне забезпечення, яке використовується для створення, модифікації та організації вихідного коду програм. Такі редактори зазвичай мають інтуїтивно зрозумілий інтерфейс, підтримують підсвічування синтаксису, автоматичне доповнення коду, пошук та заміну, а також інтеграцію з іншими інструментами розробки, наприклад, з системами контролю версій, дебагерами та компіляторами. Серед відомих редакторів коду можна виділити Visual Studio Code та Sublime Text.

У процесі вибору програмного забезпечення для редагування коду я віддав перевагу Microsoft Visual Studio. Цей вибір був обумовлений сумісністю з ігровим двигуном Unity, який використовує мову програмування C#. Важливо було вибрати середовище, яке не тільки допомагає з форматуванням коду, але й забезпечує зв'язок з іншими класами, методами, ігровими об'єктами та бібліотеками. Крім того, Microsoft Visual Studio надає доступ до проекту гри та рекомендовано для використання розробниками Unity.

Для роботи з графікою потрібно обрати графічний редактор. Графічний редактор – це програмне забезпечення, яке дозволяє створювати та модифікувати графічні зображення, використовуючи різноманітні інструменти, такі як кисті, олівці, спреї, штампи, інструменти виділення тощо. Також вони пропонують можливості для редагування кольору, розміру, прозорості та інших параметрів зображень. Серед відомих графічних редакторів можна назвати Adobe Photoshop, GIMP, CorelDRAW та Adobe Illustrator.

У контексті розробки цієї гри я вирішив використовувати Adobe Photoshop, оскільки маю досвід роботи з цим інструментом. На поточному етапі розробки гра не вимагає складної графіки, а лише створення простих спрайтів для ігрового

процесу, а також ймовірно зміну співвідношення сторін зображень.

Важливою частиною процесу розробки є використання системи контролю версій, яка дозволяє керувати змінами у вихідному коді та інших файлах проекту. Система контролю версій забезпечує відстеження змін, порівняння версій, відкат до попередніх станів та об'єднання змін від різних розробників, а також зберігає історію змін, що сприяє спільній роботі над проектом. Популярні системи контролю версій включають Git, Subversion та Mercurial.

Для мого проекту я вибрав BitBucket та SourceTree. BitBucket – це веб-базована система контролю версій, яка є безкоштовною та інтегрована з багатьма іншими додатками від Atlassian. SourceTree – це клієнтська програма з графічним інтерфейсом, яка спрощує роботу з системами контролю версій, зокрема з Git. Це програмне забезпечення дуже зручне для зберігання результатів розробки в інтернеті.

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

Для початку розробки ігрового проекту будь-яка команда розробників починає з розробки концепт-документу, який дещо схожий на дизайн-документ, але відрізняється тим, що його зміст може бути дещо змінений – скорегований – відносно реалій розробки. Концепт-документ гри – це документ, який містить основні концепції, ідеї, сюжетні елементи, механіки геймплею та інші ключові аспекти, пов'язані з розробкою гри. Цей документ зазвичай є основою для її подальшого розвитку проекту з часом дещо змінюючись. У концепт-документі гри частіше за все розглядають наступні питання:

- Опис гри: Загальний опис ігрового світу, сюжету, атмосфери та основної мети гри;
- Механіка геймплею: Опис ігрових механік, механіки управління, фізики ігрового світу, системи прогресії, системи бою та інших ігрових аспектів;
- Сюжет та персонажі: Опис сюжету гри, персонажів, їх характеристик, мотивацій та відносин між ними;

- Графіка та аудіо: Концепції та ідеї з візуального стилю, арт-дизайну, анімації та звукового оформлення гри;
- Світогляд і фон: Додаткові деталі про світ гри, його історію, культуру, технології та інші аспекти, які можуть впливати на ігровий досвід.
- Технічні вимоги та можливості: Обговорення технічних аспектів розробки гри, включаючи платформи, програмні рушії, інструменти та інше.

За допомогою концепт-документу, команда розробників має уявлення про те, який проект вони отримують в кінці, а також можуть планувати його розробку. Також такий документ дозволяє уточнити деякі аспекти розробки.

Оскільки метою розробки є створення гри в жанрі топ-даун шутер, необхідно реалізувати його основні ігрові елементи, а також механіки. Для цього було дослідження інші аналогічні проекти, та виведено з них основні положення. Також потрібно не забувати про реалізацію інтерфейсу, згідно ігрового процесу. Томи потрібно розпочати з реалізації головного меню гри. Оскільки це початкова версія гри, яка має на меті реалізувати основний ігровий функціонал та створити базу для наступних доповнень, то можна на даному етапі знехтувати графікою у головному меню гри та ігровому процесі.

Головне меню на початковому етапі існування проекту має давати основні елементи управління у грі. Мають бути реалізовані кнопки початку нової гри, налаштування гри а також виходу з гри. В свою чергу при натисканні кнопки початку нової гри, потрібно щоби з’являлась можливість обрати рівень складності. Таких рівнів має бути три, від легкого до складного. Початково, рівні складності будуть впливати на те, скільки пошкоджень буде отримувати гравець від атак ворогів. У майбутньому рівень складності може впливати і на інші параметри ігрового процесу, такі як кількість набоїв у гравця, відстань на якій вороги реагують на гравця. Переглянути концепт ігрового меню гри можна на рисунку 1.9.

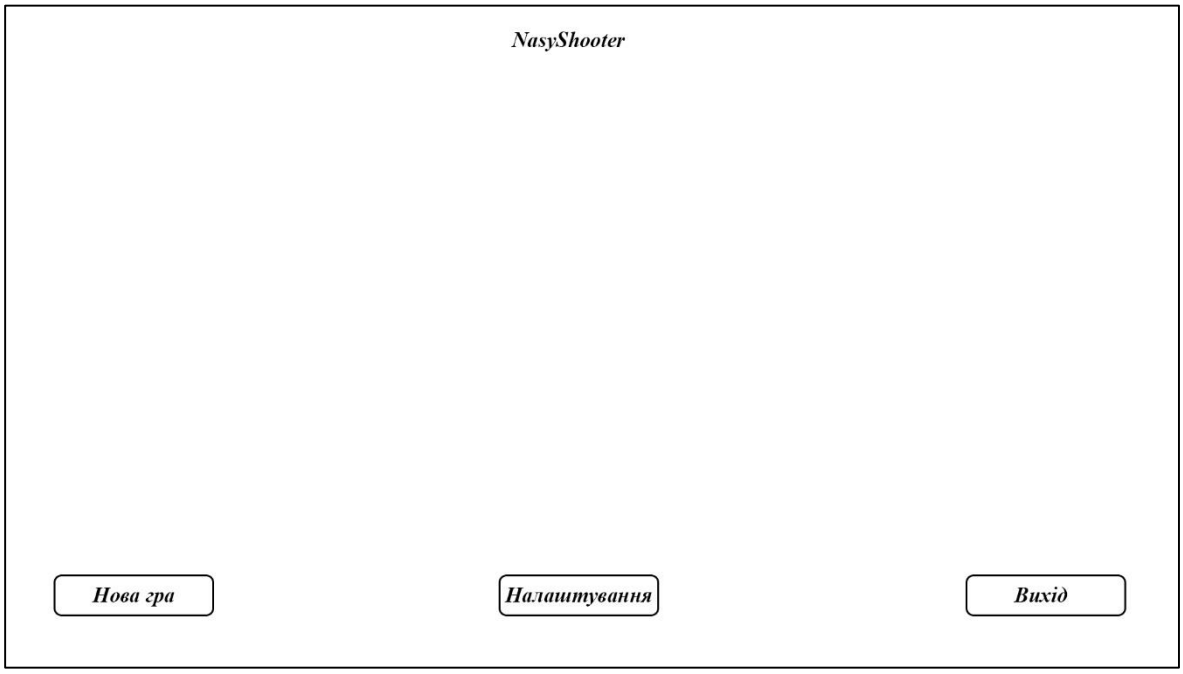


Рисунок 1.9. Концепт ігрового меню гри

Щодо реалізації ігрової частини проекту, то на рисунку 1.10 можна побачити концепт ігрового процесу. На ньому можна побачити такі елементи інтерфейсу як показник здоров'я гравця, його броні, у нижньому лівому та правому куті екрану відповідно. Також всередині ігрового інтерфейсу можна побачити панель зі зброєю.

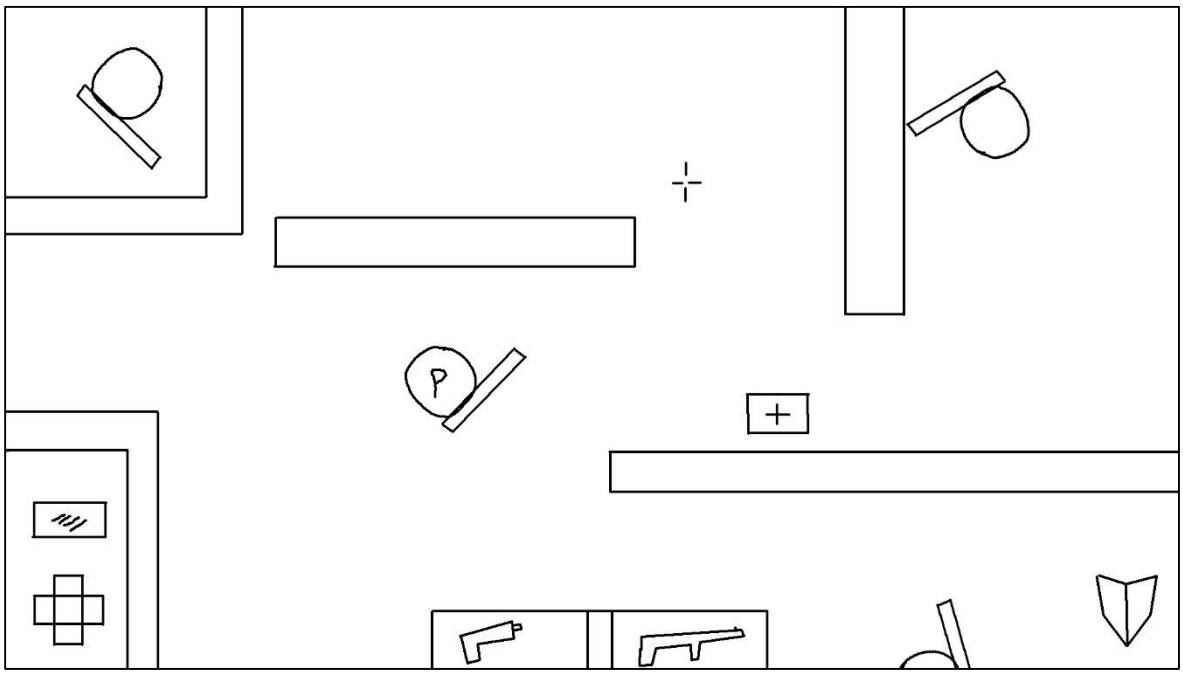


Рисунок 1.10. Концепт ігрового процесу

Виходячи з концепту, також, можна побачити елементи які гравець зможе підбирати. Такими об'єктами будуть аптечка, набої, броня. Кожний з таких

об’єктів додає гравцю його ігровий ресурс. Тобто маємо на увазі, що у гравця будуть у розпорядженні параметри здоров’я, броні, набоїв. Кількість таких ресурсів можна збільшити, додавши до них й інші види ресурсів для того щоби розширити ігрову різноманітність.

На рисунку 1.10 також можна бачити гравця, якій слідкує за перехрестям прицілу. Він має бути реалізований за допомогою прихованого курсору, а за його координатами має відображатись перехрестя. Гравець має змогу переміщатись по рівні у різні сторони – вліво, вправо, вверх та вниз, а також по діагоналям.

Разом із гравцем на рівні мають бути присутні вороги, які будуть реагувати на гравця у залежності від того, як близько гравець буде підходити до них. При цьому гравець має можливість бачити їх заздалегідь, щоби мати можливість приготуватись до бою. Постріли реалізовуються за допомогою клавіш миші, а також зміною зброї. Кожний постріл буде створювати фізичну кулю, яка буде переміщуватись по рівню, доки не зіткнеться із перешкодою, або гравцем, чи його ворогом. Тобто кулі не мають довжини дії та рухаються по вектору пострілу.

Як і у гравця, у ворогів мають бути параметри здоров’я, адже якщо цього не реалізувати, тоді ігровий процес буде досить нудним через швидку перемогу над ворогами. Слід зробити окремий параметр для швидкості пострілів для того, щоби гравець мав можливість вийти з зони ураження, використовуючи свою увагу.

1.5 Проектування основних елементів ігрового процесу та принципи їх роботи

Для вдалої реалізації ігрового проекту, необхідно попередньо виконати проектування основних ігрових елементів. Це дає змогу заздалегідь виявити можливі помилки під час реалізації елементів. Перед початком проектування, потрібно скласти структурну схему всього проекту, з урахуванням всього його функціоналу. Це допоможе структурувати розробку. На рисунку 1.11 можна побачити таку структурну схему всього ігрового проекту.

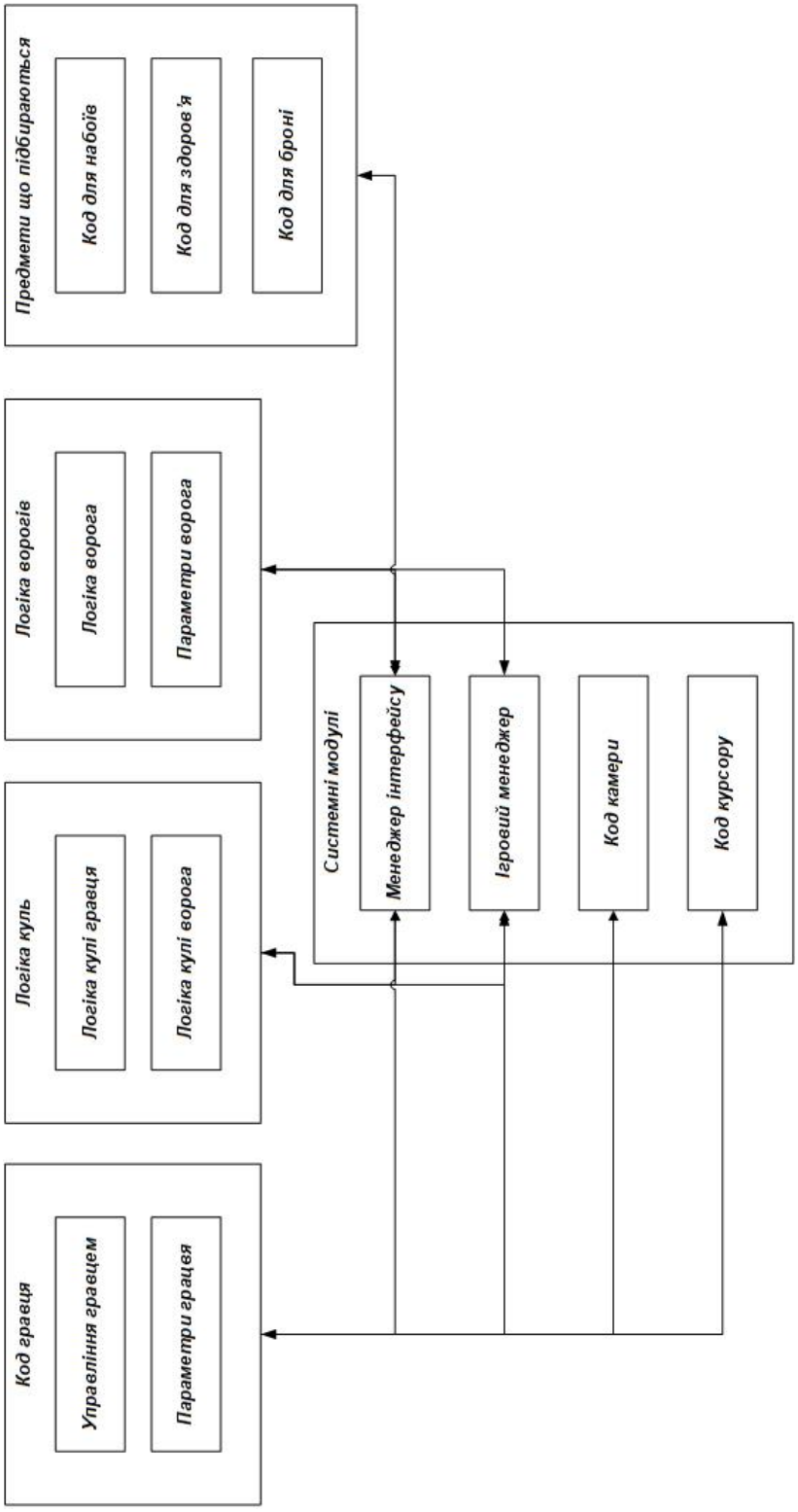


Рисунок 1.11. Структурна схема всього ігрового проекту

Розглянемо отриману схему ігрового проекту а також її модулі. Як можна з неї побачити, вона чітко ділиться на групи, які пов’язані за функціоналом. Також з цієї схеми можна побачити зв’язки між цими групами, які показують, що кожна група тісно взаємодіють одна з одною, але через посередника у лиці системних модулів. Почнемо розглядати проект саме з цієї групи.

