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

Спеціальність: 123 «Комп'ютерна інженерія»

Освітня програма: «Комп'ютерна інженерія»

Група: 2БКС-27

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

здобувача освіти денної форми навчання

БКС.27.13.000.КРБ

КОВАЛЬЧУКА КОСТЯНТИНА ОЛЕКСІЙОВИЧА

м. Одеса
2023 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітня програма: **«Комп'ютерна інженерія»**

Група: **2БКС-27**

ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи бакалавра на тему:

«Створення ігрового проекту - квесту на платформі Unity: розробка ігрової механіки та левелдизайну»

Проектний матеріал складається з пояснювальної записки на 71 сторінках та графічного (презентаційного) матеріалу на 7 аркушах (слайдах)

Виконавець _____ (Ковальчук К.О.)

Керівник проекту _____ (Іванова Л.В.)

Консультанти:

з охорони праці _____ (Чорновол Н.І.)

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

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Завідувачка кафедри _____ (Іванова Л.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «20» 06 2023 р. Протокол ДКК № 1

Оцінка ЕК 5 (відмінно

Секретар ДКК _____

АНОТАЦІЯ

Тема дипломної роботи «Створення ігрового проекту - квесту на платформі Unity: розробка ігрової механіки та левелдизайну».

У цій роботі описаний принцип створення технічного завдання для створення комп’ютерної гри у ігровому движку Unity, використовуючи мову програмування C#, сама реалізація та інструменти, які було використано в ході виконання роботи. Використання патернів проектування, різноманітних бібліотек та зручних рішень, для створення проекту.

Ключові слова: Unity3D, C#, LINQ, Factory method pattern, User Interface, конфігурації, игрові жанри, Об'єктно-орієнтоване програмування.

ANNOTATION

The topic of the thesis is "Creating a game project - a quest on the Unity platform: development of game mechanics and design level."

This work describes the principle of building maintenance for creating a computer game in the Unity game engine, I use the C# programming language, the implementation itself, and the tools that were used to complete the work. Choose project patterns, various libraries and quick solutions to create a project.

Keywords: Unity3D, C#, LINQ, factory method pattern, user interface, configuration, game genres, object-oriented programming.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Кафедра *комп'ютерної інженерії*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

Спеціальність *123 «Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР ***Беркань І.В.***

"_____" _____ 202___р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

Здобувачеві (здобувачці) освіти ***Ковальчук Костянтин Олексійович***

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи

Створення ігрового проекту – квесту на платформі Unity: розробка ігрової механіки та левелдизайну

затверджена наказом по коледжу від "17" листопада 2022 р. № 235-Ас-ОР

2. Термін здачі кваліфікаційної роботи 20.06.2023р.

3. Вихідні дані до роботи

1. Функціональні вимоги щодо розробки ігор на базі Unity застосовуючи мову програмування C#

2. Програмні вимоги до розробки ігор на базі Unity.

3. Вимоги щодо тестування кінцевого продукту.

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

Вступ. Аналітичний огляд аналогів для розробки ігор. Визначення технічного завдання на роботу. Реалізація ігрових механік. Тестування кінцевого продукту. Розділ охорони праці Висновок. Перелік використаних джерел.

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

1. Реалізація головного меню

2. Реалізація початкової кат-сцени

3. Реалізація головної сцени гри

4. Реалізація переміщення гравця

5. Реалізація складів та заводів

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів роботи, що стосується їх

Розділ	Консультант	ПІДПИС	
		Завдання видав	Завдання прийняв
Основний	*Іванова Л.В.*		
Охорона праці	*Чорновол В.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 30.11.2023

Керівник роботи Іванова Л.В.
(підпис)

Завдання прийняв до виконання Ковальчук К.О.
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Вступ*	*4.05.2023*	Виконано
2.	*Оформлення тестового завдання для додатку.*	*8.05.2023*	Виконано
3.	*Аналіз технічного завдання*	*10.05.2023*	Виконано
4.	*Реалізація переміщення гравця на сцені.*	*15.05.2023*	Виконано
5.	*Реалізація сутностей предметів для*	*17.05.2023*	Виконано
6.	*Реалізація заводів та складів*	*28.05.2023*	Виконано
7.	*Реалізація придбання заводів та покращень до них*	*30.05.2023*	Виконано
8.	*Реалізація системи квестів*	*1.06.2023*	Виконано
9.	*Тестування на виявлення критичних помилок*	*2.06.2023*	Виконано
10.	*Охорона праці*	*3.06.2023*	Виконано
11.	*Висновок*	*4.06.2023*	Виконано
12.	*Оформлення пояснювальної записки*	*7.06.2023*	Виконано
13.	*Оформлення графічної частини*	*12.06.2023*	Виконано
14.	*Малий захист кваліфікаційної роботи*	*15.06.2023*	Виконано

Виконавець Ковальчук К.О.
(підпис)

Керівник роботи Іванова Л.В.
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			БКС 27. 04 000. 00 КРБ	*Кваліфікаційна робота*		
А4			*БКС 27. 04 000. 00 КРБ ПЗ*	*Пояснювальна*	*1*	
				записка		

Зм.	Арк.	№ докум.	Підпис	Дата	*БКС 27. 13 000. 00 КРБ ПЗ*		
Розробив		Ковальчук К.О.			Створення ігрового проекту - квесту на платформі Unity: розробка ігрової механіки та левелдизайну	Літ.	Аркуш 6 / Аркушів
Перевірив		Іванова Л.В.					
Н. Контр.		Петрашова В.І.				ВСП «ОТФК ОНТУ» гр.2БКС-27	
Затверд.		Кривченко Ю.В.					

ЗМІСТ

ВСТУП

На сьогоднішній день існує безліч напрямків у сфері IT технологій, кожний з яких – по-своєму важливий. Ігрова індустрія, можна сказати - буде завжди, людям завжди буде хочеться поринути у віртуальний світ, де тільки від них залежить, як складатиметься гра.

Ігри – це величезні доходи, спільнота та свій окремий світ. Кожна гра – по-своєму унікальна. Інді розробники та великі компанії розробляють ігри, які можуть залучити навіть найвибагливіших гравців і дати їм те, чого вони хочуть.

Інді розробка сьогодні показує великі успіхи у сфері ігор. Щодня все більше і більше виходить інді ігор, які приголомшують своєю якістю, підходом до розробки та ідеєю. Існують готові рішення, щоб спростити розробникам процес розробки. Зрештою все залежить від фантазії розробника та його умінь. Інструменти дають можливість поставити вміння на другий план після фантазії.

Щодо архітектури, було вирішено розробляти гру застосовючи 3D двигун Unity, та мову програмування C#. Цей двигун має величезну функціональність. Завдяки створенню гри на цьому двигуні – вдалося створити UI частину з певними анімаціями. Інструменти дають можливість легко та зручно працювати з інтерфейсом користувача, підтримуючи адаптивність під різні розміри екрану. 3D частина була розроблена власноруч, всі будівлі та навколишнє середовище на сцені.

Всю графічну частину (3D, 2D) було взято з безкоштовних офіційних джерел по одній тематиці, щоб створити унікальну та привабливу атмосферу у грі.

Зі всіх патернів проектування було вибрано саме Factory Pattern. Породжуючий патерн проектування, який надає інтерфейс для створення об'єктів у суперкласі, але дозволяє підкласам вибирати клас екземпляра, що створюється. Фабричний метод делегує створення об'єктів підкласів, забезпечуючи більш гнучку архітектуру та позбавляючи клієнтський код від прив'язки до конкретних класів.

Основна ідея фабричного методу полягає в тому, щоб винести створення об'єктів за межі клієнтського коду та надати для цього спеціальні фабричні методи в абстрактному суперкласі чи інтерфейсі. Конкретні підкласи можуть перевизначати ці методи та повертати екземпляри відповідних класів.

Було застосовано бібліотеку LINQ, яка надає зручний та виразний спосіб для написання запитів до даних, аналогічний SQL, але інтегрований безпосередньо у мову програмування. Це дозволяє програмістам писати запити прямо в коді C#, без необхідності використовувати окремі мови запитів або специфічні API.

ТЕХНОЛОГІЧНИЙ РОЗДІЛ

1.1 Оформлення технічного завдання проекту

Технічне завдання - це документ, який містить повний опис вимог та специфікацій для розробки програмного продукту чи проекту. Воно є основою для комунікації між замовником та виконавцями проекту та є керівництвом для розробників під час виконання проекту. Технічне завдання визначає цілі, функціональність, архітектуру, вимоги до продуктивності, дизайн та інші аспекти проекту.

У технічному завданні зазвичай містяться такі розділи:

Вступ: Описує цілі та загальну концепцію проекту, а також основні завдання та вимоги.

Опис продукту: Детальний опис того, що повинен робити продукт, його основні функції та можливості. Тут вказуються вимоги до інтерфейсу користувача, функціональності, взаємодії з іншими системами і т.д.

Вимоги: У цьому розділі зазначаються функціональні та нефункціональні вимоги до проекту. Функціональні вимоги описують, які функції повинен виконувати продукт, наприклад можливості взаємодії з базою даних, обробка даних і т.д. Нефункціональні вимоги визначають обмеження, такі як продуктивність, безпека, сумісність та інші аспекти проекту.

Архітектура: Цей розділ описує загальну структуру та компоненти проекту, включаючи основні модулі, бази даних, зв'язки між компонентами та прийняті технологічні рішення.

1.2 Аналіз технічного завдання

«Moon Escape» – це проект гри (у жанрі квест, айдт, пригода), створений з використанням Unity та мови програмування C#. У цій грі головний герой, прагне відкрити та покращити заводи, які виробляють різні предмети за допомогою певних рецептів. Кожен предмет має власну вартість.

Мета гри - розвивати свої заводи, виробляти та продавати предмети, заробляти ігрову валюту та досягати заданих цілей, щоб пройти гру. Гра

пропонує гравцеві завдання, що потребують планування, ресурсного управління та оптимізації виробництва.

1.2.1 Опис продукту

«Moon escape» являє собою гру-квест, де гравець бере на себе роль астронафта, який буде відчиняти заводи за ігрову валюту, щоб виготовляти предмети. Головне завдання гравця – відкрити нові заводи та виробляти предмети, які необхідні для проходження квестів. Проходження всіх квестів буде вважатися проходженням гри.

Гра має включати в себе такі функціональності:

- Створення\покращення заводів: Гравець може будувати та покращувати різні заводи для виробництва різних предметів. Кожен завод має свої особливості. Відкривати заводи гравець має за ігрову валюту, яку він може отримати продавши предмети з його інвентаря. Сутність заводу має свою швидкість виробництва одиниці предмету та рецептуру, яка задається власноруч або динамічно в процесі гри.
- Система складів: в грі у кожного заводу має буди склад, куди головний герой може покласти предмети або забрати. Повинно бути 2 типу складів, це віддавальний та приймальний. Віддавальний склад приймає в себе предмети, вироблені заводом та може тільки віддавати їх гравцю. Приймальний склад може містити в собі тільки ті предмети, які необхідні заводу для виробництва предмету. Цей склад забирає певні предмети у гравця. Кожен склад має свою місткість, яку можна змінювати
- Виробництво предметів: Кожен завод виробляє певний предмет\предмети. Різні предмети мають різні вимоги та вартість виробництва. У кожного предмета є своя ціна. Швидкість виробництва можна покращувати через систему покращень заводу.

- Магазин, для продажу предметів: Гравець може продавати створені предмети в магазині для заробітку коштів на придбання заводів чи покращень. Магазин взаємодіє з граівцем, якщо той знаходиться у тригерній зоні магазину.
- Квести та завдання: Гравець матиме доступ до різних квестів та завдань, які потребують певних предметів. Один квест може зарахований тоді, коли гравець заповнив необхідною кількістю предметів зону квесту. При виконанні квесту – відчиняється доступ до іншого, допоки гравець не пройде останній квест. Виконання останнього квесту зараховується як прохождення гри.
- Економічна система в грі: Кожен предмет повинен мати унікальну ціну, яка має бути згідна з важкістю його виготовлення. Кожне покращення для заводів повинно мати свою коштовність згідно з його поточним рівнем та предметом, якій він створює. Кожен завод потрібен мати свою коштовність згідно з предмету, якій він виробляє.

1.2.2 Функціональні та нефункціональні вимоги

Функціональні вимоги

- Головний герой: реалізувати функціонал переміщення по мапі, взаємодією з навколишнім середовищем, тригерними зонами магазину, складу, заводу, квестів. Реалізувати систему гаманця, яка буде оброблювати всю взаємодію предметів, які є у гравця та які є на сцені. Гаманець повинен реагувати на залучення та вилучення предметів в гравця.
- Заводи: реалізувати можливість будувати заводи на ігровій карті. Поліпшення заводів підвищення їх продуктивності. Завод повинен мати тригерну зону, яка збирає в гравця певні предмети для створення заявленого предмету. Завод не має працювати, якщо предметів немає, або склад для зберігання предметів - повний. Поліпшення заводів має свою відповідну тригерну зону, де гравцю, якщо він в неї зайде, буде

відображено інтерфейс з інформацією про коштовність поліпшення. (Таблиця 1.1., Таблиця 1.2.)

- Магазин: реалізувати можливість продавати у спеціальному місці всі предмети, які є у гравця. Реалізувати діалогове вікно, яке буде відображати ціну кожного предмету. Магазин має тригерну зону, де у гравця вилучаються предмети взамін на ігрову валюту (Блок-схема 1.3)
- Система квестів: реалізувати надання різних квестів та завдань для гравця. Квести на збір ресурсів має бути завершений тільки тоді, коли гравець принесе необхідну кількість ресурсів для поточної зони квесту. Після проходження квесту має відкриватися інший, за наявністю. (Блок-схема 1.4)
- Кінцівка: реалізувати кінцівку гри. Пройшовши всі квести гравець виграє гру, запускається відповідна кат-сцена, яка демонструє закінчення гри.

