

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)
на тему:
«Гра жанру «Survival» створена засобами Unity»

Виконав студент групи КН-19-1
МОСКАЛИК Олександр Сергійович

Керівник роботи:
ЛЕВКІВСЬКИЙ Віталій Леонідович

Рецензент:
КОРОТУН Ольга Володимирівна

Житомир – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Марина ГРАФ

«05» січня 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)

Здобувач вищої освіти: МОСКАЛИК Олександр Сергійович

Керівник роботи: ЛЕВКІВСЬКИЙ Віталій Леонідович

Тема роботи: «Гра жанру «Survival» створена засобами Unity»,
затверджена наказом закладу вищої освіти від «05» січня 2023 р., №10/с

Термін здачі закінченої роботи: «10» червня 2023 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження): об'єктом дослідження є процес проектування та розробки гри жанру «Survival» засобами ігрового рушія Unity 3D .

Предметом дослідження є використання рушія Unity 3D для створення гри жанру «Survival».

Консультанти з кваліфікаційної роботи бакалавра із зазначенням розділів, що їх стосуються

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв

Календарний план

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз вимог до програмного продукту	2 лютого 2023 – 5 березня 2023	виконано
2	Пошук, огляд та аналіз аналогічних розробок	6 березня 2023 – 25 березня 2023	виконано
3	Планування структури програми	26 березня 2023 – 26 квітня 2023	виконано
4	Дизайн документ	27 квітня 2023 – 14 травня 2023	виконано
5	Реалізація алгоритмів роботи гри	15 травня 2023 – 1 червня 2023	виконано
6	Написання програмного коду	1 червня 2023 – 8 червня 2023	виконано
7	Тестування роботи програми	8 червня 2023 – 9 червня 2023	виконано
8	Оформлення пояснювальної записки	9 червня 2023 – 10 червня 2023	виконано
9	Попередній захист	10.06.2023	

Здобувач вищої освіти **Олександр МОСКАЛИК**
Керівник **Віталій ЛЕВКІВСЬКИЙ**

РЕФЕРАТ

Кваліфікаційна робота складається з пояснювальної записки та програмного комплексу – гри від першої особи жанру «Survival». Пояснювальна записка до кваліфікаційної роботи містить 83 сторінки, 47 ілюстрацій та 6 таблиць.

Метою роботи є розробка гри від першої особи жанру «Survival» за допомогою рушія Unity.

Під час розробки проекту розглянуто та проаналізовано відомі ігрові розробки жанру «Survival» та обґрунтовано вибір кожного рішення реалізації. Спроектowana структура програмної частини, наведені сценарії роботи, створено графічний контент для гри, проведено тестування ігрової розробки.

КЛЮЧОВІ СЛОВА: ГРА, ВИЖИВАННЯ, ІГРОВИЙ РУШІЙ, UNITY, ГЕЙМПЛЕЙ.

					КН.КР.Б – 122 – 23 – ПЗ				
Зм.	Арк.	№ докум.	Підпис	Дата					
Розроб.		О.С. МОСКАЛИК			Гра жанру «Survival» створена засобами Unity Пояснювальна записка		Літ.	Арк.	Аркушів
Керівник		В.Л. ЛЕВКІВСЬКИЙ						4	83
Рецензент		О.В. КОРОТУН					Житомирська політехніка, група КН-19-1		
Н. контроль									
Зав. каф.		М.С. ГРАФ							

ABSTRACT

Qualifying graduate work consist of software complex of first person game in Survival genre. The explanatory note to the final qualification work contains 83 pages, 47 illustrations and 6 tables.

The goal of the work is to develop a firs-person Survival game using the Unity engine. During development of the project, other game developments in the “Survival” genre were considered and analyzed, and the choice of each implementation solution was justified. The structure of the game was designed, the work scenarios were given, all graphic content for game was created, and the game development was tested.

KEYWORDS: GAME, SURVIVAL, GAME ENGINE, UNITY, GAMEPLAY.

					КН.КР.Б – 122 – 23 – ПЗ	
Зм.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	9
1.1. Постановка задачі	9
1.2. Аналіз аналогів програмного продукту	10
1.3. Вибір архітектури ігрового додатку	15
1.4. Обґрунтування вибору інструментів та вимоги до апаратного забезпечення.....	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ТА РЕАЛІЗАЦІЯ ДОДАТКУ	24
2.1. Визначення варіантів використання та структури системи	24
2.2. Дизайн документ.....	31
2.3. Реалізація алгоритмів роботи гри	41
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ	51
3.1. Порядок встановлення та налаштування системи	51
3.2. Інтерфейс та порядок роботи з програмним додатком.....	51
3.3. Тестування роботи програмного комплексу	57
ВИСНОВКИ.....	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	61

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

FPS(Frames per second) – кадрів на секунду.

HUD(Heads Up Display) – елементи швидкого доступу інтерфейсу користувача.

UE – Unreal Engine.

USP(Unique selling points) – унікальні точки продажу.

UI(User Interface) – інтерфейс користувача.

ОЗП – Оперативно запам'ятовуючий пристрій.

					КН.КР.Б – 122 – 23 – ПЗ	
Зм.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

У сучасному світі відеоігри стали невід'ємною частиною майже кожної людини. Вони не тільки розважають нас, але й стають засобом втілення фантазій, випробовування нашої витривалості та розвитку наших навичок. Особливо популярним серед ігор набув жанр **виживання** (англійською Survival). Такі відеоігри надають гравцям захоплюючий та неповторний досвід вирішення задач у тяжких умовах та небезпечному середовищі.

Ігри в жанрі виживання вимагають від гравця витримки, стратегічного мислення та вміння приймати рішення в умовах обмежених ресурсів та небезпеки. У таких іграх гравець зазнає випробовування, де його навички у виживанні, збиранні ресурсів, створенні предметів, побудові споруд, взаємодії з вигаданим світом стають вирішальними для його успіху.

Ігри у такому жанрі можуть мати як відкритий світ, де гравець вільно рухається та досліджує середовище, так і більш лінійний сюжетний режим, де гравець проходить певні завдання та випробовування, щоб продовжити історію.

Жанр виживання може бути представлений у вигляді суворих та реалістичних симуляторів, де гравець змушений дотримуватися реальних фізичних законів та потреб героя, а бо ж у вигляді більше аркадних ігор, де елементи виживання поєднуються з іншими жанрами та геймплеєм від третьої особи[1].

Однак, серед великої кількості ігор у цьому жанрі, наразі недостатньо уваги приділяється розробці однокористувацьких ігор у жанрі виживання. Більшість компаній зосереджують сили на створенні мультиплеєрних режимах, де гравці можуть взаємодіяти між собою, співпрацювати або змагатись. Тим не менш, інсує значний шар гравців, які віддають перевагу у виживанні наодинці без інших гравців.

Метою кваліфікаційної роботи є створення захоплюючої однокористувацької гри жанру виживання, яка надасть гравцям унікальну можливість насолоджуватись досвідом напруженості та виживання у віртуальному середовищі.

					КН.КР.Б – 122 – 23 – ПЗ	7
Зм.	Арк.	№ докум.	Підпис	Дата		

Об’єктом дослідження є процес проектування та розробки гри жанру «Survival» засобами ігрового рушія Unity 3D .

Предметом дослідження є використання рушія Unity 3D для створення гри жанру «Survival».

Методи дослідження: в роботі було використано методи об’єктно-орієнтованого проектування, програмування та дизайну.

В рамках розробки кваліфікаційної роботи були опубліковані наукові праці:

- Москалик О.С., Левківський В.Л. Жанр «виживання» у іграх на платформі Unity. Тези XIII Міжнародної науково-технічної конференції «Інформаційно-комп’ютерні технології – 2023 (ІКТ-2023)», м. Житомир, 30 - 31 березня 2023 р. – Житомир: Житомирська політехніка, 2023. С. 76-77.

- Москалик О.С., Панаріна І. В., Левченко А. Ю. Психологія людини. Чому люди грають у хоррор відеоігри. Тези доповідей IV Всеукраїнської науково-практичної інтернет-конференції «Інформаційно-комп’ютерні технології: стан, досягнення та перспективи розвитку», м. Житомир, 25 – 26 листопада 2021 р. Житомир: Житомирська політехніка, 2021. С. 96-97.

					КН.КР.Б – 122 – 23 – ПЗ	8
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1. Постановка задачі

Основною метою розробки однокористувацької гри у жанрі «Survival» є надання гравцеві позитивних емоцій та забезпечення гарного проведення часу під час користування додатком. Гра може як розслабляти, так і кинути виклик гравцеві, надаючи унікальний досвід виживання у віртуальному просторі.

У роботі необхідно розробити додаток у вигляді гри, в який потрібно виживати у віртуальному середовищі та зробити цей процес достатньо цікавим та захоплюючим. Для реалізації поставленого завдання, основними етапами є:

1. Розробити дизайн користувацького інтерфейсу.

- а. Створити загальну концепцію дизайну користувацького інтерфейсу.
- б. За допомогою графічного редактора створити основні елементи UI:
 - i. Головне меню.
 - ii. Меню налаштувань.
 - iii. Меню паузи.
 - iv. Інвентар гравця.
 - v. HUD.

2. Проектування та розробка основних ігрових модулів

- а. Модуль керування персонажем.
- б. Модуль потреб персонажа.
 - i. Голод персонажа.
 - ii. Спрага персонажа.
- с. Модуль інвентаря.
 - i. Підбирання предметів.
 - ii. Обмеження слотів інвентаря.
 - iii. Алгоритм для полегшення процесу створення значків предметів.

					КН.КР.Б – 122 – 23 – ПЗ	9
Зм.	Арк.	№ докум.	Підпис	Дата		

- d. Модуль завдань для гравця.
 - i. Створення кастомного вікна для додавання нових завдань у Unity Editor.
- e. Модуль керування звуками у грі.
- f. Модуль керування добою у грі.
- g. Модуль керування подіями.
- h. Модуль взаємодії персонажа із навколишнім середовищем.

Результатом реалізації поставленого завдання є додаток у вигляді гри-виживання у віртуальному просторі.

1.2. Аналіз аналогів програмного продукту

На сьогоднішній день існує багато ігор у жанрі виживання для різних платформ, всі вони користуються великою популярністю, на рисунку 1.1 можна побачити, що жанр «Survival» у найпопулярнішому магазині «Steam» є великою часткою серед інших жанрів, посідаючи третє місце серед кількості ігор у цьому жанрі. Буде розглянуто декілька з відомих представників цього жанру на які буде опиратись та орієнтуватись розробка, а в кінці будуть порівняні між собою[2].

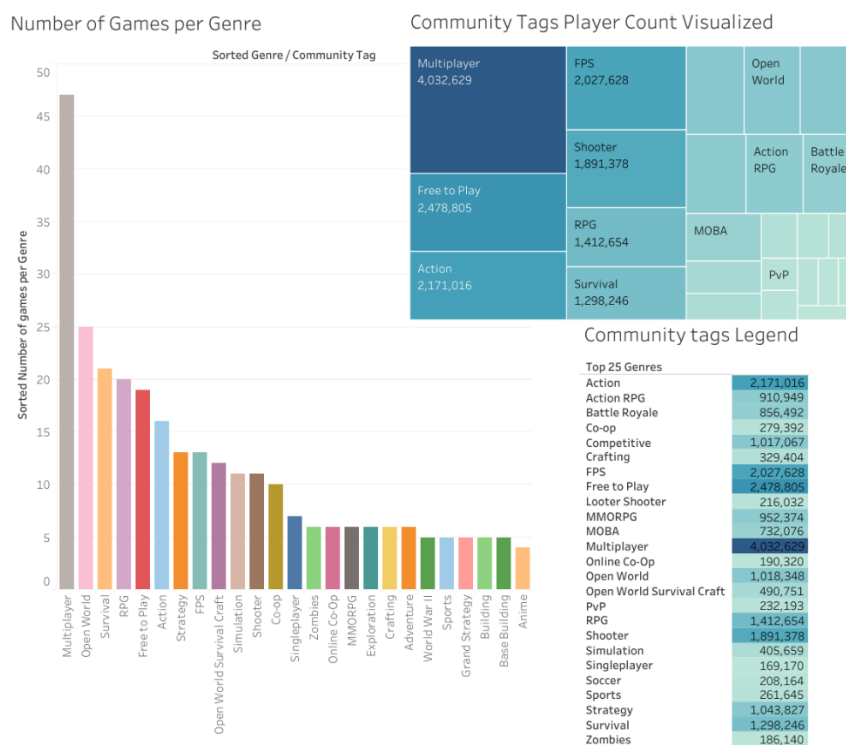


Рисунок 1.1 – Статистика кількості ігор жанрів у магазині «Steam»

«**This War of Mine**» – відеогра у жанрі виживання, розроблена польською компанією 11 bit studios та випущена у 2014 році, швидко здобула популярність завдяки своїй концепції та атмосфері. Основна ідея гри полягає у тому, що гравець керує групою цивільних які застрягли у війні. Завдання гравця – забезпечити їм виживання протягом непередбачуваного періоду, доки не скінчиться війна. Із особливостей можна відмітити:

- Моральні вибори, які гра ставить перед гравцем, вимагаючи прийняти рішення, які впливають на психологічний та моральний стан персонажів підконтрольних гравцеві;
- Гра не має строгої сюжетної структури. Замість цього вона пропонує нелінійний геймплей, де гравець формує власну історію крізь дії та рішення;
- Гра відрізняється деталізованими ручними текстурами(рисунок 1.2), що надає їй особливу атмосферу, а музичне супроводження та звукові ефекти допомагають підсилити настрій гри[3].

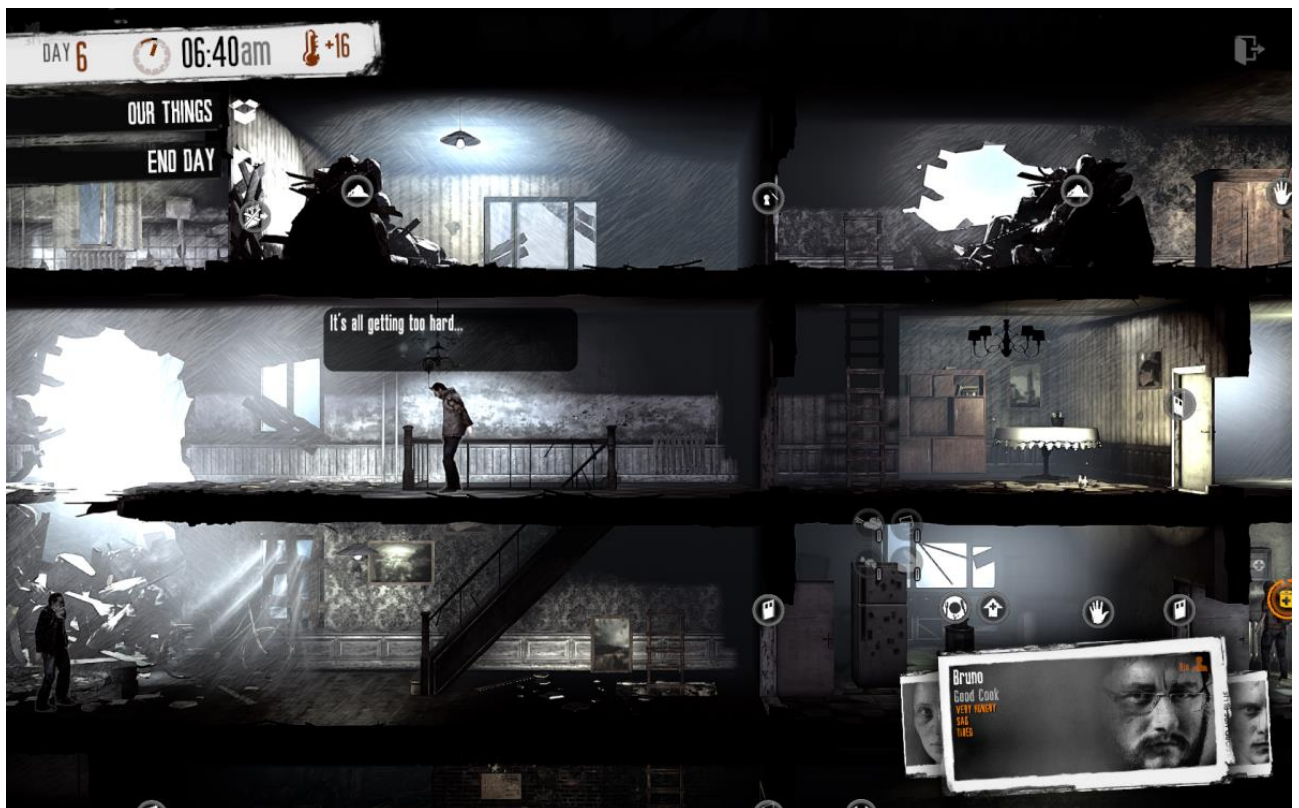


Рисунок 1.2 – Кадр з геймплею гри «This War of Mine»

«Don't Starve» – відеогра жанру виживання та пригод, розроблена канадською компанією Klei Entertainment. Гра була випущена у 2013 році і з того часу стала популярною серед шанувальників інді-ігор у жанрі виживання.

У «Don't Starve» гравець керує персонажем на ім'я Уілсон, який потрапив у дивний світ, наповнений небезпеками і тасмніцями. Основним завданням гравця є задоволення потреб персонажа, таких як, голод, спрага, здоров'я та глузд і знайти вихід із дивного світу. Гравець повинен забезпечити його їжею, будувати притулок, доглядати за здоров'ям і психічним станом у небезпечному середовищі, яке наповнене небезпеками, такими як: хижаки, монстри і ворожі створіння. Особливостями цієї гри є:

- Арт-стиль і атмосфера. Гра має темний, темно-комічний візуальний стиль(рисунок 1.3), що надає їй унікального вигляду.
- Випадково згенерований світ, що означає, що кожна нова гра буде унікальною з новими територіями і ресурсами.
- Елемент-дослідницької роботи, гравець може вивчати технології, будувати нові предмети, експериментувати з ресурсами і розблоковувати нові рецепти[4].



Рисунок 1.3 – Кадр з геймплею «Don't Starve»

					КН.КР.Б – 122 – 23 – ПЗ	12
Зм.	Арк.	№ докум.	Підпис	Дата		

Гра отримала позитивні відгуки від критиків і гравців завдяки своїй унікальній атмосфері, глибині геймплею та високій реіграбельності. Компанія випустила також кілька доповнень та спін-офф Don't Starve Together у якому став доступний мультиплеєр та розширення гри узагалом[5].

Остання гра є надпопулярною у своєму жанрі. «DayZ» – це гра у жанрі виживання з відкритим світом, розроблена бохемською студією Bohemia Interactive. Гра базується на модифікації з популярної відеогри «Arma 2» і була випущена як самостійна гра в грудні 2013 року. Основною концепцією «DayZ» є виживання в постапокаліптичному світі, зараженому зомбі-вірусом. «DayZ» покладається на реалістичність, виживання і соціальну взаємодію гравців, вона надає унікальний досвід постапокаліптичного виживання, де кожне рішення може вплинути на долю персонажа.

Особливостями цієї гри є:

- Гравці повинні стежити за своїми потребами, такими як: голод, спрага, здоров'я, кількість крові у організмі, температура тіла та інші не менш важливі потреби.
- Сотні кілометрів віртуального простору, велетенська мапа яку гравці можуть довго досліджувати.
- Мультиплеєрний режим є основним і єдиним режимом де гравці можуть взаємодіяти між собою, об'єднуватись у клани та битись один з одним.
- Реалістичний геймплей у якому є реалістична фізика, моделювання пошкоджень та зброю, яка може заклинути у самий не підходящий момент, якщо за нею не доглядати, що додає грі глибину та складність. Гравці можуть використовувати різноманітну зброю, транспортні засоби і інші предмети, а також будувати бази для захисту[6].