До складу Системних модулів входять Менеджер інтерфейсу, Ігровий менеджер, Код камери та Код курсору. Завдання Коду курсору – відстежувати положення курсору та повертати його позицію за потребою. Окрім того, цей код відповідає за відображення перехрестя прицілу. Код камери відповідає за роботу із камерою, повертає її параметри, положення курсору на ній, а також виконую код, які слідкую камерою за гравцем, повторюючи його рух в сторони. Ігровий менеджер виконує завдання з контролю ігрових подій.

Менеджер інтерфейсу відповідає за роботу з усім інтерфейсом гри, починаючи з ігрового меню, та закінчуючи ігровим процесом. На рисунку 1.12 зображено схему взаємодії модуля з іншими модулями гри. Саме цей модуль виконує зміну інтерфейсу, у разі взаємодії із ним. Також, цей модуль виконує роль відображення ресурсів гравця, які відображаються на інтерфейсі під час гри. Оперативну зміну показників здоров’я, броні та набоїв.

Я можна бачити зі схеми, даний модуль можна назвати прошарком між ігровим процесом та елементами інтерфейсу гри. У ігровому рушії Unity інтерфейс та його елементи знаходяться окремо від ігрових об’єктів, тому вони не можуть з ним взаємодіяти. Через це необхідно налагоджувати зв’язок між ігровим процесом та інтерфейсом за допомогою такого модуля Менеджеру інтерфейсу. Цікавим є той факт, що з цим модулем взаємодіє лише два елемента гри – Модуль гравця, який по суті є модулем Параметри гравця, та Ігровий процес, а саме у той частині, коли відбувається взаємодія із предметами, які можна підібрати. Інші ігрові елементи проекту не мають взаємодію із цим модулем через те, що по суті вони не виконують ніякої роботи, що впливала б на інтерфейс гравця.

Також можна відмітити той факт, що у разі звернення до цього модуля, на виході він завжди взаємодіє із інтерфейсом гравця, або ігровим меню. Це відбувається через те, що даний модуль виконує лише операції зміни інтерфейсу гравця на головного меню.

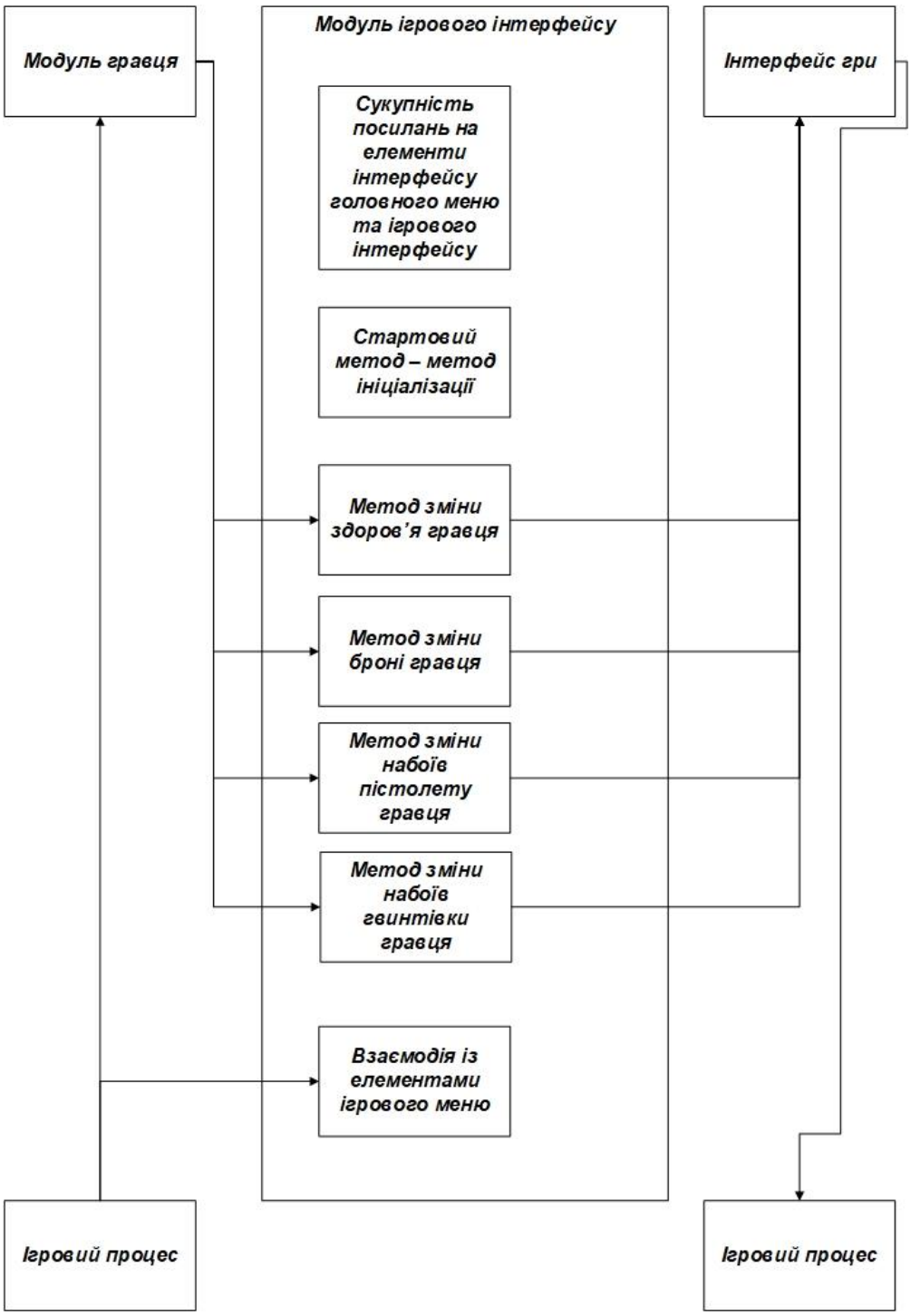


Рисунок 1.12. Схема взаємодії модуля Менеджера інтерфейсу з іншими модулями гри

У групі модулів Код гравця розміщені такі модулі як Управління гравцем та Параметри гравця. Управління гравцем реалізує механізми управління гравцем під час гри. Вмикає систему управління, слідкує за швидкістю гравця, а також виконує роль коду, що ініціює постріли при натисканні лівої кнопки миші. Параметри гравця зберігають у собі дані про параметри гравця, або його ігрові

ресурси. Такими значеннями є здоров'я гравця, його броня, кількість набоїв у зброї а також її наявність. Також саме цей модуль має виконувати процес віднімання значень здоров'я, броні, набоїв тощо, за допомогою спеціальних методів. Більше детально на схему взаємодії модулю Параметри гравця з іншими модулями гри можна побачити на рисунку 1.13:

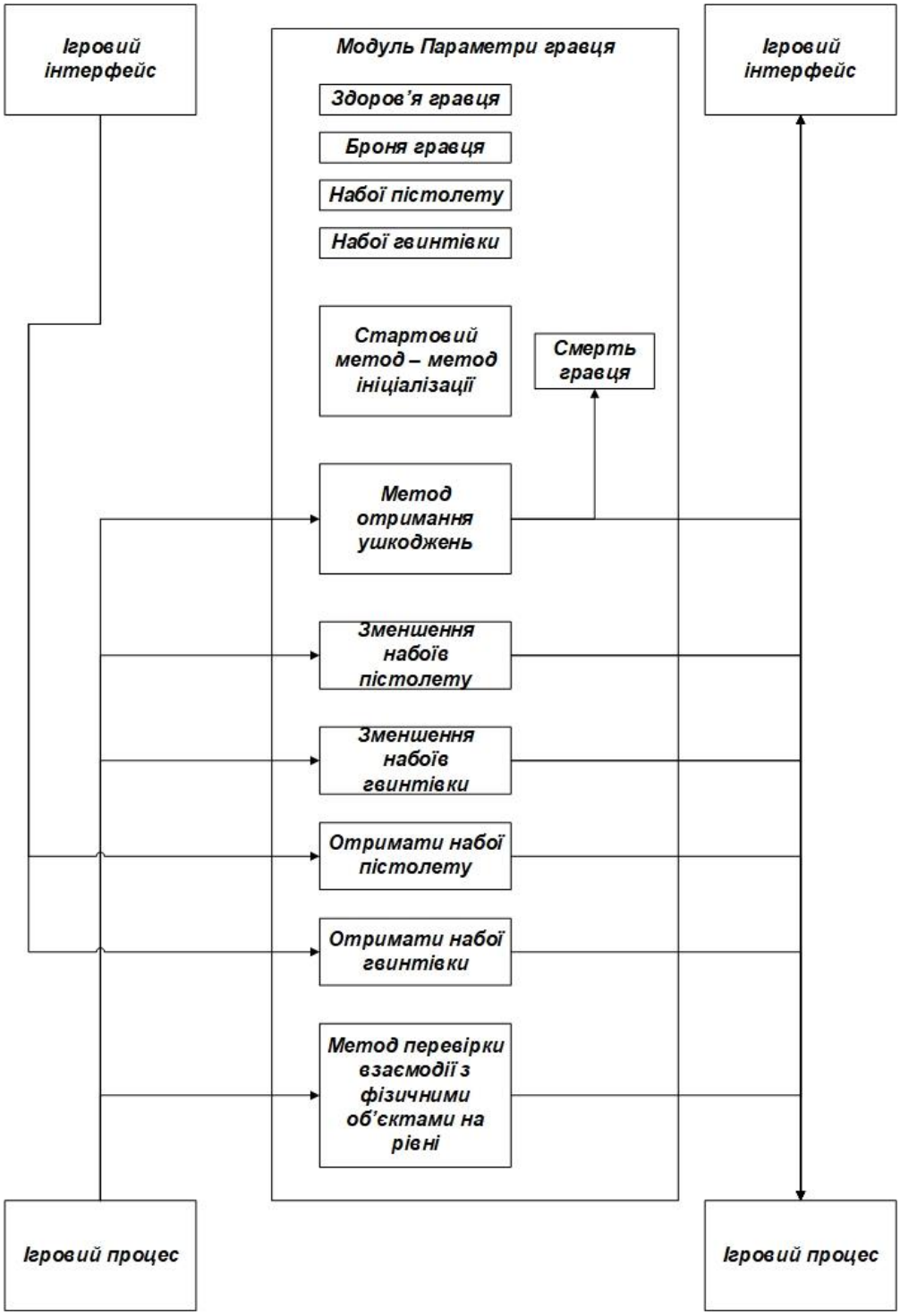


Рисунок 1.13. Схема взаємодії модулю Параметри гравця з іншими модулями гри

Група Логіки куль відповідає за реалізацію поведінки куль у грі. Оскільки кожна куля фізично знаходиться на рівні, виходить, що кожна куля має свою чергову поведінку, яка не може бути представлена простим кодом. По суті модулі Логіка кулі гравця та Логіка кулі ворога реалізують різницю між цими кулями не даючи їм можливості виконувати неправильні дії.

Група Логіки ворогів, як слідує з її імені, відповідає за логіку ворогів. На початковому рівні проекту, буде реалізовано лише один тип ворогів, але це закладає основу для майбутнього розвитку проекту. Модуль Логіка ворога відповідає за поведінку ворога, а саме його реакцію на гравця, положення на рівні. Цей модуль є основою для майбутніх покращень поведінки ворогів. Модуль Параметри ворога виконую роль схожу із тим, що робить модуль Параметри гравця. Тобто стежить за характеристиками ворога, його здоров'ям та іншими параметрами, які можуть бути в ньому реалізовані. Оскільки кожний ворог генерується як незалежна одиниця, то відповідно для кожного ворога створюється свій екземпляр такого коду.

Нарешті група Предмети що підбираються поєднує в собі модулі, які програмно реалізують ігрові об'єкти, які гравець може використати для отримання ігрових ресурсів. Наприклад в поточній версії проекту мають бути реалізовані аптечки, броня та набої, які гравець має підбирати на рівні. По суті кожний такий модуль зберігає в собі значення ресурсів, які необхідно передати гравцю, а також код, який і реалізує передачу. На рисунку 1.14 можна побачити типову структурну схему модуля для об'єкту, який можна підібрати. Як видно із схеми, даний модуль є лише допоміжним ланцюгом у алгоритмі передачі значень під час ігрового процесу. По суті, якщо гравець під час ігрового процесу створює ситуацію, коли натикається на такий предмет, то він викликає метод для отримання ресурсу, що зберігається в такому предметі. Після цього рушій передає управління назад з модулю предмету до модуля гравця та повертає його назад, до ігрового процесу.

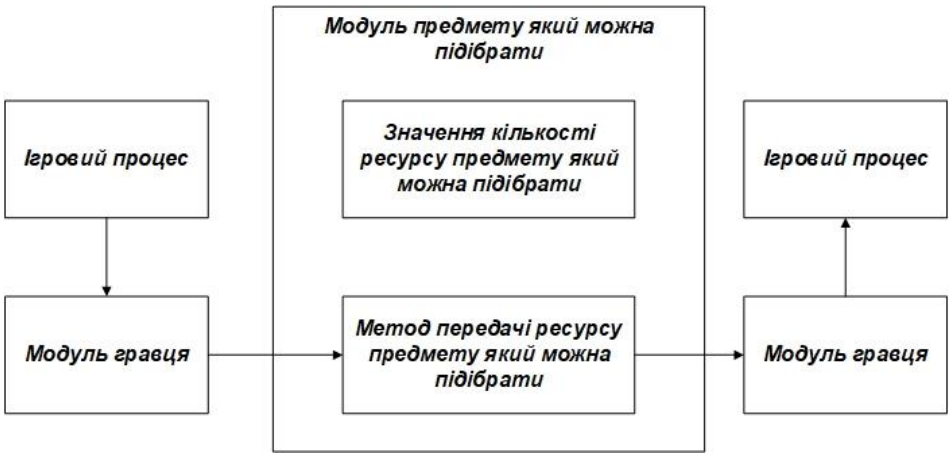


Рисунок 1.14. Типова структурна схема модуля для об’єкту, який можна підібрати

Як результат розробки проектування, було отримано структурні схеми ключових модулів гри. Деякі елементи ігрового процесу виявились простими у проектуванні, а інші досить складними у своїй роботі та взаємодії з іншими елементами розроблюємого проекту.

1.6 Реалізація основних елементів ігрового процесу

Процес реалізації основних елементів ігрового інтерфейсу розпочинається з реалізації графічного матеріалу за допомогою графічного редактору Adobe Photoshop. Для даного проекту я вирішив використовувати стилізацію під гру на листку бумаги в клітинку, тому вирішив реалізовувати елементи графіки в такому вигляді, як показано на рисунку 1.15. В такому ж стилі реалізовані й графічні інші елементи гри.

Окремо слід зазначити про реалізацію елементів графіки, які пов’язані із побудовою рівнів. Оскільки гра реалізується виключно у 2D-просторі, а також гра створюється в жанрі топ-даун шутер, потрібно створити комплект графічних елементів, з яких можна було б скласти рівень, ніби з конструктору. Для цього потрібно реалізувати комплекс спрайтів для полу, клітинки-стіни, групи клітинок-стіни. Маючи такий набор можна створювати рівні різної комплекції, від прямих до діагональних. Такий простий набор дає змогу швидко створювати локації, групуючи отримані об’єкти в один для того, щоби вони не заважали. На рисунку 1.16 можна побачити графічні елементи для створення рівнів.