Нефункціональні вимоги

- Платформа: Гра розроблятиметься для платформи PC з можливістю.
- Програмні вимоги: Гра має розроблятися з використанням ігрового двигуна Unity та мови програмування C#. Немає ніяких обмежень щодо використання різноманітних бібліотек, плагінов або рішень, які вже є у відкритому доступі для мови програмування C# та двигуна Unity.
- Графіка: Гра матиме якісні 2D або 3D графічні елементи. 3D моделі мають бути частково анімовані, задля поліпшення візуального вигляду гри. Потрібно використовувати систему часток, яка буде наповнювати гру динамічністю. Інтерфейс користувача: Інтуїтивний і привабливий інтерфейс користувача. Інтерфейс потрібен бути зручний та доступний для розуміння користувачем. Інтерактивний інтерфейс буде використано лише в декількох випадках.
- Продуктивність: Гра повинна працювати плавно та ефективно навіть на старих комп'ютерах. Продуктивність гри треба забезпечити рішеннями

щодо компресії великих текстур, часток інтерфейсу та 3D моделей. Встановити певні налаштування сцени та білду для додатку, щоб зберегти продуктивність гри.

- Економічна частина гри:

Таблиця 1.1.

Ціни на предмети	
Назва предмету	**Ціна**
Hummer	500
Tool	750
Tape	1800
Gas	3500
Chip	6000
Microchip	10000

Таблиця 1.2.

Ціна на відкриття заводів	
Назва заводу	**Ціна**
Hummer	-
Tool	8500
Tape	15400
Gas	20700
Chip	60000
Microchip	100000

1.2.3 Архітектура

При розробці гри є обов.язковим використання паттерну проектування Factory Pattern, який забезпечить доступність коду для подальших змін та покращувань гри. Реалізовано паттерн має бути наступним чином:

- Створити окремий компонент, який відповідає за створення об'єктів, та перемістіть код створення об'єктів у цей компонент.
- Створити фабричний метод або фабричний клас, який інкапсулює складність створення об'єктів із певними залежностями. Надайте простий інтерфейс для створення об'єктів із заданими параметрами.
- Визначити абстрактний інтерфейс заводу, з яким клієнтський код взаємодіятиме. Реалізувати конкретні фабрики, які створюють конкретні класи об'єктів, та приховати деталі реалізації від клієнтського коду.
- Створити новий клас, що реалізує інтерфейс заводу, для кожного нового типу об'єкта, який ви хочете додати. Додати нові класи до заводу, щоб клієнтський код міг створювати ці нові типи об'єктів, не змінюючи свій код.
- Створити підроблені об'єкти або заступники для фабрик під час тестування, щоб перевірити різні сценарії та поведінку клієнтського коду. Це забезпечить більш легке та надійне тестування коду.

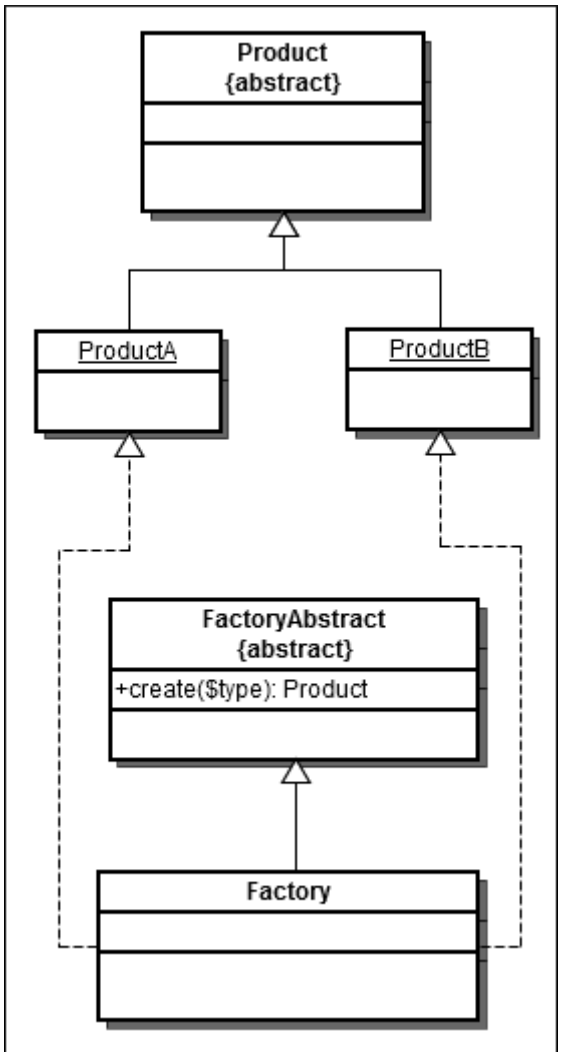


Рисунок 1.1. Блок-схема прикладу роботи патерну проектування Factory Pattern

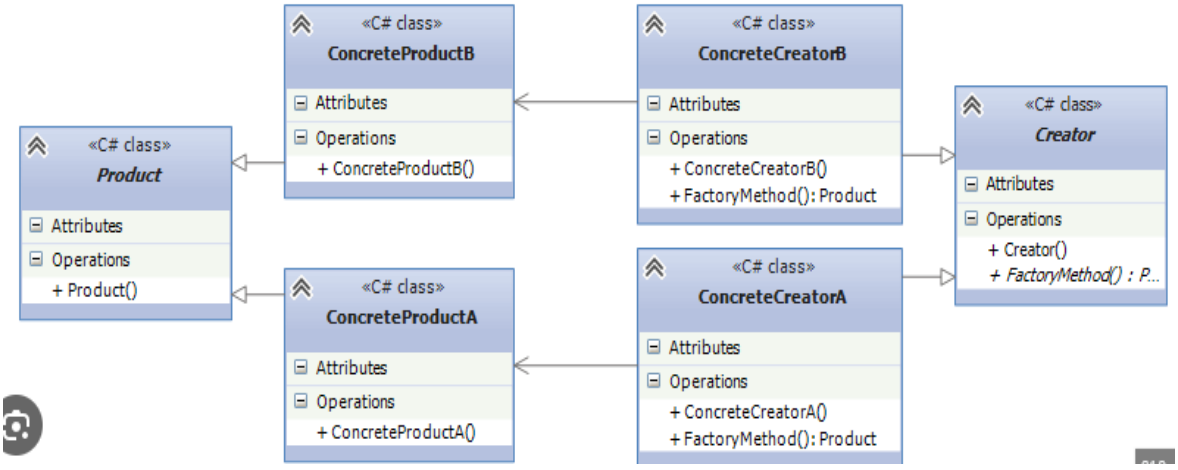


Рисунок 1.2. Модель роботи патерну проектування Factory Pattern

Основний компонент гри складатиметься зі сцени ігрового світу, де будуть розміщені ігрові об'єкти (заводи, склади, тощо).

Реалізація окремих сутностей в грі має бути виконано згідно з наступними блок-схемами:

- Сутність заводу повинна бути виконана згідно блок-схеми, яка описує принцип роботы:

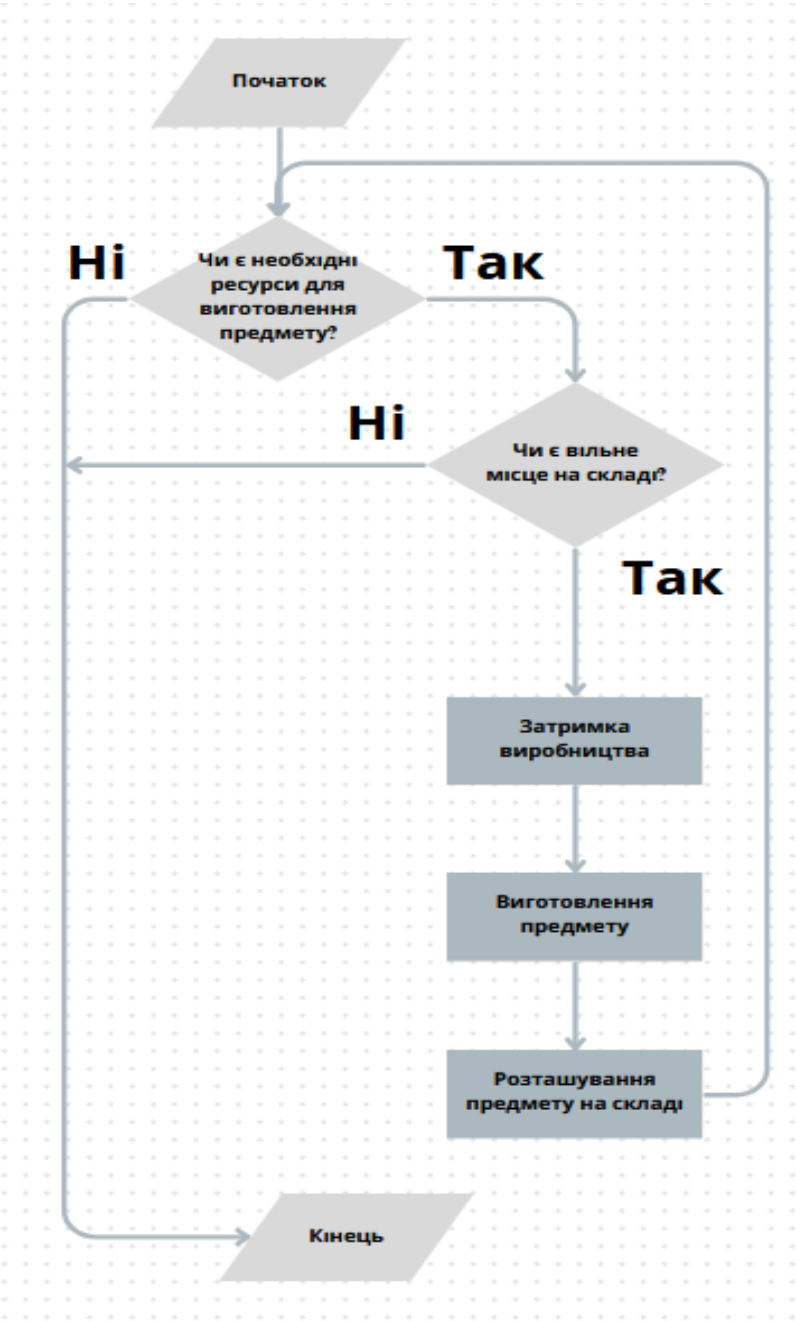


Рисунок 1.3. Сутність заводу

- Сутність складу повинна бути виконана згідно блок-схеми, яка описує принцип роботи:

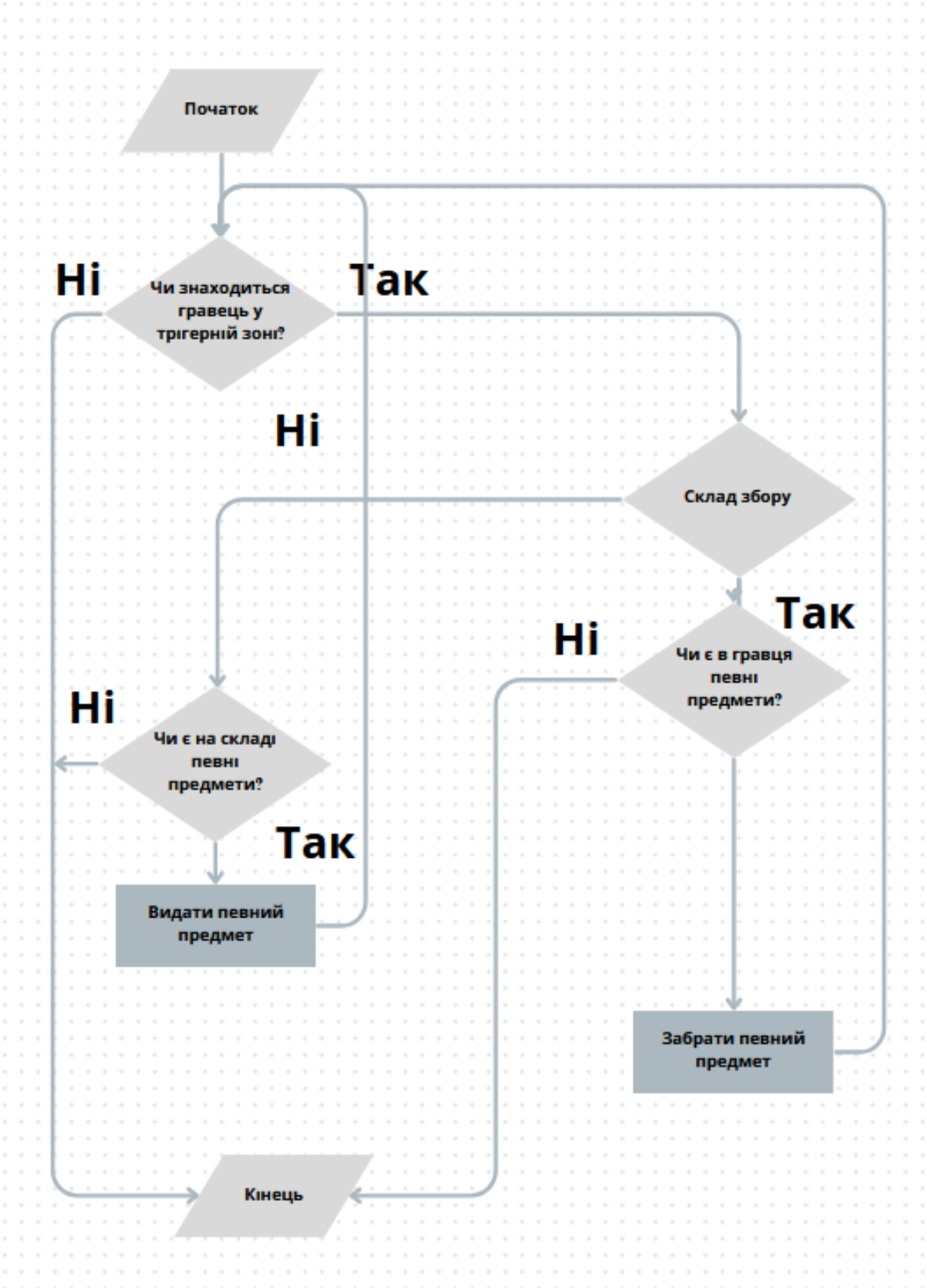


Рисунок 1.4. Сутність складу

- Сутність магазину повинна бути виконана згідно блок-схеми, яка описує принцип роботи:

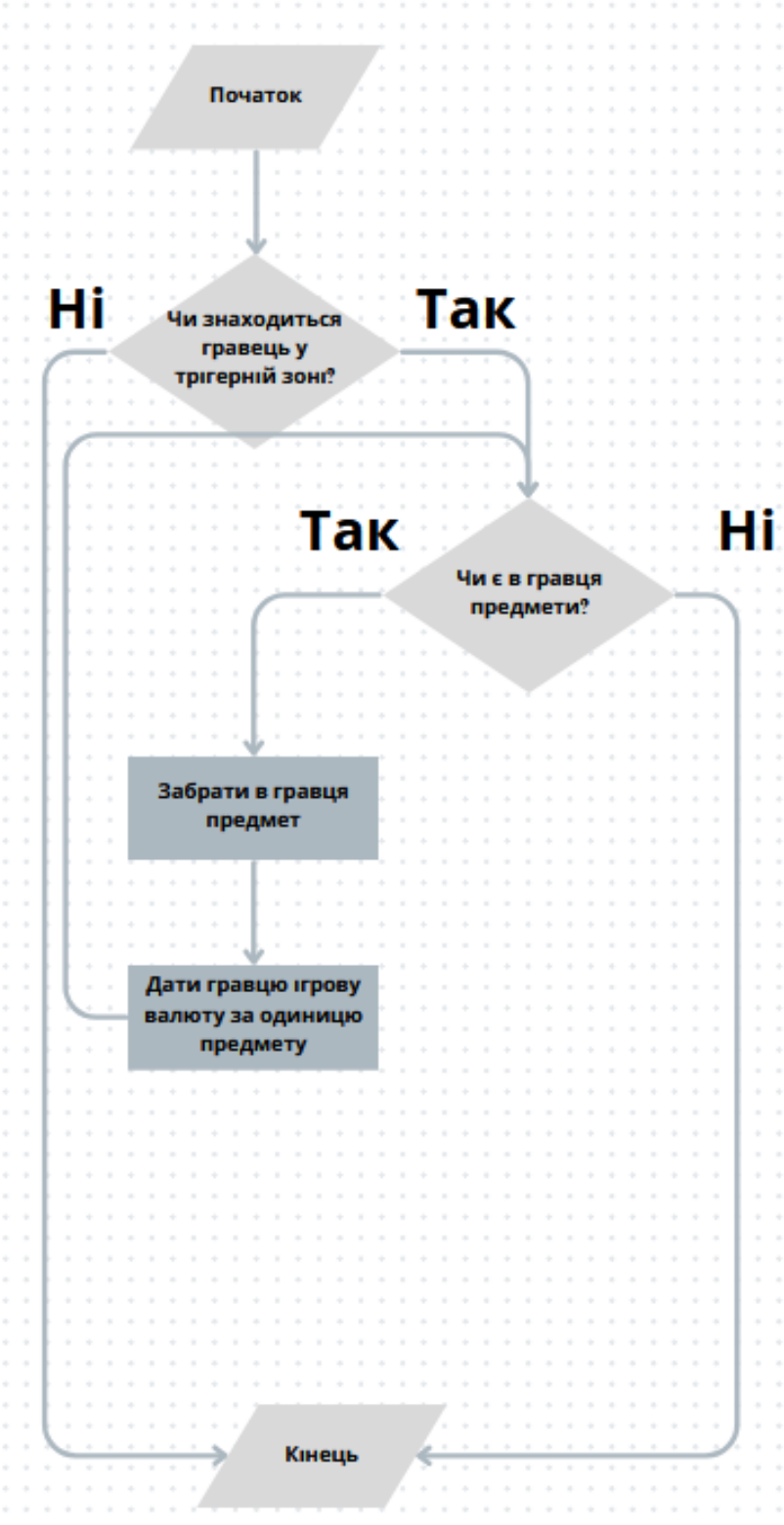


Рисунок 1.5. Сутність магазину

- Сутність квесту повинна бути виконана згідно блок-схеми, яка описує принцип роботи:

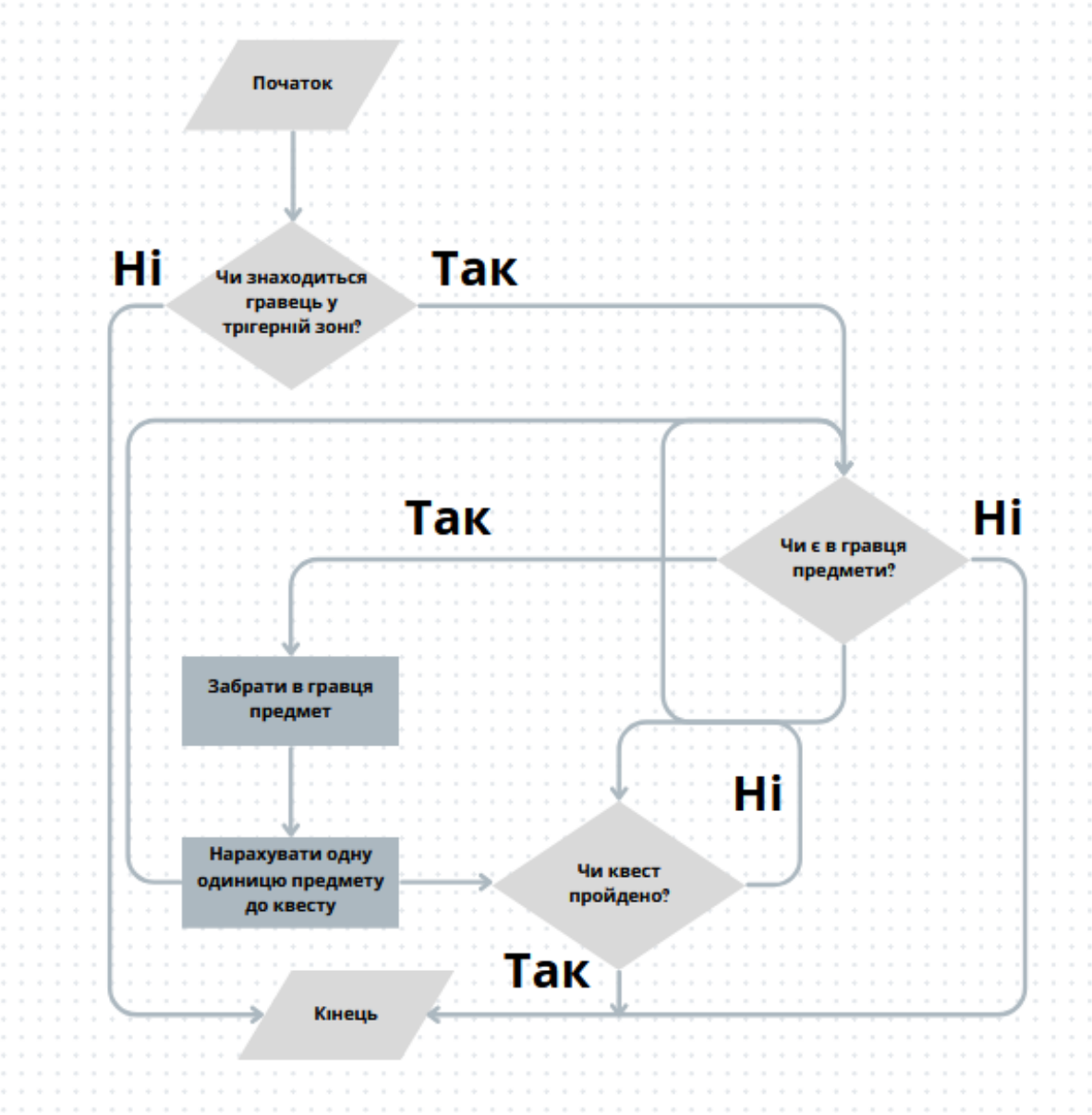


Рисунок 1.6. Сутність квесту

1.2.4 Розробка та реалізація

Розробка гри повинна виконатись у такому порядку:

- Створення ігрового світу та сцени, включаючи об'єкти заводів.
- Реалізація архітектури проекту та компонентів згідно з вимогами патерну проектування Factory Pattern.
- Реалізація функціональності управління заводами, включаючи будівництво, покращення та виробництво предметів.
- Розробка системи магазину та продажів предметів.
- Реалізація системи квестів та завдань.
- Налаштування балансу гри.
- Заповнення конфігураційних файлів гри.
- Тестування.

1.3 Реалізація переміщення гравця на сцені

Реалізувати переміщення гравця на сцені можна декількома способами:

- Використовуючи фізику.
- Використовуючи параметри двигуна, які не враховують наявність фізики на сцені.

В даному проекті необхідна взаємодія з фізикою, тому обрано перший варіант для розробки логіки. Скрипт, якій відповідає за переміщення гравця виглядає наступним чином:

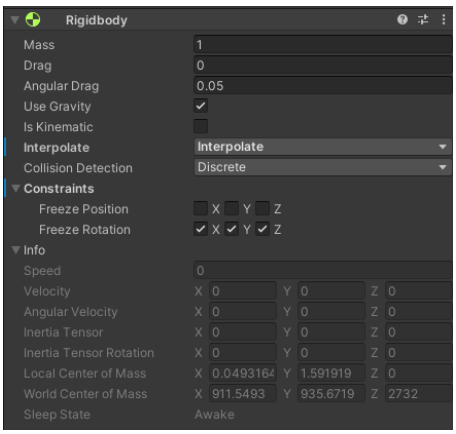


Рисунок 1.7. Код переміщення головного герою

```
public void Move(Vector2 axises)
{
    ApplyMovement(axises);
    playerAnimator.SetRunAnimation(1);

}

ссылка: 1
private void ApplyMovement(Vector2 axises)
{
    var velocity = GetYVelocity();
    Vector3 direction = new Vector3(axises.x, 0f, axises.y);
    rigidBody.velocity = direction * (Time.deltaTime * movementSpeed) * movementForce;
    UpdateRotation(rigidBody.velocity);
    rigidBody.velocity = new Vector3(rigidBody.velocity.x, -3f, rigidBody.velocity.z);
}

ссылка: 1
public bool IsGrounded()
{
    return Physics.Raycast(transform.position, Vector3.down, colider.bounds.size.y + distanceBetweenGround);
}

Ссылок: 2
public void ResetVelocity(Vector2 direction)
{
    playerAnimator.SetRunAnimation(0);
    rigidBody.velocity = new Vector3(0f, 0f, 0f);
}

ссылка: 1
private void UpdateRotation(Vector3 direction)
{
    if (direction == Vector3.zero)
    {
        return;
    }
    transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.LookRotation(direction),
        rotationSpeed * Time.deltaTime);
}
```

Рисунок 1.8. Код переміщення головного герою

На цій ілюстрації можна бачити декілька методів, які виконують різні дії.

Метод ApplyMovement виконує основну дію – переміщення гравця на сцені.

Також є важливий метод UpdateRotation, який розгортає гравця в напрямку руху. Використовую вбудовані функції Unity :

- Quaternion.Rotation();
- Physics.RayCast();
- RigidBody.SetVelocity();

1.4 Реалізація сутностей предметів, які будуть вироблятися на заводах

Код, який використовується для опису сутності предмету:

```
namespace Gameplay.FactorySystem.Items
{
    [Serializable]
    Ссылок: 6
    public class Item
    {
        public int id;
    }
}
```

Рисунок 1.9. Код сутності предмету

Знаючи ID предмета, можна описати його характеристики. Було створено ще один скрипт:

```
namespace Gameplay.FactorySystem.Configurations
{
    [Serializable]
    ссылка: 1
    public class ItemConfigEntity
    {
        public int id;
        public int price;
        public Sprite sprite;
    }

}
```

Рисунок 1.10. Код сутності для опису характеристик предмету

Тут ми можемо бачити характерні поля:

- id – ідентифікатор предмета
- price – ціна предмету
- Sprite – іконка предмету

1.5 Реалізація системи заводів/складів.