					КН.КР.Б – 122 – 23 – ПЗ	13
Зм.	Арк.	№ докум.	Підпис	Дата		



Рисунок 1.4 – Кадр геймплею з «DayZ»

Після того як було оглянуто декілька аналогів, тепер розглянемо їх у вигляді таблиці(таблиця 1.1).

Таблиця 1.1.

Порівняння аналогів за функціональними можливостями

Назва гри	Графіка	Режим	Наявність сюжету	Мапа
This War of Mine	3D з деталізованими ручними текстурами	Тільки однокористувацький	-	Випадкові мапи, які випадють гравцеві кожного проходження
Don't Starve	2D, особливий арт-стиль	Однокористувацький(та мультиплеєр у спін-офі)	Є сюжетний режим під назвою «Пригоди»	Процедурно згенерована
DayZ	3D, реалістична	Тільки мультиплеєрний	-	На разі є дві мапи: «Чорнорусія» та «Лівонія»

Таблиця 1.2.

Мінімальні системні вимоги до розглянутих ігор

Назва гри	Процесор	ОЗУ	Відеокарта	Місце на диску
This War of Mine	2.4 GHz Dual Core	2 гб	GTX 260, 1024 мегабайт відеопам'яті	5 гігабайт
Don't Starve	1.7+ Ghz або ліпше	1+ гб	Radeon HD5450 або ліпше; 256 мегабайт відеопам'яті або ліпше	500 мб
DayZ	Intel Core i5-4430	8 гб	NVIDIA GeForce GTX 760 або AMD R9 270X - серії	25 гігабайт

Переглянувши та проаналізувавши аналоги відеоігор у жанрі виживання, можна сказати, що кожен з них відрізняється від своїх конкурентів, у кожного є свої особливості, як і недоліки, що створює певну різноманітність та свободу вибору для користувача.

Враховуючи дані противоріччя виникає потреба спроектувати та розробити додаток поєднавши особливості проаналізованих аналогів та позбутись їх недоліків.

Таким чином, основними рисами даного продукту можуть бути: сучасний фізичний рушій, наявність інтуїтивно-зрозумілого інтерфейсу, зручне управління персонажем, взаємодія із навколишнім середовищем та основні механіки які визначають жанр гри.

1.3. Вибір архітектури ігрового додатку

Для вибору відповідної архітектури, яка б найкраще підходила для розробки, розглянемо декілька з популярних паттернів та підходів:

					КН.КР.Б – 122 – 23 – ПЗ	15
Зм.	Арк.	№ докум.	Підпис	Дата		

- MVC (Model-View-Controller) – цей паттерн розділяє гру на три компоненти: модель, представлення та контроллер. Модель представляє дані та бізнес-логіку, представлення відображає інтерфес користувача, а контроллер відповідає за обробку вводу від користувача та зміну моделі. Із переваг можна відмітити:

- Полегшення розробки та подальшої підтримки гри завдяки розділенню відповідальностей;
- Можливість змінювати окремі компоненти не впливаючи на інші;

Недоліки:

- Довге налаштування паттерну;
- Шанс створення zombie – екземплярів, які впливають не лише на продуктивність, а можуть бути ще й причиною багів;
- При великій кількості об'єктів виникає плутанина між їхніми взаємодіями[7];

- ECS (Entity-Component-System) – паттерн заснований на концепції розділення сутностей, компонентів та систем. Кожна сутність складається з компонентів, які визначають її властивість на поведінку. Системи оброблюють компоненти та виконують необхідні операції. Перевагами такого підходу можна назвати:

- Високу продуктивність та ефективне використання ресурсів;
- Легке створення нових типів сутностей і компонентів;
- Ідеально підходить для гри з великою кількістю об'єктів;

Із недоліків відзначимо:

- Вимагає більше часу для задіяння паттерну;
- Потребує грамотного керування життєвим циклом сутностей та компонентів;

Також варто відмітити, що Unity має власний фреймворк Unity ECS який полегшує розробку гри з використанням архітектури ECS, але використання такого фреймворку ліпше розглядати у проєкті зі

					КН.КР.Б – 122 – 23 – ПЗ	16
Зм.	Арк.	№ докум.	Підпис	Дата		

складною логікою та великою кількістю об'єктів таких як у іграх-стратегіях[8].

- MVP (Model-View-Presenter) – це архітектурний паттерн, де модель представляє дані і логіку, представлення відповідає за відображення інтерфейсу користувача, а презентер обробляє події від користувача та спілкується з моделлю та представленням. Серед переваг MVP є:

- Розділення обов'язків;
- Легко тестувати, оскільки логіка зосереджена в презентері, його можна легко тестувати без залежності від інтерфейсу користувача;
- Модифікування додатку є простою, без впливу на інші компоненти;

З недоліків є:

- Складність реалізації. Іноді реалізація цього паттерну може бути складною і призводити до написання зайвого коду у великих проектах;
- Багато зв'язків, оскільки презентер повинен взаємодіяти з моделлю і представленням, це також збільшить кількість залежностей між компонентами;

- MVVM (Model-View-ViewModel) – архітектурний паттерн, в якому модель відповідає за дані та бізнес-логіку, представлення відображає графічний інтерфейс, а модель-представлення забезпечує зв'язок між моделлю та представленням. Перевагами цього паттерну є:

- Як і у MVC та MVP, MVVM розділяє логіку додатку, що спрощує розробку та підтримку кода;
- Представлення-модель може використовувати прив'язки даних для автоматичного оновлення представлення при зміні даних;
- Зручність тестування, оскільки логіка розташована у представленні-моделі, її можна зручно та легко тестувати[9].

					КН.КР.Б – 122 – 23 – ПЗ	17
Зм.	Арк.	№ докум.	Підпис	Дата		

Розглянувши декілька архітектурних паттернів можна зробити висновок, що найбільш відповідним паттерном є ECS, оскільки він має високу продуктивність та ефективно використовує ресурси на відміну від сімейства паттернів MVC, також з ним легко створювати нові об'єкти. Більше того, Unity орієнтована на використання дизайнів паттерну компонентного підходу, що підходить для використання вкупі з ECS.

Отже, можна зробити висновок, що даний архітектурний шаблон спростить написання коду та підвищить продуктивність додатку, що дозволяє реалізувати всі поставлені завдання.

Також у процесі проектування сутностей було прийнято рішення використовувати патерни проектування оскільки шаблони дозволяють стандартизувати код та використовувати готові рішення що в свою чергу спростить та пришвидшить розробку додатку. Обрані патерни проектування:

Спостерігач — це поведінковий патерн проектування, який створює механізм підписки, що дає змогу одним об'єктам стежити й реагувати на події, які відбуваються в інших об'єктах[10].

У Unity, такий патерн реалізовується доволі легко, за допомогою подій, у нашому випадку його було реалізовано за допомогою делегатів для передачі змінної та статичної події для зручності виклику.

Стан — це поведінковий патерн проектування, що дає змогу об'єктам змінювати поведінку в залежності від їхнього стану. Ззовні створюється враження, ніби змінився клас об'єкта. Основна ідея в тому, що програма може знаходитися в одному з кількох станів, які увесь час змінюють один одного. Набір цих станів, а також переходів між ними, визначений наперед та скінченний. Перебуваючи в різних станах, програма може по-різному реагувати на одні і ті самі події, що відбуваються з нею. Патерн Стан пропонує створити окремі класи для кожного стану, в якому може перебувати контекстний об'єкт, а потім винести туди поведінки, що відповідають цим станам. Замість того, щоб зберігати код всіх станів, початковий об'єкт, який зветься контекстом, міститиме

посилання на один з об'єктів-станів і делегуватиме йому роботу в залежності від стану[11].

Одинак — це породжувальний патерн проектування, який гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього. Всі реалізації Одинака зводяться до того, аби приховати типовий конструктор та створити публічний статичний метод, який і контролюватиме життєвий цикл об'єкта-одинака.

Враховучи вище написане, патерни дозволять оптимізувати роботу над проектом, таким чином дозволяючи сконцентрувати увагу на інших важливих частинах додатку.

1.4. Обґрунтування вибору інструментів та вимоги до апаратного забезпечення

Оскільки будемо реалізовувати додаток у готовому рушії тому почнемо з нього. На вибір є декілька популярних безкоштовних рушіїв: Unity, Unreal Engine та Godot. Але оскільки потрібно реалізувати кроссплатформену гру з використанням фізики та невибагливою у графіці, в рушії в який буде гнучким, інтерфейс якого буде інтуїтивно зрозумілим, а сам рушій легко засвоїти, під цей опис ідеально підходить Unity. Вибір даного рушія зумовлено декількома перевагами.

Перша перевага – інтуїтивно зрозумілий та дружелюбний інтерфейс(рисунок 1.4), він є легким у використанні, що дозволяє швидко орієнтуватись та приступити до розробки ігор навіть новачкам, довго в ньому не розбираючись.

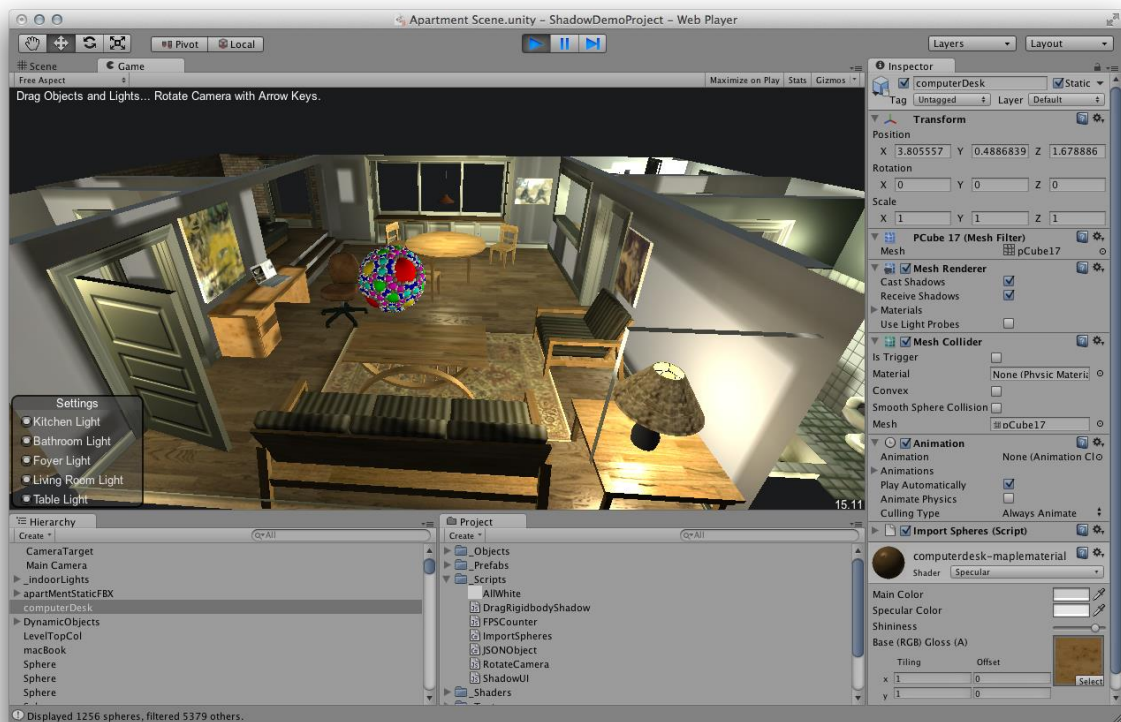


Рисунок 1.5 – Інтерфейс Unity

Друга перевага – скрипти пишуться мовою C#. Дана мова є вискорівневою, що дозволяє легко її вивчити, засвоїти і приступити до розробки гри. Також, у Unity є ще візуальна мова програмування. C# є важливою перевагою, оскільки дозволяє швидше приступити до розробки, на відмінну від Unreal Engine, де скрипти пишуться C++ мовою, що є складнішою, так, у UE є blue prints – що є візуальним програмуванням, але так чи інакше скрипти є набагато гнучкішими та ліпшими у створенні ігрової логіки. Також у Unity реалізовано багато функцій які б у UE довелося писати самостійно.

Ще однією перевагою є кроссплатформеність додатків скомпільованих на Unity, це дозволить випускати додатки на більшу кількість платформ з мінімальними змінами[13].

Іншою перевагою є документація рушія. Документація Unity є також дружельною та зрозумілою. Документація охоплює всі аспекти розробки. Розробки можуть знайти відповіді на свої питання щодо графіки, шейдерів, звуку, фізики, мережевої взаємодії та багато іншого.

В документації також практичного до кожного методу/класу є велика кількість прикладу коду, що допомагає розробникам зрозуміти, як використовувати різні функції і компоненти рушія. Це особливо корисно для початківців, які можуть вивчати шаблони і «best practices», переглядаючи існуючі приклади[14].

І останньою перевагою є спільнота рушія. Unity має велику та активну спільноту розробників, яка допомагає одне одному та ділиться знаннями. На форумах Unity розробники можуть знайти відповіді на свої запитання від розробників самого рушія, або ж інших учасників, обговорювати проблеми та отримувати поради від інших досвідчених розробників.

Наступний інструмент ми будемо розглядати 3D-редактор. Із запропонованих програм є 3Ds Max, та Maya які є платними і Blender який є безкоштовним та з відкритим кодом, тому поліпшувати програму може будь-хто. Blender – це повнофункціональний пакет для 3D-моделювання, анімації, текстурування, рендерингу, композитингу і багато іншого, це програма яка може задовольнити як початківця так і професіоналів[15]. Також перевагою Blender є у його кроссплатформенності і гнучкості, можна налаштувати інтерфейс як заманеться і навіть розширити його функціонал за допомогою інших плагінів і скриптів python. Також, розробники постійно вдосконалюють програму, випускаючи нові версії зі значними оновленнями.

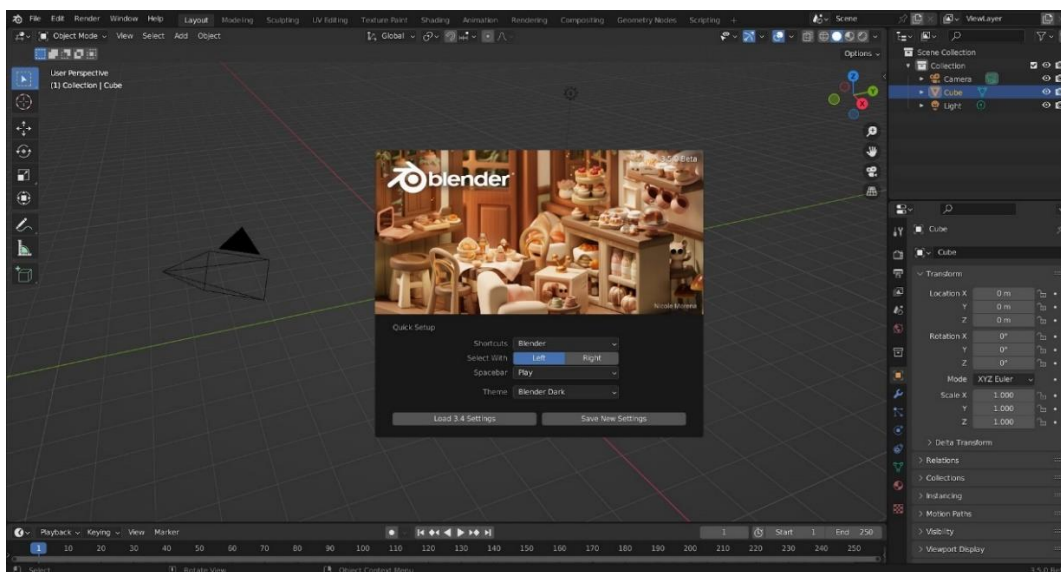


Рисунок 1.6 – Інтерфейс програми Blender

Звукозапис. Із доступних програм є Audacity і вона є найліпшою по декільком причинам, так само як і Blender вона є безкоштовною з відкритим кодом. Вона є гнучкою, оскільки її можна модифікувати згідно своїх потреб. Audacity надає можливість запису аудіо з різних джерел, атких як мікрофон, лінійний вхід, або навіть потокове аудіо з комп'ютера. У програмі можна дуже легко редагувати, вирізати, копіювати, вставляти і багато іншого завдяки своєму дружньому та інтуїтивно зрозумілому інтерфейсу(рисунок 1.6). Також є важливим, що Audacity має широкий набір вбудованих ефектів, таких як еквалайзер, ефекти зміни швидкості, темпу, тону, видалення шуму та багато чого іншого. Як було згадано раніше, Audacity має інтуїтивний і простий у використанні інтерфейс, що робить його доступним для новачків у сфері аудіообробки. Іншою особливістю є те, що програма підтримує широкий спектр аудіоформатів, включаючи WAV, MP3, AIFF, FLAC та багато інших, можна імпортувати і експортувати факли у різних форматах залежно від потреб. Audacity також підтримує багатоканальність[16].

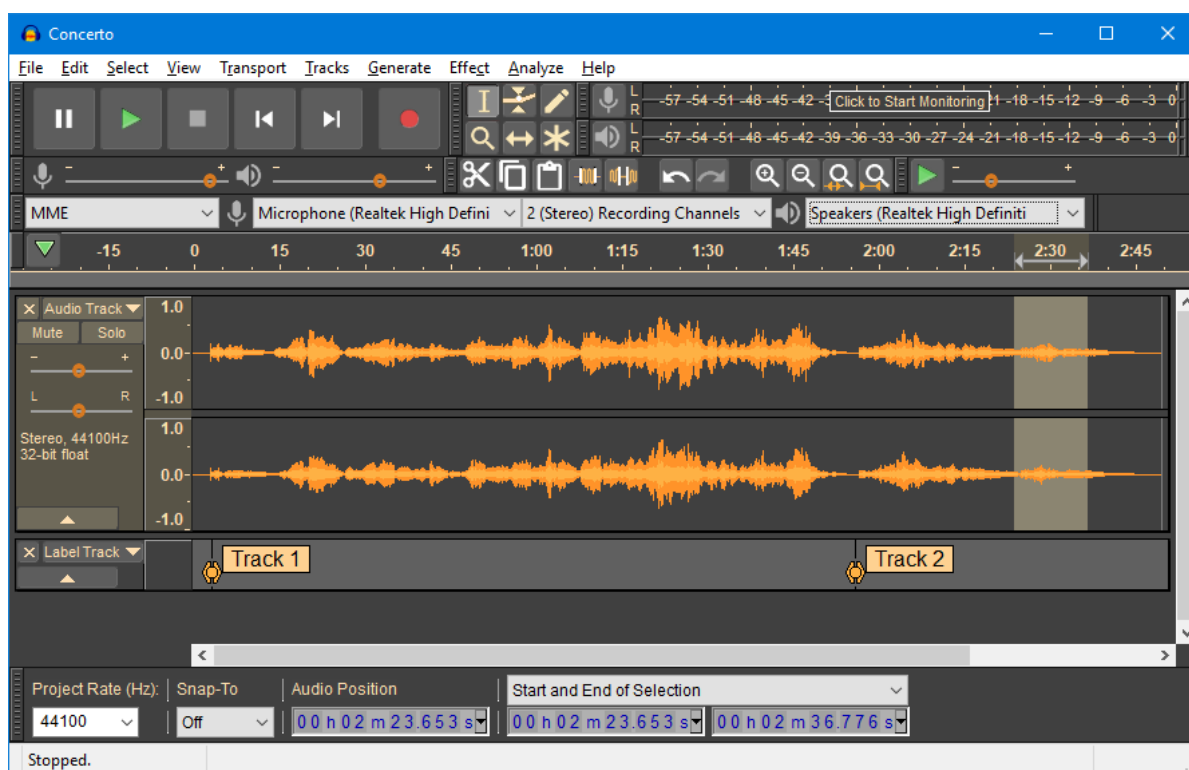


Рисунок 1.7 – Інтерфейс програми Audacity

Отже, використано такі програмні продукти та середовища розробки: рушій Unity, редактор Blender 3D та для звукозапису – Audacity.