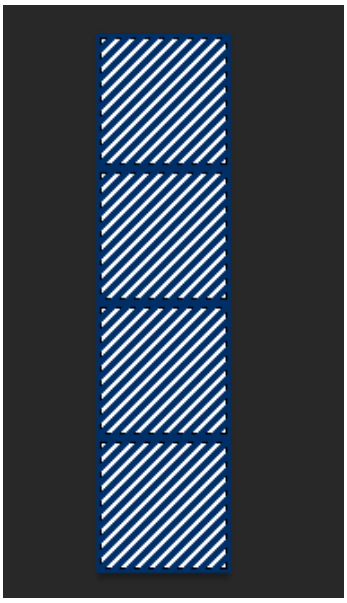


Рисунок 1.15. Приклад стилістики графічних елементів проекту

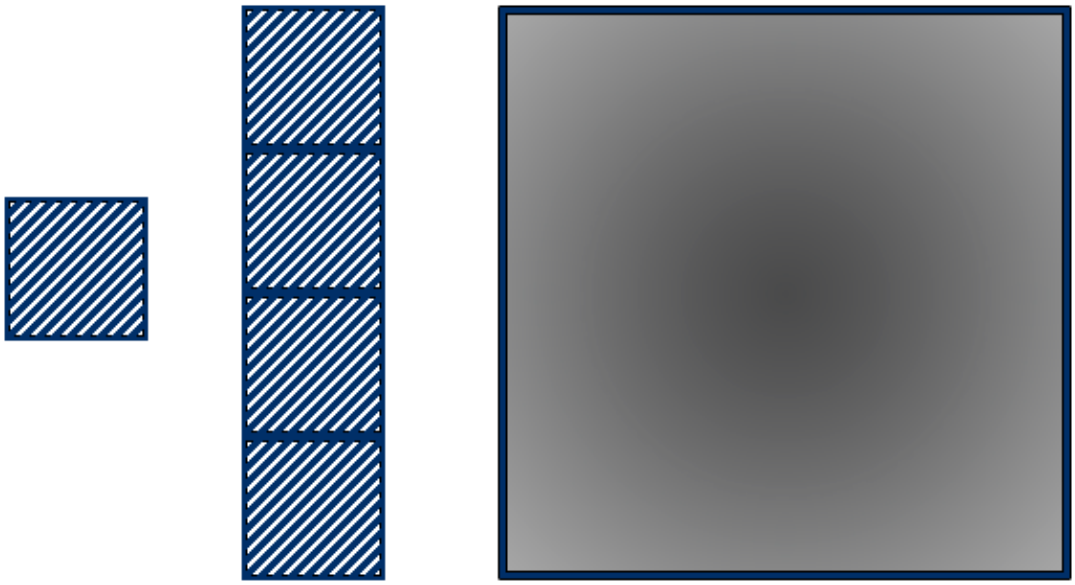


Рисунок 1.16. Графічні елементи для створення рівнів гри

Також, потрібно створити графічні елементи для інтерфейсу. Потрібно реалізувати елементи для відображення значення здоров’я, броні зброї, набоїв та прицілу. Для цього також, як і для рівнів буде використано стилістику паперової гри. Всі графічні елементи створено власноруч.

Таким же чином у Adobe Photoshop створювались спрайти для гравця, його ворогів та об’єктів, які можна підібрати та кулі, які також відображаються на рівні під час свого переміщення по ньому. Також були створені спрайти для карток дверей на рівні. Всі реалізовані спрайти можна переглянути на рисунку 1.17.

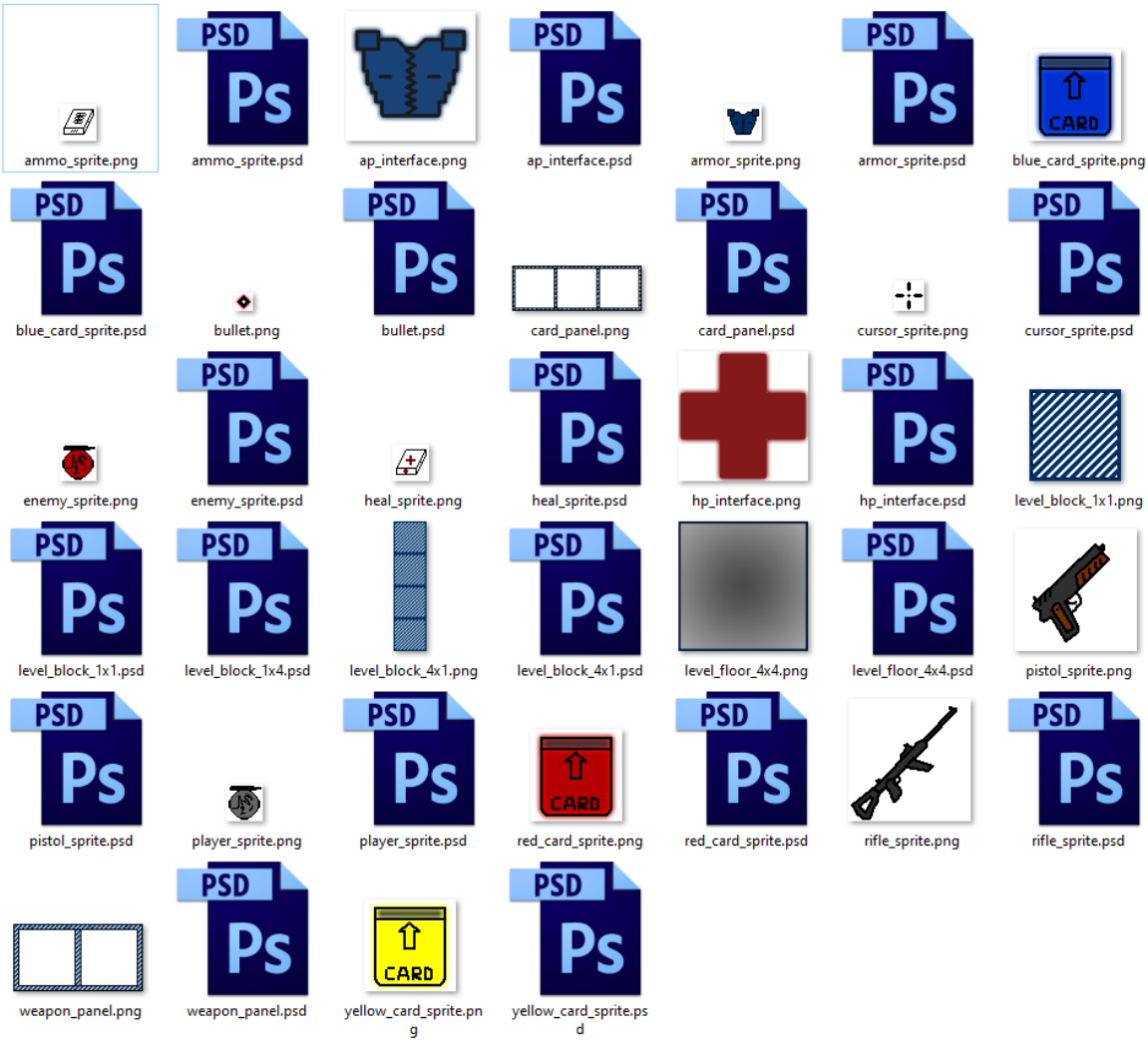


Рисунок 1.17. Всі спрайти реалізовані для проекту гри

Перед початком роботи над кодом проекту, потрібно розмістити елементи інтерфейсу. Те, як має виглядати головне меню гри було розроблено на етапі побудови концепту гри та ігрового процесу. Для створення ігрового меню я використовував елемент рушія Canvas. Через нього створюються всі елементи UI, або інтерфейсу користувача. Собою він представляє окремий простір, який можна редагувати у відриві від безпосередньо ігрового простору.

Елементи меню реалізовуються за допомогою елементів Panel, Button та Text. Panel використовується як основа для меню, на яку у майбутньому можна розмістити зображення концепт-арту гри. Елементи Button будуть викликати відповідний функціонал, як початок нової гри, налаштування гри або вихід з гри. Нарешті елемент Text використовується для зображення найменування гри. На рисунку 1.18 можна побачити кінцевий вид ігрового меню. Функціонал кнопки

Нова гра створюється таким же чином за допомогою такого ж набору елементів. Результат створення підменю для вибору складності нової гри можна переглянути на рисунку 1.19. Налаштування реалізовуються таким же чином.

Рисунок 1.18. Кінцевий вид ігрового меню

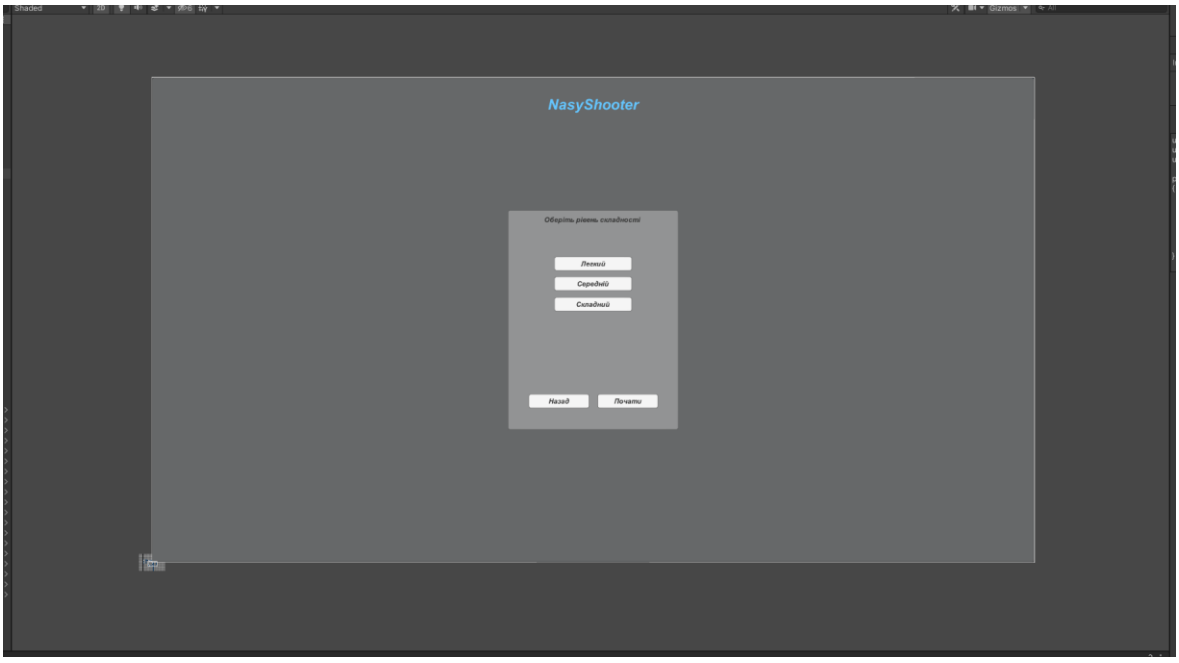


Рисунок 1.19. Підменю вибору рівня складності для нової гри

Після створення інтерфейсу для головного меню та його підменю, було реалізовано інтерфейс ігрового процесу. Для цього були використані елементи Image та Text. Оскільки інтерфейс гравця не функційний – гравець не може натискати на нього, він реалізований як зображення за допомогою відповідного

елемента Image. Такі елементи інтерфейсу як показник здоров’я та кількість набоїв гравця відображаються за допомогою текстового елемента. На рисунку 1.20 можна побачити готовий інтерфейс гравця у режимі розробки. Слід зазначити, що відсутність цифрових позначень на параметрах здоров’я, броні та набоїв зброї не є помилкою, оскільки цифри відображаються під час ігрового процесу.

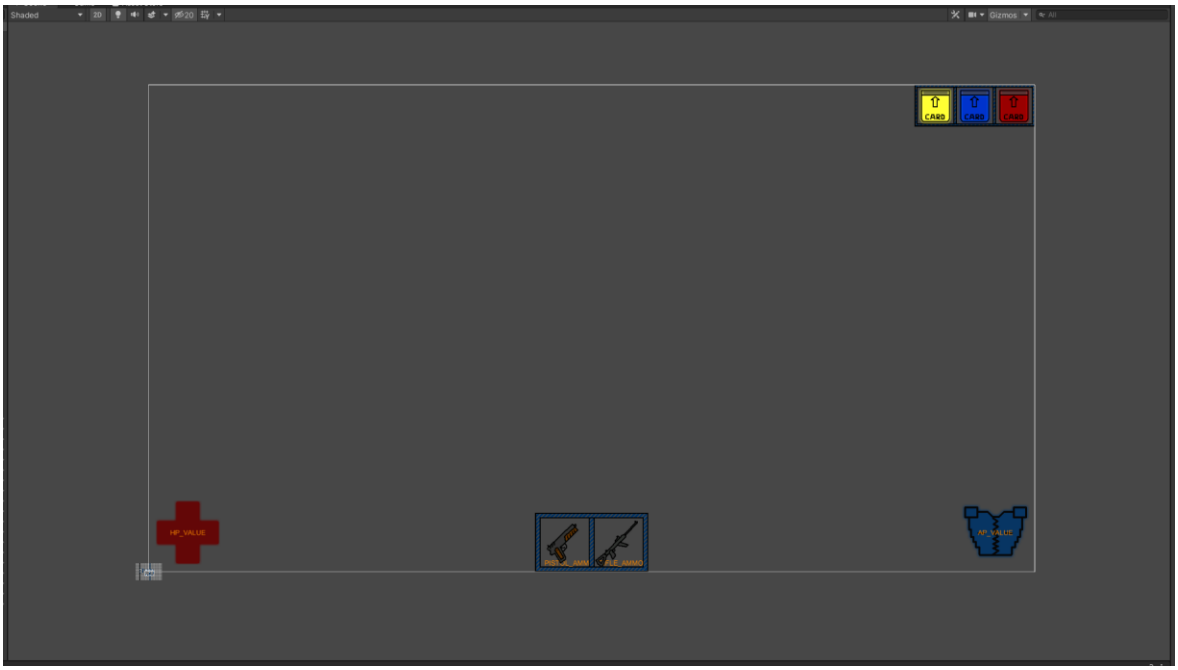


Рисунок 1.20. Готовий інтерфейс гравця у режимі розробки

Після реалізації графічної частини інтерфейсу гри, необхідно створити рівень, на якому буде виконуватись процес розробки. Рівень створюється за допомогою спрайтів, які були створені раніше. З елементів рівня збирається стартовий рівень. На ньому розміщуються гравець, вороги, предмети, які можна підібрати. Елементи рівня групуються у один пустий ігровий об’єкт, для біль зручної навігації по інтерфейсу ігрового рушія.

Слід зазначити, що ігрові об’єкти рівня розміщувались за допомогою ігрового об’єкту Grid, який дозволяє розмістити на ігровій поверхні уявну сітку, на якій можна розміщувати спрайти рівня. Сітка дозволяє легко пересувати спрайти, прикріпляючи їх до країв сітки. На рисунку 1.21 можна побачити скріншот готового рівня гри.

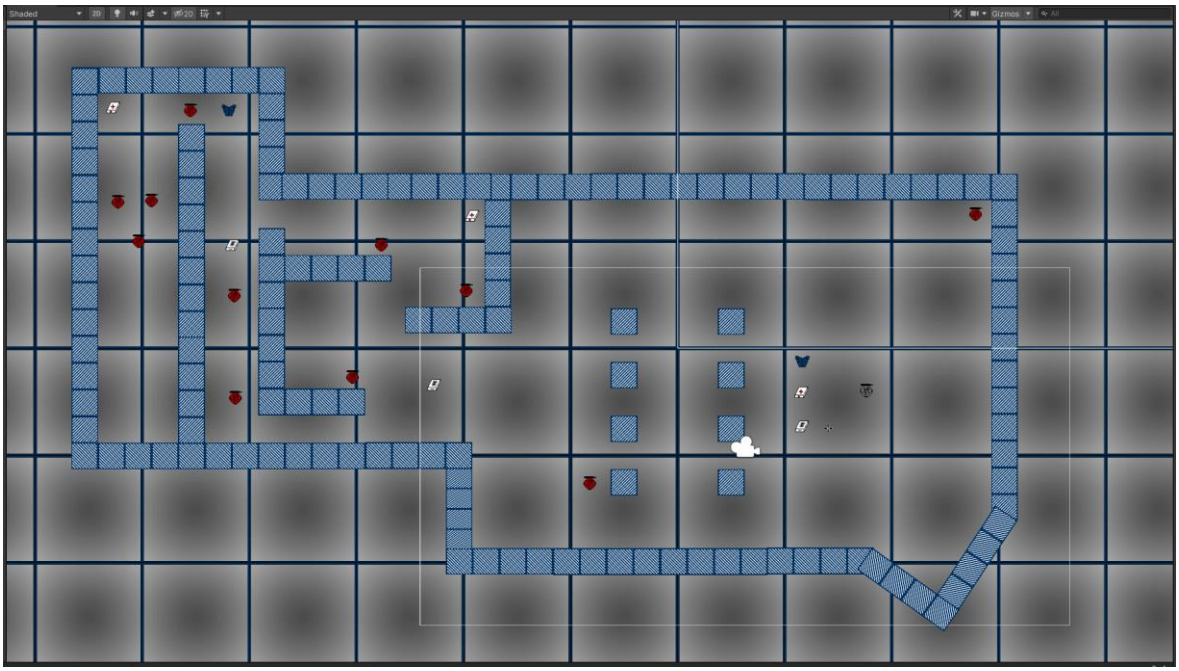


Рисунок 1.21. Скріншот готового рівня гри

Після закінчення реалізація елементів інтерфейсу та рівня необхідно перейти до реалізації безпосередньо коду гри та основних ігрових модулів. Їх перелік можна було побачити на структурній схемі гри. Розробку необхідно розпочати з коду параметрів гравця. Його структура була наведена раніше. У ігровому рушії Unity код гри пишеться у скриптах, які додаються до ігрових об’єктів. Топу код параметрів – player_parameters.cs – гравця ми додаємо до ігрового об’єкту гравця.