Завод і склад – це такі-ж сутності, як і предмети. У кожного заводу повинен бути склад, який буде накопичувати вироблені матеріали, та, на вимогу, повинен буди склад для складання в нього ресурсів самим гравцем, які будуть перероблюватися на заводі та видавати інший предмет. Завод та склад повинні мати певні набори даних, якими вони будуть керуватися під час складування/вироблення предметів.

```
namespace Gameplay.FactorySystem
{
    [Serializable]
    ссылка: 1
    public class FactoryEntity
    {
        [SerializeField] private int id;
        [SerializeField] private FactoriesUpgradeConfig factoriesUpgradeConfig;
        [Header("Produce settings")]
        [SerializeField] private float manufactureDuration;
        [SerializeField] private ConveyorBelt conveyor;
        [Space]
        [Header("Items settings")]
        public List<Item> consumableItems;
        public List<Item> manufacturedItems;

        [Space]
        [Header("Storages settings")]
        public StorageBase storageToFill;
        public StorageBase storageToGrab;

        public event Action<int, Transform> OnGrabItem;
        public event Action<int, Transform> OnPutItem;

        ссылка: 1
        public float ManufactureDuration => manufactureDuration;
```

Рисунок 1.11. Код сутності заводу

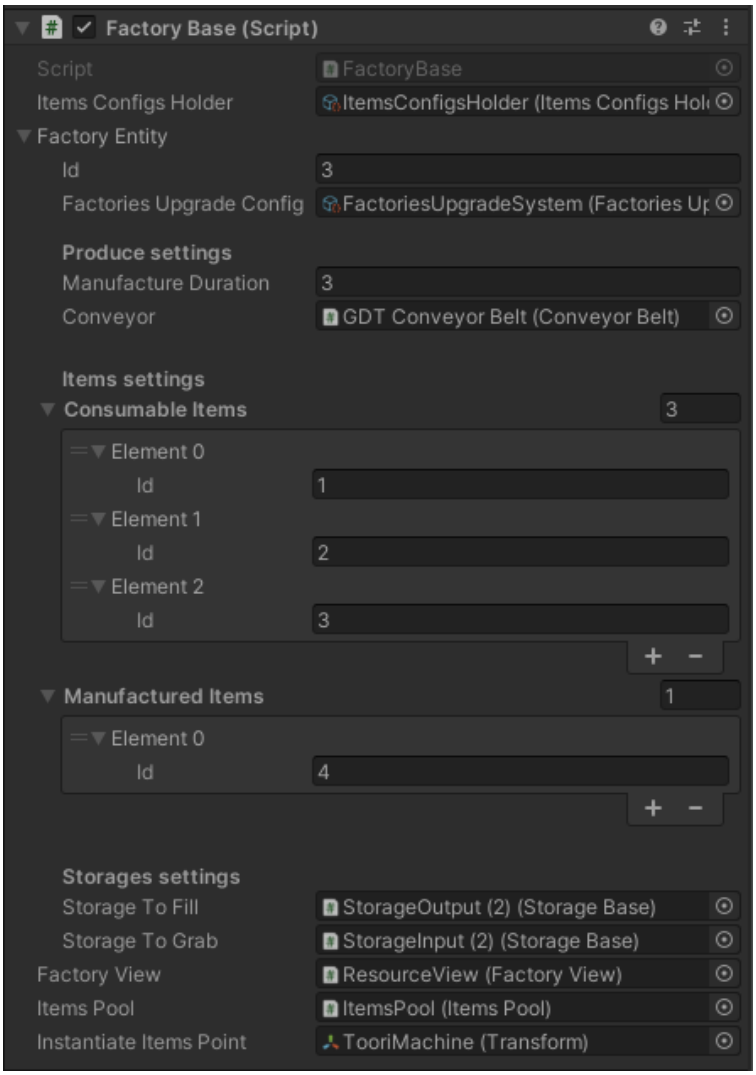


Рисунок 1.12. Конфігурація сутності заводу

Скрипт заводу містить в собі такі поля:

- id – ідентифікатор заводу
- FactoriesUpgradeConfig – конфігурація покращень заводу
- manufactureDuration – затримка перед виробництвом одної одиниці предмету
- consumableItems – предмети, які завод поглинає
- manufacturedItems – предмети, які завод виробляє
- storageToFill – склад, звідки завод бере предмети.
- storageToGrab - склад, куди с конвейору йдуть предмети
- OnGrabItem – дія, яка виконується, якщо грок забирає предмети.

- OnPutItem - дія, яка виконується, якщо грок ставить предмети.

```
ссылка: 1
public void InitializeStorages()
{
    if (HasConsumableItems())
    {
        storageToGrab.InitializeStorage(consumableItems);
    }

    storageToFill.InitializeStorage(manufacturedItems);
}

ссылка: 1
public bool HasConsumableItems()
{
    return consumableItems.Any();
}

ссылка: 1
public bool StorageToGrabHasItems()
{
    foreach (var item in consumableItems)
    {
        if (!storageToGrab.HasItems(item.id))
        {
            conveyor.SetActive(false);
            return false;
        }
    }

    conveyor.SetActive(true);
    return true;
}

ссылка: 1
public bool StorageToFillIsFull()
{
    conveyor.SetActive(!storageToFill.StorageIsFull());
    return  storageToFill.StorageIsFull();
}
```

Рисунок 1.13. Код сутності заводу

```
ссылка: 1
private void InitializeFactory()
{
    factoryEntity.InitializeStorages();
    factoryEntity.InitializeFactory();
    InitializeView();

    StartCoroutine(ManufactureCoroutine());
}

ссылка: 1
private void InitializeView()
{
    factoryView.SetSprite(itemsConfigsHolder.itemsConfigs.Find(c => c.id == factoryEntity.manufacturedItems[0].id).sprite);
}

ссылка: 1
private IEnumerator ManufactureCoroutine()
{
    while (true)
    {
        yield return new WaitUntil(() => !factoryEntity.StorageToFillIsFull() && factoryEntity.StorageToGrabHasItems());
        factoryEntity.GrabItems();
        yield return new WaitForSeconds(factoryEntity.ManufactureDuration);
        factoryEntity.PutItems();
    }
}


Ссылок: 2
private void AnimateItemIn(int id, Transform startTransform)
{
    //var currentItem = itemsPool.GetFreeItemById(id);
    //itemsPool.ItemsVisualAnimation.MakeTransitionAnimation(currentItem, startTransform, transform, true);
}

Ссылок: 2
private void AnimateItemOut(int id, Transform endTransform)
{
    var currentItem = itemsPool.GetFreeItemById(id);
    itemsPool.ItemsVisualAnimation.InstantiateItemOnPoint(currentItem, instantiateItemsPoint);
}
```

Рисунок 1.14. Код функціональної частини заводу

```
[SerializeField] private List<ItemView> itemsViews;
[SerializeField] private ItemsVisualAnimation itemsAnimation;
[SerializeField] private List<ItemView> itemsToInstantiate;

Сообщение Unity | Ссылок: 0
private void Awake()
{
    itemsToInstantiate.ForEach(x =>
    {
        for(int i = 0; i < 20; i++)
        {
            var item = Instantiate(x, transform);
            item.gameObject.SetActive(false);
            itemsViews.Add(item);
        }
    });
}

ссылка: 1
public ItemsVisualAnimation ItemsVisualAnimation => itemsAnimation;

ссылка: 1
public ItemView GetFreeItemById(int id)
{
    var currentItem = itemsViews.FindAll(i => !i.gameObject.activeSelf).Find(i => i.Id == id);
    currentItem.transform.parent = null;

    return currentItem;
}
```

Рисунок 1.15. Код функціональної частини заводу

```
ссылка: 1
public void GrabItems()
{
    consumableItems.ForEach(i =>
    {
        storageToGrab.GrabFromStorage(i.id, 1);
        OnGrabItem?.Invoke(i.id, storageToGrab.transform);
    });
}

ссылка: 1
public void PutItems()
{
    manufacturedItems.ForEach(i =>
    {
        storageToFill.FillStorage(i.id, 1);
        OnPutItem?.Invoke(i.id, storageToFill.transform);
    });
}

ссылка: 1
public void InitializeFactory()
{
    conveyor.SetActive(false);
    var factoryUpgradeConfig = factoriesUpgradeConfig.upgradeEntities.Find(upgrade => upgrade.factoryId == id);
    var manufactureSpeed = factoryUpgradeConfig.productionSpeedLevels[factoryUpgradeConfig.level];
    var capacity = factoryUpgradeConfig.capacityLevels[factoryUpgradeConfig.level];
    manufactureDuration = manufactureSpeed;
    storageToFill.InitializeCapacity(capacity);
    storageToGrab.InitializeCapacity(capacity);
}
```

Рисунок 1.16. Код функціональної частини заводу

```
Скрипт Unity | ссылка: 1
public class PurchaseFactoryView : MonoBehaviour
{
    [SerializeField] private string textTemplate;
    [SerializeField] private TextMeshProUGUI label;

    ссылка: 1
    public void SetText(string name, string price)
    {
        label.text = string.Format(textTemplate, name, price);
    }

    ссылка: 1
    public void Activate(bool state)
    {
        gameObject.SetActive(state);
    }
}
```

Рисунок 1.17. Код UI частини заводу

Скрипт сутності складу виглядає наступним чином:

```
namespace Gameplay.FactorySystem.Storage
{
    Ссылок: 4
    public enum StorageTypes
    {
        Output,
        Input
    }
}
```

Рисунок 1.18. Код сутності складу

```
namespace Gameplay.FactorySystem.Storage
{
    [Serializable]
    ссылка: 1
    public class StorageEntity
    {
        [SerializeField] private int capacity;
        [SerializeField] private int maxCapacity;

        [SerializeField] private List<ItemsHolder> _itemsHolders;

        Ссылок: 7
        public List<ItemsHolder> ItemsHolder => _itemsHolders;
        ссылка: 1
        public int MaxCapacity => maxCapacity;

        ссылка: 1
        public void InitializeMaxCapacity(int currentValue)
        {
            maxCapacity = currentValue;
        }

        ссылка: 1
        public void SetItemsToStorage(List<Item> items)
        {
            _itemsHolders = new List<ItemsHolder>();
            items.ForEach(i =>
            {
                _itemsHolders.Add(new ItemsHolder() {itemId = i.id, itemsCapacity = 0, maxCapacity = (maxCapacity / items.Count)});
            });
        }
    }
}
```

Рисунок 1.19. Код сутності складу

Застосовуються поля :

- Capacity – поточна кількість предметів на складі
- MaxCapacity – максимальна кількість предметів на складі

Частина коду з скрипта, якій функціонує між складом і заводом:

```
public class StorageBase : MonoBehaviour
{
    [SerializeField] private StorageTypes type;
    [SerializeField] private ItemsConfigsHolder itemsConfigHolder;
    [SerializeField] private StorageEntity storageEntity;
    [SerializeField] private List<StorageView> storageView;

    private List<StorageView> _activeStorageViews;

    public event Action<StorageBase, List<int>, int> OnPlayerEnter;
    public event Action OnPlayerExit;

    ссылка: 1
    public StorageTypes StorageTypes => type;

    Ссылок: 2
    public void InitializeStorage(List<Item> items)
    {
        storageEntity.SetItemsToStorage(items);

        InitializeViews();
    }

    Ссылок: 2
    public void FillStorage(int id, int value)
    {
        var currentStorage = storageEntity.ItemsHolder.Find(i => i.itemId == id);
        currentStorage.itemsCapacity += value;
        var currentView = storageView.Find(v => v.ItemId == id);
        currentView.UpdateView(currentStorage.itemsCapacity);
        currentView.SetFullMarkActive(StorageIsFull());
    }

    Ссылок: 2
    public void GrabFromStorage(int id, int value)
    {
        var currentStorage = storageEntity.ItemsHolder.Find(i => i.itemId == id);
        currentStorage.itemsCapacity -= value;
        var currentView = storageView.Find(v => v.ItemId == id);
        currentView.UpdateView(currentStorage.itemsCapacity);
        currentView.SetFullMarkActive(StorageIsFull());
    }
```

Рисунок 1.20. Код функціональної частини складу

Тут можна бачити наступні поля:

- type – тип складу (Приймаючий, або віддаючий)
- itemsConfigHolder – конфігурація всіх існуючих предметів
- storageEntity – єдина сутність складу
- storageView – інтерфейс, який розташовано на сцені над складом.

```
    private void InitializeViews()
    {
        _activeStorageViews = new List<StorageView>();
        storageEntity.ItemsHolder.ForEach(x =>
        {
            var currentView = storageView.Find(v => !v.IsActive());
            currentView.Activate();
            currentView.ItemId = x.itemId;
            currentView.SetSprite(itemsConfigHolder.itemsConfigs.Find(c => c.id == currentView.ItemId).sprite)
            currentView.UpdateView(x.itemsCapacity);
            _activeStorageViews.Add(currentView);
        });
    }

    Сообщение Unity | Ссылок: 0
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            OnPlayerEnter?.Invoke(this, GetListOfIds(), 1);
        }
    }
    Сообщение Unity | Ссылок: 0
    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            OnPlayerExit?.Invoke();
        }
    }

    ссылка: 1
    private List<int> GetListOfIds()
    {
        var ids = new List<int>();
        storageEntity.ItemsHolder.ForEach(i =>
        {
            ids.Add(i.itemId);
        });
        return ids;
    }
}
```

Рисунок 1.21. Код функціональної частини складу

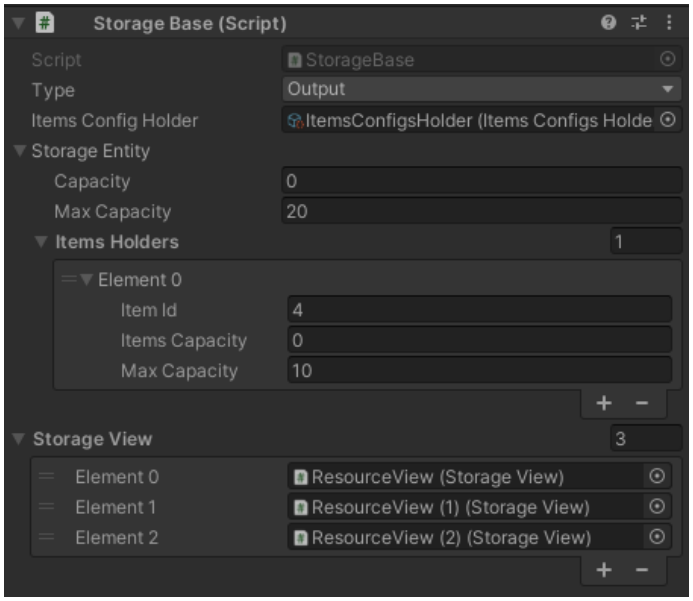


Рисунок 1.22. Вигляд конфігурації складу на сцені

```
public class StorageView : MonoBehaviour
{
    [SerializeField] private string capacityLabelTemplate;
    [SerializeField] private Image image;
    [SerializeField] private Image warningMarkImage;
    [SerializeField] private TextMeshProUGUI capacityLabel;

    private int _itemId;

    Ссылок: 4
    public int ItemId
    {
        get => _itemId;
        set => _itemId = value;
    }

    ссылка: 1
    public void SetSprite(Sprite sprite)
    {
        image.sprite = sprite;
    }

    Ссылок: 3
    public void UpdateView(int value)
    {
        capacityLabel.text = string.Format(capacityLabelTemplate, value);
    }

    Ссылок: 2
    public void SetFullMarkActive(bool state)
    {
        warningMarkImage.gameObject.SetActive(state);
    }

    ссылка: 1
    public void Activate()
    {
        gameObject.SetActive(true);
    }

    ссылка: 1
    public bool IsActive()
    {
        return gameObject.activeSelf;
    }
}
```