Висновки до першого розділу

Проведений аналіз предметної області, визначено основні компоненти які має мати гра. Було розглянуто наявні аналоги та виявлення особливості які має мати майбутні додаток: зручний UI, відмінну особливість у геймплеї та наявність якісного аудіо.

Визначено оптимальну архітектуру, розглянуто та визначено патерни проектування які будуть використані у проектуванні сутностей і обрано необхідний пакет програмних продуктів для реалізації поставлених задач.

					КН.КР.Б – 122 – 23 – ПЗ	23
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ТА РЕАЛІЗАЦІЯ ДОДАТКУ

2.1. Визначення варіантів використання та структури системи

Для опису функціональних вимог додатку або його підсистем зазвичай використовується діаграма варіантів використання. Вона представляє загальну картину того як додаток буде використовуватись. **Діаграма варіантів використання (Use Case Diagram)** – це діаграма яка допомагає відображати взаємодію між акторами(користувачами системи) та системою у рамках конкретних сценаріїв використання[17].

Діаграма варіантів використання складається з акторів, безпосередньо самих варіантів використання та взаємозв'язків між ними. Актори представляють зовнішні сутності, такі як, користувачі, системи або зовнішні сервіси, які взаємодіють з системою. Варіанти використання описують функціональні можливості системи та взаємодію між акторами та системою.

Для розробки діаграми варіантів використання було визначеного одного актора – гравця. На рисунку 2.1 зображена діаграма з основним механіками з якими гравець може взаємодіяти.

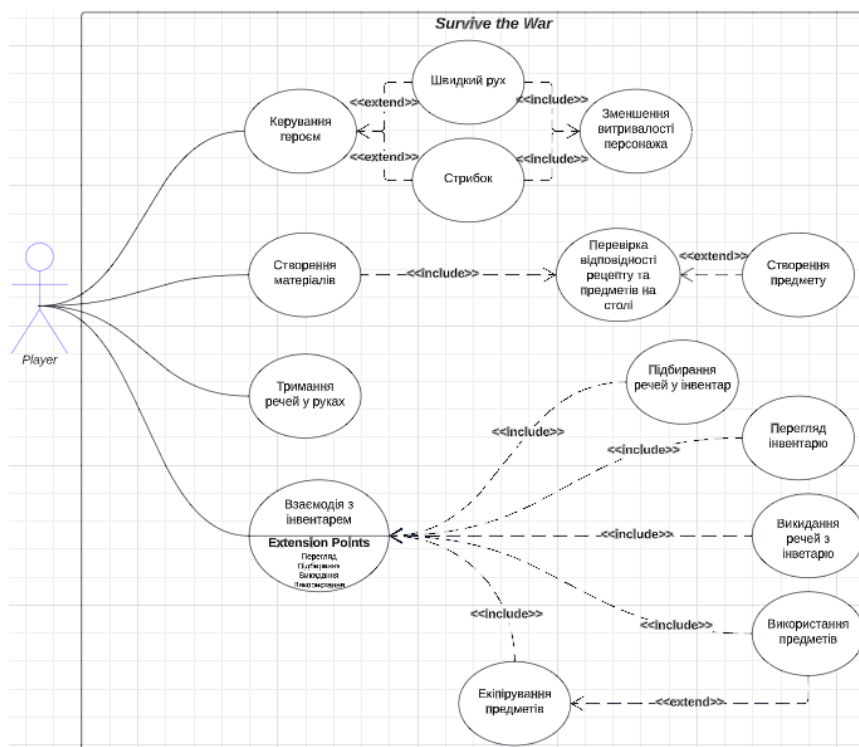


Рисунок 2.1 – Діаграма варіантів використання основних механік

На рисунку 2.2 є діаграма варіантів використання, на якій продемонстровано механіки потреб персонажа, виконання завдань та взаємодії з навколишнім середовищем.

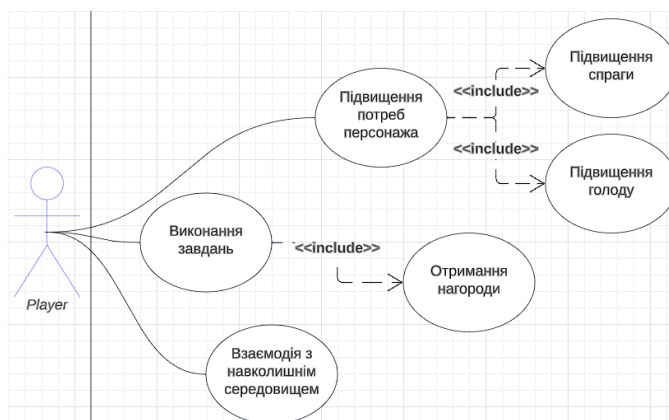


Рисунок 2.2 Діаграма варіантів використання додаткових механік

На діаграмах варіантів використання(рисунок 2.1, рисунок 2.2) зображений один актор – гравець. Він може керувати персонажем, створювати різного роду речі, тримати речі, взаємодіяти зі своїм інвентарем, переглядати його, підбирати речі у інвентар, викинути річ з інвентарю, використати річ з інвентаря або екіпірувати. Також у гравця є потреби, які поступово підвищуються, гравець може виконувати випадкові завдання та взаємодіяти із навколишнім середовищем(відкривати двері, вмикати світло та вимикати і т.д.).

Функціональні вимоги:

- керування героєм: користувач має можливість керувати героєм на локації. Герой може як стрибати так і стрибати;
- взаємодіяти з інвентарем: користувач має можливість взаємодіти із інвентарем героя, переглядати, додавати або викидати речі і екіпірувати їх;
- створення речей: користувач має можливість створювати речі на верстаку;
- підвищення потреб персонажа: у персонажа мають з часом підвищуватись потреби, такі як голод та спрага і способи знижувати їх;

- виконання завдань: герой має можливість виконувати завдання і отримувати за них випадкові нагороди;
- взаємодія зі світом: гравець має можливість взаємодіяти з віртуальним середовищем через героя;
- Не потрібно перебувати онлайн: користувач може грати без підключення до інтернету у довільний момент часу.

Не функціональні вимоги:

- Продуктивність: гра повинна бути добре оптимізована та підтримувати безперебійну роботу;
- Можливість експлуатації: після оновлення все повинно працювати без збоїв. Оновлення версій гри повинно здійснюватись шляхом випускання патчів.

Проаналізувавши функціональні вимоги були побудовані сутності з яких складається гра. Для графічного зображення сутностей та зв'язків між ними виористовують діаграму класів. **Діаграма класів** – діаграма на якій представлена сукупність декларативних, або статичних елементів моделі, таких як класи з атрибутами й операціями, а також відношення. На рисунку 2.3 показана діаграма класів, модуля завдань, ці класи відповідають за створення та відображення завдань гравцеві[18].

Клас Task є основним у завданні у ньому початково зберігається інформація про завдання, назва, опис та нагорода. Є методи щоб дістати опис завдання – GetDescription(). Метод Initialize() потрібен для початкової ініціалізації завдання. Метод Evaluate() викликається коли пункт із завдання виконується гравцем там перевіряється чи кількість виконаних пунктів дорівнює кількості всіх пунктів у завданні, якщо перевірка повертає true тоді викликається метод Complete() який ініціює виконання завдання. CheckGoals() викликається коли потрібно перевірити чи виконав гравець всі пункти у завданні, якщо виконав, тоді завдання помічаються як виконані і гравцю дається нагорода.

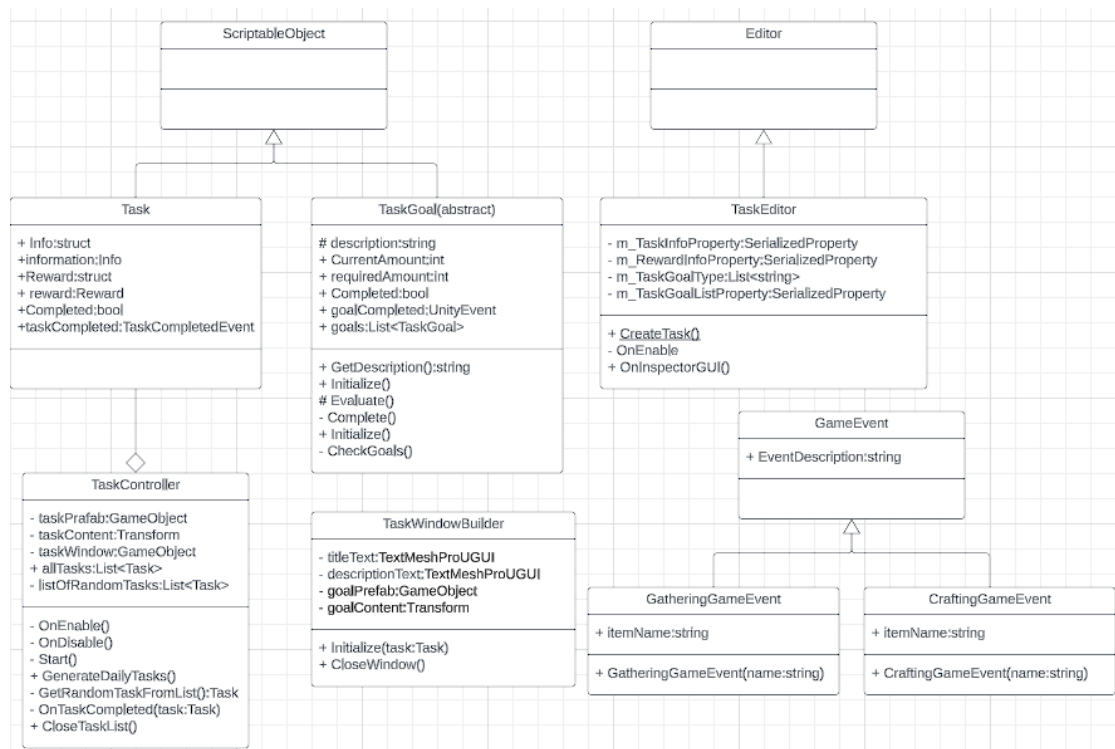


Рисунок 2.3 – Діаграма класів модуля завдань

Також важливим класом являється TaskEditor який не задіюється безпосередньо у грі, але потрібен для створення самих завдань у рушії. Він наслідує клас Editor, що дає йому змогу розширити інтерфейс Unity та писати власні вікна. Завдяки йому створюється власний «інтерфейс» у рушії для створення завдання(рисунок 2.4).

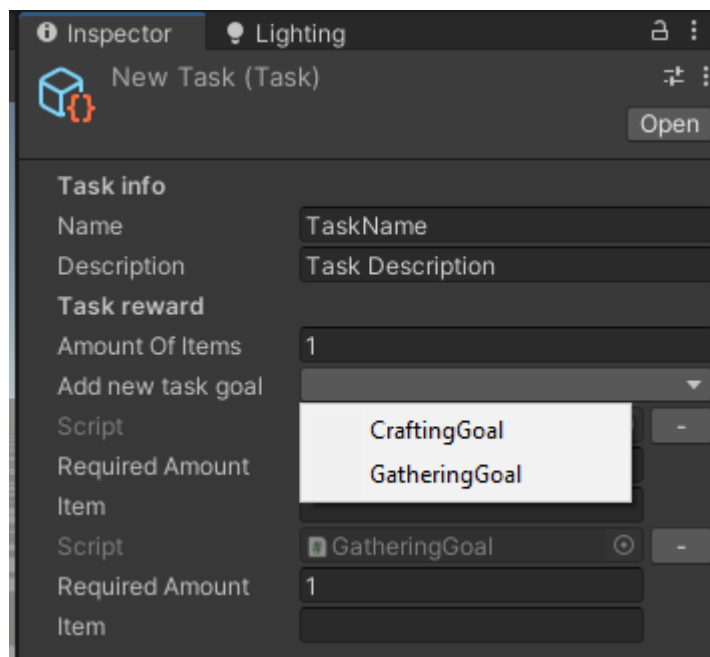
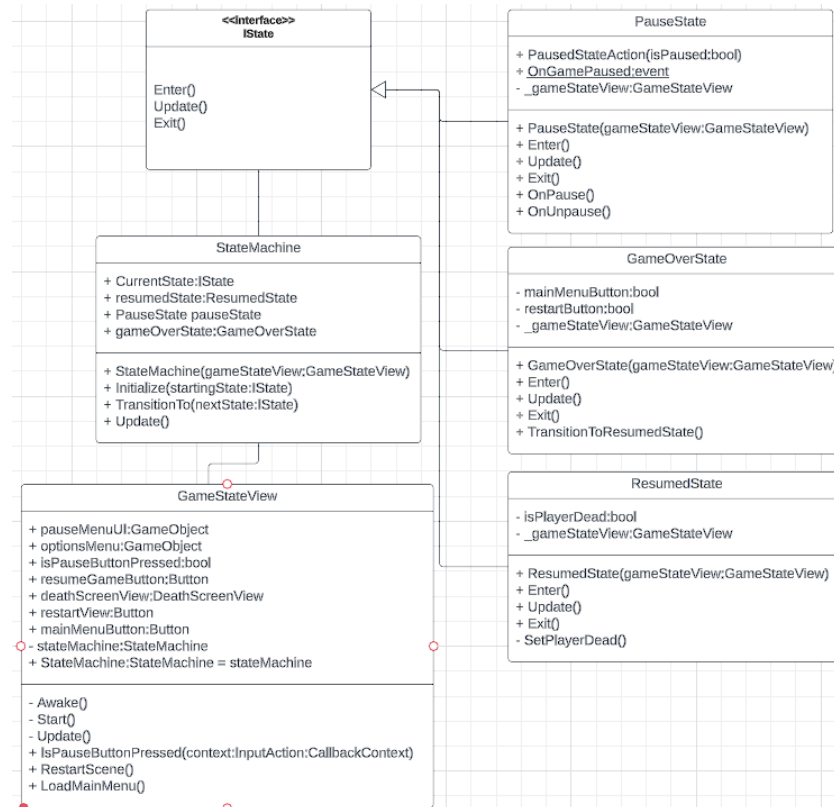


Рисунок 2.4 – Інтерфейс створення задвання

У наступній діаграмі(Рисунок 2.5), класи були спроектовані за допомогою паттернів «Стан»(StateMachine) та «Спостерігач»(Observer) який викистимувавсь у проекті неодноразово.



У випадку нашого проекту, шаблон «Стан» було реалізовано за допомогою інтерфейсу `IState` який наслідували інші стани, такі як, `PauseState`, `GameOverState` та `ResumedState` та безпосередньо самого класу `StateMachine` який налаштовував те, як будуть працювати переходи між станами.

- Enter – викликається при переході у поточний стан;
- Update – викликається кожен кадр, у ньому перевіряється коли потрібно перейти у наступний стан;
- Exit – викликається при виході з поточного стану;

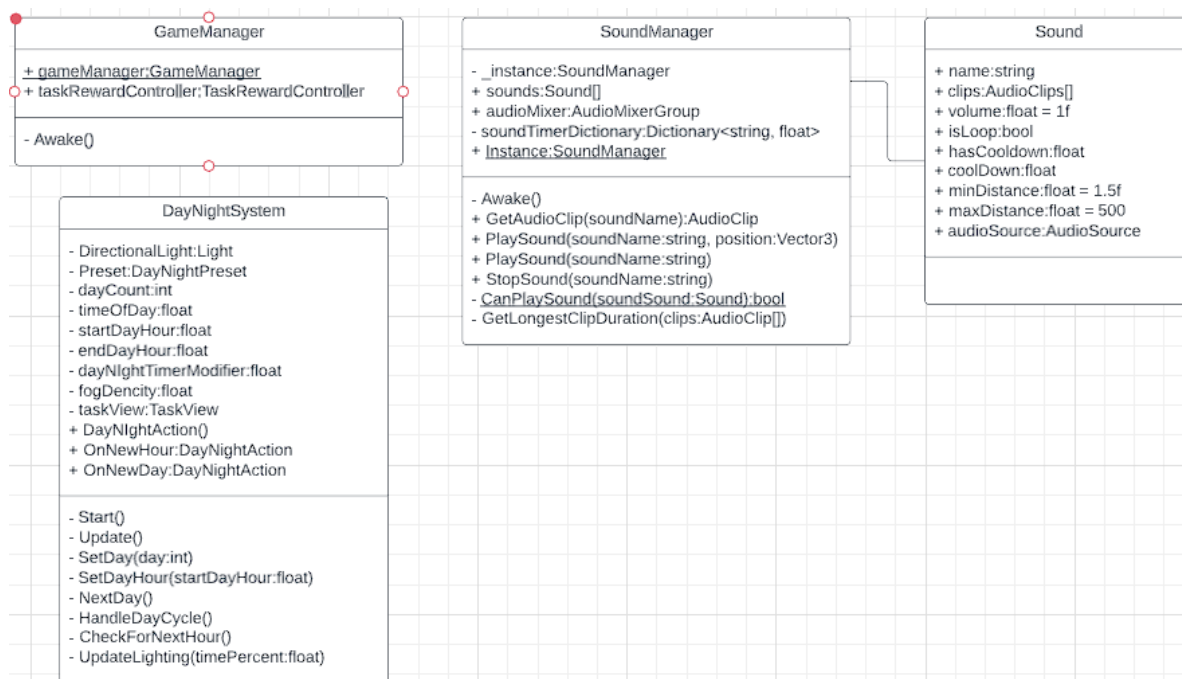


Рисунок 2.6 – Діаграма обслуговуючих класів

Клас GameStateView отримує інформацію від станів та ініціалізує інтерфейс користувача в залежності від стану, тобто якщо теперішнім станом є PauseState тоді потрібно показати гравцеві екран(меню) паузи.

На рисунку 2.6 зображена діаграма на якій є важливі класи, оскільки відповідають за аудіо гри та цикл дня і видачу нагороди гравцю. Для GameManager та SoundManager було прийняте рішення використати шаблон «Одинак», для того щоб можна було контролювати гру та програвати звуки у будь-якому стані гри.

У SoundManager є декілька методів:

- GetAudioClip – метод, який за назвою з класу Sound знаходить audio clip;
- PlaySound має дві варіації, у випадку з аргументом де передається назва та позиція, відбувається створення дочірнього елемента з назвою звуку та програвання звуку у заданій позиції у 3D-просторі, у іншому випадку з одним аргументом, програв звук по назві у 2D просторі.

- StopSound знаходить дочірній об'єкт SoundManager, зупиняє AudioSource та знищує об'єкт;

Клас Sound використовується для налаштування звуків і має такі параметри як, назва звука, гучність, чи відтворювати його повторно, чи є в нього в перезарядка.

DayNightSystem контролює добу у грі та налаштування світла. На діаграмі зображеній на рисунку 2.6 показані класи, які відповідають за пересування гравця, його потреби, взаємодії. Розбиття функціоналу на такі класи – компоненти є вимогою архітектури яку ми обрали на початку та заохочення самим рушієм Unity, оскільки такі компоненти легко тестувати, та по бажанню вмикати і вимикати.