Виходячи зі схеми модуля, можна поділити скрипт на частину із оголошенням змінних, їх ініціалізацією. За це відповідає метод Start() який є стандартним модулем будь-якого скрипта Unity. Всередині цього метода я ініціалізую параметри гравця значеннями для того, щоби не створювати проблем зі зверненням до змін у майбутньому. У подальшому, в багатьох скриптах буде виконуватись цей процес, тому нижче представлений приклад методу Start():

```
void Start(){
    player_hp = 100;
    player_ap = 100;
    pistol_ammo = 0;
    rifle_ammo = 0;
    yellow_card = false;
    blue_card = false;
    red_card = false;
```

```
        _UI_manager._set_player_hp(player_hp);
        _UI_manager._set_player_ap(player_ap);
        _UI_manager._set_pistol_ammo_text(pistol_ammo);
        _UI_manager._set_rifle_ammo_text(rifle_ammo);
    }
```

Слід виділити тут звернення до такого об’єкту як _UI_manager. Це змінна-посилання на об’єкт який зберігає в собі скрипт ui_manager.cs. Ці звернення виконуються для передачі у модуль інтерфейсу даних про кількість здоров’я гравця, його броні та кількості набоїв. Оскільки ініціалізація виконується у методі Start(), то цей код буде викликаний відразу, як буде створений ігровий об’єкт гравця. А це значить, що показники параметрів гравця відразу ж, із першим кадром, будуть передані до інтерфейсу гравця. Код менеджеру інтерфейсу розглянемо пізніше.

Ще одною важливою частиною коду параметрів гравця можна виділити метод take_damage(), який отримує параметр значення пошкоджень, які отримує гравець. Метод цікавий тим, що виконує підрахунок пошкоджень з урахуванням параметру броні. Тобто, якщо в гравця є броня, то в першу чергу кількість пошкоджень уходить в броні, якщо вона закінчилась, тоді решка пошкоджень віднімають здоров’я. Якщо ж у гравця при отриманні пошкоджень немає броні, тоді вони відразу віднімають параметр здоров’я. Після виконання маніпуляцій, код передає до модуля ігрового інтерфейсу команду замінити поточні значення текстових полів здоров’я та броні гравця на нові. Якщо ж після всіх цих дій виявилось, що кількість здоров’я гравця дорівнює, або менше нуля, тоді ігровий об’єкт гравця знищується. Даний метод має публічний доступ, який дає змогу викликати його зовні. Це дуже зручно, оскільки під час ігрового процесу велика кількість скриптів зовні буде мати змогу нанести гравцю пошкодження. Нижче приведено код методу take_damage():

```
public void take_damage(int damage_value)    {
        if(player_ap > 0){
            player_ap -= damage_value;
            if(player_ap < 0){
                player_hp += (player_ap);

                player_ap = 0;
```

```
            }
        }
        else{
            player_hp -= damage_value;
        }
        _UI_manager._set_player_hp(player_hp);
        _UI_manager._set_player_ap(player_ap);
        if (player_hp <= 0)
            Destroy(this.gameObject);
    }
```

Окрім того, в даному скрипті реалізовано методи для зменшення кількості набоїв, та передачі до зовнішніх скриптів даних про кількість набоїв у гравця. Оскільки ці методи можуть бути викликані зовні, то їх модифікатор доступу встановлений у public.

Гравець отримав параметри, але не може рухатись. Для реалізації цієї важливої частини ігрового процесу топ-даун шутеру створимо скрипт player_control.cs. Це досить простий у своєму початковому вигляді код. Рух ігрового об’єкту по поверхні рівня виконується у одні площині, тому достатньо отримувати параметри руху по осям горизонтального та вертикального руху, після чого передавати його до фізичного тіла гравця у вигляді сили руху. Отримання параметрів руху по горизонталі та вертикалі виконується через спеціальний об’єкт типу Input. Цей об’єкт зберігає всі дані, які стосуються вводу гравця, будь то клавіатура, миша, або гарнітура віртуальної реальності.

Втім є одна особливість у роботі з вводом гравця. Якщо ми говоримо про рух об’єкту у просторі, то логічним є те, що він отримує свої данні для руху у парадигмі простір-час. У ігрового рушія є метод Update який виконує звій зміст кожний кадр, який малює ігрова станція. Якщо було створено 64000 кадрів у секунду, то код цього методу буде виконаний 64000 разів. Насправді ми можемо зчитувати натискання клавіш клавіатури та інших пристроїв вводу таку кількість разів, але функціонал руху вписувати у цей метод заборонено! Для руху об’єктів існує інший метод FixedUpdate – він використовується фіксовану кількість кадрів, а це значить, що його зміст буде виконаний фіксованих 60 разів. На рисунку 1.22 схематично зображено різницю між цими методами.

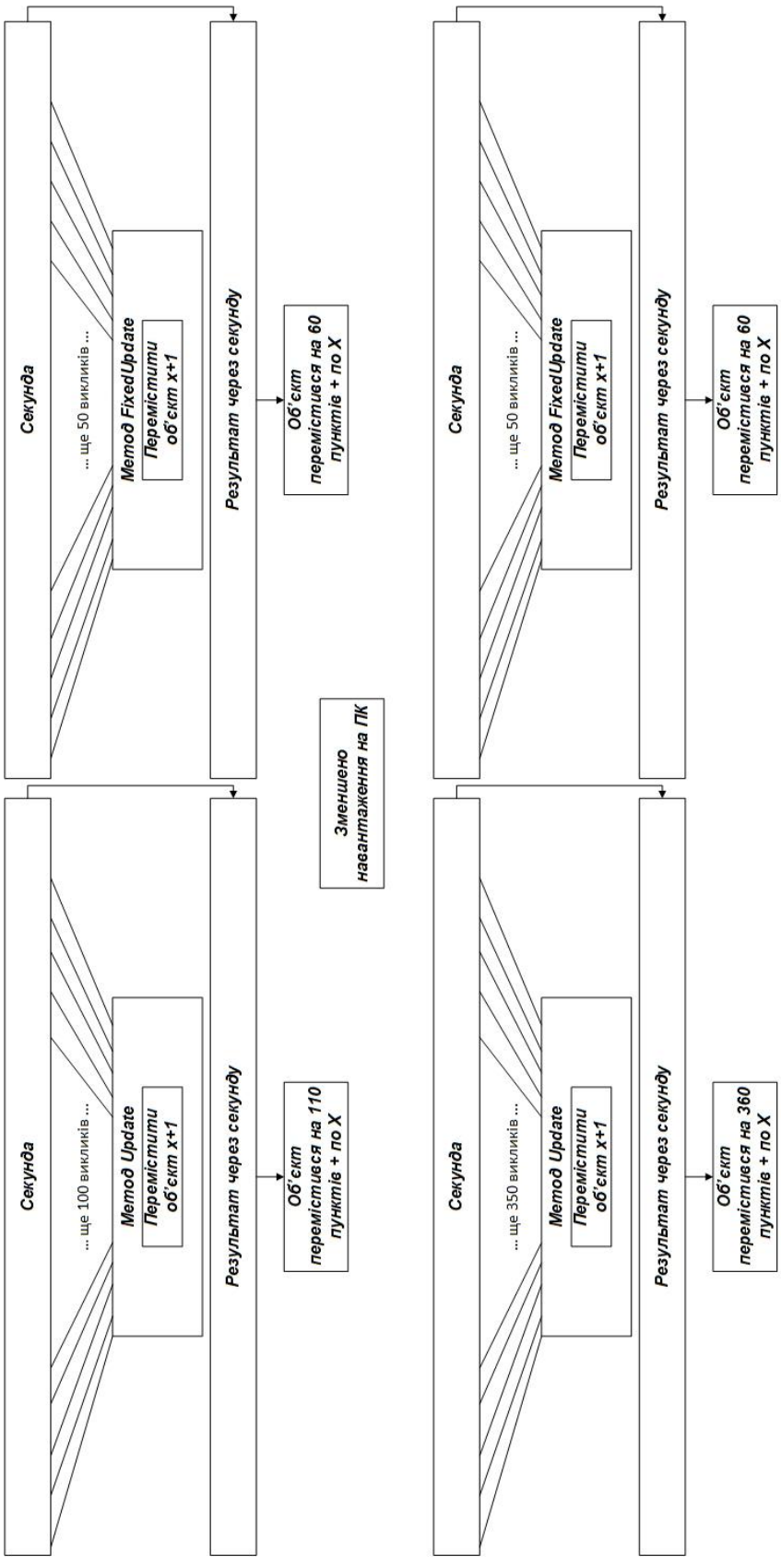


Рисунок 1.22. Різниця між методами Update та FixedUpdate

Як можна побачити з рисунку 1.22, метод Update дає нестабільний кінцевий результат, який залежить від продуктивності системи. На одному ПК гравець буде переміщуватись з однією швидкістю, а на іншому з іншою. В той же час цей метод

ідеально підходить для виконання швидкого оновлення даних кожний кадр гри, а такими даними можуть буду текстові поля інтерфейсу, або розрахунки. Але треба пам’ятати, що деякі розрахунки залежать від часу, тому для таких потрібно використовувати FixedUpdate. Тому отримувати натискання на клавіши клавіатури ми будемо у методі Update, а рухатись у FixedUpdate. Нижче приведено фрагменти коду процесу генерації руху:

```
void Update()
   {
      horizontal_input = Input.GetAxis("Horizontal");
      vertical_input = Input.GetAxis("Vertical");
   }
private void FixedUpdate()
   {
      player_rb.velocity = new Vector2(horizontal_input * player_speed, vertical_input * player_speed);
   }
```

Як можна бачити у першому методі мі кожний кадр отримаємо натискання на клавіатуру. І використовуємо ці дані тільки у фіксованому виклику 60 кадрів в секунду. Але реалізований код ніяким чином не дає змогу обертатись гравцю навколо себе. Для цього потрібно додати до методу Update код для оберту. Слід зазначити, що оберти виконуються саме в цьому методі тому, що чим частіше будуть розраховуватись на малюватись оберти на екрані, тим плавніше цей процес буде проходити. Нижче приведено код для оберту гравця навколо себе:

```
Vector2 mouse_position = player_camera. ScreenToWorldPoint (Input.mousePosition) - transform.position;
float rotation_z = Mathf.Atan2(mouse_position.y, mouse_position.x) * Mathf.Rad2Deg;
Quaternion rotation = Quaternion.AngleAxis(rotation_z, Vector3.forward);
transform.rotation = rotation;
```

Даний код створює вектор позиції миші на екрані ігрового монітору, після чого інтерполює його у світові координати, для того, щоби гра знали у яку точку ігрового світу вказує курсор на камері, яка рухається по цьому світу. Після цього вираховується кут між поточним вектором напрямку об’єкту гравця та точкою в яку вказує курсор. Цей кут передається до кута параметрів повороту ігрового об’єкту гравця.

Коли наш гравець може рухатись необхідно зробити так, щоби він не проходив крізь об’єкти світу. Для цього слід використовувати два компоненти фізичного рушія ігрового рушія Unity, а саме RigidBody2D та Collider. Перший компонент симулює тверде тіло для ігрового об’єкту, а другий компонент утворює границі цього твердого тіла, для того, щоби фізичний рушій знав, коли це тіло взаємодіє з іншими такими тілами у ігровому просторі. Лише потрібно вимкнути масу, щоби об’єкти на падали в пустий простір рушія. Такі компоненти потрібно додати до всіх об’єктів, які мають властивість фізичної перешкоди для гравця. На рисунку 1.23 зображено ігровий об’єкт гравця з його Collider-ами:

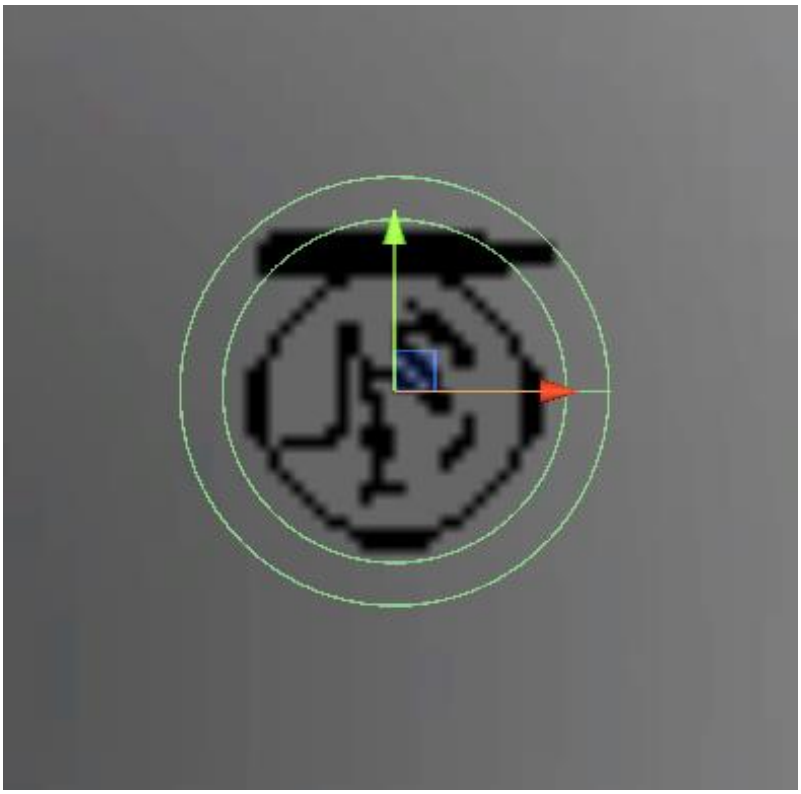


Рисунок 1.23. Ігровий об’єкт гравця з його Collider-ами.