Рисунок 1.23. Код UI частини складу

Також, була створена система, яка відстежує дії гравця на сцені, щоб можна було зрозуміти, яку логіку впроваджувати якому елементу. Декілька фрагментів коду:

```
public class GameController : MonoBehaviour
{
    [SerializeField] private PlayerController playerController;
    [SerializeField] private List<StorageBase> storages;
    [SerializeField] private List<FactoryBase> factories;
    [SerializeField] private List<QuestSpot> questSpots;
    [SerializeField] private ShopBase shop;
    [SerializeField] private WalletController walletController;
    [SerializeField] private ItemsPool itemsPool;
    [SerializeField] private ItemsConfigsHolder itemsConfigsHolder;

    private Coroutine _triggerActionRoutine;

    Сообщение Unity | Ссылок: 0
    private void Awake()
    {
        Initialize();
    }

    ссылка: 1
    private void Initialize()
    {
        shop.OnPlayerEnter += StartShopTriggerActionTracking;
        shop.OnPlayerExit += StopTriggerActionTracking;

        questSpots.ForEach(q =>
        {
            q.OnPlayerEnter += StartQuestTriggerActionTracking;
            q.OnPlayerExit += StopTriggerActionTracking;
        });

        storages.ForEach(s =>
        {
            s.OnPlayerEnter += StartTriggerActionTracking;
            s.OnPlayerExit += StopTriggerActionTracking;
        });
    }

    ссылка: 1
    private void StartTriggerActionTracking(StorageBase storage, List<int> itemsIds, int value)
    {
        _triggerActionRoutine = StartCoroutine(TriggerActionCoroutine(storage, itemsIds, value));
    }
```

Рисунок 1.24. Код компоненту GameController

```
ссылка: 1
private IEnumerator ShopTriggerActionCoroutine(ShopBase shopBase, int value)
{
    while (true)
    {
        yield return new WaitForSeconds(playerController.GrabDuration);
        SellTriggerAction(shopBase, value);
    }
}

ссылка: 1
private IEnumerator QuestTriggerActionCoroutine(SideQuest sideQuest)
{
    while (true)
    {
        yield return new WaitForSeconds(0.3f);
        QuestTriggerAction(sideQuest);
    }
}

ссылка: 1
private IEnumerator TriggerActionCoroutine(StorageBase storage, List<int> itemsIds, int value)
{
    while (true)
    {
        yield return new WaitForSeconds(playerController.GrabDuration);
        TriggerAction(storage, itemsIds, value);
    }
}

ссылка: 1
private void SellTriggerAction(ShopBase shopBase, int value)
{
    walletController.GetExistingItemsIds().ForEach(itemId =>
    {
        if (walletController.HaveNoItems(itemId))
        {
            return;
        }
        walletController.IncreaseBalance(itemsConfigsHolder.itemsConfigs.Find(i => i.id == itemId).price);
        walletController.DecreaseWallet(itemId, value);
```

Рисунок 1.25. Код компоненту GameController

```
ссылка: 1
private void TriggerAction(StorageBase storage, List<int> itemsIds, int value)
{
    itemsIds.ForEach(itemId =>
    {
        switch (storage.StorageTypes)
        {
            case StorageTypes.Input:
            {
                if (walletController.HaveNoItems(itemId) || storage.FullOfItemsById(itemId))
                {
                    return;
                }
                walletController.DecreaseWallet(itemId, value);
                storage.FillStorage(itemId, value);
                break;
            }
            case StorageTypes.Output:
            {
                if (walletController.StackIsFull(itemId) || storage.StorageIsEmpty())
                {
                    return;
                }

                walletController.IncreaseWallet(itemId, value);
                storage.GrabFromStorage(itemId, value);
                break;
            }
        }
    });
}
}
```

Рисунок 1.26. Код компоненту GameController

1.6 Реалізація рецептів для предметів.

На заводах є поля, які в собі тримають список із унікальних номерів існуючих предметів. Один список бере в себе предмети, які будуть поглинатися зі ПРИЙМАЮЧОГО складу біля заводу, другий список створює предмет\предмети, які будуть переходить до ВІДДАЮЧОГО складу. Взагалом, комбінації можуть бути безкінечно різними. Нижче наведено приклад, як це виглядає:

Задано заводу ID предметів, які необхідно поглинути:

Рисунок 1.27. Вигляд рецепту в inspector

Задано заводу ID предметів, які необхідно виробити:

Рисунок 1.28. Вигляд виробничого предмету в inspector

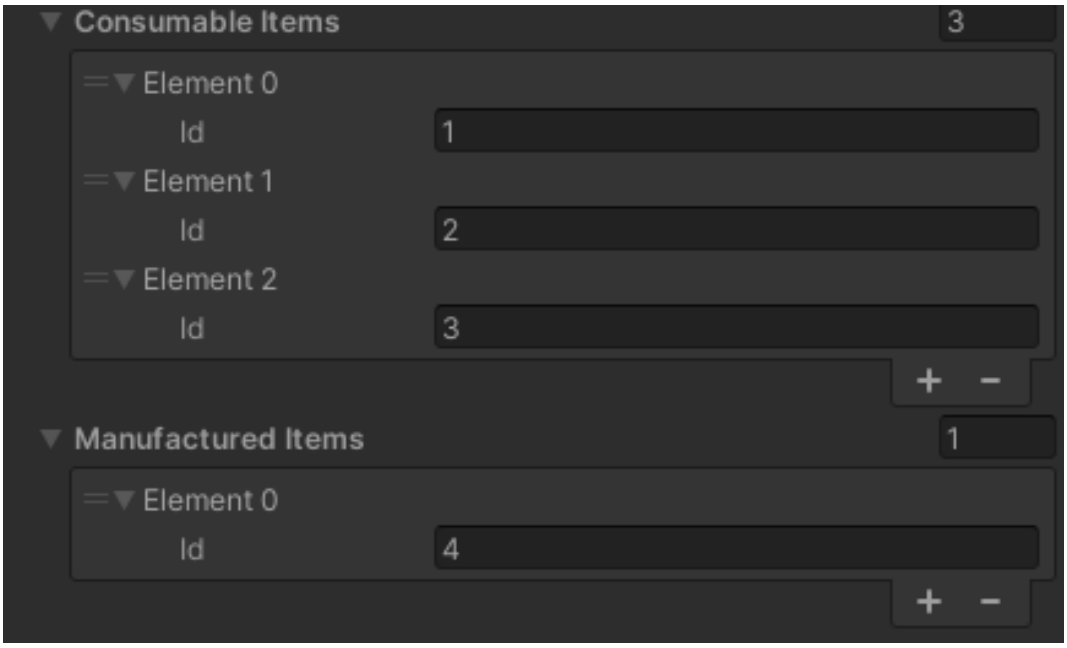


Рисунок 1.29. Вигляд виробничого предмету в inspector

Зовнішній вигляд рецептів на ігровій сцені:

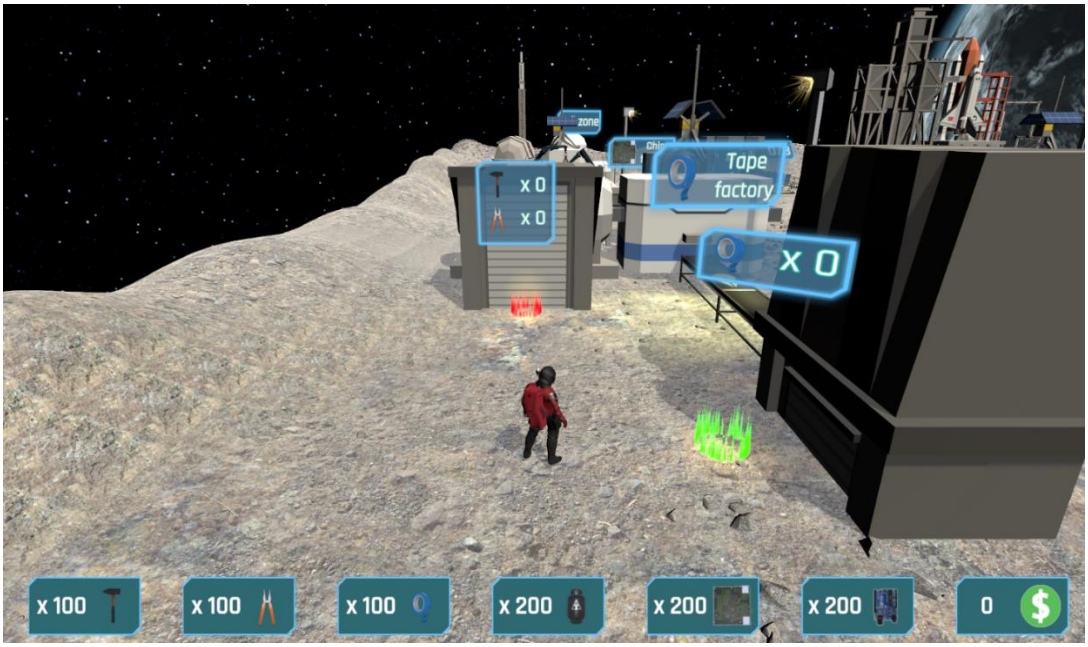


Рисунок 1.30. Вигляд рецептів, заводів та складів на ігровій сцені.

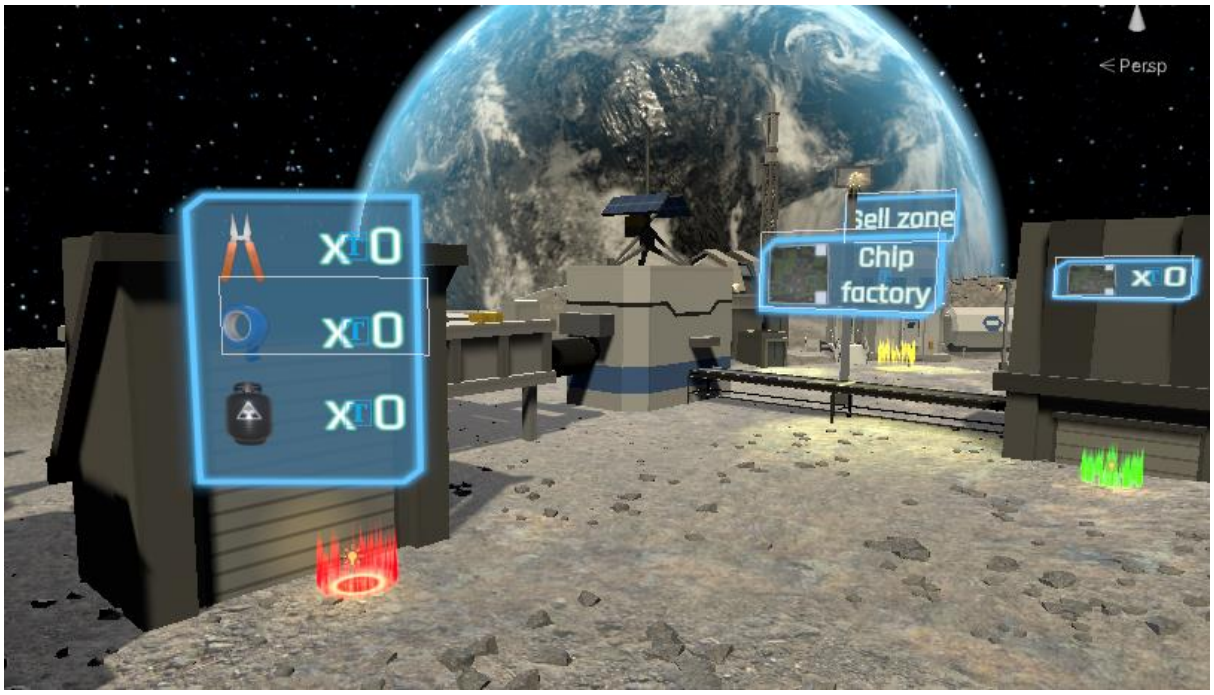


Рисунок 1.31. Вигляд рецептів, заводів та складів на ігровій сцені.