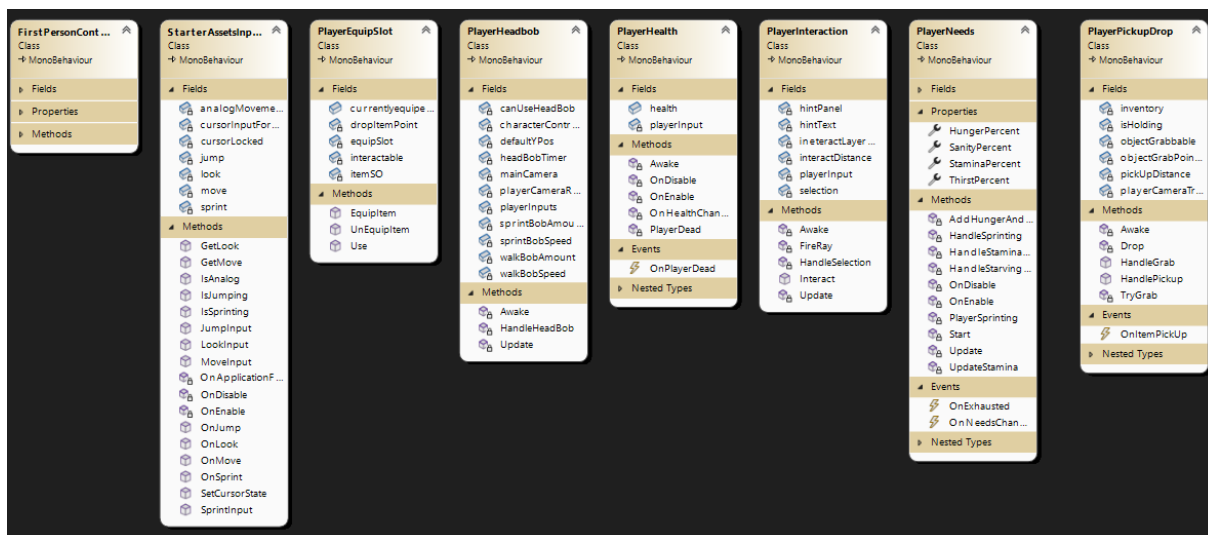


Рисунок 2.6 – Діаграма класів які стосується взаємодії гравця

Клас StarterAssetInputs – компонент який реєструє кнопки які натискає гравець та передає дані у FirstPesrsonController де реалізоване базове керування гравцем(біг, стрибок, огляд мишею). Наступні класи є модульними та опціональними, їх можна на далі без проблем розширювати. PlayerEquipSlot відповідає та екіпірування та використання предметів, має методи EquipItem() – Екіпірувати предмет, Unequip() – прибрати з рук, Use() – Використати активний предмет у руці.

PlayerHeadBob – використовується для надання ефекту кивання голови при ходьбі. Компонент PlayerNeeds реалізовує потреби персонажа(голод, спрагу,

витривалість) головний метод це HandleStarvingAndThirsting який часом знижує потреби та PlayerSprinting який знижує витривалість, якщо герой довгий час бігає. Також у цьому класі присутні дві події, які є шаблоном «Наглядач», завдяки їм, оновлюється UI частина потреб(спрайти голоду та спраги, полоска витривалості). У PlayerPickUpDrop реалізовується функціонал завдяки якому гравець може просто утримувати річ у руках, або підняти у інвентар.

2.2. Дизайн документ

Перед створенням гри зазвичай створюється дизайн документ для спрощення розробки проекту, допомогти іншим та самому орієнтуватись у розробці гри. **Геймдизайн документ (Game Design Document)** – це повноцінна технічна документація гри, «бібліотека» проекту, допоміжний інструмент з основами, які будуть керівництвом до дії команди[19].

Гра розробляється під назвою: Survive The War.

Тема сеттинг.

Це гра у жанрі Survival події якої відбуваються у сучасності під час війни. Гравець бере на себе роль звичайного мешканця який залишився у місті на переживає війну.

Геймплей є основною складовою гри, тому наступним буде розглянуто основний цикл геймплею та його особливості. Керування ігровим персонажем відбувається від першої особи. У гравця є можливість вільно оглядатись, швидко бігати та стрибати. У грі є лише один режим, він йде до тих пір доки гравець не помре. У героя є інвентар, він може підбирати речі, викидати та деякі екіпірувати, також він може просто тримати їх у руках, це знадобиться для механіки створення речей, оскільки щоб зробити якусь річ, потрібно фізично покласти складники на верстак перед тим обравши потрібний рецепт. Гравець може використовувати предмети які є активними у його руці. Також у нього є потреби, голод, спрага, є полоска здоров'я та глузду.

Потреби збільшуються з часом, їх можна зменшити відповідно з'ївши їжу або випивши води. У грі також присутня симуляція доби, тобто є день та ніч.

					КН.КР.Б – 122 – 23 – ПЗ	31
Зм.	Арк.	№ докум.	Підпис	Дата		

Днем гравцю потрібно шукати ресурси для створення корисних предметів та для того, аби пережити успішно ніч. У ночі, гравцю необхідно заправляти та вмикати генератор, щоб було освітлення, оскільки якщо гравець не буде знаходитись у світлі в ночі, він буде втрачати глузд.

При нульовому глузді у гравця почне або відніматись здоров'я. Отже гравцю потрібно також шукати паливо для генератора та/або робити освітлення власноруч що всеодно віднімало б глузд.

Гравець ще може втратити глузд при випадкових подіях, таких як, наприклад, повітряна тривога, при якій гравцю потрібно сховатись у безпечне місце, якщо ж під час події гравець не буде знаходити у безпечному місці він також буде втрачати глузд. Отже, крім стандартних механік притаманних іграм у жанрі виживання, можна відмітити особливості(USP):

- Задля того щоб зробити щось на робочому місці, потрібно покласти предмет на верстак та вибрати потрібний рецепт;
- Днем гравцеві потрібно шукати ресурси, у тому числі бензин для генератора, аби пережити ніч, оскільки уночі, якщо гравець не буде знаходитись у світлі він буде втрачати глузд;
- Під час лунавання сирени у місті, гравцю необхідно сховатись у будь-яке доступне приміщення, таким чином він збереже себе від отримання шкоди.

Інтерфейсу гри має бути якомога менше щоб гравець більше занурився у атмосферу гри.

Також важливим елементом є світ у якому буде знаходить гравець, у нашому випадку це не величке місце скомпоноване з різних ассетів. Події гри розгортаються локації міста. На рисунках зображено основні будинки та навколишнє середовище в якому буде знаходитись гравець.

					КН.КР.Б – 122 – 23 – ПЗ	32
Зм.	Арк.	№ докум.	Підпис	Дата		



Рисунок 2.7 – Дім у якому живе головний герой

На рисунку 2.7 зображений будинок у якому живе головний герой, всього у ньому 4 поверхи, герой має житло на першому поверсі. У своїх апартаментах(рисунок 2.10) гравець зможе знайти ресурси та майстерню на якій зможе зробити потрібні речі.



Рисунок 2.9 – Перший поверх дому



Рисунок 2.10 – Квартира головного героя

На рисунку 2.11 зображена невелика покинута місцевість, яка складається з декількох будинків. У цій області гравець може знайти більше металобрухту та інших вироничих матеріалів ніж на інших місцевостях.



Рисунок 2.11 – Покинуті споруди на локації

Повністю локацію можна побачити на рисунку 2.12. Оглядаючи локацію окрім звичайних багатоповерхових будинків можна побачити маленькі лісові хатинки до яких ведуть стежинки, саме у них гравець зможе переховуватись під

час сирени та знаходити різноманітні речі. Також там є покинуті кафе у яких гравець зможе знайти більше їжі.



Рисунок 2.12 – Інші споруди

Для надання 3D-моделі зображення, нерівностей та інших деталей, використовуються текстури. У відеогрі, текстура – це графічний елемент, який використовується для створення вигляду поверхонь об’єктів у відеогрі. Вона надає деталізацію, кольори, структуру та інші візуальні аспекти об’єктів у грі.

Текстури використовуються для створення ілюзії реалістичного відображення поверхонь. Спочатку текстури створюють художники, а потім вже ці зображення використовуються у рушіях для накладання їх на геометрію об’єктів у грі. Текстури можуть бути статичними або динамічними, у випадку проекту, будуть використовуватись лише статичні текстури.

Також варто зазначити, що є декілька типів текстур:

- Кольорова текстура(Albedo, Base map) – це текстура яка визначає колір кожної точки поверхні об’єкта. Тобто кольорова текстура це те, що розфарбовує саму модель, але без тіней, матеріалу, тощо;

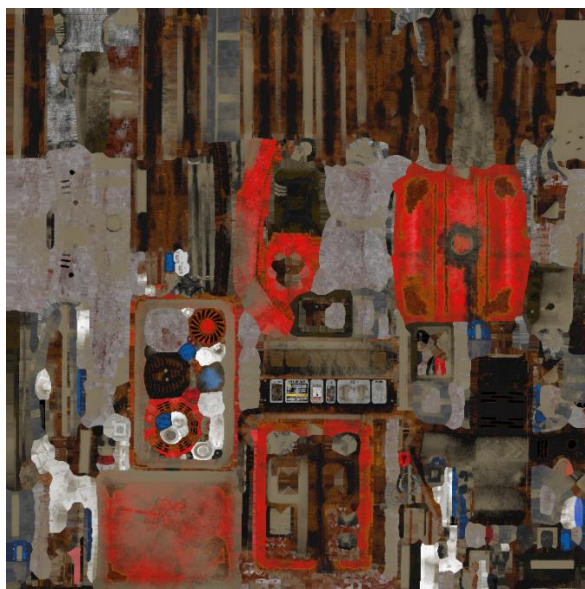


Рисунок 2.7 – Кольорова текстура генератора

- Текстура нормалей(Normal map) – ця текстура використовується для симуляції деталей на поверхні об’єкта, таких як рельєф або текстурні нерівності;

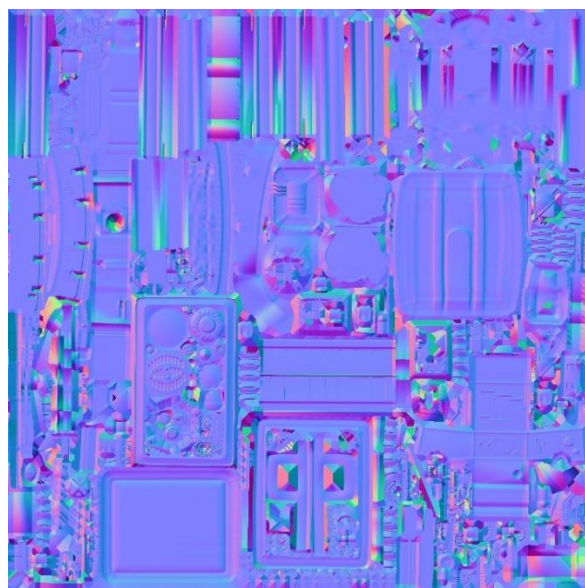


Рисунок 2.8 – Текстура нормалей генератора

- Текстура висот(Height map) – текстура що використовується для створення тривимірного ефекту на поверхні об’єкта. Вона використовується для визначення висоти кожної точки поверхні, дозволяючи створити враження рельєфу та глибини;

- Бамп-мап текстура(Bump map). Ця текстура подібна до текстури нормалей, оскільки вона також використовується для створення рельєфу та деталей на поверхні об'єкта. Однак, вона використовує сірі відтінки, а не RGB-канали, для представлення зміни висоти поверхні.

Є і інші види текстур:

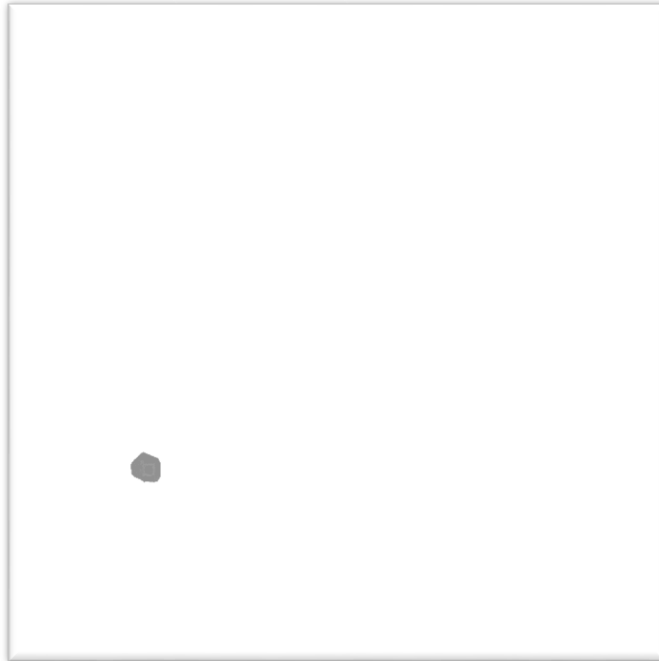


Рисунок 2.9 – Текстура прозорості

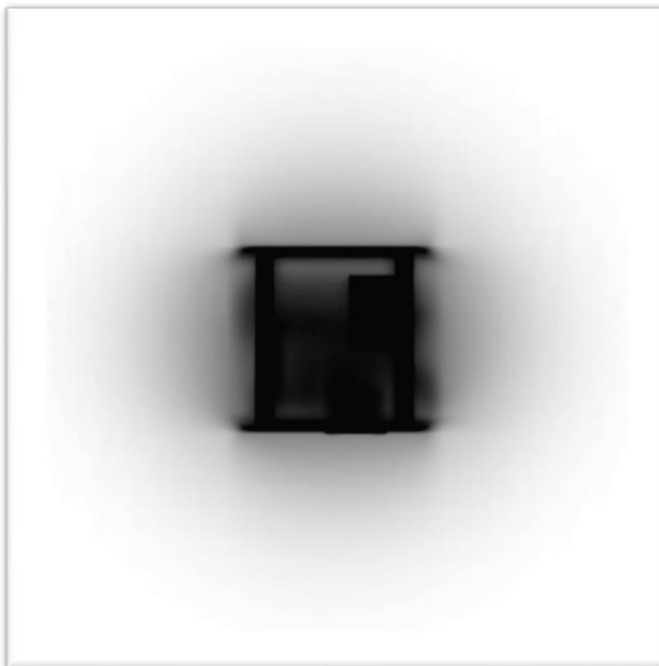


Рисунок 2.10 – Ambient Occlusion(AO) текстура

Зображення на рисунках 2.11 та 2.12 використовуються для відображення потреб персонажу, рисунок 2.11 – потреби персонажа задоволені, рисунок 2.12 – персонаж голодний та має спрагу.

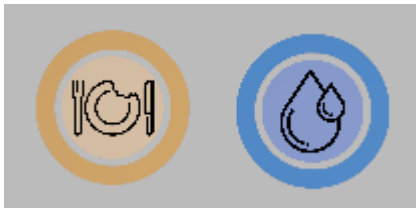


Рисунок 2.11 – Спрайти голоду та спраги



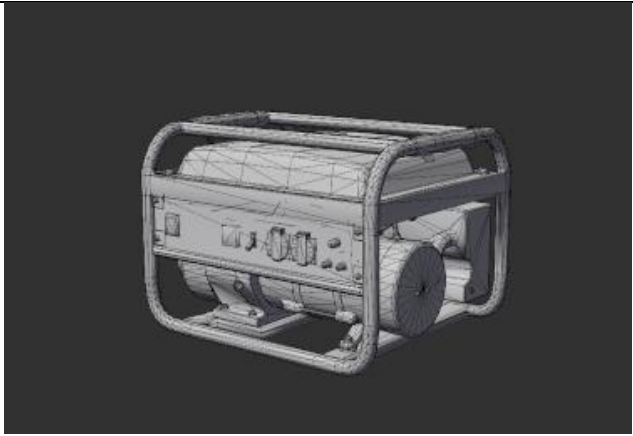
Рисунок 2.12 – Змінені спрайти голоду та спраги

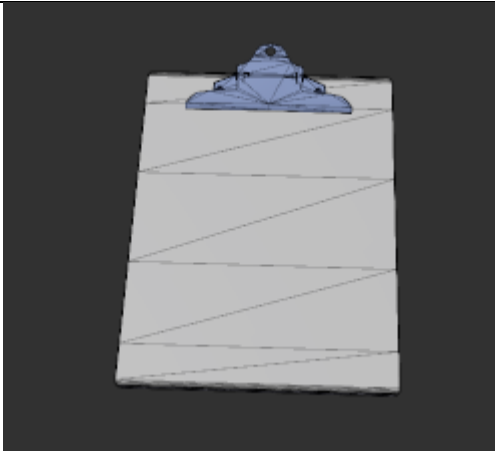
У таблиці 2.1 наведено декілька 3D-моделей, що використовуються у грі.

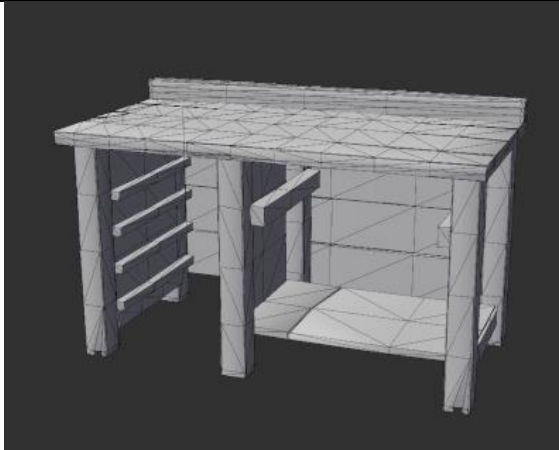
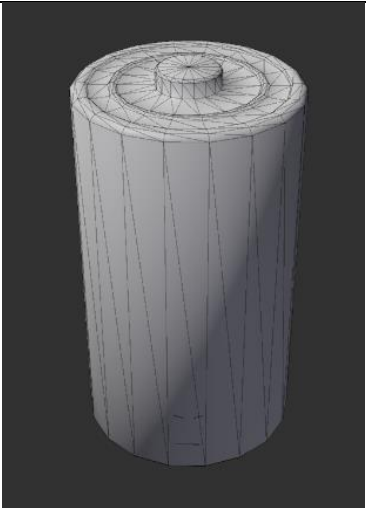
Деякі ассети, включно з 3D-моделями були взяті або придбані у Unity Asset Store. Ассет це будь-який ресурс, що використовується в грі для створення візуального або аудіо-візуального вмісту. Ассет може бути будь-яким елементом, який використовується у грі, включаючи моделі персонажів, об'єкти, текстури, анімації, звукові ефекти, музику, фони, інтерфейси та багато іншого.

Таблиця 2.1

3D-моделі

Сітка 3D-моделі	3D-модель з текстурами
	

Звук – важливий компонент будь-якої гри та відіграє велику роль у сприйманні гри та її атмосфері. Всі налаштування звуку здійснюються у

спроектованому класі SoundManager, він має масив Sound який у свою чергу містить всі налаштування потрібні для звуку(рисунок 2.14)

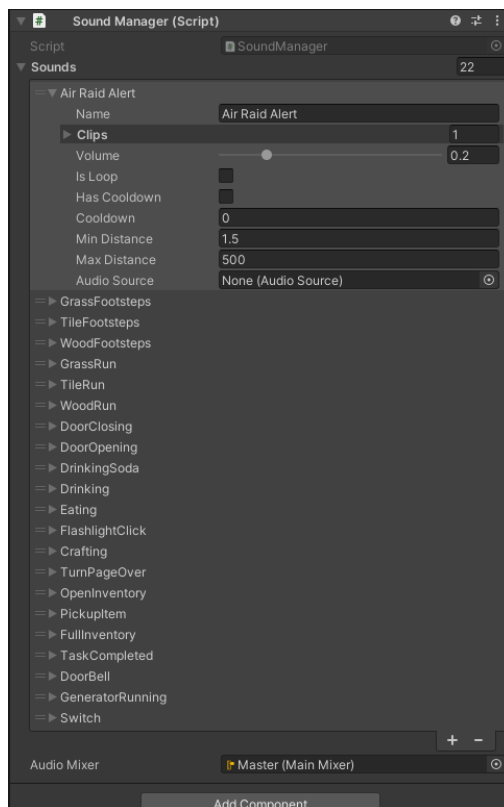


Рисунок 2.14 – Налаштування звуків гри

2.3. Реалізація алгоритмів роботи гри

У даному розділі розглянуто частину створеного коду, що дасть основні уявлення про процес розробки. Детальніше з кодом можна ознайомитись з важливою частиною коду в додатку А.