Необхідно пояснити, для чого гравцю дві фізичні грані. У компоненту Collider є ще одна функція – альтернативна стандартній. Цей компонент може виступати в ролі тригеру. Тобто при взаємодії з іншими Collider-ами, які я гранями твердих тіл, викликати подію тригеру. Це дуже зручно, коли потрібно реалізувати зчитування зіткнення з іншими об’єктами. Такий функціонал ми використаємо для реалізації механіки підбору предметів з підлоги гри. Перед тим переглянемо код аптечки:

```
[SerializeField] int heal_value = 0;
```

```
public void pick_up_heal(ref int player_hp_param){
    player_hp_param += heal_value;
}
```

Вище показано весь код для предметів, які можна підібрати з полу. По суті, все, що потрібно для таких ігрових об’єктів – це значення, яке цей предмет буде передавати гравцю, а також метод для передачі цього значення. Слід загострити увагу на той факт, що у цьому методі передачі використовується не звичайний параметр а параметр з модифікатором ref, який передає посилання на змінну, в яку будуть вноситись зміни. Це зроблено через те, що всі предмети які можна підібрати на рівні є префабами – клонами одного предмету, який знаходиться в ресурсах гри. Через це немає можливості прив’язати такі об’єкти до інших. Тепер необхідно реалізувати код для підбирання предметів у скрипті player_parameters.cs, нижче показано цей код:

```
private void OnTriggerEnter2D(Collider2D collision)
   {
      ammo_pickup_params _ammo_pickup_params;
      heal_pickup_params _heal_pickup_params;
      armor_pickup_params _armor_pickup_params;
      _ammo_pickup_params = collision.GetComponent<ammo_pickup_params>();
      _heal_pickup_params = collision.GetComponent<heal_pickup_params>();
      _armor_pickup_params = collision.GetComponent<armor_pickup_params>();
      if(_ammo_pickup_params != null)
      {
         _ammo_pickup_params.pick_up_ammo(ref pistol_ammo, ref rifle_ammo);
         Destroy(collision.gameObject);
         _UI_manager._set_pistol_ammo_text(pistol_ammo);
         _UI_manager._set_rifle_ammo_text(rifle_ammo);
      }
      else if(_heal_pickup_params != null)
      {
         _heal_pickup_params.pick_up_heal(ref player_hp);
         Destroy(collision.gameObject);
         _UI_manager._set_player_hp(player_hp);
      }
      else if (_armor_pickup_params != null)
      {
```

```
            _armor_pickup_params.pick_up_heal(ref player_ap);
            Destroy(collision.gameObject);
            _UI_manager._set_player_ap(player_ap);
        }
    }
```

Розглянемо цей код детальніше. По-перше, це не простий метод, а метод який викликається тригером Collider-а гравця. Коли спрацьовує тригер, він автоматично отримає Collider з яким зіштовхнувся. Оскільки я маю доступ до одного компоненту ігрового об'єкту, то в свою чергу маю змогу отримати доступ до інших компонентів цього об'єкту та їх параметрів, якщо будуть дозволяти модифікатори доступу. Тому при зустрічі тригера гравця з будь-яким фізичним об'єктом на рівні, ми будемо отримувати його зміст.

По-друге, оскільки нам потрібно робити конкретні дії у відповідності до того, з якими компонентами ми зустрілись, потрібно визначити, що це за об'єкт. Ми очікуємо, що це об'єкти, які можна підібрати, тому створюємо змінні-посилання типів таких об'єктів, після чого намагаємось ці посилання отримати. У разі якщо ми не отримали жодного співпадіння, то тригер зустрів будь-шо, але не предмети для підбору. У іншому разі, якщо отримане посилання збігається із його типом, ми викликаємо відповідний метод із цього типу. На рисунку 1.24 зображена схема роботи коду тригеру-Collider-а гравця.

Зі схеми видно, що тригер буде перебирати варіанти об'єктів, які ми йому вказуємо, після чого виконувати ту, чи іншу дію. Наприклад у разі знаходження тригером броні, спочатку буде перевірка на аптечку, потім, оскільки перевірка дала негативну відповідь, код перевірить предмет на наявність коду предмету броні і тоді буде отримано позитивну відповідь. В той же час буде викликано відповідний код із об'єкту броні, з яким тригер-Collider гравця зіткнувся. Таким чином можуть бути створені будь-які перевірки у майбутньому. Будь-то предмети на землі, вороги, або елементи рівня, наприклад двері із реакцією на об'єкти поряд, або графічні елементи, як освітлення.

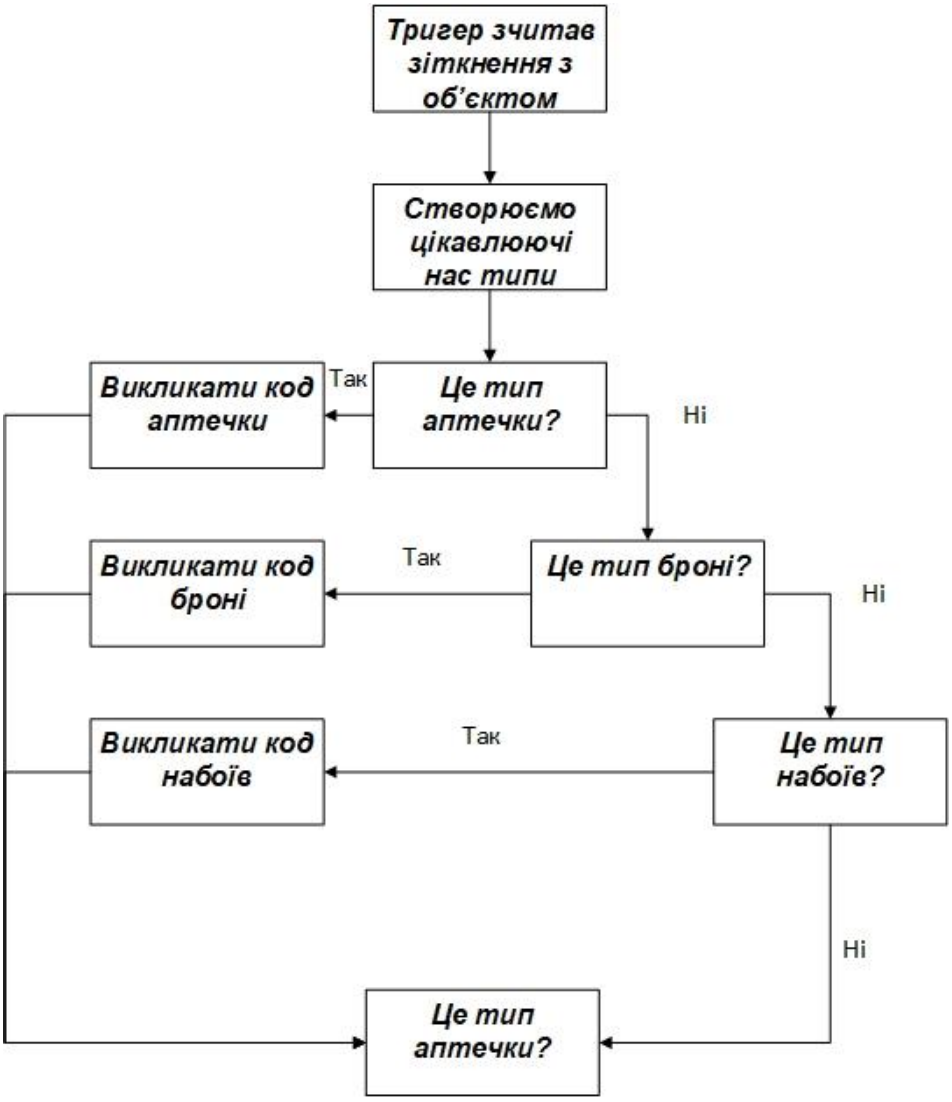


Рисунок 1.24. Схема роботи коду тригеру-Collider-а гравця

Наступним основним елементом ігрового процесу для реалізації є механіка пострілів. Вона має бути реалізована у скрипті управління гравцем, а також захоплює створення коду логіки для кулі гравця. Спочатку потрібно написати код кулі. У кулі є всього один параметр – її швидкість. Вона ініціалізується під час створення кулі. З точки зору структури скрипту, то він має у собі методи руху кулі, та перевірки кулею об'єктів, з якими вона зіткнулась.

Переміщення кулі дуже просте – вона просто рухається вперед по своєму вектору направлення. Вектор направлення куля отримує від гравця, коли той робить постріл.

Код перевірки зіткнення з об'єктами такий самий, як код перевірки предметів, які можна підбирати. Всередині цього метода створюється перелік об'єктів, з якими куля може взаємодіяти, після чого окремо прописується реакція

кулі на всі такі об’єкти. Слід зазначити, що куля буде реагувати зі всіма фізичними об’єктами, втім вона не повинна мати фізичного тіла для того, щоби проходити скрізь деякі об’єкти, наприклад предмети на підлозі. Поки в нас немає коду ворогу, ми не будемо прописувати жодний код.

Більш складна структура коду на стороні гравця, коли він виконує постріл. Потрібно урахувати, зо гравець може виконувати постріли з пістолету та гвинтівки. Таким чином, у методі Update або FixedUpdate ми зчитуємо натискання на пробіл, який переключає зброю. Код пострілу реалізовано в двох методах через швидкість їх зчитування. Просте натискання миші дуже складно зловити FixedUpdate-ом, тому код пістолету реалізовано у Update, який виконує більше перевірок у секунду. В той же час стрільба з гвинтівки реалізована через затискання лівої клавіши миші, тому його можна реалізувати у FixedUpdate, адже скільки б разів ми не перевіряли, статус натиснутої кнопки миші буде зчитаний у будь-якому разі. Отже, перевіривши натискання лівої кнопки миші, якщо режим зброї виставлений на пістолет, тоді ми викликаємо метод стрільби, його код представлений нижче:

```
void weapon_shoot()
{
    Vector3 direction;
    _cursor_follower.take_mouse_position(out direction);
    direction -= transform.position;
    float rotation_z = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
    Quaternion rotation = Quaternion.AngleAxis(rotation_z, Vector3.forward);
    bullet_game_object = Instantiate(_bullet_prefab, transform.position, rotation) as GameObject;
}
```

Код дуже простий і складається у тому, щоби отримати напрямок для кулі із напрямку положення курсору, створити кулю та передати їй цей напрямок. Як тільки куля буде створена в ній спрацює код переміщення вперед та перевірки зіткнень, якщо такі будуть. В той же час, після генерації пулі, код повернеться до частини, де була перевірка натискання пробілу та виконає віднімання набоїв у пістолету, або гвинтівку, в залежності від того, який код викликав постріл. В той же час код зменшення кількості набоїв пістолету, або гвинтівки звернеться до

інтерфейсу через менеджер інтерфейсу та відобразить зміни на інтерфейсі.

Коли були реалізовані кулі, потрібно реалізувати ворогів. Вони складаються майже з того ж комплексу скриптів, що і гравець. У ворогів також є параметри, але вони не звертаються до інтерфейсу. Єдиний метод, що реалізовується в них – код отримання пошкоджень, який зменшує значення здоров'я ворога і у разі зменшення цього показника до 0 або менше, викликає знищення ігрового об'єкту ворога. Код цього методу представлено нижче:

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

Ще один скрипт який реалізовується для ворогів – enemy_logic.cs. Цей код відповідає за поведінку ворогів. В поточній версії вороги будуть стояти на місці, але вони завжди будуть слідкувати за гравцем. Логіка їх поведінки побудована таким чином, що вони відстежують положення гравця, але не виконують ніяких дій до тих пір, поки гравець не підійде занадто близько. В такому разі буде виконуватись код пострілів. Нижче показаний фрагмент коду скрипта ворога, який слідкує за гравцем:

```
private void FixedUpdate(){
    direction = player_game_object.transform.position - transform.position;
    distance = direction.magnitude;
    if(distance < 5f){
       StartCoroutine(weapon_shoot());
    }
}
```

Цей код працює таким чином, що отримує положення гравця у світі, після чого виконує поворот вектору напрямку ворога в сторону гравця. Для розрахунку відстані виконується приведення вектору напрямку до модулю. Якщо значення відстані менше 5 пунктів, тоді виконується код пострілу, код якого показано нижче:

```
private IEnumerator weapon_shoot(){
```

```
            if (can_shoot) {
                can_shoot = false;
                float rotation_z = Mathf.Atan2(direction.y, direction.x) *
Mathf.Rad2Deg;
                Quaternion rotation = Quaternion.AngleAxis(rotation_z,
Vector3.forward);
                enemy_bullet_game_object = Instantiate(_enemy_bullet_prefab,
transform.position, rotation) as GameObject;
                yield return new WaitForSeconds(1f);
                can_shoot = true;
            }
        }
```

Цей код дещо відрізняється від коду пострілу гравця, оскільки для полегшення гри потрібно, щоби вороги виконували постріл із паузою. Для цього виконуються так звані сопроцеси, або IEnumerator з командою yield. В данному випадку код пострілу виконується із затримкою в 1 секунду. Тобто ворог виконує постріл лише 1 раз в секунду.

У ворога також є і своя куля, код якою у здебільшому схожий до коду кулі гравця. Але вона зчитує зіткнення з об'єктом гравця. Оскільки в нас вже є об'єкт ворога, ми можемо також реалізувати логіку пулі і для гравця, тому додаємо відповідний код у скрипт логіки кулі гравця, звертаючись до параметрів ворога та викликаючи метод нанесення йому ушкоджень.

Останній код реалізація якого буде проілюстрована – код менеджеру інтерфейсу. Цей модуль відповідаю за передачу даних до інтерфейсу гравця. В ході реалізації попередніх скриптів було неодноразово виконано звернення до інтерфейсу. У здебільшому структура цього скрипта проста. Він складається із посилань на елементи інтерфейсу ігрового процесу, а також коду для заміни значень у текстових полях, або властивостей зображень зброї. Нижче приведений код, який спочатку ігрової сесії робить всі зображення зброї прозорими, для створення у гравця враження, що в нього її немає:

```
    void Start(){
        _set_Image_color_transperensy(ref yellow_card, 0f);
        _set_Image_color_transperensy(ref blue_card, 0f);
        _set_Image_color_transperensy(ref red_card, 0f);
        _set_Image_color_transperensy(ref pistol_image, 0f);
        _set_Image_color_transperensy(ref rifle_image, 0f);
```

```
    }
    void _set_Image_color_transperensy(ref Image target_image, float transperensy){
        Color _color;
        _color = target_image.color;
        _color.a = transperensy;
        target_image.color = _color;
    }
```

Було реалізовано основні ігрові функції та модулі проекту. Реалізовані спрайти та префаби ігрових об’єктів, які можна підібрати. Таким чином проект отримав свою кінцеву структуру, яку можна побачити на рисунку 1.25:

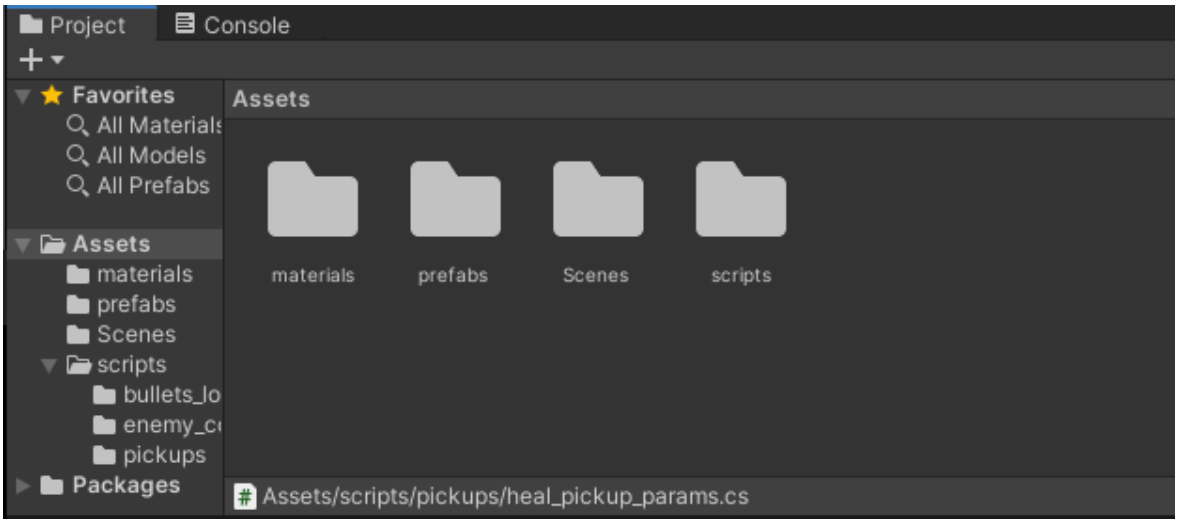


Рисунок 1.25. Кінцева структура проекту

Як можна бачити всі основні файли, з якими проводилась робота розміщені у підкаталогах. Також до складу проекту входять файли ігрового програмного рушія Unity. Всі файли проекту було розподіллено по підкаталогам розділу проекту Assets, розміщення котрих можна побачити на рисунку 1.26.

Поточна реалізація проекту дає змогу зіграти у топ-даун шутер, створити більш складні рівні та ігрові ситуації. Втім така реалізація також дає можливість розширювати ігровий процес. Додавати нових ворогів, зброю, предмети, або ігрові механіки, а також у подальшому додавати нові режими гри, наприклад для гри на одному комп’ютері, або за допомогою мережі Інтернет.