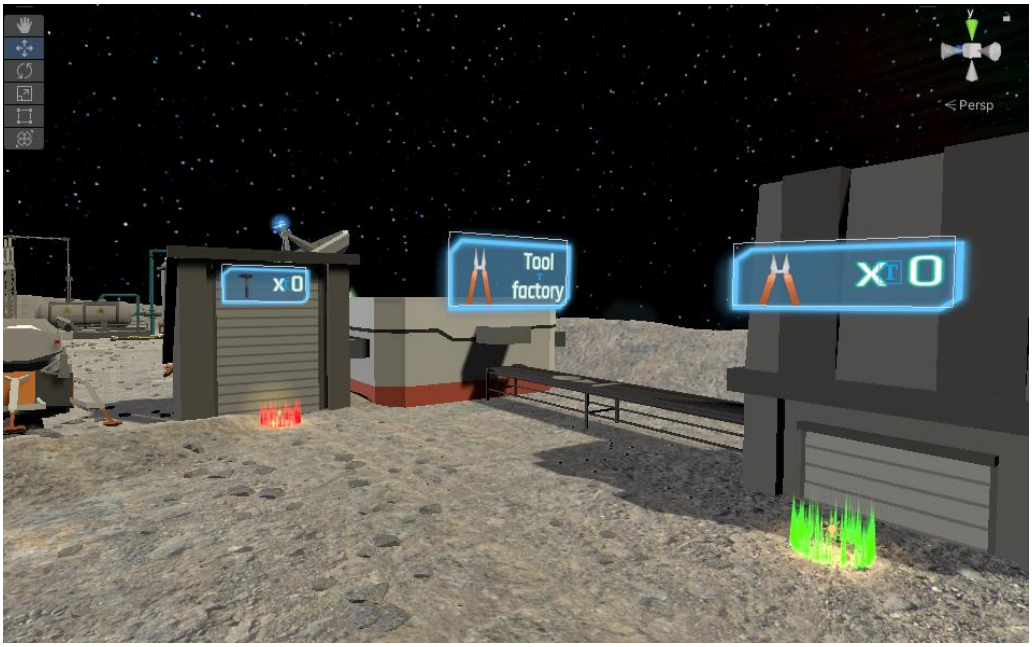


Рисунок 1.32. Вигляд рецептів, заводів та складів на ігровій сцені.

Згідно з конфігурацією предметів, яка зображена на рисунку 2.2, можна побачити, що завод поглинає предмети:

- Hummer
- Tool

а виробляє :

- Tape;

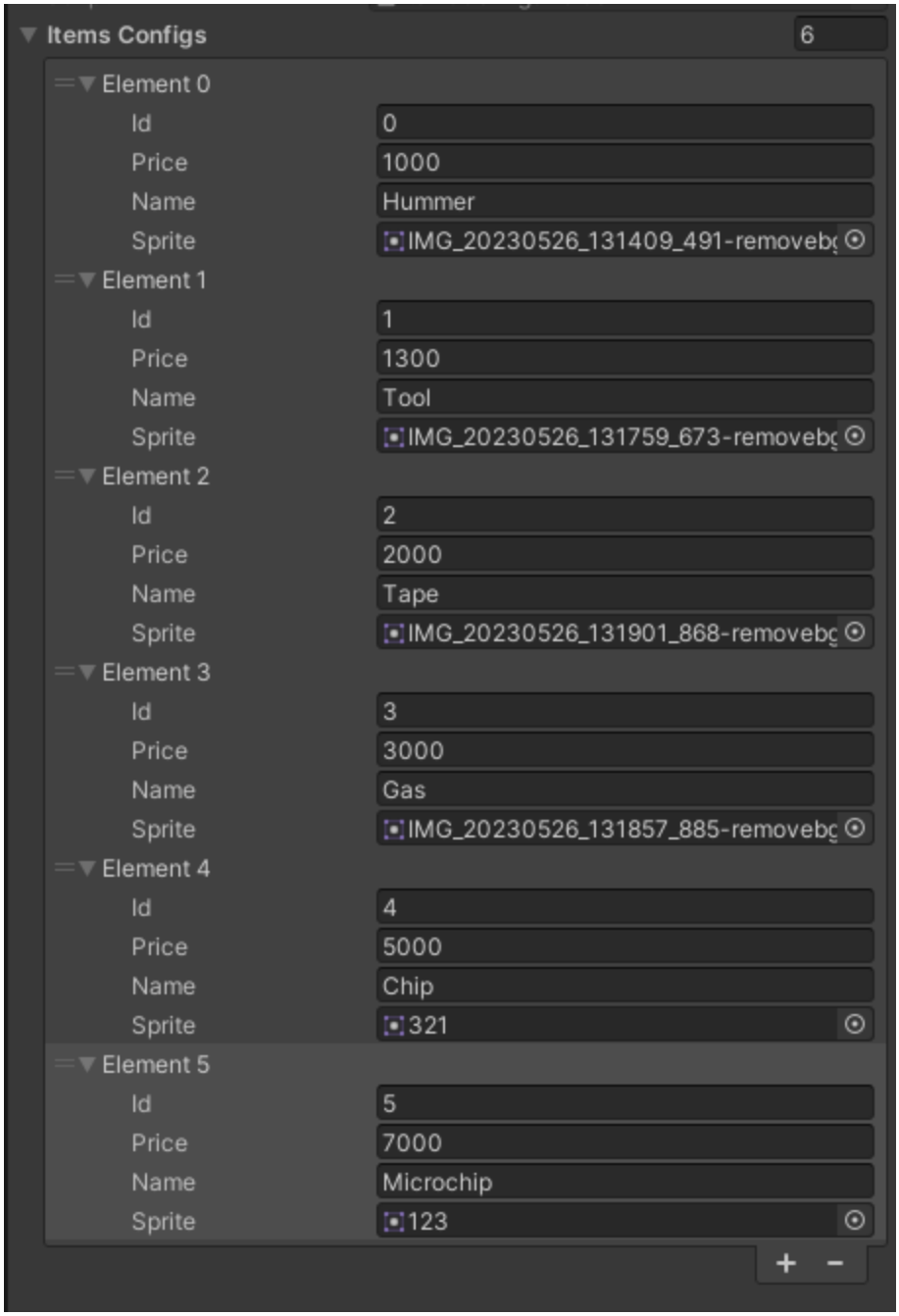


Рисунок 1.33. Вигляд конфігурації предметів в inspector

Таким чином було зроблено для всіх заводів. Кожен завод виробляє унікальний предмет, а поглинає один чи декілька тих, що в процесі гри гравець міг здобути.

1.7 Реалізація системи придбання заводів та покращення для них.

В грі був продуманий варіант, при якому йому необхідно заробляти кошти та відкривати нові заводи за них. Це стосується і відкритих заводів, вони можуть бути покращені, що призведе до прискорення виробництва.

Логіка придбання заводу теж пов.язана з ID, щоб було зрозуміло, який завод конкретно було придбано. Була розроблена сутність заводу, яка буде тримати в собі інформацію:

```
[Serializable]
ссылка: 1
public class FactoryShopEntity
{
    public int id;
    public string name;
    public int price;
    public bool unlocked;
}
```

Рисунок 1.34. Код сутності заводу

- Id – ідентифікатор заводу, якій купляє гравець
- Name – назва заводу, якій купляє гравець
- Price – ціна заводу, якій купляє гравець
- Unlocked – статус заводу, чи є він відкритим

На ігровій сцені було заповнено ці поля даними:

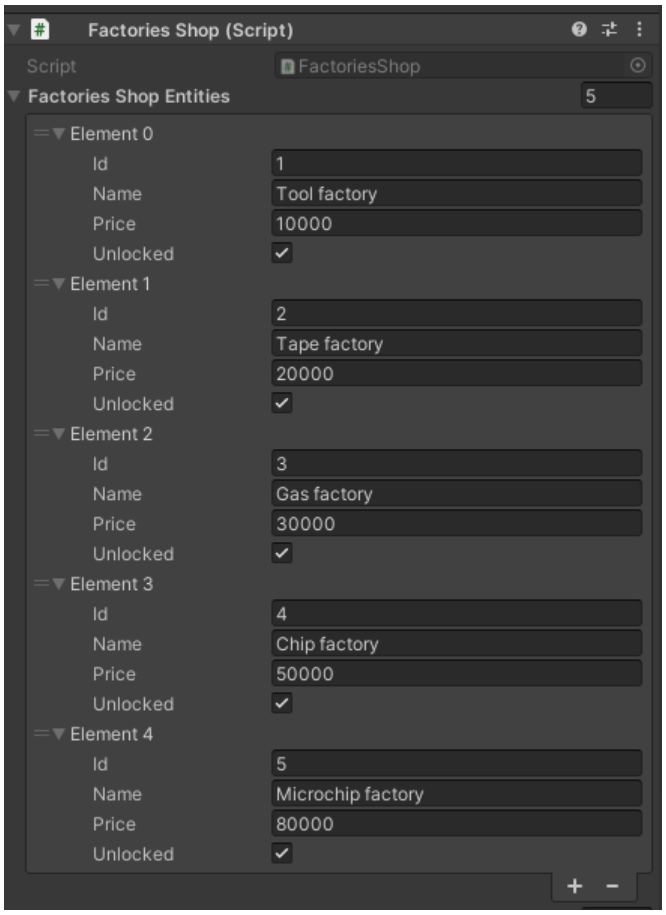


Рисунок 1.35. Вигляд конфігурації магазину

```
public class ShopBase : MonoBehaviour
{
    public event Action<ShopBase, int> OnPlayerEnter;
    public event Action OnPlayerExit;


    Сообщение Unity | Ссылок: 0
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            OnPlayerEnter?.Invoke(this, 1);
        }
    }
    Сообщение Unity | Ссылок: 0
    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            OnPlayerExit?.Invoke();
        }
    }
}
```

Рисунок 1.36. Код покупки в магазині

Код придбання заводу виглядає наступним чином :

```
public class FactoryBuySpot : MonoBehaviour
{
    [SerializeField] private int id;

    public event Action<int> OnBuy;
    public event Action<int, bool> OnActive;

    private bool _isActive;

    private void Update()
    {
        if (!_isActive)
        {
            return;
        }

        if (Input.GetKeyDown(KeyCode.E))
        {
            OnBuy?.Invoke(id);
        }
    }


    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _isActive = true;
            OnActive?.Invoke(id, true);
        }
    }


    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _isActive = false;
            OnActive?.Invoke(id, false);
        }
    }
}
```

Рисунок 1.37.Код функціонала магазину.

Конфігурація системи покращень виглядає наступним чином:

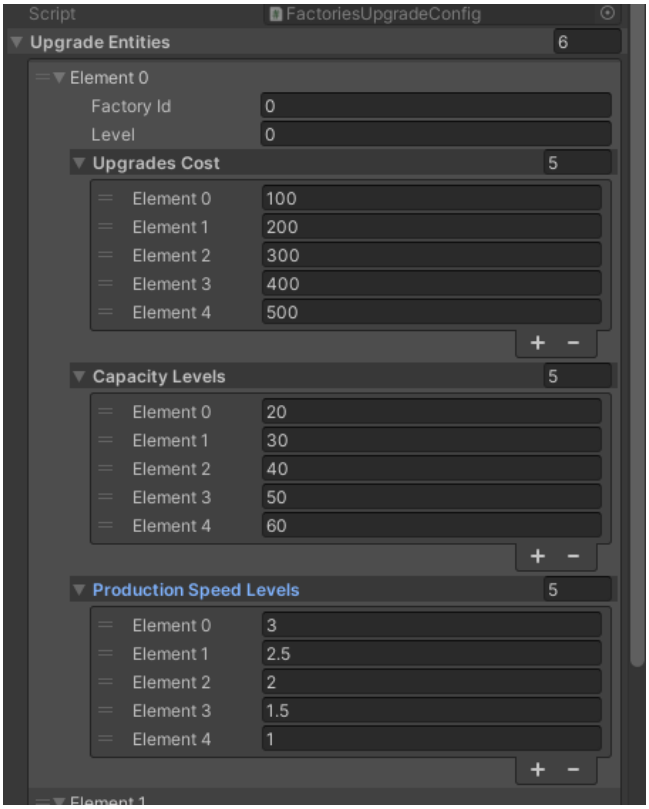


Рисунок 1.38. Вигляд конфігурації системи покращень в inspector.

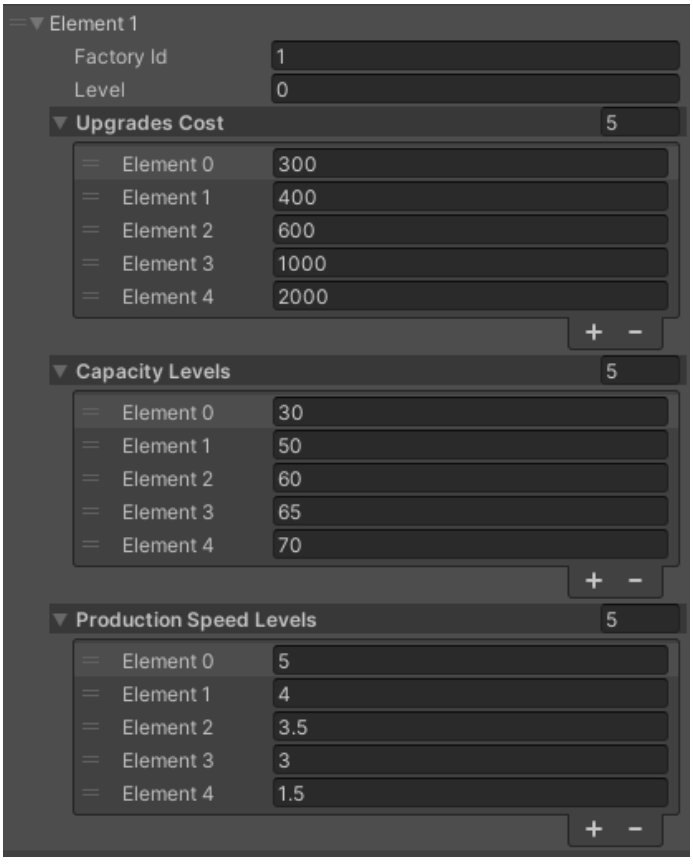


Рисунок 1.39. Вигляд конфігурації системи покращень в inspector.