Далі наведено реалізацію кількох важливих класів та алгоритм роботи нової системи вводу Unity.

Керування у грі є простим та інтуїтивно зрозумілим, у грі також будуть присутні підказки які будуть відображатись у нижньому лівому кутку екрану гравця(рисунок 2.15) та адаптуватись до поточної мови клавіатури користувача(рисунок 2.16).

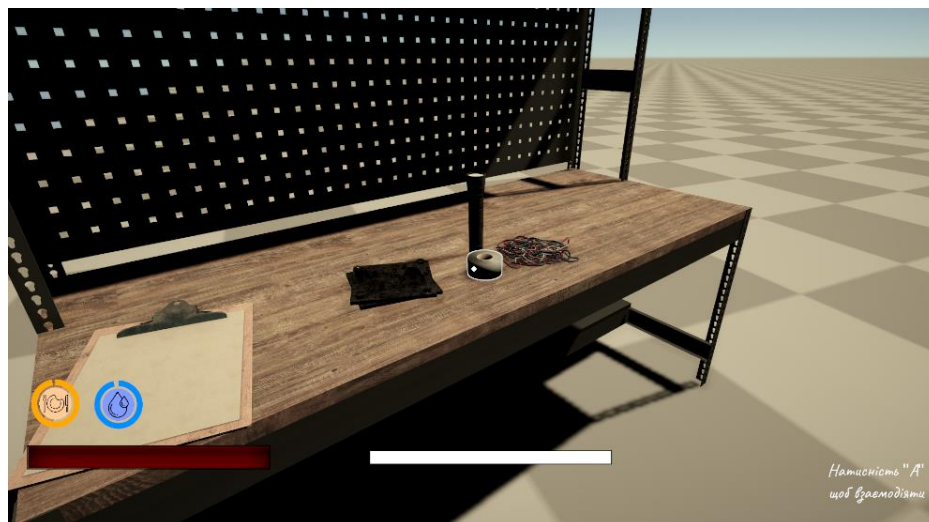


Рисунок 2.15 – Підказка гравця з українською клавіатурою



Рисунок 2.16 – Підказка гравця з англійською клавіатурою

Також, через те, що у проекті було використано нову систему вводу Unity(рисунок 2.17), налаштування, зміна та додавання нових дій є надзвичайно простою.

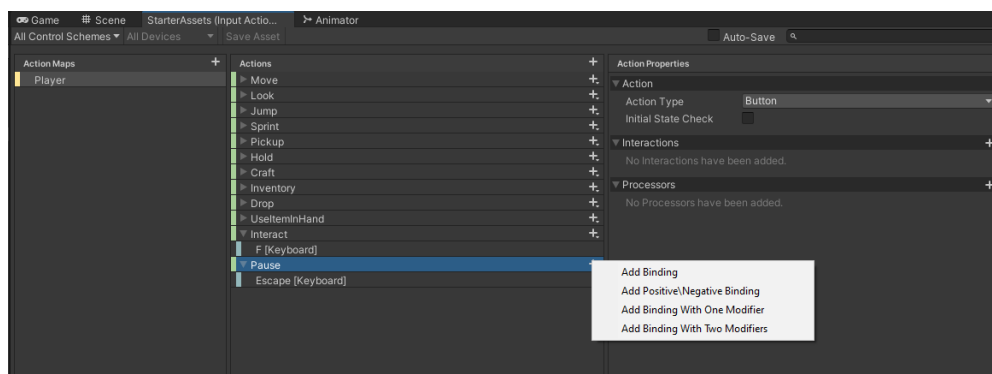


Рисунок 2.17 – Нова система вводу Unity

Нова система вводу є event-based, тобто працює за допомогою подій, є декілька способів використовувати її, зокрема напряду через згенерований ассетом клас, використовуючи події мови C# та, найзручніший, який і використовувався у даному проєкті, це використовувати Unity Event у компоненті Player Input(Рисунок 2.18).

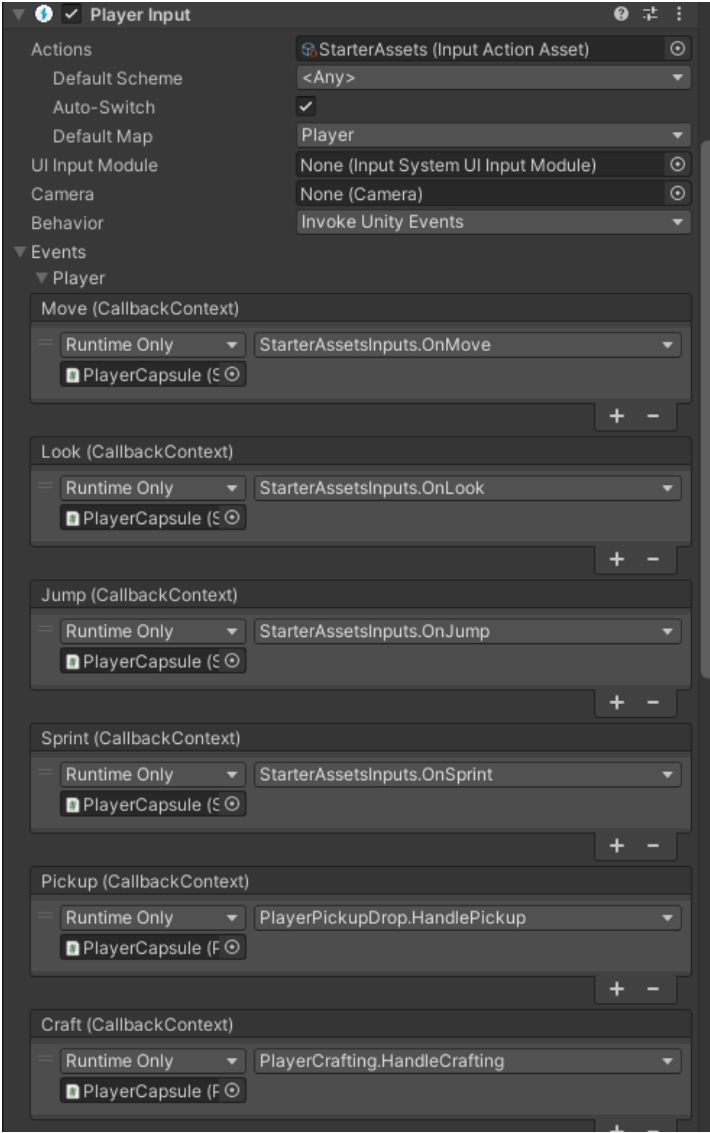


Рисунок 2.18. Компонент Player Input

Player Input це спеціальний компонент завдяки якому система вводу отримує значення з клавіатури. Як тільки у ассеті створюється нова дія (див. рисунок 2.17) вона відображається й і у компоненті PlayerInput де можна підписати функцію на подію. Для того щоб метод можна було підписати на подію та отримувати у методі дані про конкретну кнопку, потрібно аби функція у параметрах приймала об'єкт типу InputAction.CallbackContext:

```

public void UnequipItem(InputAction.CallbackContext context)
{
    if(context.performed)
    {
        if (currentlyequipedItem == null)
            return;

        currentlyequipedItem.gameObject.SetActive(false);
        GameObject dropItem = Instantiate(itemSO.Prefab, new
        Vector3(dropItemPoint.position.x, dropItemPoint.position.y, dropItemPoint.position.z),
        Quaternion.identity);
        itemSO = null;
        interactable = null;
        currentlyequipedItem = null;
    }
}

```

Отже, нова система вводу Unity є зручною та надає безліч переваг, завдяки їй можна легко змінити, переназначити кнопку а також, додати нову дію, на кнопку клавіатури, телефону, або геймпаду, тому, якщо потрібно буде реалізувати управління з геймпаду, для цього буде достатньо додати нові кнопки до дій.

Основний клас InventorySystem який відповідає за основний функціонал, додати предмет, видалити, повернути предмет такого типу, створити нову клітинку в інвентарі.

Лістинг коду:

```

using System.Collections.Generic;
using UnityEngine;

namespace OM
{
    public class InventorySystem
    {
        public int slotsLimit = 6;
        public Dictionary<ItemSO, InventoryItem> ItemDictionary;
        public List<InventoryItem> Inventory { get; set; }

        public delegate void InventoryAction();
        public static event InventoryAction OnInventoryChange;

        public InventoryItem Get(ItemSO item)
        {
            if (ItemDictionary.TryGetValue(item, out InventoryItem inventoryItem))
            {
                return inventoryItem;
            }
            return null;
        }

        public void Add(ItemSO referenceData)
        {
            if (ItemDictionary.TryGetValue(referenceData, out InventoryItem value))
            {
                value.AddToStack();
            }
        }
    }
}

```

```

    }
    else
    {
        CreateNewItemCell(referenceData);
    }
    SoundManager.Instance.PlaySound("PickupItem");
    OnInventoryChange();
}

private void CreateNewItemCell(ItemSO referenceData)
{
    InventoryItem newItem = new InventoryItem(referenceData);
    Inventory.Add(newItem);
    ItemDictionary.Add(referenceData, newItem);
}

public void Remove(ItemSO referenceData)
{
    if (ItemDictionary.TryGetValue(referenceData, out InventoryItem value))
    {
        if (value.StackSize == 1)
        {
            Inventory.Remove(value);
            ItemDictionary.Remove(referenceData);
        }
        value.RemoveFromStack();
    }
    OnInventoryChange();
}
}
}

```

Тут здебільшого прості методи, у які обов'язково передаються об'єкти типу ItemSO – це Scriptable Object, тобто сам предмет, який має властивості, такі як, тип предмету, назву, префаб та спрайт.

Лістинг коду ItemSO:

```

using UnityEngine;

namespace OM
{
    [CreateAssetMenu(menuName = "ScriptableObjects/ItemSO")]
    public class ItemSO : ScriptableObject
    {
        public enum ItemType
        {
            None,
            CraftingMaterial,
            Equipable,
            Eatable
        }

        [SerializeField] public ItemType itemType;
        [SerializeField] public string itemName;
        [SerializeField] private GameObject prefab;
        [SerializeField] public Sprite image;
        [SerializeField] public int MaxStackSize;

        public GameObject Prefab { get { return prefab; } }
    }
}

```

Атрибут CreateAssetMenu потрібен для можливості створення самого асету у рушії, таким чином при кліку лівої кнопки миші у папці проекту, з'являється спеціальне контекстне меню, де можна створити об'єкт цього типу(рисунок 2.19).

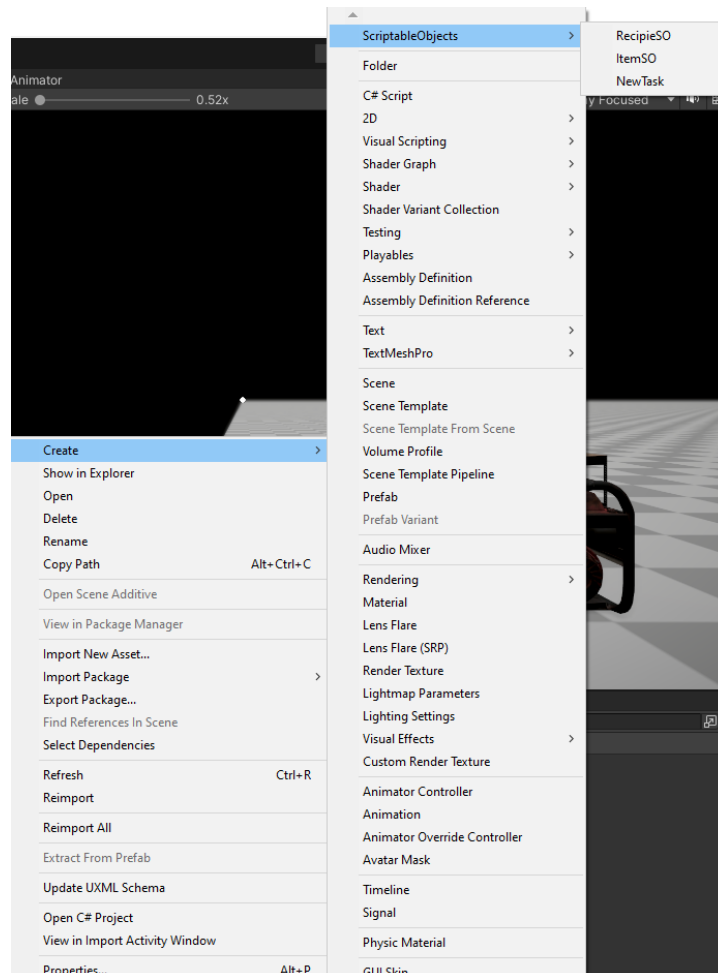


Рисунок 2.19 – Спеціальне контекстне меню

InventoryItem потрібен у системі інвентаря для того, щоб відслідковувати кількість одного предмету у слоті.

Лістинг InventoryItem:

```
using System;

namespace OM
{
    [Serializable]
    public class InventoryItem
    {
        public ItemSO itemData { get; set; }
        public int StackSize { get; set; }
        public int MaxStackSize { get; set; }

        public InventoryItem(ItemSO source)
        {
            itemData = source;
        }
    }
}
```

```

        MaxStackSize = source.MaxStackSize;
        AddToStack();
    }

    public void AddToStack()
    {
        StackSize++;
    }

    public void RemoveFromStack()
    {
        StackSize--;
    }
}
}

```

Також є клас InventoryMediator що є своєрідним патерном Посередника(Mediator). Він ініціалізує інвентар, оскільки InventorySystem не може, через те, що на початку гри, gameObject до якого прикріплений скрипт є не активним і скрипт не може бути активним, також це потрібно, для оновлення інвентаря коли він закритий як вище було, через те що gameObject не активний і скрипт не буде працювати.

Лістинг InventoryMediator

```

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;

namespace OM
{
    public class InventoryMediator : MonoBehaviour
    {
        [SerializeField] private InventoryView inventoryView;
        [SerializeField] private TaskWindowBuilder taskWindow;
        [SerializeField] private GameObject inventoryMenu;
        [SerializeField] private Transform playerDropPoint;

        public InventorySystem InventorySystem = new InventorySystem();

        void OnEnable()
        {
            InventorySystem.OnInventoryChange += UpdateInventory;
            PlayerPickupDrop.OnItemPickUp += AddItemToInventory;
        }

        void OnDisable()
        {
            InventorySystem.OnInventoryChange -= UpdateInventory;
            PlayerPickupDrop.OnItemPickUp -= AddItemToInventory;
        }

        private void Awake()
        {
            InventorySystem.Inventory = new List<InventoryItem>();
            InventorySystem.ItemDictionary = new Dictionary<ItemSO, InventoryItem>();
        }

        public void UpdateInventory()
        {
            inventoryView.OnUpdateInventory();
        }
    }
}

```

```

public bool IsInventoryFull(ItemSO item)
{
    if (InventorySystem.Get(item) == null)
    {
        if (InventorySystem.Inventory.Count == InventorySystem.slotsLimit)
        {
            SoundManager.Instance.PlaySound("FullInventory");
            return true;
        }
    }
    return false;
}

public void AddItemToInventory(ItemSO item)
{
    InventorySystem.Add(item);
    EvaluateGatheringGoal(item.itemName);
}

public void RemoveItemFromInventory(ItemSO item)
{
    InventorySystem.Remove(item);
}

public void RemoveItemAndDrop(ItemSO item)
{
    if(InventorySystem.Get(item) != null)
    {
        InventorySystem.Remove(item);
    }
    GameObject dropItem = Instantiate(item.Prefab,
        new Vector3(playerDropPoint.position.x, playerDropPoint.position.y,
playerDropPoint.position.z),
        Quaternion.identity);
}

public void InventoryVisibility(InputAction.CallbackContext ctx)
{
    if (ctx.performed)
    {
        inventoryMenu.SetActive(!isInventoryMenuActive());
        if (isInventoryMenuActive())
        {
            Cursor.lockState = CursorLockMode.None;
            SoundManager.Instance.PlaySound("OpenInventory");
        }
        else
        {
            Cursor.lockState = CursorLockMode.Locked;
            taskWindow.CloseWindow();
        }
    }
}

private bool isInventoryMenuActive()
{
    return inventoryMenu.activeSelf;
}

public void EvaluateGatheringGoal(string itemName)
{
    EventManager.Instance.QueueEvent(new GatheringGameEvent(itemName));
}
}
}

```

Також, InventoryMediator виконує деякі функції в UI частині, тому він є таким собі посередником, не тільки ініціалізовує інвентар, а й оновлює його і UI частину та встановлює його видимість.

На діаграмі активностей (рисунок 2.20) можна побачити загальний алгоритм додавання предмету у інвентар.

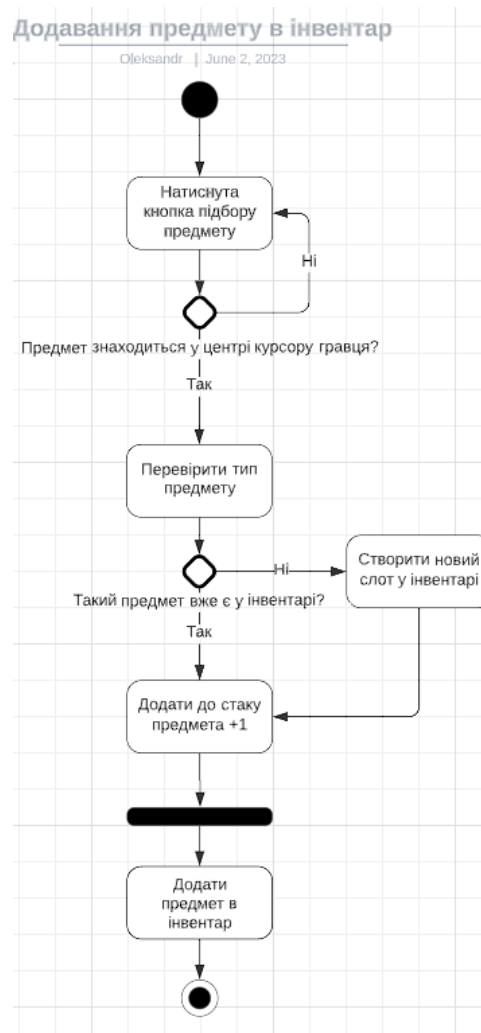


Рисунок 2.20 – Алгоритм додавання предмету

Наступним алгоритмом є реалізація подій, які завдають шкоду гравцеві(рисунок 2.21)

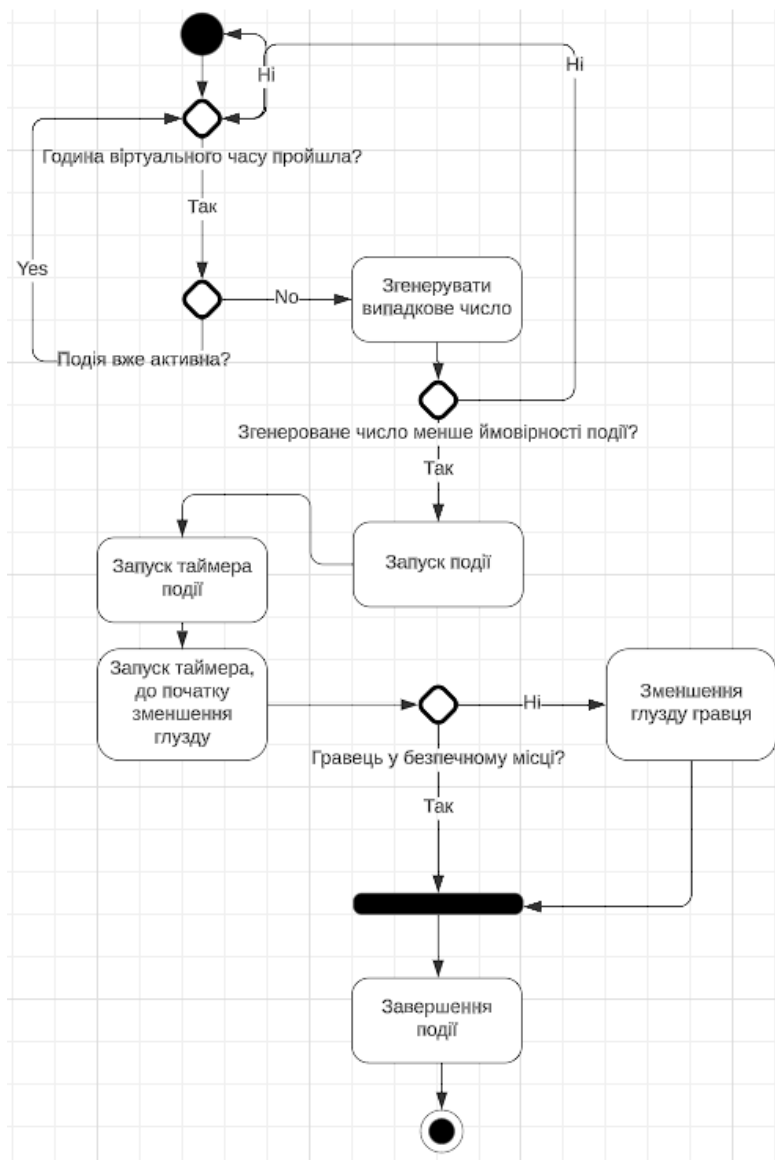


Рисунок 2.21 – Алгоритм роботи подій

Висновки до другого розділу

Проведено розробу програмного забезпечення, яке створене на основі дизайн документу та технічного завдання. Розроблено діаграму варіантів використання загального геймплею. Розроблено діаграму прецедентів та класів.