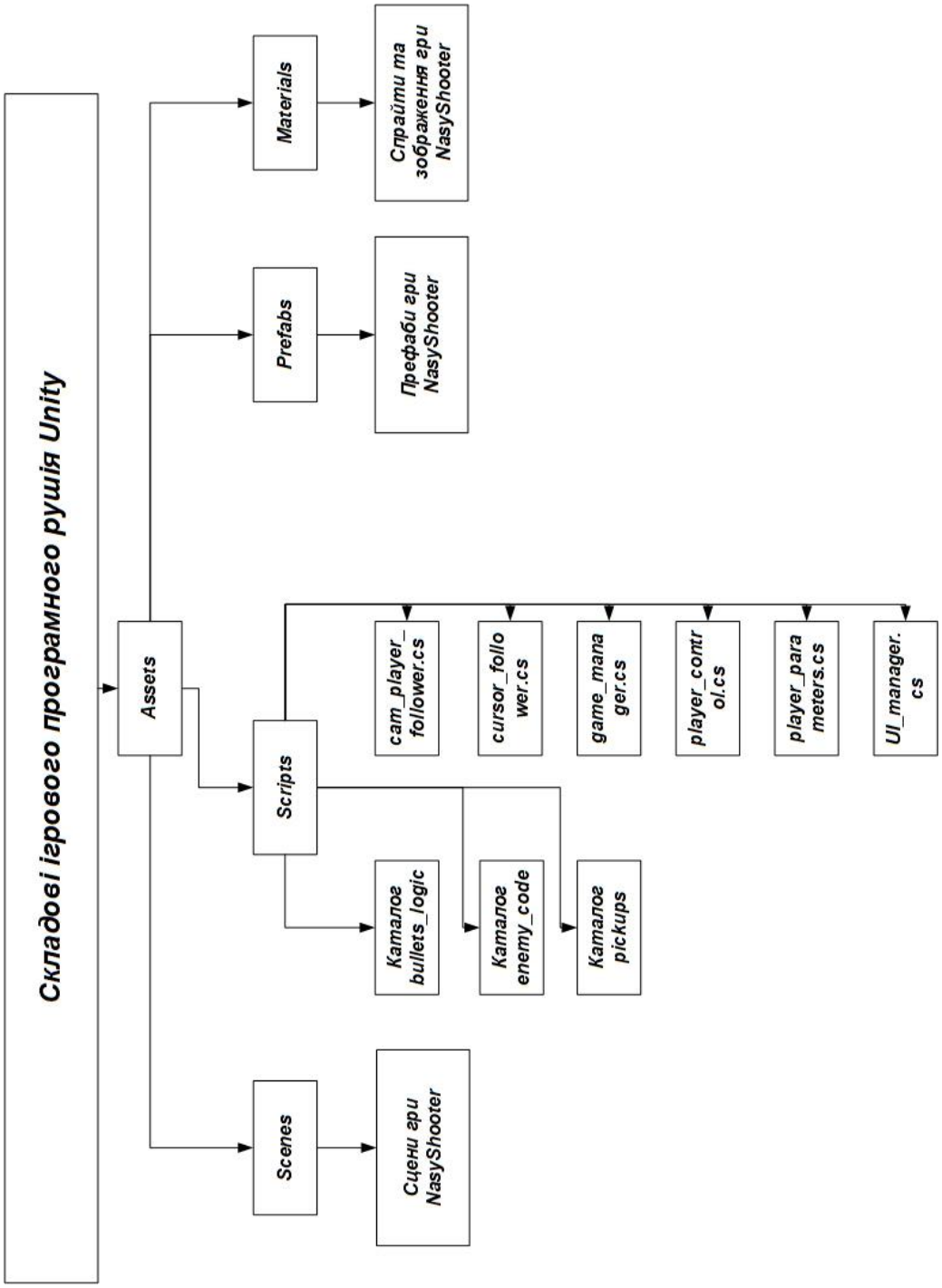


Рисунок 1.26. Розміщення файлів проекту по підкаталогам розділу Assets

1.7 Тестування працездатності ігрових елементів

Використовуючи функціонал ігрового програмного рушія Unity, тестування та відладку гри можна проводити у процесі її створення. Для цього достатньо розмістити ігрові об’єкти на сцені, розмістити елементи інтерфейсу у необхідне положення, налаштувати відповідні параметри та натиснути кнопку «Play». Після цього гра запускається середою розробки, що показано нижче на рисунку 1.27.

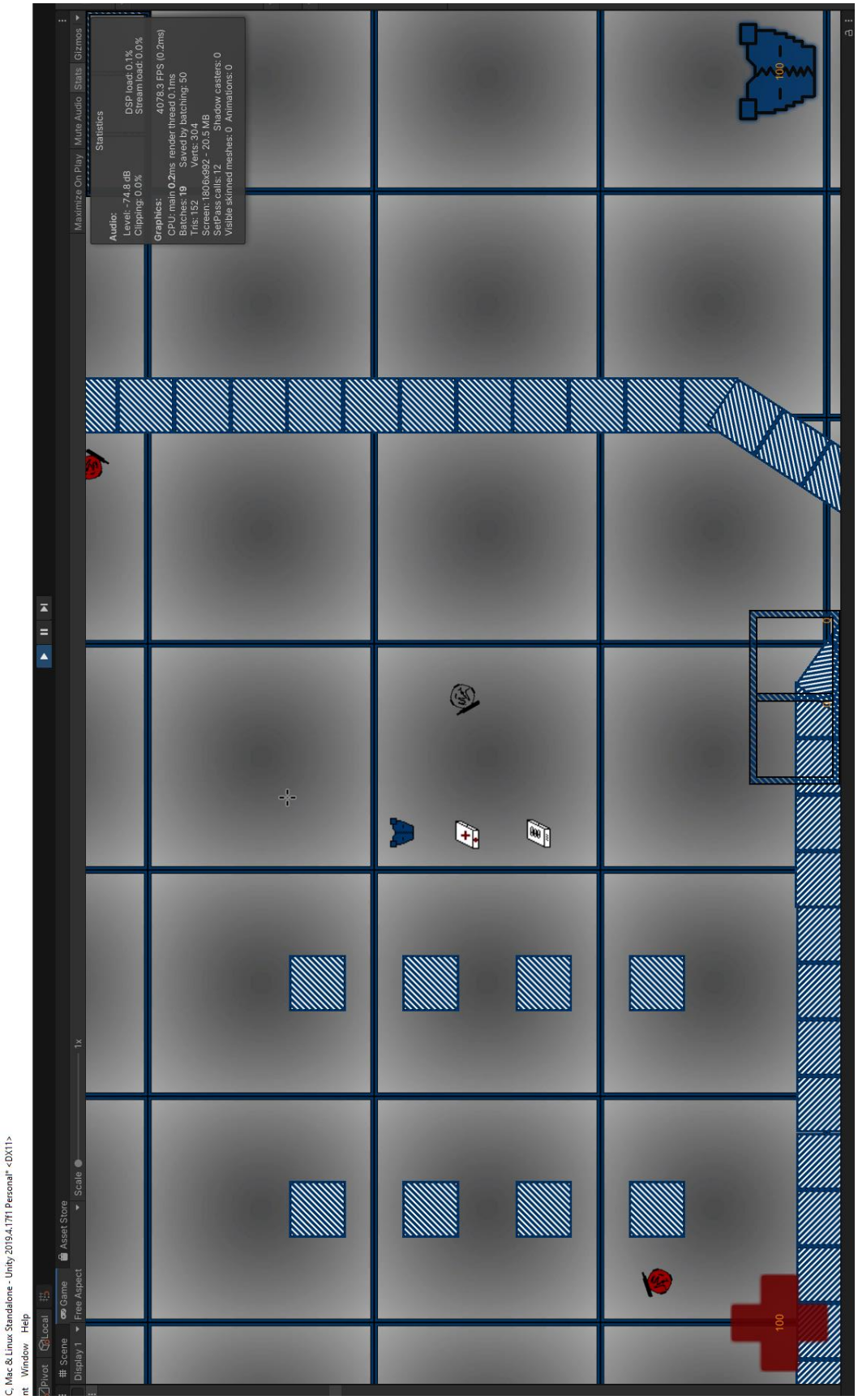


Рисунок 1.27. Відладка гри у середі розробки Unity

Основною метою тестування є тестування працездатності ігрового меню, його елементів, а безпосередньо ігрового процесу. Наскільки коректним є

взаємодія між елементами меню. Чи не виникає помилок під час звернення до якихось кнопок, або у разі натискання кнопок, які не було передбачено налаштуваннями гри. Чи справно працює система управління гравцем, як взаємодіє гравець із оточенням та ворогами на рівні.

В ході тестування було знайдено помилки у роботі логіки куль, які під час пострілу знищувались як тільки створювались. Причиною цієї поведінки ігрової логіки було у тому, що не було реалізовано виключення ініціатора пострілу з переліку цілей для ураження.

Іншою проблемою була надзвичайно велика швидкість пострілів у гравця та ворогів. Якщо для гравця це стало частиною ігрового процесу, то така ситуація у ворогів створювала неможливість пройти гру. Помилка була у відсутності коду, що чекав би деякий час перед наступним пострілом.

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 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

В даному дипломному проекті розроблено та реалізовано 2D-гру в жанрі топ-даун шутер ігровому програмному рушії Unity. Виготовлений програмний продукт дає змогу зіграти в гру, яку можна запустити на операційний системі Microsoft Windows. Розробка програмного продукту виконувалась за допомогою умовно безкоштовного ігрового програмного рушія Unity із використанням редактора коду Microsoft Visual Studio, систем контролю версій та графічного редактора. Розроблене програмне забезпечення було протестовано на помилки в коді та графічному супроводженні. Виявлені недоліки було усунуто за допомогою інструментів програмного рушія Unity.

Оцінка якості програмного продукту з точки зору користувача визначається необхідним на стадії функціонування розміром оперативної пам'яті ЕОТ, витратами машинного часу, пропускною спроможністю каналів передачі даних. 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

Каталог аналогів

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. Аналоги програмного забезпечення

Найменування ПЗ	*Обсяг функції ПП = V_0, ум. машинних команд*
1. ПП СУБД	1300 – 8600
2. ПП введення інформації	1800 – 8800
3. ПП оптимізаційних розрахунків	13000 – 10200

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

Таблиця.2.2. Трудомісткість

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

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 = 244 х 0,8 = 195.20 (люд/годин).

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

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

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

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

Для розрахунку необхідні наступні коефіцієнти:

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

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.3.); Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.4.).

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

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

Для нашого варіанта виділено сірим кольором.

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	Ступінь новизни	Значення $К_н$
А	Принципово нові ПЗ	1,75 – 1,2
Б	ПЗ – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПЗ 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 ПО 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

$$Т_{тз}=Та*L_1*К_н=195,2 *0,12*0,7= 16,40 \text{ (люд/годин)}$$

Трудомісткість розробки технічного проекту

$$Т_{тп}=Та*L_2*К_н =195,2 *0,11*0,7 = 15,04 \text{ (люд/годин)}$$

Трудомісткість розробки робочого проекту

$$Т_{рп}=Та*L_3*К_н*К_т=195,2 *0,61*0,7*0,7= 58,35 \text{ (люд/годин)}$$

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 $N_{рп}$=25 (стр), пояснювальна записка відповідно $N_{пз}$=50 (стр)

Розрахунок зведений у таблицю 2.6

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{Ртз}$=16,40	$T_{кк}=0,7*N_{тз}$= 0,7*2=1,4	$T_{нк}=0,15*N_{тз}$=0,15*2=0,30
2.Розробка ТП	$T_{Ртп}$=15,04	$T_{кк}=0,7*N_{тп}$=0,7*15=10,50	$T_{нк}=0,15*N_{тп}$=0,15*15=2,25
3.Розробка РП	$T_{Ррп}$=58,35	$T_{кк}=0,7*N_{рп}$=0,7*25=17,5	$T_{нк}=0,15*N_{рп}$=0,15*25=3,75
4.Розробка ПЗ	$T_{пз}=1,5**N_{пз}$=1,5*50 =75	$T_{кк}=0,7*N_{тз}$=0,7*50=35	$T_{нк}=0,15*N_{пз}$=0,15*50 =7.5
Усього, в т.ч.:	230,2		
- на розробку	$\Sigma T_р$=152		
- контроль керівника		$\Sigma T_{кк}$= 64,4	
- нормоконтроль			$\Sigma T_{нк}$=13,8

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 ПЗ. Розрахунок основної заробітної плати виконавців приведений у таблиці 2.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2024» встановлено мінімальну заробітну плату.

Таблиця 2.7 Розрахунок основної заробітної плати виконавців

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн.	Розрахунок, грн.
1.Розробка ПП	152	39.26	5958,15
2.Контроль керівника	65	38,50	2502,50
3.Нормоконт-роль	14	38,50	539,00
Усього	-	-	ΣЗо= 8999,15

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

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПЗ

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	60	2.60	160,0
Разом	-	-	-	$В_{м\,і}$ =160,0
Транспортно– заготівельні витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$= 0,1х160.0=16,0
Усього				$В_м=В_{мі}+В_{тр\_з=}$ 176,0

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

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	176,0	$В_м$ (див. табл. 2.7)
2. Основна заробітна плата	8999,15	$З_о$ (див. табл. 2.6)
3.Додаткова заробітна плата	1349,87	$Зд = 0{,}15 \times Зо$=8999,15х0,15
4.Відрахування до єдиного фонду соціального внеску	2276,78	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$=0,22х (8999,15+ 1349,87)
5. Накладні витрати	2699,75	$Внак. = 0{,}3 \times Зо$ =0,3 х 8999,15
6. Повна собівартість	15501,55	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.= 176,0+8999,15+1349,87+2276,78+ 2699,75

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

$$П=(С_п*Р)/100= (15501,55*10)/100 = 1550,15грн \quad (2.4)$$

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

Оптова ціна (кошторисна вартість) визначається по формулі:

$$Ц_о =С_п+П =15501,55+1550,15= 17051,70 \text{ грн} \quad (2.5)$$

Податок на додану вартість визначаємо по наступній формулі:

$$ПДВ = 0.2 * Ц_о = 17051,70 *0,2 = 3410,34грн; \quad (2.6)$$

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

$$Ц_р=Ц_о+ПДВ = 17051,70 + 3410,34 = 20462,04 \text{ грн} \quad (2.7)$$

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Вступ

Охорона праці, як соціальний чинник, відіграє на підприємстві важливу, роль оскільки, якими б важливими не були трудові здобутки, вони не можуть компенсувати людині втраченого здоров'я, а тим більше життя. Те і інше дається лише один раз. Необхідно пам'ятати, що внаслідок нещасних випадків та аварій гинуть на виробництві не просто робітники та службовці, на підготовку яких держава вкладає значні кошти, а перш за все люди – годувальники сімей, батьки та матері дітей. Незадовільний стан охорони праці відображається на економіці держави. Служба охорони праці створюється на підприємствах незалежно від форми власності та видів діяльності для виконання правових, організаційно-технічних, санітарно-гігієнічних, соціально-економічних і лікувально-профілактичних заходів, спрямованих на запобігання нещасних випадків, професійних захворювань і аваріям в процесі праці. Основна мета всіх цих заходів – створити на підприємстві безпечні та здорові умови праці.

У розділі охорона праці дипломного проекту наведені характеристики приміщень, де експлуатуються ВДТ. До розгляду взято робоче місце програміста (оператора ЕОМ).

3.2 Аналіз небезпечних та шкідливих чинників, що впливають на працівника

Оператори ПК і програмісти зіштовхуються із впливом таких фізично небезпечних і шкідливих виробничих факторів, як підвищений рівень шуму, підвищена температура зовнішнього середовища, недостатня освітленість робочої зони, електричний струм та інші. Тому на робочому місці програміста повинні бути створені умови для високопродуктивної праці.

Перетворення і обробка інформації проводиться за допомогою ПК. Робота може кваліфікуватися як робота оператором ЕОМ.

3.3 Розробка заходів з охорони праці

3.3.1 Виробничі приміщення

При плануванні виробничого приміщення врахована санітарна характеристика виробничих процесів, дотримуються норми корисної площі для працюючих, а також нормативи площ для розташування устаткування, що забезпечують безпечну роботу та зручне обслуговування устаткування.

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 ніж 6,0 $м^2$, а об'єм – не менше ніж 20,0$м^3$.

Виробничі приміщення повинні обладнуватися шафами для зберігання документів, полицями, стелажами, тумбами тощо, з урахуванням вимог до плащі приміщення.

У приміщеннях з ВДТ слід щоденно робити вологе прибирання. Приміщення повинні бути оснащені аптечками першої медичної допомоги.

3.3.2 Мікроклімат робочої зони працівників, вентиляція

У виробничих приміщеннях на робочих місцях з ВДТ мають забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря (ДСанПіН 3.3.2.007-98).