Конфігурація містить в собі інформацію:

- Id – ідентифікатор заводу, якій покращується.
- Level – поточний рівень покращеня заводу
- Upgrade cost – цини покращеннь
- Capacity levels – максимальна вмістимість після покращення
- Production speed levels – швидкість виробництва після покращення.

1.8 Реалізація системи квестів.

Систему квестів реалізовано по тому ж принципу, що й попередні системи. Необхідним є інформація про предмети, які необхідні для квету, необхідна їх кількість для того, щоб завершити квест та його ID.

Фрагмент коду QuestController скрипта:

```
Скрипт Unity | Ссылок: 0
public class Quest : MonoBehaviour
{
    public int id;
    public List<SideQuest> sideQuests;
    public event Action OnGamePassed;


    Сообщение Unity | Ссылок: 0
    private void Awake()
    {
        InitializeQuest();
    }

    Сообщение Unity | Ссылок: 0
    private void Start()
    {
        InitializeSideQuest();
    }

    ссылка: 1
    private void InitializeQuest()
    {
        sideQuests.ForEach(q => q.OnQuestPassed += InitializeSideQuest);
    }

    Ссылок: 2
    private void InitializeSideQuest()
    {
        sideQuests.ForEach(q => q.QuestIsActive = false);
        var gameIsPassed = !sideQuests.Any(q => q.questIsPassed != true);
        if (gameIsPassed)
        {
            OnGamePassed?.Invoke();
            Debug.Log("GAME IS PASSED");
            return;
        }
        var quest = sideQuests.FirstOrDefault(q => !q.questIsPassed && !q.QuestIsActive).QuestIsActive = true;
    }
}
```

Рисунок 1.40. Код компоненту QuestController

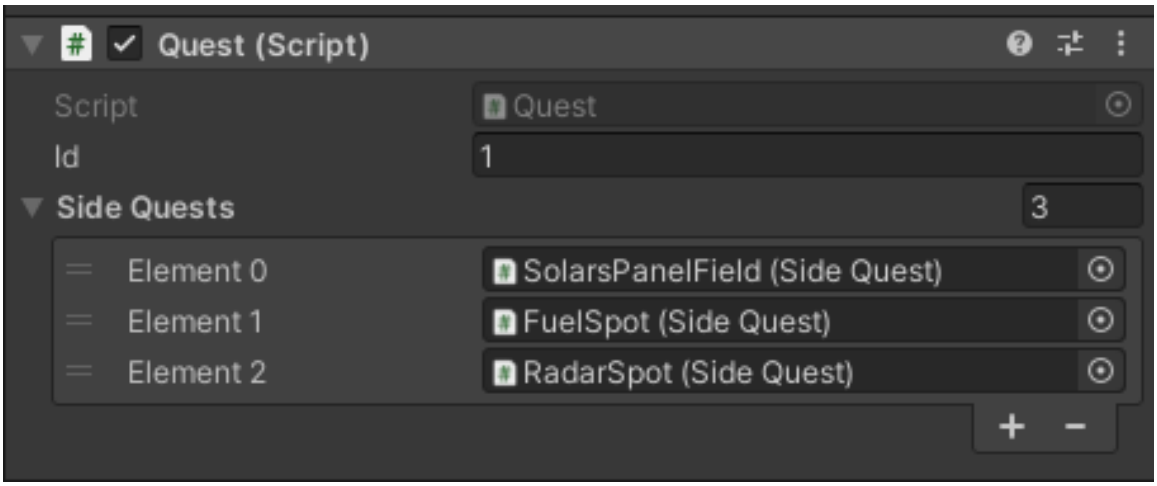


Рисунок 1.41. Конфігурація скрипту QuestController

Фрагмент коду QuestSpot скрипта, який обробляє взаємодію з гравцем:

```
Скрипт Unity | ссылка: 1
public class QuestSpot : MonoBehaviour
{
    [SerializeField] private string textTemplate;
    [SerializeField] private SideQuest sideQuest;
    [SerializeField] private FactoryView spotView;
    [SerializeField] private ItemsConfigsHolder itemsConfigs;
    [SerializeField] private GameObject ui;

    public event Action<SideQuest> OnPlayerEnter;
    public event Action OnPlayerExit;

    Сообщение Unity | Ссылок: 0
    private void Awake()
    {
        sideQuest.OnQuestActive += InitializeSpot;
        sideQuest.OnQuestPassed += DeinitializeSpot;
        sideQuest.OnQuestUpdate += UpdateView;
    }

    ссылка: 1
    private void InitializeSpot()
    {
        ui.SetActive(true);
        var icon = itemsConfigs.itemsConfigs.Find(x => x.id == sideQuest.ItemForQuest.id).sprite;
        spotView.gameObject.SetActive(true);
        spotView.SetText(string.Format(textTemplate, sideQuest.CurrentAmount, sideQuest.NeededAmound));
        spotView.SetSprite(icon);
    }

    ссылка: 1
    private void UpdateView()
    {
        spotView.SetText(string.Format(textTemplate, sideQuest.CurrentAmount, sideQuest.NeededAmound));
    }

    ссылка: 1
    private void DeinitializeSpot()
    {
        spotView.gameObject.SetActive(false);
        ui.SetActive(false);
    }
```

Рисунок 1.42. Код компоненту QuestSpot

Система квестів на ігровій сцені має вигляд:

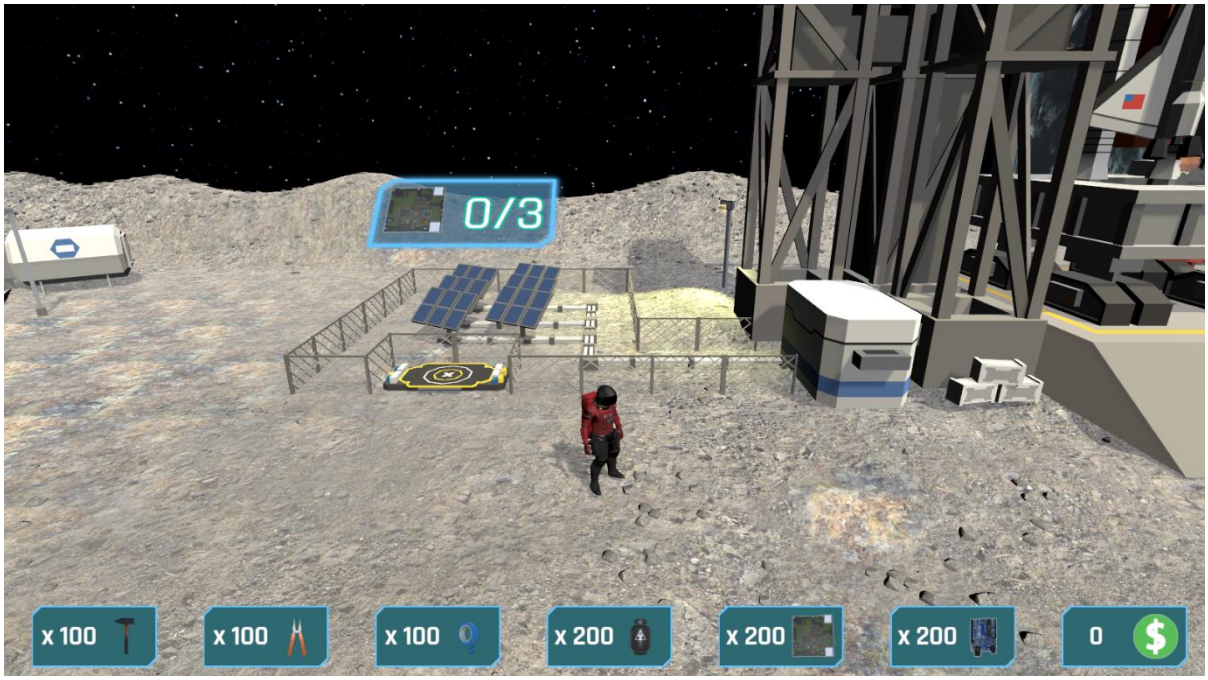


Рисунок 1.43. Зовнішній вигляд квесту.

Рисунок 1.44. Зовнішній вигляд квесту з наявним прогресом.

1.9 Реалізація UI частини головного меню

Задля реалізації візуальної частини головного меню – застосовувалися доступні асети та 2D матеріали. Приклад матеріалів для головного меню:

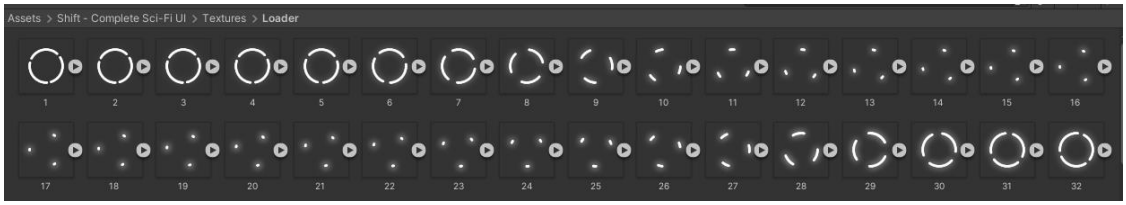


Рисунок 1.45. Матеріали для реалізації інтерфейсу головного меню

Рисунок 1.46. Іконки для реалізації інтерфейсу головного меню

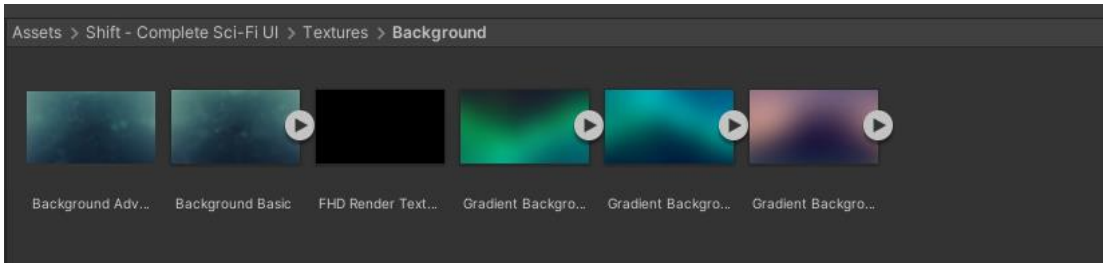


Рисунок 1.47. Фонові матеріали для головного меню

Кінцевий результат головного меню має вигляд:

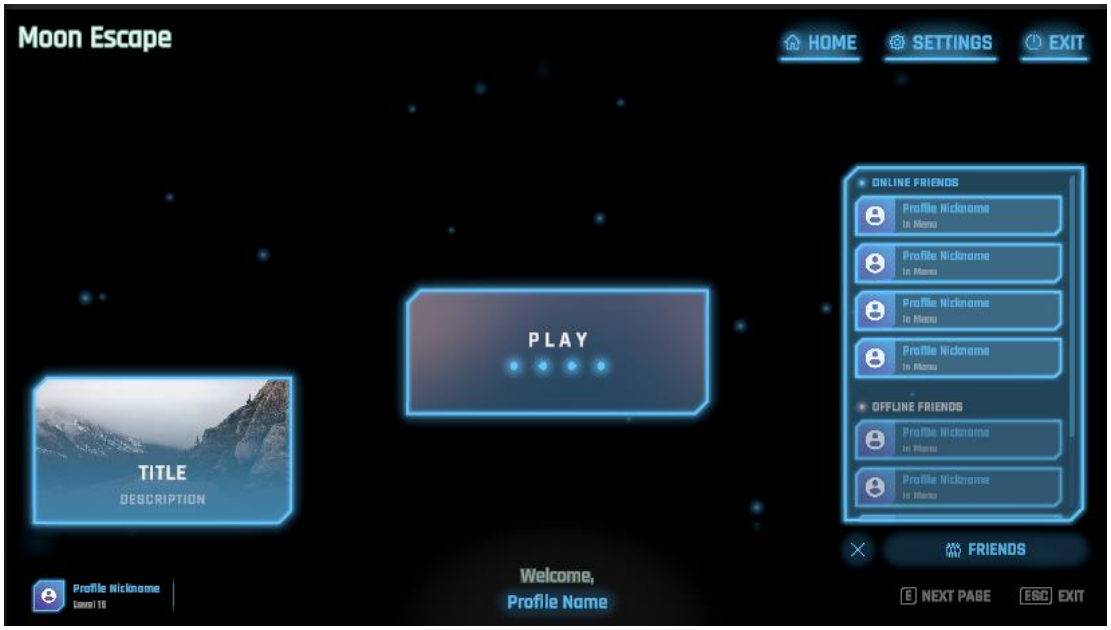


Рисунок 1.48. Кінцевий результат головного меню

1.10 Тестування на виявлення критичних помилок.

1.10.1 Тестування переміщення головного герою по сцені.

Функціональне тестування відбуватиметься на тестовій сцені з різними вхідними даними, для запобігання помилок при динамічній зміні даних.

Тестування переміщення головного герою по сцені включає в себе декілька етапів:

- Наявність взаємодії з гравітацією.
- Наявність взаємодії з колайдерами навколишнього середовища.
- Наявність анімацій при переміщені.

Вхідні дані, при яких проводилось функціональне тестування переміщення головного героя по мапі:

Рисунок 1.49. Вхідні дані для компоненту Player Movement.