Разом з цим було розроблено та проаналізовано геймплей, 3D-моделі та керування. На наступному етапі створення гри був розроблений програмний код та розглянуто деякі компоненти з нього. Також було розглянуто деякі алгоритми гри разом із програмним кодом.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ

3.1. Порядок встановлення та налаштування системи

Гра у жанрі «Survival» під назвою «Survive The War» це гра яка була створена у рушії Unity. Після релізу гри, її можна випускати на будь-який девайс, що відповідає цільовій операційній системі збірки, адже рушій має широку підтримку багатьох платформ і гра була побудована таким чином, щоб її можна було з мінімальними зусиллями портувати на інші системи. Оригінальний білд було виконано під Windows, тому, гра комфортно запуститься на будь-якій системі, конфігурація якої задовольняє наступним мінімальним вимогам:

- Процесор з тактовою частотою 2.4 ГГц або вище;
- 4 ГБ ОЗП;
- Графічний пристрій з підтримкою DX 11 та 4 ГБ відеопам'яті;
- 1 гб вільного місця на жорсткому диску;

Таким чином, якщо дотримуватись мінімальним вимогам, гра матиме достатню частоту кадрів(60 fps).

3.2. Інтерфейс та порядок роботи з програмним додатком

Після початкового запуску гри, користувач опиняється у головному меню (рисунок 3.1).

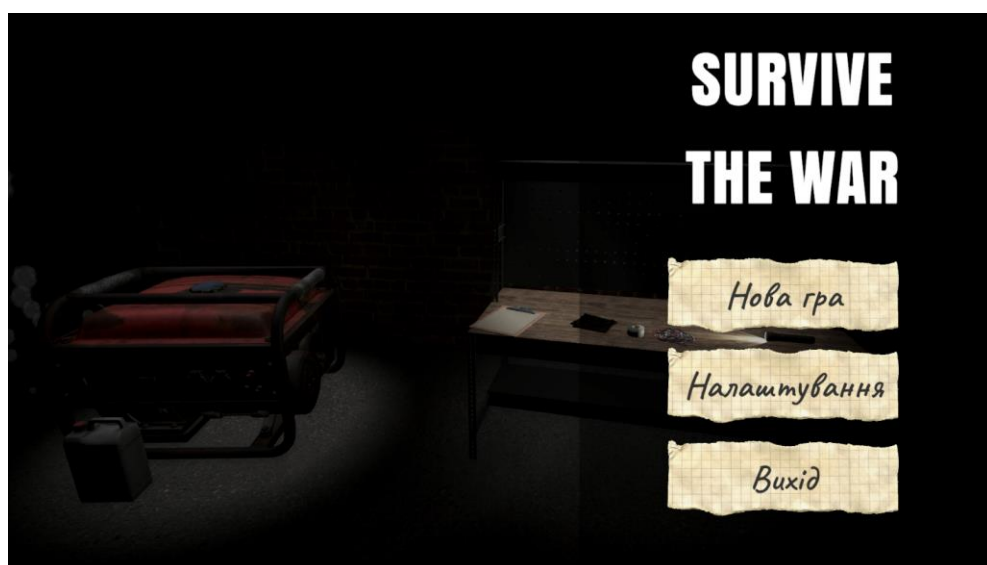


Рисунок 3.1 – Головне меню

Головне меню є анімованим та стилізованим для приємного вигляду. Для меню була скомпонована невелика 3D-сцена у якій симулюється цикл дня, і коли наступає ніч, включається генератор та запалюється світло. Також присутній легкий «шум» камери при якому камера ніби повільно та з маленькою амплітудою трясеться.

У головному меню(див. рисунок 3.1) гравець може почати гру, зайти у налаштування гри та налаштувати необхідні параметри і вийти з гри. Натиснувши на налаштування, камера плавно переходить у іншу позицію та з'являється меню налаштувань(рисунок 3.2).

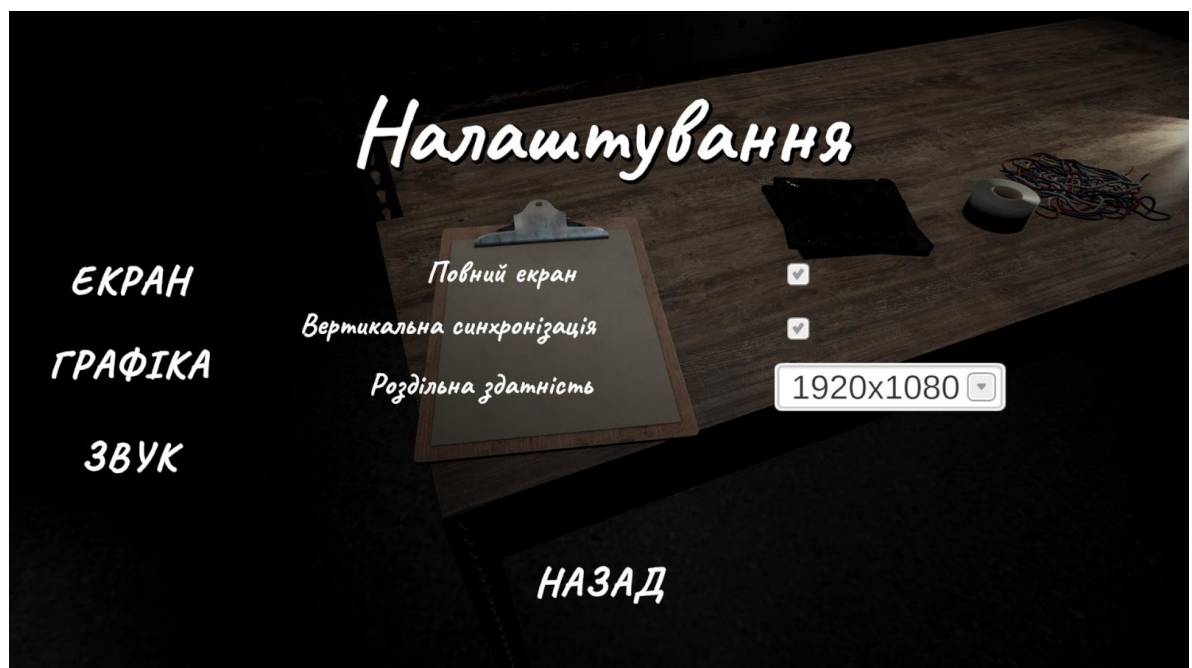


Рисунок 3.2 – Меню налаштувань

У меню налаштувань є 3 пункти:

- Налаштування екрану(див. Рисунок 3.2), де гравець зможе налаштувати роздільну здатність, увімкнення вертикальної синхронізації та повного екрану;
- Налаштування графіки(рисунок 3.3), де можна налаштувати якість графіки та згладжування;
- Налаштування звуку(рисунок 3.4), де можна налаштувати загальну гучність гри;

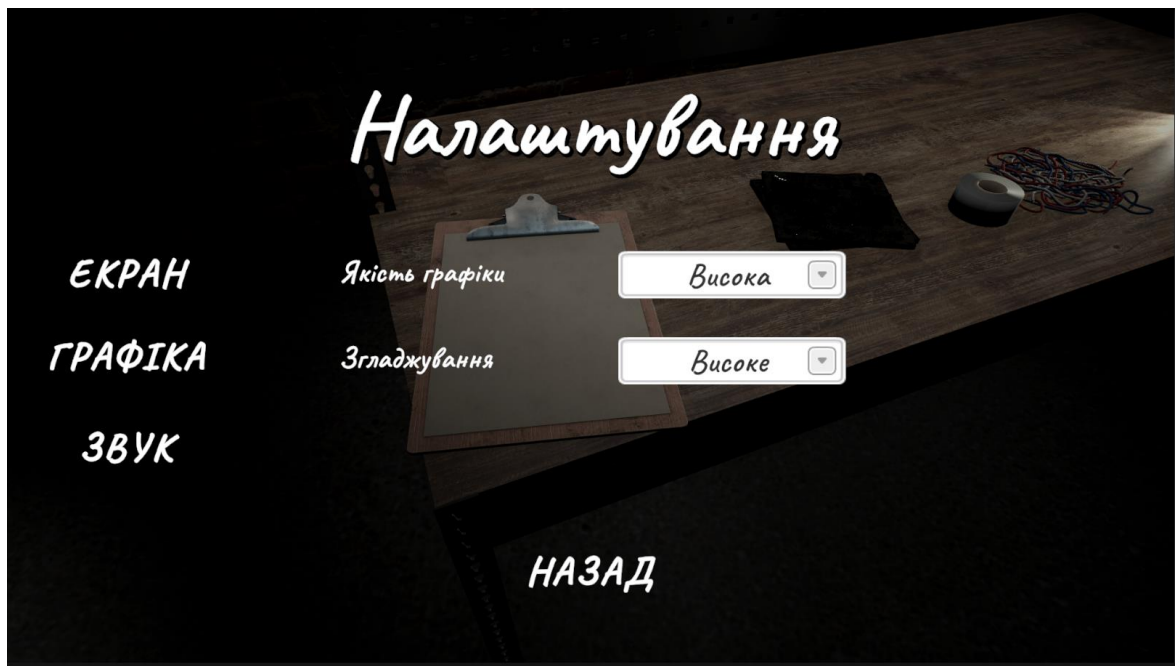


Рисунок 3.3 – Налаштування графіки

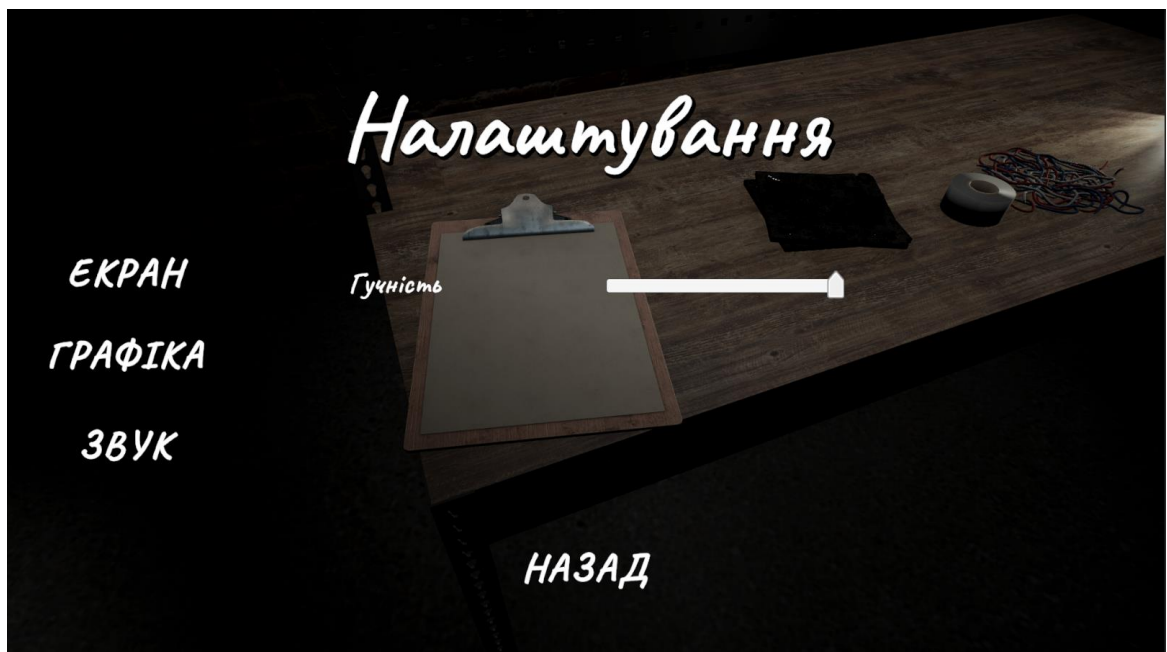


Рисунок 3.4 – Налаштування звуку

Після того як гравець налаштував гру, він може натиснути у головному меню «Нова гра» аби почати гру.

Інтерфейс гравця(рисунок 3.10) складається із таких компонентів:

- Полоска здоров'я;



Рисунок 3.5 Полоска здоров'я персонажа

- Полоска глузду;



Рисунок 3.6 Полоска глузду персонажа

- Полоска голоду;



Рисунок 3.7 Полоска голоду персонажа

- Полоска спраги;



Рисунок 3.8 Полоска спраги персонажа

- Полоска витривалості;

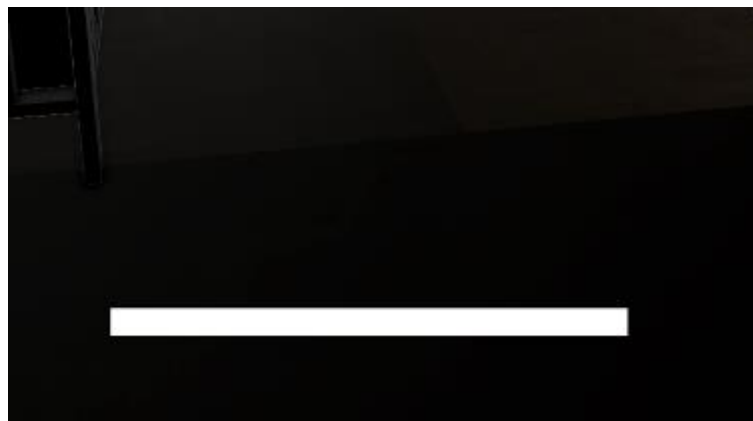


Рисунок 3.9 Полоска витривалості персонажа

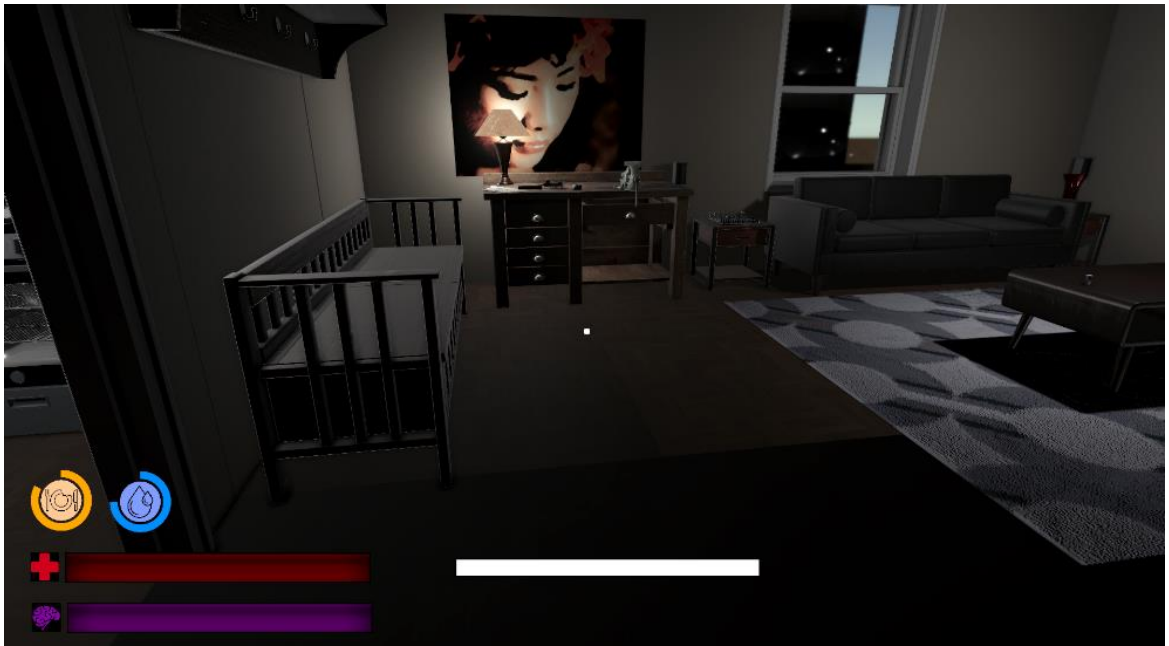


Рисунок 3.10 – Інтерфейс гравця

Після початку гри, герой з'являється у себе в кімнаті яка зображена на рисунку 3.10). Як тільки гра розпочинається у гравця одразу починають рости потреби(див. рисунок 3.10). Задовольняти їх, та своє здоров'я він може з'ївши або випивши їжу та застосувавши аптечку. У гравця також є інвентар який зображений на рисунку 3.11.