Таблиця 3.1. Норми мікроклімату для приміщень з ВДТ ЕОМ та ПЕМ

Пора року	Категорія робіт	Температура повітря, С, не більше	Відносна вологість повітря %	Швидкість руху повітря, м/с
Холодна	Легка-1а	22-24	40-60	0,1
	Легка-1б	21-23	40-60	0,1
Тепла	Легка-1а	23-25	40-60	0,1
	Легка-1б	22-24	40-60	0,1

Рівні позитивних і негативних іонів у повітрі приміщень з ВДТ мають відповідати санітарно-гігієнічним нормам № 2152-80.

Таблиця 3.2. Рівні позитивних і негативних іонів

Рівні	Число іонів в 1 $см^3$ повітря	Число іонів в 1 $см^3$ повітря
	n+	n-
Мінімально необхідні	400	600
Оптимальні	1500-3000	3000-5000
Максимально допустимі	50000	50000

3.3.3 Освітлення робочого місця, шум, вібрація

Штучне освітлення в приміщеннях з робочими місцями, обладнаними ВДТ має здійснюватись системою загального рівномірного освітлення. У виробничих та адміністративних приміщеннях, у разі переважної роботи з документами, допускається застосування системи комбінованого освітлення – крім системи загального освітлення додатково встановлюються світильники місцевого освітлення.

Значення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300-500лк.

Як джерела світла для штучного освітлення мають застосовуватись переважно люмінесцентні лампи типу ЛД. Допускається застосування ламп розжарювання у світильниках місцевого освітлення.

3.3.4 Організація робочого місця користувача ПК

Робочі місця слід так розташовувати відносно світових прорізів, щоб природне світло падало збоку, переважно зліва. При розміщенні робочих столів з ВДТ слід дотримуватися таких відстаней: між бічними поверхнями ВДТ -1,2м; від тильної поверхні одного ВДТ до екрану іншого – 2,5м.

Екран ВДТ має розташовуватися на оптимальній відстані від очей користувача, що становить 600…700 мм, але не ближче ніж за 600 мм з урахуванням розміру літерно-цифрових знаків і символів.

Клавіатуру розташовують на поверхні столу на відстані 100…300 мм від краю, зверненого до працюючого. У конструкції клавіатури має передбачатися опорний пристрій, який дає змогу змінювати кут нахилу поверхні клавіатури у межах $5…15^0$.

При оснащенні робочого місця лазерним принтером параметри лазерного випромінювання повинні відповідати вимогам СанПіН № 5804-91.

ЕОМ ВДТ і ПК , інше устаткування , електропроводи та кабелі за виконанням і ступенем захисту мають відповідати класу зони за НПАОП 40.1-1.01-97, мати апаратуру захисту від струму короткого замикання та інших аварійних режимів. У приміщеннях, де одночасно експлуатується понад п’ять ЕОМ встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення. Не допускається підключати ЕОМ з ВДТ і ПК до звичайної двопровідної електромережі, в тому числі – з використанням перехідних пристроїв

3.3.5 Електробезпека

Це система організаційних і технічних заходів та засобів, що забезпечують захист людей від шкідливої і небезпечної дії електричного струму, електричної дуги, електричного поля і статичної електрики.

Основні технічні засоби і заходи забезпечення електробезпеки при нормальному режимі роботи електроустановок включають:

- ізоляцію струмовідних частин;
- недоступність струмовідних частин;
- блоківки безпеки;
- засоби орієнтації в електроустановках;
- виконання електроустановок, ізольованих від землі;
- захисне розділення електричних мереж;
- компенсацію ємнісних струмів замикання на землю;
- вирівнювання потенціалів.

Із метою підвищення рівня безпеки, залежно від призначення, умов експлуатації і конструкції, в електроустановках застосовується одночасно більшість з перерахованих технічних засобів і заходів.

Особа відповідальна за електрогосподарство призначається з числа працівників, які мають не нижче IV групи з електробезпеки та відповідний стаж роботи для обслуговування електроустановок несе персональну відповідальність за допущення працівника використовувати в роботі електричну енергію

3.4 Пожежна безпека

Пожежна небезпека – це можливість виникнення та розвитку пожежі в будь-якій речовині, процесі, стані.. Коли людина перебуває в зоні впливу пожежі, то вона може потрапити під дію наступних небезпечних та шкідливих факторів: токсичні продукти згорання, вогонь, підвищена температура середовища, дим, недостатність кисню, руйнування будівельних конструкцій, вибухи, паніка.

Усі працівники повинні вміти користуватись наявними вогнегасниками, іншими первинними засобами пожежогасіння, знати місце їх знаходження.

До первинних засобів пожежогасіння відносяться:

- вогнегасники;
- пожежний інвентар (покривала з негорючого теплоізоляційного полотна, грубововняної тканини або повсті;
- ящики з піском;
- бочки з водою, пожежні відра, совкові лопати) та пожежний інструмент (гаки, ломи, сокири тощо).

Пожежні щити (стенди) встановлюються на території об’єкта з розрахунку один щит (стенд) на площу 5000 м^2. Ящики для піску повинні мати місткість 0,5, 1,0 або 3,0 м3 та бути укомплектованими совковою лопатою.

ВИСНОВКИ

Виконання дипломного проектування почалось з пошуку аналогічних ігрових проектів у жанрі топ-даун шутеру. Було виділено декілька ключових проектів, з яких було взято основні елементи ігрових механік, управління та способу ігрового процесу. Вони дали основу для розробки не великого дизайн-документу, в який увійшло основні положення про візуальну складову, гри, управління, ігрові ресурси та інше.

Ігровий проект був поділений на модулі, які були спроектовані з урахуванням ймовірної подальшої необхідності їх використання. Кожний модуль виконував свої функції та мав працювати зі своїм ігровим об’єктом. Такий підхід полегшив розробку, оскільки не потрібно було повертатись назад, тому що весь проект був заздалегідь спроектований.

Розробка виконувалась на програмному рушії Unity на мові програмування C#. Графічна частина створювалась у Adobe Photoshop. Виконання спроектованих модулів не зайняло багато часу, після чого вони були додані до відповідних ігрових об’єктів.

Після відладки, в результаті було отримано 2D-гру в жанрі топ-даун шутер. Вона повністю відповідає основним рисам цього жанру та дає змогу гравцю відчути добре знайомий ігровий процес. Реалізовані основні ігрові механіки, зо дозволяє зайняти гравця.

Водночас створений проект дає основу для додавання нових елементів гри, або навіть режимів. Модульність розробки дозволяє дуже легко додавати нові елементи до ігрового процесу, або структури гри, у випадку, якщо необхідно значно змінити її ієрархію.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ 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 мовою C#

Скрипт player_parameters.cs

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

public class player_parameters : MonoBehaviour
{
    [SerializeField] UI_manager _UI_manager;
    int player_hp;
    int player_ap;
    int pistol_ammo;
    int rifle_ammo;
    bool yellow_card;
    bool blue_card;
    bool red_card;

    void Start()
    {
        player_hp = 100;
        player_ap = 100;
        pistol_ammo = 0;
        rifle_ammo = 0;
        yellow_card = false;
        blue_card = false;
        red_card = false;
        _UI_manager._set_player_hp(player_hp);
        _UI_manager._set_player_ap(player_ap);
        _UI_manager._set_pistol_ammo_text(pistol_ammo);
        _UI_manager._set_rifle_ammo_text(rifle_ammo);
    }

    public void take_damage(int damage_value)
    {
        if(player_ap > 0)
        {
            player_ap -= damage_value;
            if(player_ap < 0)
            {
                player_hp += (player_ap);
                player_ap = 0;
            }
        }
        else
        {
            player_hp -= damage_value;
        }
        _UI_manager._set_player_hp(player_hp);
```

```
    _UI_manager._set_player_ap(player_ap);
    if (player_hp <= 0)
        Destroy(this.gameObject);
 }

public void reduce_pistol_ammo()
 {
    --pistol_ammo;
    _UI_manager._set_pistol_ammo_text(pistol_ammo);
 }

 public void reduce_rifle_ammo()
 {
    --rifle_ammo;
    _UI_manager._set_rifle_ammo_text(rifle_ammo);
 }

 public int get_pistol_ammo()
 {
    return pistol_ammo;
 }

 public int get_rifle_ammo()
 {
    return rifle_ammo;
 }

 private void OnTriggerEnter2D(Collider2D collision)
 {
    ammo_pickup_params _ammo_pickup_params;
    heal_pickup_params _heal_pickup_params;
    armor_pickup_params _armor_pickup_params;
    _ammo_pickup_params = collision.GetComponent<ammo_pickup_params>();
    _heal_pickup_params = collision.GetComponent<heal_pickup_params>();
    _armor_pickup_params = collision.GetComponent<armor_pickup_params>();
    if(_ammo_pickup_params != null)
    {
        _ammo_pickup_params.pick_up_ammo(ref pistol_ammo, ref rifle_ammo);
        Destroy(collision.gameObject);
        _UI_manager._set_pistol_ammo_text(pistol_ammo);
        _UI_manager._set_rifle_ammo_text(rifle_ammo);
    }
    else if(_heal_pickup_params != null)
    {
        _heal_pickup_params.pick_up_heal(ref player_hp);
        Destroy(collision.gameObject);
        _UI_manager._set_player_hp(player_hp);
    }
    else if (_armor_pickup_params != null)
    {
```

```
            _armor_pickup_params.pick_up_heal(ref player_ap);
            Destroy(collision.gameObject);
            _UI_manager._set_player_ap(player_ap);
        }
    }
}
```

Скрипт player_control.cs

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

public class player_control : MonoBehaviour
{
    player_parameters _player_parameters;
    [SerializeField] cursor_follower _cursor_follower;
    [SerializeField] GameObject _bullet_prefab;
    GameObject bullet_game_object;
    float horizontal_input;
    float vertical_input;
    float player_speed = 4f;
    bool weapon_regine = false;
    Rigidbody2D player_rb;
    [SerializeField] Camera player_camera;

    private void Start()
    {
        player_rb = GetComponent<Rigidbody2D>();
        _player_parameters = GetComponent<player_parameters>();
    }

    void Update()
    {
        horizontal_input = Input.GetAxis("Horizontal");
        vertical_input = Input.GetAxis("Vertical");
        Vector2 mouse_position = player_camera.ScreenToWorldPoint(Input.mousePosition) -
transform.position;
        float rotation_z = Mathf.Atan2(mouse_position.y, mouse_position.x) * Mathf.Rad2Deg;
        Quaternion rotation = Quaternion.AngleAxis(rotation_z, Vector3.forward);
        transform.rotation = rotation;
        if (Input.GetKeyUp(KeyCode.Space))
        {
            weapon_regine = !weapon_regine;
        }
        if (!weapon_regine && Input.GetMouseButtonUp(0) &&
(_player_parameters.get_pistol_ammo() > 0))
        {
            weapon_shoot();
            _player_parameters.reduce_pistol_ammo();
        }
    }
```

```
    private void FixedUpdate()
    {
        player_rb.velocity = new Vector2(horizontal_input * player_speed, vertical_input *
player_speed);
        if (weapon_regine && Input.GetMouseButton(0) && (_player_parameters.get_rifle_ammo() >
0))
        {
            weapon_shoot();
            _player_parameters.reduce_rifle_ammo();
        }
    }

    void weapon_shoot()
    {
        Vector3 direction;
        _cursor_follower.take_mouse_position(out direction);
        direction -= transform.position;
        float rotation_z = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
        Quaternion rotation = Quaternion.AngleAxis(rotation_z, Vector3.forward);
        bullet_game_object = Instantiate(_bullet_prefab, transform.position, rotation) as GameObject;
    }
}
```

Скрипт cam_player_follower.cs

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

public class cam_player_follower : MonoBehaviour
{
    [SerializeField] Transform player_transform;
    void Update()
    {
        transform.position = new Vector3(player_transform.position.x, player_transform.position.y, -
10);
    }
}
```

Скрипт cursor_follower.cs

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

public class cursor_follower : MonoBehaviour
{
    Vector3 mouse_position;
    void Update()
    {
        mouse_position = Input.mousePosition;
        mouse_position = Camera.main.ScreenToWorldPoint(mouse_position);
```

```
        transform.position = new Vector3(mouse_position.x, mouse_position.y, 0f);;
    }

    public void take_mouse_position(out Vector3 mouse_pos_var)
    {
        mouse_pos_var = new Vector3(mouse_position.x, mouse_position.y, 0f);
    }
}
```

Скрипт game_manager.cs

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

public class game_manager : MonoBehaviour
{
    void Start()
    {
        Cursor.visible = false;
    }
```

Скрипт UI_manager.cs

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

public class UI_manager : MonoBehaviour
{
    [SerializeField] Image yellow_card;
    [SerializeField] Image blue_card;
    [SerializeField] Image red_card;
    [SerializeField] Image pistol_image;
    [SerializeField] Image rifle_image;
    [SerializeField] Text hp_value_text;
    [SerializeField] Text ap_value_text;
    [SerializeField] Text pistol_ammo_text;
    [SerializeField] Text rifle_ammo_text;

    void Start()
    {
        _set_Image_color_transperensy(ref yellow_card, 0f);
        _set_Image_color_transperensy(ref blue_card, 0f);
        _set_Image_color_transperensy(ref red_card, 0f);
        _set_Image_color_transperensy(ref pistol_image, 0f);
        _set_Image_color_transperensy(ref rifle_image, 0f);
    }

    void _set_Image_color_transperensy(ref Image target_image, float transperensy)
    {
        Color _color;
```

```
        _color = target_image.color;
        _color.a = transperensy;
        target_image.color = _color;
    }

    public void _set_player_hp(int hp_value)
    {
        hp_value_text.text = hp_value.ToString();
    }

    public void _set_player_ap(int ap_value)
    {
        ap_value_text.text = ap_value.ToString();
    }

    public void _set_pistol_ammo_text(int ammo_value)
    {
        pistol_ammo_text.text = ammo_value.ToString();
        if(ammo_value > 0)
        {
            _set_Image_color_transperensy(ref pistol_image, 100f);
        }
    }

    public void _set_rifle_ammo_text(int ammo_value)
    {
        rifle_ammo_text.text = ammo_value.ToString();
        if (ammo_value > 0)
        {
            _set_Image_color_transperensy(ref rifle_image, 100f);
        }
    }
}
```

Скрипт ammo_pickup_params.cs

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

public class ammo_pickup_params : MonoBehaviour
{
    [SerializeField] int pistol_ammo_value = 0;
    [SerializeField] int rifle_ammo_value = 0;
    public void pick_up_ammo(ref int pistol_ammo_param, ref int rifle_ammo_param)
    {
        pistol_ammo_param += pistol_ammo_value;
        rifle_ammo_param += rifle_ammo_value;
    }
}
```

Скрипт armor_pickup_params.cs

```
using System.Collections;
```

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

public class armor_pickup_params : MonoBehaviour
{
    [SerializeField] int armor_value = 0;
    public void pick_up_heal(ref int player_ap_param)
    {
        player_ap_param += armor_value;
    }
}
```

Скрипт heal_pickup_params.cs

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

public class heal_pickup_params : MonoBehaviour
{
    [SerializeField] int heal_value = 0;
    public void pick_up_heal(ref int player_hp_param)
    {
        player_hp_param += heal_value;
    }
}
```

Скрипт enemy_logic.cs

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

public class enemy_logic : MonoBehaviour
{
    [SerializeField] GameObject player_game_object;
    [SerializeField] GameObject _enemy_bullet_prefab;
    GameObject enemy_bullet_game_object;
    Vector3 direction;
    bool can_shoot = true;
    float distance = 0f;

    void Update()
    {
        float rotation_z = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
        Quaternion rotation = Quaternion.AngleAxis(rotation_z, Vector3.forward);
        transform.rotation = rotation;
    }

    private void FixedUpdate()
    {
        direction = player_game_object.transform.position - transform.position;
        distance = direction.magnitude;
        if(distance < 5f)
```

```
        {
            StartCoroutine(weapon_shoot());
        }
    }

    private IEnumerator weapon_shoot()
    {
        if (can_shoot)
        {
            can_shoot = false;
            float rotation_z = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
            Quaternion rotation = Quaternion.AngleAxis(rotation_z, Vector3.forward);
            enemy_bullet_game_object = Instantiate(_enemy_bullet_prefab, transform.position,
rotation) as GameObject;
            yield return new WaitForSeconds(1f);
            can_shoot = true;
        }
    }
}
```