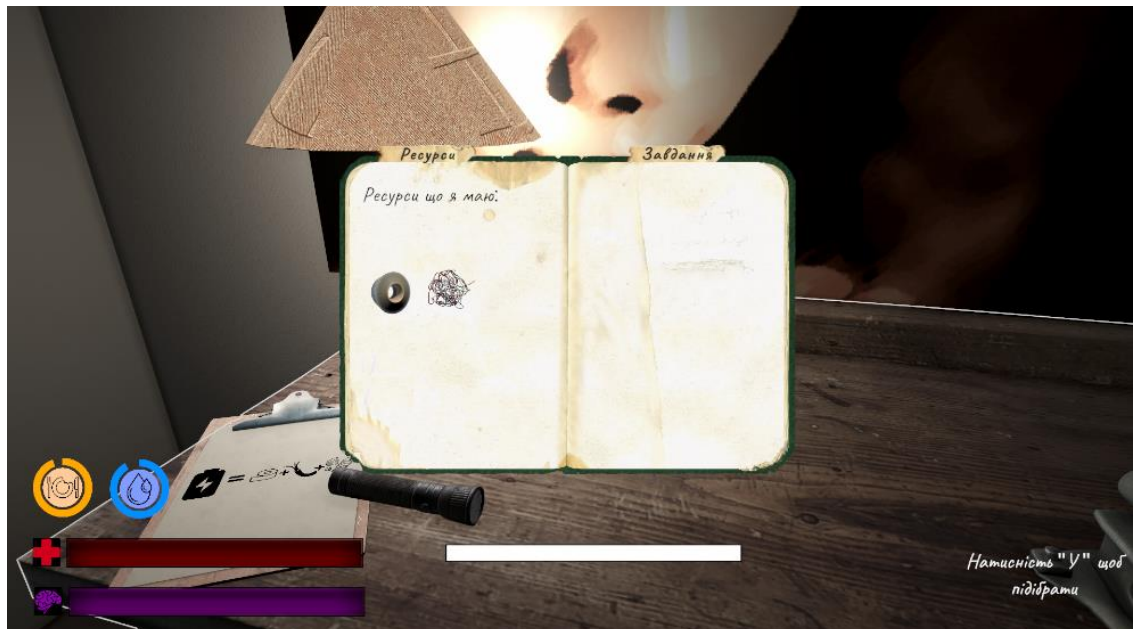


Рисунок 3.11 – Інтерфейс інвентаря гравця

У інвентарі є дві вкладинки, «ресурси»(див. рисунок 3.11) та «завдання»(рисунок 3.12).

					КН.КР.Б – 122 – 23 – ПЗ	55
Зм.	Арк.	№ докум.	Підпис	Дата		



Рисунок 3.12 – Вкладка «завдання»

У вкладці «ресурси», будуть відображатись всі предмети що гравець підняв, а у вкладці «завдання» кожного дня будуть видаватись різні завдання. Якщо натиснути по назві завдання, можна прочитати опис та проглянути те, що потрібно зробити(рисунок 3.13).



Рисунок 3.13 – Опис та завдання

З предметами у інвентарі можна взаємодіяти, якщо предмет можна взяти до рук, наприклад, ліхтарик, з'явиться додаткова кнопка(рисунок 3.14), якщо ж ні, тоді можна викинути з інвентаря(див. рисунок 3.14).



Рисунок 3.14 – Взаємодія з предметами у інвентарі

На планшеті(див. рисунок 3.14.) можна змінювати рецепти, на предмети які гравець хоче створити, там, у вигляді іконок, буде показано які предмети потрібні для створення того чи іншого предмету.

3.3. Тестування роботи програмного комплексу

Тестування програмного забезпечення є процесом перевірки і оцінки програмного продукту з метою виявлення дефектів, помилок або недоліків в ньому. Це важлива фаза в життєвому циклі розробки програмного забезпечення, яка допомагає забезпечити якість та надійність програми перед його випуском.

Для гри буде проведено функціональне тестування модулів. Для функціонального тестування у даній ситуації є доречним використання тест кейсів.

Тест кейси (Test Cases) є документованими інструкціями, що описують конкретні кроки, умови і очікувані результати для виконання тестування певної функціональності або властивості програмного продукту[20].

Першими модулями були протестовані модулі графічного інтерфейсу користувача та геймплею. Результати тестування були записані у вигляді таблиць 3.1, 3.2 та 3.3.

Таблиця 3.1

Тест кейс №1

Назва	Перевірка кнопки «Нова гра» у головному меню
Опис	Перевіряється правильність опрацювання кнопки при натисненні на неї.
Кроки	1. Запустити гру. 2. Натиснути на кнопку «Нова гра»
Очікуваний результат	Екран затемниться та завантажиться рівень гри.
Результат	Екран затемнився та завантажився рівень гри.

Таблиця 3.2

Тест кейс №2

Назва	Перевірка кнопки «Налаштування» у головному меню
Опис	Перевіряється правильність опрацювання кнопки при натисненні на неї.
Кроки	1. Запустити гру. 2. Натиснути на кнопку «Налаштування»
Очікуваний результат	Камера плавно перенесеться до столу та з'явиться меню налаштувань.
Результат	Камера плавно перенеслась до столу та з'явилося меню налаштувань.

Таблиця 3.3

Тест кейс №3

Назва	Налаштування гри
Опис	Перевіряється функціонал налаштування гри.

Кроки	1. Запустити гру. 2. Натиснути на кнопку «Налаштування». 3. Обрати довільний пункт налаштувань. 4. Змінити налаштування.
Очікуваний результат	Змінені налаштування застосовуються до гри.
Результат	Змінені налаштування застосувались до гри.

Було проведено тестування геймплейних особливостей, включаючи основні механіки – збільшення потреб персонажа, споживання продуктів для зменшення потреб. Взаємодія із інвентарем та крафтинг.

Під час тестування доступного функціоналу у нього не було виявлено критичних недоліків чи помилок у роботі.

Висновки до розділу 3

В даному розділі було наведено порядок встановлення та налаштування додатку, визначені системні вимоги для гри. Розглянуто та протестовано головні компоненти інтерфейсу користувача.

Опис роботи включає в себе розробку текстової та ілюстраційної частини які демонструють зовнішній вигляд закінченої гри. В грі реалізований інтерфейс головного меню, меню налаштувань та користувацький інтерфейс.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було спроектовано та створено гру від першої особи в жанрі виживання, яка повністю відповідає поставленим на початку проектування вимогам.

Було розглянуто та проаналізовано аналогічні ігри. При аналізі розглянуто основні елементи гри, їх геймплей та тематику. Розібрано основні елементи та характеристики жанру. Отримано теоретичний фундамент для подальшої роботи і сформовано уявлення про гру, що створюється.

Також були розглянуті засоби розробки. В ході аналізу було проведено їх порівняння та підібрано засоби реалізації. Технології, програмне забезпечення та інші інструменти є обґрунтованими.

Наступним кроком було проектування основних процесів системи, діаграм класів, діаграми варіантів використання. Описано деякі з важливих скриптів, їх методи та патерни проектування які використовувались при проектуванні класів.

Описано дизайн документ, у якому було розглянуто геймплей, сетинг, цільову аудиторію. Наведені моделі будинків та предметів на локації. Описано та розглянуто керування персонажем, при розробці керування використовувалась нова система вводу Unity, яка є зручнішою і спрощує розробку проекту. Далі були реалізовані та розглянуті алгоритми гри та описано декілька основних систем гри, спроектовані діаграми активностей.

Реалізована система містить інтерфейси головного меню, меню налаштувань, інвентаря та завдань.

Також проведено функціональне і геймплейне тестування додатку. Критичних помилок не було виявлено. Система працює коректно та виконує поставлені задачі.

Реалізована гра готова до використання. У подальшому гра може бути розширена шляхом впровадження нових механік, функцій та розширення світу (локації).

					КН.КР.Б – 122 – 23 – ПЗ	60
Зм.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The History of Survival Games - From Archaic to ARK. *Guided*. URL: <https://guided.news/en/gaming/history-of-survival-games/> (дата звернення: 24.06.2023).
2. Top 25 Games. *Tableau Public*. URL: <https://public.tableau.com/app/profile/stuart.townley/viz/Top25Games/Dashboard1#1>
3. This War of Mine. Final Cut. 11 bit studios. URL: https://store.steampowered.com/app/282070/This_War_of_Mine/.
4. Don't Starve. Klei Entertainment. URL: https://store.steampowered.com/app/219740/Dont_Starve/.
5. Don't Starve. *Metacritic*. URL: <https://www.metacritic.com/game/pc/dont-starve> (дата звернення: 24.06.2023).
6. DayZ. Bohemia Interactive. URL: <https://store.steampowered.com/app/221100/DayZ/>.
7. Hernandez R. D. The Model View Controller Pattern – MVC Architecture and Frameworks Explained. *freeCodeCamp.org*. URL: <https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/> (дата звернення: 24.06.2023).
8. ECS for Unity. *Unity*. URL: <https://unity.com/ecs> (дата звернення: 24.06.2023).
9. Difference Between MVC, MVP and MVVM Architecture Pattern in Android - GeeksforGeeks. *GeeksforGeeks*. URL: <https://www.geeksforgeeks.org/difference-between-mvc-mvp-and-mvvm-architecture-pattern-in-android/> (дата звернення: 24.06.2023).
10. Спостерігач. *Refactoring and Design Patterns*. URL: <https://refactoring.guru/uk/design-patterns/observer> (дата звернення: 24.06.2023).
11. Стан. *Refactoring and Design Patterns*. URL: <https://refactoring.guru/uk/design-patterns/state> (дата звернення: 24.06.2023).
12. Одинак. *Refactoring and Design Patterns*. URL: <https://refactoring.guru/uk/design-patterns/singleton> (дата звернення: 24.06.2023).

					КН.КР.Б – 122 – 23 – ПЗ	
Зм.	Арк.	№ докум.	Підпис	Дата		61

13. Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine. *Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine*. URL: <https://unity.com/> (дата звернення: 24.06.2023).

14. Unity - Manual: Unity User Manual 2021.3 (LTS). *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/Manual/index.html> (дата звернення: 24.06.2023).

15. blender.org - Home of the Blender project - Free and Open 3D Creation Software. *blender.org*. URL: <https://www.blender.org/> (дата звернення: 24.06.2023).

16. Home. *Audacity* ®. URL: <https://www.audacityteam.org/> (date of access: 24.06.2023).

17. Застосування UML в дипломних роботах :: Державний університет телекомунікацій. *Головна :: Державний університет телекомунікацій*. URL: <https://dut.edu.ua/ua/news-1-626-7758-zastosuvannya-uml-v-diplomnih-robotah-kafedra-kompyuternih-nauk-ta-informaciy-nih-tehnologiy> (дата звернення: 24.06.2023).

18. Діаграми UML для моделювання процесів і архітектури проекту. *Evergreen - web розробка і діджиталізація бізнесу за допомогою AI продуктів*. URL: <https://evergreens.com.ua/ua/articles/uml-diagrams.html> (дата звернення: 24.06.2023).

19. Як написати геймдизайн-документ: структура, поради та популярні помилки • VOKI Games. *Voki Games*. URL: <https://vokigames.com/ua/yak-napisati-gejmdizajn-dokument-struktura-poradi-ta-populyarni-pomilki/> (дата звернення: 24.06.2023).

20. Правила оформлення теми тест-кейса | Безкоштовний онлайн-курс від компанії QATestLab. *Безкоштовний онлайн-курс від компанії QATestLab | Головна сторінка*. URL: <https://training.qatestlab.com/blog/course-materials/test-case-topic/> (дата звернення: 24.06.2023).

					КН.КР.Б – 122 – 23 – ПЗ	62
Зм.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

					КН.КР.Б – 122 – 23 – ПЗ	
Зм.	Арк.	№ докум.	Підпис	Дата		63

StateMachine.cs:

```

using System;

namespace OM
{
    [Serializable]
    public class StateMachine
    {
        public IState CurrentState { get; private set; }

        public ResumedState resumedState;
        public PauseState pauseState;
        public GameOverState gameOverState;

        public StateMachine(GameStateView gameStateView)
        {
            resumedState = new ResumedState(gameStateView);
            pauseState = new PauseState(gameStateView);
            gameOverState = new GameOverState(gameStateView);
        }

        public void Initialize(IState startingState)
        {
            CurrentState = startingState;
            startingState.Enter();
        }

        public void TransitionTo(IState nextState)
        {
            CurrentState.Exit();
            CurrentState = nextState;
            nextState.Enter();
        }

        public void Update()
        {
            if (CurrentState != null)
            {
                CurrentState.Update();
            }
        }
    }
}

```

IState:

```

using System;

namespace OM
{
    [Serializable]
    public class StateMachine
    {
        public IState CurrentState { get; private set; }

        public ResumedState resumedState;
        public PauseState pauseState;
        public GameOverState gameOverState;
    }
}

```

```

public StateMachine(GameStateView gameStateView)
{
    resumedState = new ResumedState(gameStateView);
    pauseState = new PauseState(gameStateView);
    gameOverState = new GameOverState(gameStateView);
}

public void Initialize(IState startingState)
{
    CurrentState = startingState;
    startingState.Enter();
}

public void TransitionTo(IState nextState)
{
    CurrentState.Exit();
    CurrentState = nextState;
    nextState.Enter();
}

public void Update()
{
    if (CurrentState != null)
    {
        CurrentState.Update();
    }
}
}

```

PauseState.cs:

```

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

namespace OM
{
    public class PauseState : IState
    {
        public delegate void PausedStateAction(bool isPaused);
        public static event PausedStateAction OnGamePaused;

        private bool isExitButtonPressed;

        private GameStateView _gameStateView;

        public PauseState(GameStateView gameStateView)
        {
            _gameStateView = gameStateView;
        }

        public void Enter()
        {
            OnGamePaused?.Invoke(true);

            _gameStateView.resumeGameButton.onClick.AddListener(delegate
            {
                _gameStateView.isPauseButtonPressed = false;
            });

            _gameStateView.exitButton.onClick.AddListener(delegate
            {

```

```

        isExitButtonPressed = true;
    });

    _gameStateView.pauseMenuUI.SetActive(true);
    OnPause();
}

public void Update()
{
    if (!_gameStateView.isPauseButtonPressed)
    {
        _gameStateView.StateMachine.TransitionTo(_gameStateView.StateMachine.resumedState);
    }
    else if (isExitButtonPressed)
    {
        _gameStateView.StateMachine.TransitionTo(_gameStateView.StateMachine.resumedState);
        _gameStateView.LoadMainMenuScene();
    }
}

public void Exit()
{
    OnGamePaused?.Invoke(false);

    _gameStateView.optionsMenu.SetActive(false);

    _gameStateView.pauseMenuUI.SetActive(false);

    _gameStateView.isPauseButtonPressed = false;

    isExitButtonPressed = false;

    OnUnpause();
}

public void OnPause()
{
    Cursor.lockState = CursorLockMode.None;

    Time.timeScale = 0f;

    AudioListener.pause = true;
}

public void OnUnpause()
{
    Cursor.lockState = CursorLockMode.Locked;

    Time.timeScale = 1f;

    AudioListener.pause = false;
}
}
}

```

PlayerNeeds.cs:

```

using StarterAssets;
using UnityEngine;
using UnityEngine.InputSystem;

namespace OM

```

					КН.КР.Б – 122 – 23 – ПЗ	66
Зм.	Арк.	№ докум.	Підпис	Дата		

```

{
    public class PlayerNeeds : MonoBehaviour
    {
        [SerializeField] private int starvingDamagePerTick = 5;
        [SerializeField] private int onFullNeedsHealPoints = 10;

        #region Hunger

        [Header("Hunger")]
        [SerializeField] private float maxHunger = 100f;

        [SerializeField] private float decreaseHungerRate = 1f;
        [SerializeField] private float currentHunger;

        [SerializeField, Tooltip("Used to simulate short time when player is not hungry")]
        private int bonusSatedPoints = 10;
        public float HungerPercent => currentHunger / maxHunger;

        #endregion

        #region Thirst
        [Header("Thirst")]
        [SerializeField] private float maxThirst = 100f;

        [SerializeField] private float decreaseThirstRate = 1f;
        [SerializeField] private float currentThirst;
        public float ThirstPercent => currentThirst / maxThirst;
        #endregion

        #region Sanity
        [Header("Sanity")]
        [SerializeField] private float maxSanity = 100f;

        [SerializeField] private float decreaseSanityRate = 1f;
        [SerializeField] private float hightDecreaseSanityRate = 5f;
        [SerializeField] private float currentSanity;

        [SerializeField] private int sanityDamage = 10;

        [SerializeField] public int lightPoints = 0;
        public float SanityPercent => currentSanity / maxSanity;

        #endregion

        #region Stamina
        [Header("Stamina")]
        [SerializeField] private float maxStamina = 100f;

        [SerializeField] private float decreaseStaminaRate = 1f;

        [SerializeField] private float rechargeStaminaRate = 2f;
        [SerializeField] private float rechargeStaminaDelay = 1f;

        [SerializeField] private float currentStamina;
        [SerializeField] private float currentStaminaDelayCounter;

        public float StaminaPercent => currentStamina / maxStamina;
        #endregion

        [SerializeField] private DayNightCycle dayNightCycle;

        public delegate void CharacterStaminaAction(bool HasStamina);
        public static event CharacterStaminaAction OnExhausted;
    }
}

```

```

public delegate void CharacterNeedsAction(
    float Hunger,
    float Thirst,
    float Stamina);

public static event CharacterNeedsAction OnNeedsChanged;

public delegate void CharacterSanityAction(float maxSanity, float
currentSanity);
public static event CharacterSanityAction OnSanityChanged;

[Header("Player Refernces")]
private StarterAssetsInputs playerInputs;
private PlayerHealth playerHealth;

private void OnEnable()
{
    ObjectEatable.OneItemConsuming += AddHungerAndThirst;
}

private void OnDisable()
{
    ObjectEatable.OneItemConsuming -= AddHungerAndThirst;
}

private void Start()
{
    currentThirst = maxThirst;
    currentHunger = maxHunger;
    currentStamina = maxStamina;
    currentSanity = maxSanity;

    playerInputs = GetComponent<StarterAssetsInputs>();
    playerHealth = GetComponent<PlayerHealth>();
}

private void Update()
{
    HandleStarvingAndThirst();
    UpdateStamina();
    HandleSanity();

    OnNeedsChanged?.Invoke(HungerPercent, ThirstPercent, StaminaPercent);
}

private void HandleSanity()
{
    SanityStatus();

    if (!dayNightCycle.IsNight())
    {
        return;
    }

    if (lightPoints <= 0)
    {
        ReduceSanity(hightDecreaseSanityRate);
    }

    else if (lightPoints < 100)
    {
        ReduceSanity(decreaseSanityRate);
    }
}

```

```

    }
    OnSanityChanged?.Invoke(maxSanity, currentSanity);
    //Debug.Log("Sanity: " + currentSanity);
}

private void SanityStatus()
{
    if (currentSanity <= 0)
    {
        // Damage player for 0 sanity
        playerHealth.health.DamageEntity(sanityDamage);
        currentSanity = Mathf.Clamp(currentSanity, 0f, maxStamina);
        OnSanityChanged?.Invoke(maxSanity, currentSanity);
    }
}

private void HandleStarvingAndThirst()
{
    currentHunger -= decreaseHungerRate * Time.deltaTime;
    currentThirst -= decreaseThirstRate * Time.deltaTime;

    if (currentHunger <= 0 || currentThirst <= 0)
    {
        playerHealth.health.DamageEntity(starvingDamagePerTick);

        currentHunger = Mathf.Clamp(currentHunger, 0f, maxHunger);
        currentThirst = Mathf.Clamp(currentThirst, 0f, maxThirst);
    }

    if (currentHunger >= maxHunger && currentThirst >= maxThirst)
    {
        playerHealth.health.HealEntity(onFullNeedsHealPoints);
    }
}

private void UpdateStamina()
{
    currentStamina = Mathf.Clamp(currentStamina, 0f, maxStamina);

    if (PlayerSprinting())
    {
        HandleSprinting();
    }
    else if (currentStamina < maxStamina)
    {
        HandleStaminaRecharging();
    }
}

private bool PlayerSprinting()
{
    return playerInputs.IsSprinting() && playerInputs.GetMove().y > 0.1f;
}

private void HandleSprinting()
{
    currentStamina -= decreaseStaminaRate * Time.deltaTime;
    if (currentStamina <= 0)
    {
        OnExhausted(false);
        currentStamina = 0;
    }
    currentStaminaDelayCounter = 0f;
}