Скрипт enemy_parameters.cs

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

public class enemy_parameters : MonoBehaviour
{
    int enemy_hp = 40;
    public void take_damage(int damage_value)
    {
        enemy_hp -= damage_value;
        if (enemy_hp <= 0)
            Destroy(this.gameObject);
    }
}
```

Скрипт bullet_logic.cs

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

public class bullet_logic : MonoBehaviour
{
    float bullet_speed = 0f;
    void Start()
    {
        bullet_speed = 0.2f;
    }

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

```
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        player_parameters _player_parameters;
        heal_pickup_params _heal_pickup_params;
        armor_pickup_params _armor_pickup_params;
        ammo_pickup_params _ammo_pickup_params;
        bullet_logic _bullet_logic;
        enemy_parameters _enemy_parameters;
        _player_parameters = collision.GetComponent<player_parameters>();
        _heal_pickup_params = collision.GetComponent<heal_pickup_params>();
        _armor_pickup_params = collision.GetComponent<armor_pickup_params>();
        _ammo_pickup_params = collision.GetComponent<ammo_pickup_params>();
        _bullet_logic = collision.GetComponent<bullet_logic>();
        _enemy_parameters = collision.GetComponent<enemy_parameters>();
        if (_player_parameters)
        {
        }
        else if(_enemy_parameters)
        {
            _enemy_parameters.take_damage(30);
            Destroy(this.gameObject);
        }
        else if(_heal_pickup_params || _armor_pickup_params || _ammo_pickup_params)
        {
        }
        else if(_bullet_logic)
        {
        }
        else
        {
            Destroy(this.gameObject);
        }
    }
}
```

Скрипт enemy_bullet_logic.cs

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

public class enemy_bullet_logic : MonoBehaviour
{
    float bullet_speed = 0f;

    void Start()
    {
        bullet_speed = 0.2f;
    }
```

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

    private void OnTriggerEnter2D(Collider2D collision)
    {
        player_parameters _player_parameters;
        heal_pickup_params _heal_pickup_params;
        armor_pickup_params _armor_pickup_params;
        ammo_pickup_params _ammo_pickup_params;
        bullet_logic _bullet_logic;
        enemy_logic _enemy_logic;
        _player_parameters = collision.GetComponent<player_parameters>();
        _heal_pickup_params = collision.GetComponent<heal_pickup_params>();
        _armor_pickup_params = collision.GetComponent<armor_pickup_params>();
        _ammo_pickup_params = collision.GetComponent<ammo_pickup_params>();
        _bullet_logic = collision.GetComponent<bullet_logic>();
        _enemy_logic = collision.GetComponent<enemy_logic>();

        if (_player_parameters)
        {
            _player_parameters.take_damage(15);
            Destroy(this.gameObject);
        }
        else if (_enemy_logic)
        {
        }
        else if (_heal_pickup_params || _armor_pickup_params || _ammo_pickup_params)
        {
        }
        else if (_bullet_logic)
        {
        }
        else
        {
            Destroy(this.gameObject);
        }
    }
}
```

ДОДАТОК Б. Слайди мультимедійної презентації

Розробка 2D-гри у жанрі топ-даун шутер на програмному рушії Unity

ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТА ГРУПИ 4РП-07: ЛАТИШЕВА КИРИЛА ОЛЕКСАНДРОВИЧА

КЕРІВНИК: ДЖАБРАІЛОВ Д.В.

Особливості ігрового жанру 2D топ-даун шутер

Топ-даун шутер – це жанр відеоігор, в яких ігровий світ відображається зверху вниз. У таких іграх гравець керує персонажем, який часто озброєний, і прагне вижити або виконати ціль, стріляючи по ворогах та уникаючи їх атак.

Основними характеристиками жанру є:

- Перспектива польоту;
- Стрілянина та бій;
- Стратегія та тактика;
- Мультиплеєр;
- Прогресія та розвиток;

Особливості програмного рушія Unity та причини його вибору для розробки проекту

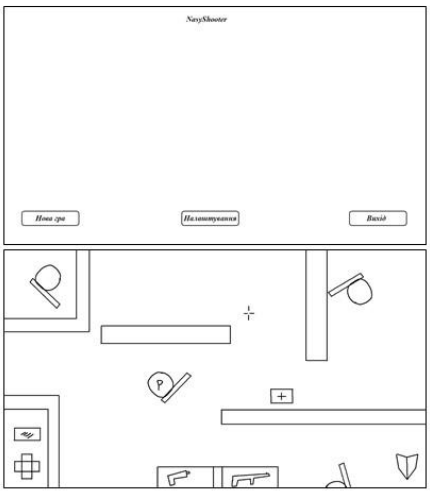
Ігровий двигун Unity є одним із популярніших ігрових двигунів у світі. За його допомогою створюють як мобільні, так і комп'ютерні ігри. Цей двигун досі активно розвивається!

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

Особливості ігрових механік розробляємого проекту

Можна виділити такі особливості розробляємого проекту:

- У гравця є особові параметри персонажу – його ігрові ресурси;
- Гравець може підбирати предмети на рівні для підвишення кількості ігрових ресурсів;
- Управління поглядом персонажу незалежне від руху – слідкування за прицілом;
- Вороги слідкуючі за гравцем;
- У ворогів є свої ігрові ресурси;
- Фізичні кулі.



Огляд основних елементів ігрового процесу

Основні елементи ігрового процесу можна поділити на два етапи:

- Робота у головному меню гри;
- Ігровий процес.

Головне меню дає змогу гравцю:

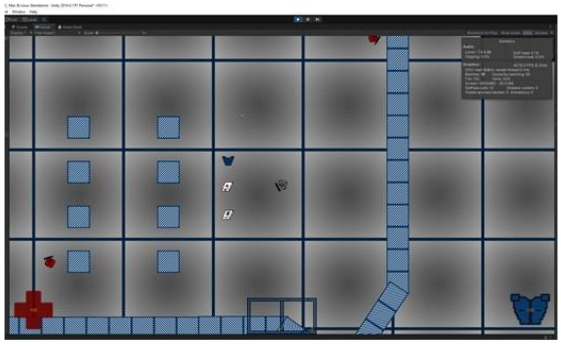
- Почати нову гру;
- Виконати налаштування гри;
- Вийти з гри.

Огляд основних елементів ігрового процесу

Етап ігрового процесу починається після обрання рівня складності гри.

Складається з:

- Переміщення по рівню;
- Збору предметів на рівні;
- Знищення ворогів;
- Пошуку виходу з рівня.

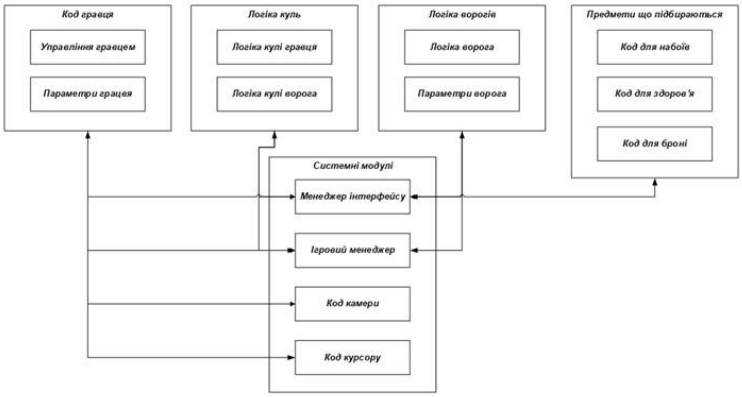


Принцип роботи основних елементів ігрового процесу

Структурно гра має чітку будову і складається з груп модулів:

- Код гравця;
- Логіка куль;
- Логіка ворогів;
- Предмети, що підбираються;
- Системні модулі.

Кожна група займає своє місце у ігровому процесі.



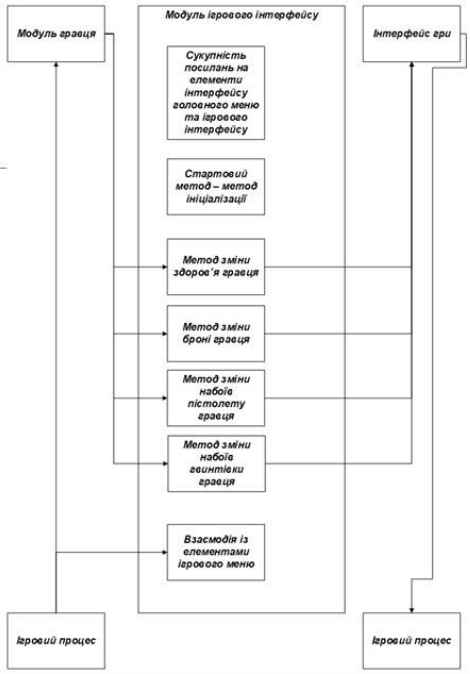
Принцип роботи основних елементів ігрового процесу

Окреме місце займає модуль інтерфейсу, оскільки він контролює передачу інформації від одного модулю гри до іншого.

За своєю суттю, модуль інтерфейсу є прошарком між ігровим процесом та всім іншим кодом проекту.

Модуль обробляє такі події як:

- Зміну здоров'я гравця у інтерфейсі;
- Зміну броні гравця у інтерфейсі;
- Зміну набоїв гравця у інтерфейсі;
- Взаємодію із елементами ігрового меню.



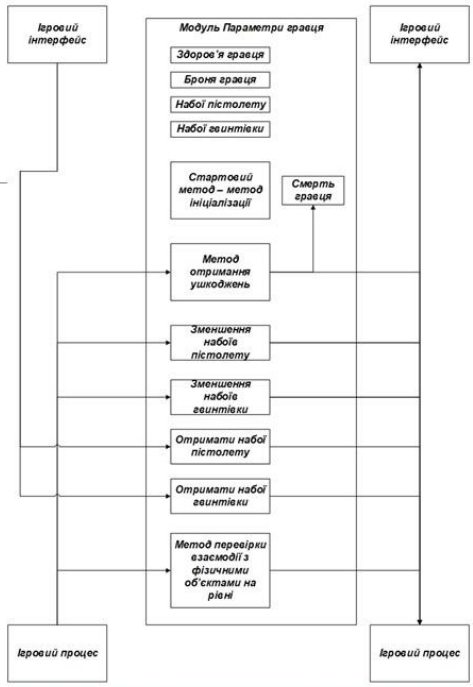
Принцип роботи основних елементів ігрового процесу

Модуль параметрів гравця є ще одним важливим модулем гри.

Він відповідає за можливості у ігровому процесі, від можливості грати, до стрільби та проходу до закритих місць рівнів. Тісно працює з модулем інтерфейсу.

Модуль обробляє такі задачі як:

- Метод отримання ушкоджень;
- Зменшення набоїв різних типів;
- Отримання набоїв різних типів;
- Перевірка взаємодії з іншими об’єктами на рівні.



Етапи реалізації основних елементів ігрового процесу

Реалізація основних елементів гри складалась з:

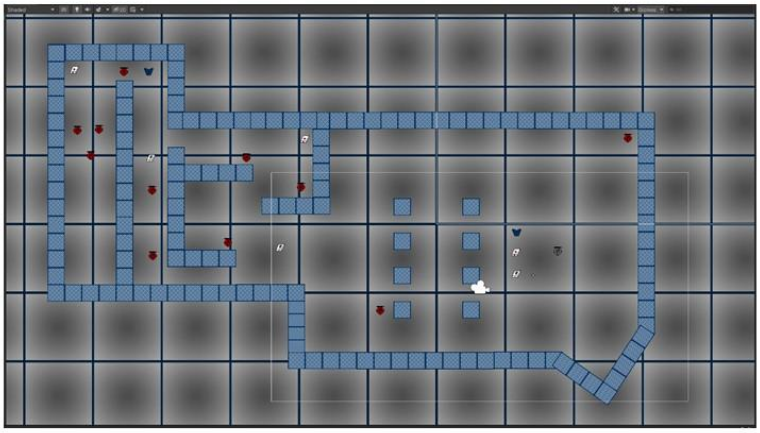
- Реалізації спрайтів для гри;
- Реалізація головного меню гри;
- Реалізація рівня для гри;
- Реалізація елементів управління;
- Реалізація логіки гравця управління;
- Реалізація логіки ворогів;
- Реалізація логіки куль;
- Реалізація взаємодії ігрового процесу з інтерфейсом.

Хід тестування гри

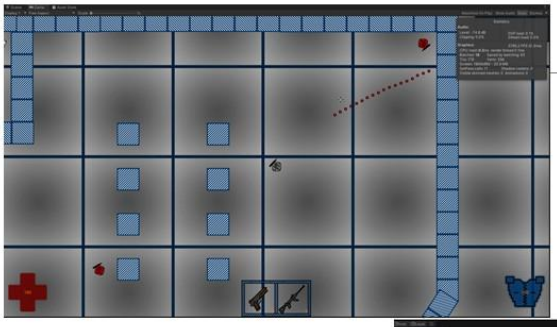
Тестування гри проводилось з допомогою вбудованого у ігровий програмний рушій Unity інструмент.

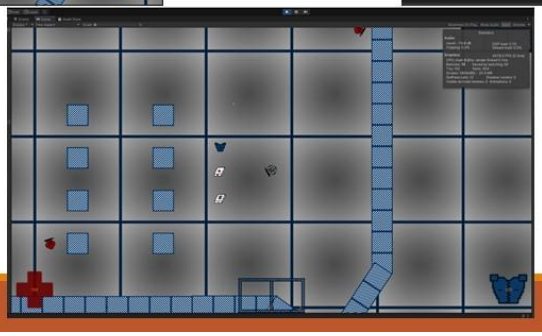
Метою тестування було:

- Визначити справність роботи ігрового меню;
- Визначити справність роботи систем управління гравцем;
- Визначити справність роботи ігрової логіки та взаємодії з іншими елементами рівня та ворогами.



Скріншоти розробленої гри





ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Латишева Кирила Олександровича

(прізвище, ім`я та по батькові)

Спеціальність: 121 "Інженерія програмного забезпечення"

Освітня програма: «Розробка програмного забезпечення»

Тема дипломного проекту: Розробка 2D-гри у жанрі топ-даун шутер на програмному рушії Unity

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка містить 75 сторінок. У пояснювальній записці виконано опис предметної області, способи створення 2D-гри в жанрі топ-даун шутер. Проведено проектування та розробка основних елементів гри. Графічна частина складається з 12 слайдів мультимедійної презентації, які передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.

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

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

Вважаю, що теоретична підготовка дипломника добра і він готовий до захисту дипломного проекту

г) вміння розв'язувати виробничі та конструкторські питання

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

Оцінка розрахункової частини 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

« 10 » 06 2024 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Латишева Кирила Олександровича

(прізвище, ім`я та по батькові)

Спеціальність 121 " Інженерія програмного забезпечення "

Освітня програма «Розробка програмного забезпечення»

Керівник дипломного проекту (роботи) Джабраілов Дмитро Володимирович

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка 2D-гри у жанрі топ-даун шутер на програмному рушії Unity*

Обсяг розрахунково-пояснювальної записки 75 сторінок

Обсяг графічної (презентаційної) частини 12 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

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

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

б) характеристика виконання кожного розділу дипломного проекту (роботи)

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

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

г) перелік позитивних якостей дипломного проекту (роботи)

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

д) основні недоліки дипломного проекту (роботи)

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

Оцінка розрахункової частини Добре

Оцінка графічної частини Добре

Загальна оцінка Добре

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

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

Підпис:

«14» червня 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

ID перевірки:
1016325049

Дата перевірки:
05.06.2024 21:11:57 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
05.06.2024 21:20:53 EEST

ID користувача:
100011688

Назва документа: 4РП-07_Латишев

Кількість сторінок: 42 Кількість слів: 8145 Кількість символів: 55829 Розмір файлу: 2.70 MB ID файлу: 1016123819

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

5.5%
Схожість

Найбільша схожість: 3.44% з Інтернет-джерелом (https://ela.kpi.ua/bitstream/123456789/28218/1/Alpert_bakalavr.docx)

5.5% Джерела з Інтернету | 127 ..... Сторінка 44

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Підозріле форматування | 9 сторінок

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

Ми, що нижче підписалися,

Латишев Кирило Олександрович,
здобувач освіти гр. 4РП-07, та

Джабраілов Дмитро Володимирович,
керівник дипломного проекту,

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

«Розробка 2D-гри у жанрі топ-даун шутер на програмному рушії Unity» (автор роботи – Латишев К.О., керівник роботи – Джабраілов Д.В.)

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

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

Виконавець _____ / Латишев К.О. /

Керівник _____ / Джабраілов Д.В./

«10» 06 2024 р.