```

```

private void HandleStaminaRecharging()
{
    if (currentStaminaDelayCounter < rechargeStaminaDelay)
    {
        currentStaminaDelayCounter += Time.deltaTime;
    }
    else
    {
        currentStamina += rechargeStaminaRate * Time.deltaTime;
    }
}

private void AddHungerAndThirst(float hungerAmount, float thirstAmount)
{
    currentHunger += hungerAmount;
    currentThirst += thirstAmount;

    currentHunger = Mathf.Clamp(currentHunger, 0f, maxHunger +
bonusSatedPoints);
    currentThirst = Mathf.Clamp(currentThirst, 0f, maxThirst +
bonusSatedPoints);
}

public void ReduceSanity(float valueAmount)
{
    currentSanity -= valueAmount * Time.deltaTime;
}
}
}

```

PlayerInteraction.cs:

```

using TMPPro;
using UnityEngine;
using UnityEngine.InputSystem;

namespace OM
{
    public class PlayerInteraction : MonoBehaviour
    {
        [Header("Interaction settings")]
        [SerializeField, Tooltip("Distance for player interaction")]
        private float interactDistance = 4f;
        private int ineteractLayerMask = 1 << 10;
        [SerializeField] private GameObject hintPanel;
        [SerializeField] private TextMeshProUGUI hintText;

        private PlayerInput playerInput;

        // for storing selectable interface
        private Transform selection;

        private void Awake()
        {
            playerInput = GetComponent<PlayerInput>();
        }

        void Update()
        {
            HandleSelection();
        }

        private void HandleSelection()
        {
            // if seleacttion was setted
            if (selection != null)

```

```

    {
        if (selection.TryGetComponent(out ISelectable interactable))
        {
            interactable.Dehighlight();
        }
        // set selection to null so we don't deselect it
        selection = null;
    }

    RaycastHit hit;
    Ray ray = FireRay();
    var pickUpKey = playerInput.actions["Pickup"].GetBindingDisplayString();
    var interactKey =
playerInput.actions["Interact"].GetBindingDisplayString();

    if (Physics.Raycast(ray, out hit, interactDistance, ineteractLayerMask))
    {
        // get a hit transform
        Transform selected = hit.transform;
        if (hit.transform.TryGetComponent(out ISelectable selectable))
        {
            selectable.Highlight();
        }
        if (hit.transform.TryGetComponent(out IInteractable interactable))
        {
            if (interactable.IsPickable)
            {
                hintText.text = $"Натисніть \"{pickUpKey}\" щоб підібрати";
                hintPanel.SetActive(true);
            }
            else
            {
                hintText.text = $"Натисніть \"{interactKey}\" для
взаємодії";
                hintPanel.SetActive(true);
            }
        }

        // set selection tranform to hit interface transofm
        selection = selected;
    }
    else
    {
        hintPanel.SetActive(false);
    }
}

private static Ray FireRay()
{
    return Camera.main.ScreenPointToRay(new Vector3(Screen.width / 2f,
Screen.height / 2f, 0f));
}

public void Interact(InputAction.CallbackContext context)
{
    if (context.performed)
    {
        RaycastHit hit;
        Ray ray = FireRay();
        if (Physics.Raycast(ray, out hit, interactDistance,
ineteractLayerMask))
        {
            if (hit.transform.TryGetComponent(out IInteractable interactable))
            {
                interactable.Interact();
            }
        }
    }
}

```

```

    }
    }
    }
    }
}

```

InventorySystem.cs:

```

using System.Collections.Generic;
using UnityEngine;

namespace OM
{
    public class InventorySystem
    {
        public int slotsLimit = 6;
        public Dictionary<ItemSO, InventoryItem> ItemDictionary;
        public List<InventoryItem> Inventory { get; set; }

        public delegate void InventoryAction();
        public static event InventoryAction OnInventoryChange;

        public InventoryItem Get(ItemSO item)
        {
            if (ItemDictionary.TryGetValue(item, out InventoryItem inventoryItem))
            {
                return inventoryItem;
            }
            return null;
        }

        public void Add(ItemSO referenceData)
        {
            if (ItemDictionary.TryGetValue(referenceData, out InventoryItem value))
            {
                value.AddToStack();
            }
            else
            {
                CreateNewItemCell(referenceData);
            }
            SoundManager.Instance.PlaySound("PickupItem");
            OnInventoryChange();
        }

        private void CreateNewItemCell(ItemSO referenceData)
        {
            InventoryItem newItem = new InventoryItem(referenceData);
            Inventory.Add(newItem);
            ItemDictionary.Add(referenceData, newItem);
        }

        public void Remove(ItemSO referenceData)
        {
            if (ItemDictionary.TryGetValue(referenceData, out InventoryItem value))
            {
                if (value.StackSize == 1)
                {
                    Inventory.Remove(value);
                    ItemDictionary.Remove(referenceData);
                }
                value.RemoveFromStack();
            }
            OnInventoryChange();
        }
    }
}

```

```

    }
}

```

InventoryMediator.cs:

```

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;

namespace OM
{
    public class InventoryMediator : MonoBehaviour
    {
        [SerializeField] private InventoryView inventoryView;
        [SerializeField] private TaskWindowBuilder taskWindow;
        [SerializeField] private GameObject inventoryMenu;
        [SerializeField] private Transform playerDropPoint;

        public InventorySystem InventorySystem = new InventorySystem();

        void OnEnable()
        {
            InventorySystem.OnInventoryChange += UpdateInventory;
            PlayerPickupDrop.OnItemPickUp += AddItemToInventory;
        }

        void OnDisable()
        {
            InventorySystem.OnInventoryChange -= UpdateInventory;
            PlayerPickupDrop.OnItemPickUp -= AddItemToInventory;
        }

        private void Awake()
        {
            InventorySystem.Inventory = new List<InventoryItem>();
            InventorySystem.ItemDictionary = new Dictionary<ItemSO, InventoryItem>();
        }

        public void UpdateInventory()
        {
            inventoryView.OnUpdateInventory();
            foreach (InventoryItem item in InventorySystem.Inventory)
            {
                Debug.Log(item.itemData.itemName);
            }
        }

        public bool IsInventoryFull(ItemSO item)
        {
            if (InventorySystem.Get(item) == null)
            {
                if (InventorySystem.Inventory.Count == InventorySystem.slotsLimit)
                {
                    SoundManager.Instance.PlaySound("FullInventory");
                    return true;
                }
            }
            return false;
        }

        public void AddItemToInventory(ItemSO item)
        {
            InventorySystem.Add(item);
            EvaluateGatheringGoal(item.itemName);
        }
    }
}

```

```

public void RemoveItemFromInventory(ItemSO item)
{
    InventorySystem.Remove(item);
}

public void RemoveItemAndDrop(ItemSO item)
{
    if(InventorySystem.Get(item) != null)
    {
        InventorySystem.Remove(item);
    }
    GameObject dropItem = Instantiate(item.Prefab,
        new Vector3(playerDropPoint.position.x, playerDropPoint.position.y,
playerDropPoint.position.z),
        Quaternion.identity);
}

public void InventoryVisibility(InputAction.CallbackContext ctx)
{
    if (ctx.performed)
    {
        inventoryMenu.SetActive(!isInventoryMenuActive());
        if (isInventoryMenuActive())
        {
            Cursor.lockState = CursorLockMode.None;
            SoundManager.Instance.PlaySound("OpenInventory");
        }
        else
        {
            Cursor.lockState = CursorLockMode.Locked;
            taskWindow.CloseWindow();
        }
    }
}

private bool isInventoryMenuActive()
{
    return inventoryMenu.activeSelf;
}

public void EvaluateGatheringGoal(string itemName)
{
    EventManager.Instance.QueueEvent(new GatheringGameEvent(itemName));
}
}
}

```

DayNightCycle.cs:

```

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

public class DayNightCycle : MonoBehaviour {

    [Header("Time")]
    [Tooltip("Day Length in Minutes")]
    [SerializeField]
    private float _targetDayLength = 0.5f; //length of day in minutes

    [SerializeField] private float dayStart = 8f;
    public float targetDayLength
    {
        get
        {

```

					КН.КР.Б – 122 – 23 – ПЗ	74
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        return _targetDayLength;
    }
}
[SerializeField]

private float elapsedTime;
[SerializeField]
private bool use24Clock = true;
//[SerializeField]
//private Text clockText;
[SerializeField]
[Range(0f, 1f)]
private float _timeOfDay;
public float timeOfDay
{
    get
    {
        return _timeOfDay;
    }
}

[SerializeField]
private int _dayNumber = 0; //tracks the days passed
public int dayNumber
{
    get
    {
        return _dayNumber;
    }
}

[SerializeField]
private int _yearNumber = 0;
public int yearNumber
{
    get
    {
        return _yearNumber;
    }
}

private float _timeScale = 100f;
[SerializeField]
private int _yearLength = 100;
public float yearLength
{
    get
    {
        return _yearLength;
    }
}

public bool pause = false;
[SerializeField]
private AnimationCurve timeCurve;
private float timeCurveNormalization;

[Header("Sun Light")]
[SerializeField]
private Transform dailyRotation;
[SerializeField]
private Light sun;
private float intensity;
[SerializeField]
private float sunBaseIntensity = 1f;
[SerializeField]
private float sunVariation = 1.5f;
[SerializeField]

```

```

private Gradient sunColor;

[Header("Seasonal Variables")]
[SerializeField]
private Transform sunSeasonalRotation;
[SerializeField]
[Range(-45f, 45f)]
private float maxSeasonalTilt;

[Header("Modules")]
private List<DNModuleBase> moduleList = new List<DNModuleBase>();

[Header("Quest View")]
[SerializeField] private TaskView taskView;

public delegate void DayNightAction();
public static event DayNightAction OnNewHour;
public static event DayNightAction OnNewDay;

private int previousHour = 0;

private void Start()
{
    NormalTimeCurve();
    SetDayTime(dayStart);
}

private void Update()
{
    if (!pause)
    {
        UpdateTimeScale();
        UpdateTime();
        UpdateClock();
    }
    AdjustSunRotation();
    SunIntensity();
    AdjustSunColor();
    UpdateModules(); //will update modules each frame
}

private void UpdateTimeScale()
{
    _timeScale = 24 / (_targetDayLength / 60);
    _timeScale *= timeCurve.Evaluate(elapsedTime / (targetDayLength * 60));
    //changes timescale based on time curve
    _timeScale /= timeCurveNormalization; //keeps day length at target value
}

private void NormalTimeCurve()
{
    float stepSize = 0.01f;
    int numberSteps = Mathf.FloorToInt(1f / stepSize);
    float curveTotal = 0;

    for (int i = 0; i < numberSteps; i++)
    {
        curveTotal += timeCurve.Evaluate(i * stepSize);
    }

    timeCurveNormalization = curveTotal / numberSteps; //keeps day length at
    target value
}

```

```

private void UpdateTime()
{
    _timeOfDay += Time.deltaTime * _timeScale / 86400; // seconds in a day
    elapsedTime += Time.deltaTime;

    CheckNewHour();

    if(_timeOfDay > 1) //new day!!
    {
        elapsedTime = 0;
        _dayNumber++;
        _timeOfDay -= 1;

        OnNewDay?.Invoke();
        if (taskView != null)
        {
            taskView.GenerateDailyTasks(); // Generate new tasks
        }
        else
        {
            Debug.LogWarning(taskView + " is null");
        }

        if(_dayNumber > _yearLength) //new year!
        {
            _yearNumber++;
            _dayNumber = 0;
        }
    }
}

private void CheckNewHour()
{
    int currentHour = Mathf.FloorToInt(_timeOfDay * 24);
    if (currentHour != previousHour)
    {
        // Наступила нова година
        //Debug.Log("New Hour");
        Debug.Log(GetHour());
        OnNewHour?.Invoke();
        previousHour = currentHour;
    }
}

private void UpdateClock()
{
    float time = elapsedTime / (targetDayLength * 60);
    float hour = Mathf.FloorToInt(time * 24);
    float minute = Mathf.FloorToInt(((time * 24) - hour) * 60);

    string hourString;
    string minuteString;

    if (!use24Clock && hour > 12)
        hour -= 12;

    if (hour < 10)
        hourString = "0" + hour.ToString();
    else
        hourString = hour.ToString();

    if (minute < 10)
        minuteString = "0" + minute.ToString();
    else

```

```

        minuteString = minute.ToString();

        // if(use24Clock)
        //     clockText.text = hourString + " : " + minuteString;
        // else if (time > 0.5f)
        //     clockText.text = hourString + " : " + minuteString + " pm";
        // else
        //     clockText.text = hourString + " : " + minuteString + " am";
    }

    //rotates the sun daily (and seasonally soon too);
    private void AdjustSunRotation()
    {
        float sunAngle = timeOfDay * 360f;
        dailyRotation.transform.localRotation = Quaternion.Euler(new Vector3(0f, 0f,
sunAngle));

        float seasonalAngle = -maxSeasonalTilt * Mathf.Cos(dayNumber / yearLength * 2f
* Mathf.PI);
        sunSeasonalRotation.localRotation = Quaternion.Euler(new
Vector3(seasonalAngle, 0f, 0f));
    }

    private void SunIntensity()
    {
        intensity = Vector3.Dot(sun.transform.forward, Vector3.down);
        intensity = Mathf.Clamp01(intensity);

        sun.intensity = intensity * sunVariation + sunBaseIntensity;
    }

    private void AdjustSunColor()
    {
        sun.color = sunColor.Evaluate(intensity);
    }

    public void AddModule(DNModuleBase module)
    {
        moduleList.Add(module);
    }

    public void RemoveModule(DNModuleBase module)
    {
        moduleList.Remove(module);
    }

    //update each module based on current sun intensity
    private void UpdateModules()
    {
        foreach (DNModuleBase module in moduleList)
        {
            module.UpdateModule(intensity);
        }
    }

    private int GetHour()
    {
        float time = elapsedTime / (targetDayLength * 60);
        int hour = Mathf.FloorToInt(time * 24);
        return hour;
    }

    public bool IsNight()
    {

```

```

        if (GetHour() >= 18)
        {
            return true;
        }
        return false;
    }

    private void SetDayTime(float hour)
    {
        _timeOfDay = hour / 24f;
        elapsedTime = _timeOfDay * targetDayLength * 60f;
        AdjustSunRotation();
        SunIntensity();
        AdjustSunColor();
    }
}

```

Task.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using UnityEditor;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.Serialization;

namespace OM
{
    public class Task : ScriptableObject
    {
        [Serializable]
        public struct Info
        {
            public string name;
            public string description;
        }

        [Header("Info")] public Info information;

        [Serializable]
        public struct Reward
        {
            [FormerlySerializedAs("money")] public int amountOfItems;
        }

        [Header("Reward")] public Reward reward = new Reward { amountOfItems = 1 };

        public bool Completed { get; protected set; }
        public TaskCompletedEvent taskCompleted;

        public abstract class TaskGoal : ScriptableObject
        {
            protected string description;
            public int CurrentAmount { get; protected set; }
            public int requiredAmount = 1;

            public bool Completed { get; protected set; }
            [HideInInspector] public UnityEvent goalCompleted;

            public virtual string GetDescription()
            {
                return description;
            }
        }
    }
}

```

```

        public virtual void Initialize()
        {
            Completed = false;
            CurrentAmount = 0;
            goalCompleted = new UnityEvent();
        }

        protected void Evaluate()
        {
            if (CurrentAmount >= requiredAmount)
            {
                Complete();
            }
        }

        private void Complete()
        {
            Completed = true;
            goalCompleted.Invoke();
            goalCompleted.RemoveAllListeners();
        }
    }

    public List<TaskGoal> goals;

    public void Initialize()
    {
        Completed = false;
        taskCompleted = new TaskCompletedEvent();

        foreach (var goal in goals)
        {
            goal.Initialize();
            goal.goalCompleted.AddListener(delegate { CheckGoals(); });
        }
    }

    private void CheckGoals()
    {
        Completed = goals.All(g => g.Completed);
        if (Completed)
        {
            //Give reward

            GameManager.gameManager.taskRewardController.SpawnRandomReward(reward.amountOfItems);
            taskCompleted.Invoke(this);
            taskCompleted.RemoveAllListeners();
        }
    }

    public class TaskCompletedEvent : UnityEvent<Task> { }

    #if UNITY_EDITOR
    [CustomEditor(typeof(Task))]
    public class TaskEditor : Editor
    {

        SerializedProperty m_TaskInfoProperty;
        SerializedProperty m_TaskRewardProperty;

        List<string> m_TaskGoalType;
        SerializedProperty m_TaskGoalListProperty;

        [MenuItem("Assets/Create/ScriptableObjects/NewTask", priority = 0)]
    }
    #endif

```

```

public static void CreateTask()
{
    var newTask = CreateInstance<Task>();

    ProjectWindowUtil.CreateAsset(newTask, "NewTask.asset");
}

private void OnEnable()
{
    m_TaskInfoProperty =
serializedObject.FindProperty(nameof(Task.information));
    m_TaskRewardProperty = serializedObject.FindProperty(nameof(Task.reward));

    m_TaskGoalListProperty =
serializedObject.FindProperty(nameof(Task.goals));

    var lookup = typeof(Task.TaskGoal);
    m_TaskGoalType = AppDomain.CurrentDomain.GetAssemblies()
        .SelectMany(assembly => assembly.GetTypes())
        .Where(x => x.IsClass && !x.IsAbstract && x.IsSubclassOf(lookup))
        .Select(type => type.Name)
        .ToList();
}

public override void OnInspectorGUI()
{
    var child = m_TaskInfoProperty.Copy();
    var depth = child.depth;
    child.NextVisible(true);

    EditorGUILayout.LabelField("Task info", EditorStyles.boldLabel);
    while (child.depth > depth)
    {
        EditorGUILayout.PropertyField(child, true);
        child.NextVisible(false);
    }

    child = m_TaskRewardProperty.Copy();
    depth = child.depth;
    child.NextVisible(true);

    EditorGUILayout.LabelField("Task reward", EditorStyles.boldLabel);
    while (child.depth > depth)
    {
        EditorGUILayout.PropertyField(child, true);
        child.NextVisible(false);
    }

    int choice = EditorGUILayout.Popup("Add new task goal", -1,
m_TaskGoalType.ToArray());

    if (choice != -1)
    {
        var newInstance =
ScriptableObject.CreateInstance(m_TaskGoalType[choice]);

        AssetDatabase.AddObjectToAsset(newInstance, target);

        m_TaskGoalListProperty.InsertArrayElementAtIndex(m_TaskGoalListProperty.arraySize);
        m_TaskGoalListProperty.GetArrayElementAtIndex(m_TaskGoalListProperty.arraySize - 1)
            .objectReferenceValue = newInstance;
    }
}

```

```

    }

    Editor ed = null;
    int toDelete = -1;
    for (int i = 0; i < m_TaskGoalListProperty.arraySize; ++i)
    {
        EditorGUILayout.BeginHorizontal();
        EditorGUILayout.BeginVertical();
        var item = m_TaskGoalListProperty.GetArrayElementAtIndex(i);
        SerializedObject obj = new
SerializedObject(item.objectReferenceValue);

        CreateCachedEditor(item.objectReferenceValue, null, ref ed);

        ed.OnInspectorGUI();
        EditorGUILayout.EndVertical();

        if (GUILayout.Button("-", GUILayout.Width(32)))
        {
            toDelete = i;
        }

        EditorGUILayout.EndHorizontal();
    }

    if (toDelete != -1)
    {
        var item =
m_TaskGoalListProperty.GetArrayElementAtIndex(toDelete).objectReferenceValue;
        DestroyImmediate(item, true);

        m_TaskGoalListProperty.DeleteArrayElementAtIndex(toDelete);
        m_TaskGoalListProperty.DeleteArrayElementAtIndex(toDelete);
    }

    serializedObject.ApplyModifiedProperties();
}
}
#endif
}

```