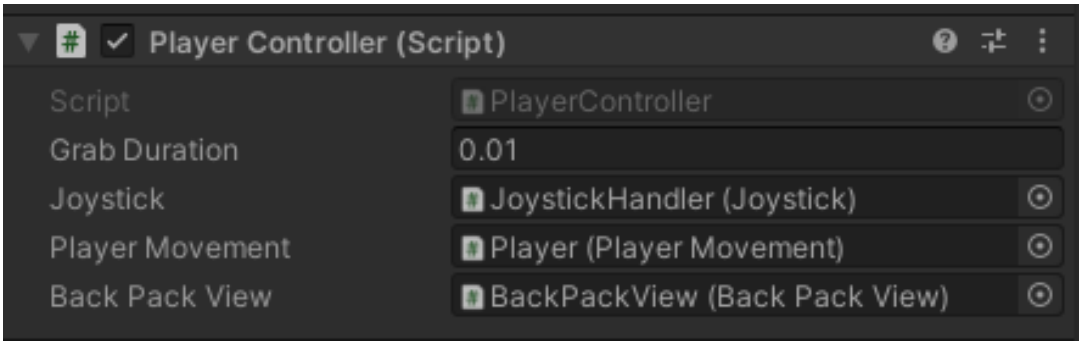


Рисунок 1.50. Вхідні дані для компоненту Player Movement

Тестування гравітації та з колізією між головним героєм та навколишнім середовищем:

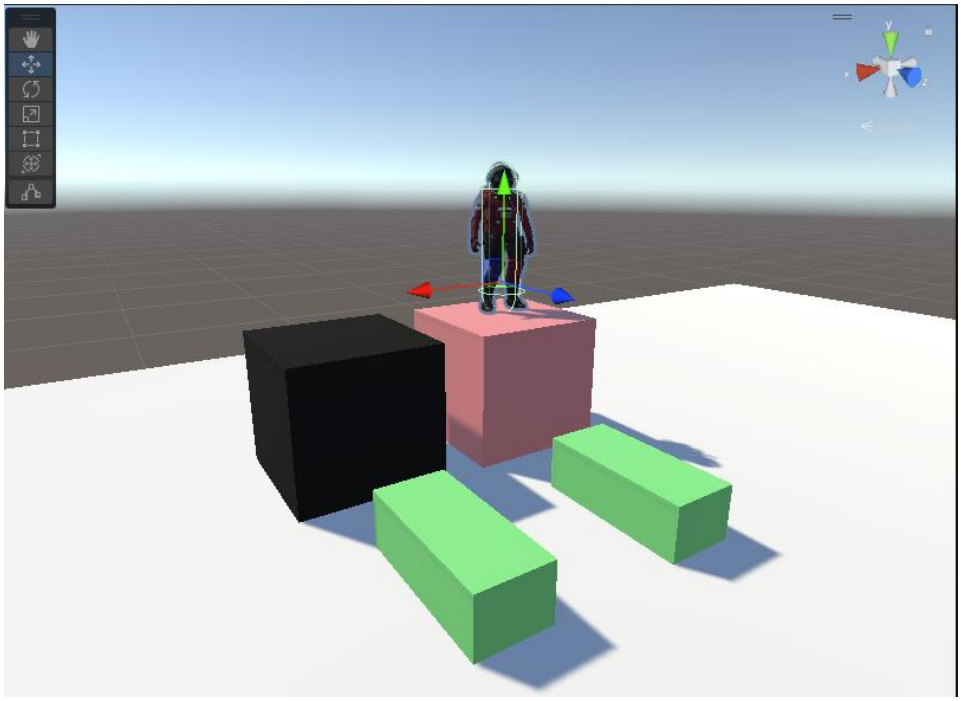


Рисунок 1.51. Тестування гравітації та колізії

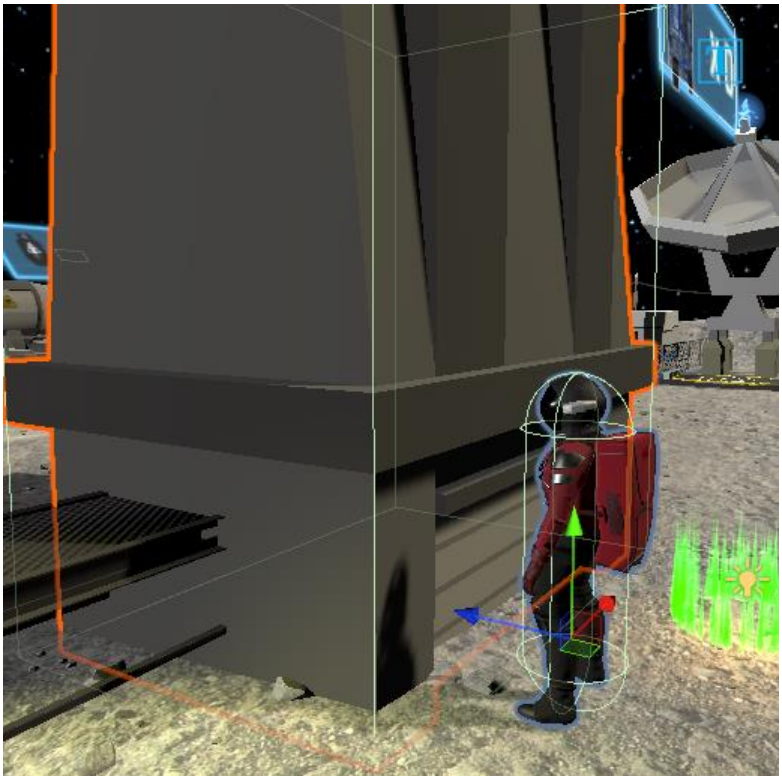


Рисунок 1.52. Тестування гравітації та колізії

Тестування анімації при переміщенні головного герою:

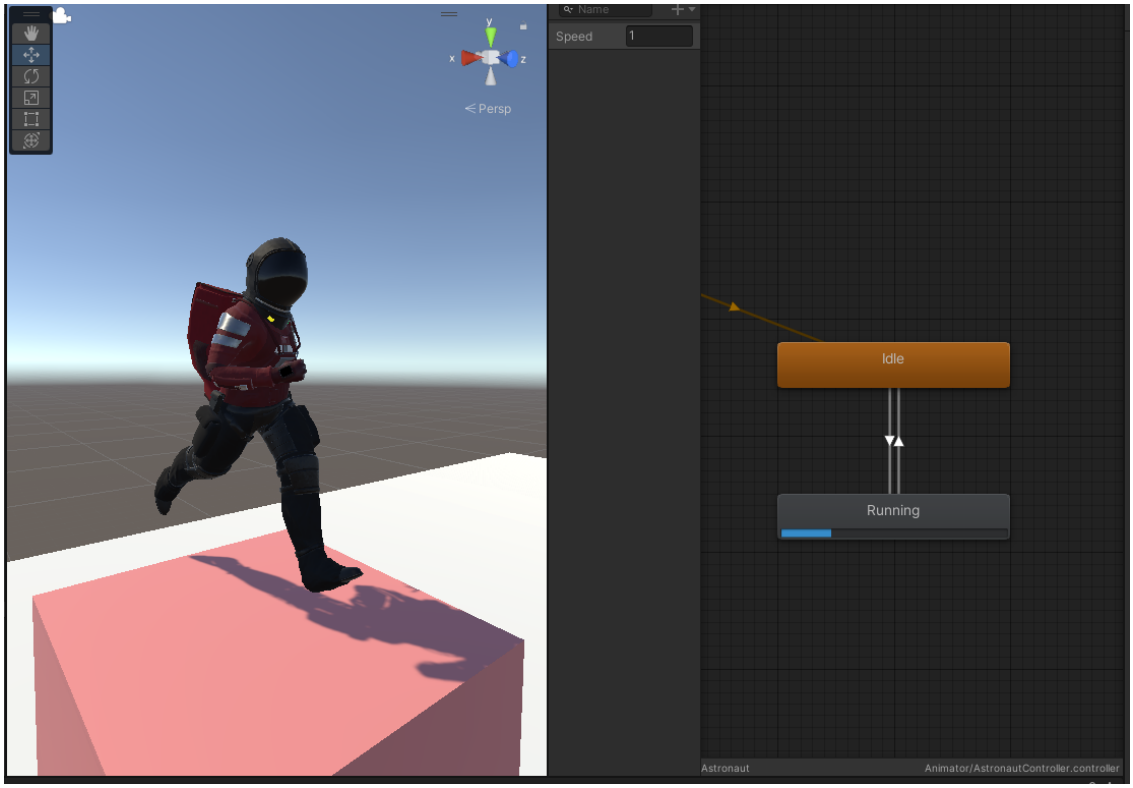


Рисунок 1.53. Тестування анімацій

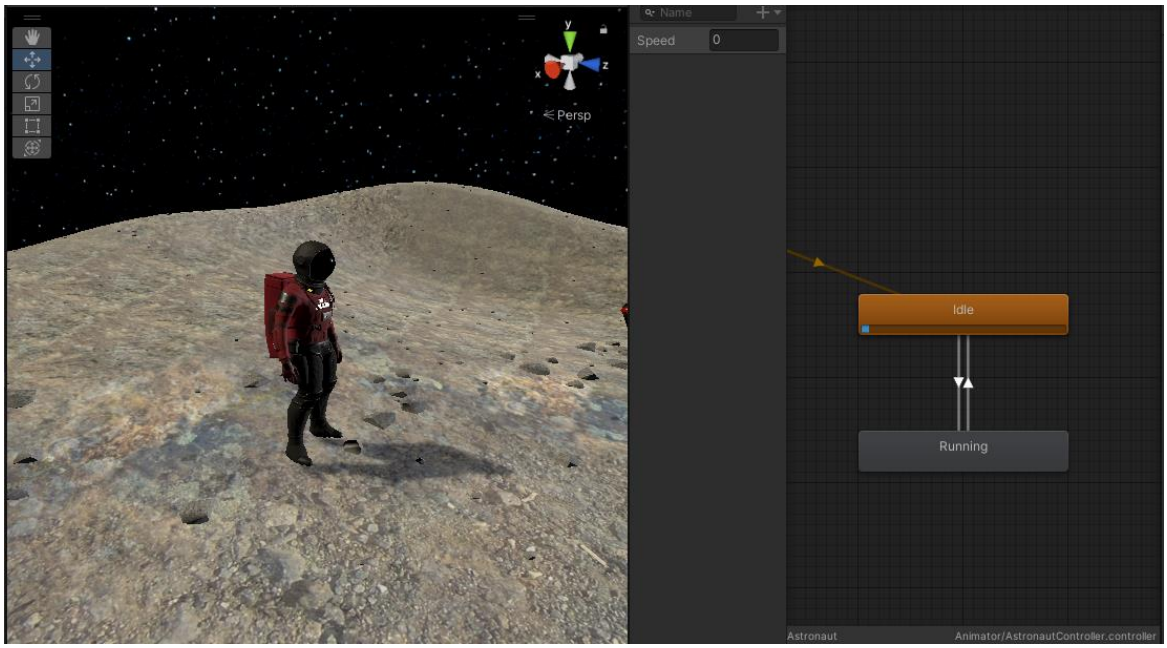


Рисунок 1.54. Тестування анімацій

1.10.2 Тестування заводу та складів.

Тестування системи заводів та складів включає в себе:

- Можливість мати динамічні змінні вмісткості та швидкості виробництва одиниці продукту.
- Можливість мати різноманітні комбінації рецептів.
- Можливість взаємодії головного герою зі складами.

Вхідні дані тестового заводу:

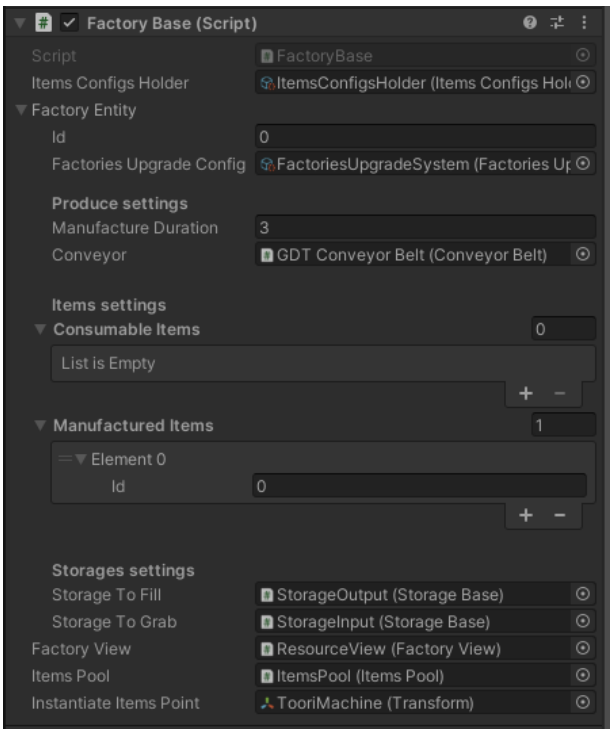


Рисунок 1.55. Вхідні дані тестового заводу

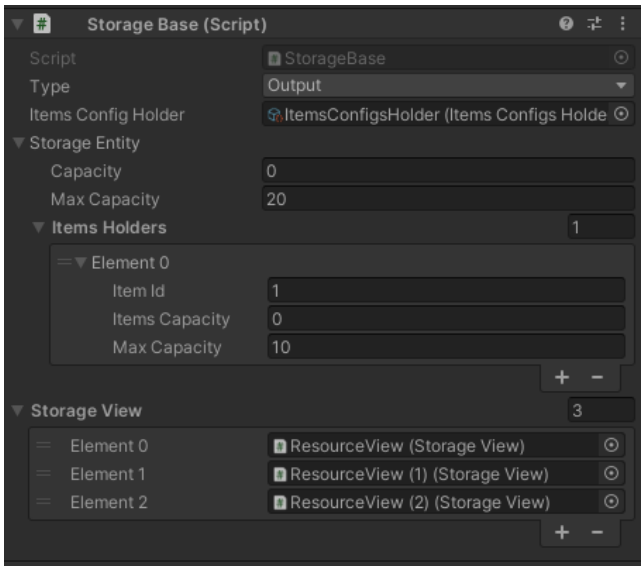


Рисунок 1.56. Вхідні дані тестового заводу

Тестування підтримки динамічних змінних місткості та швидкості виробництва:

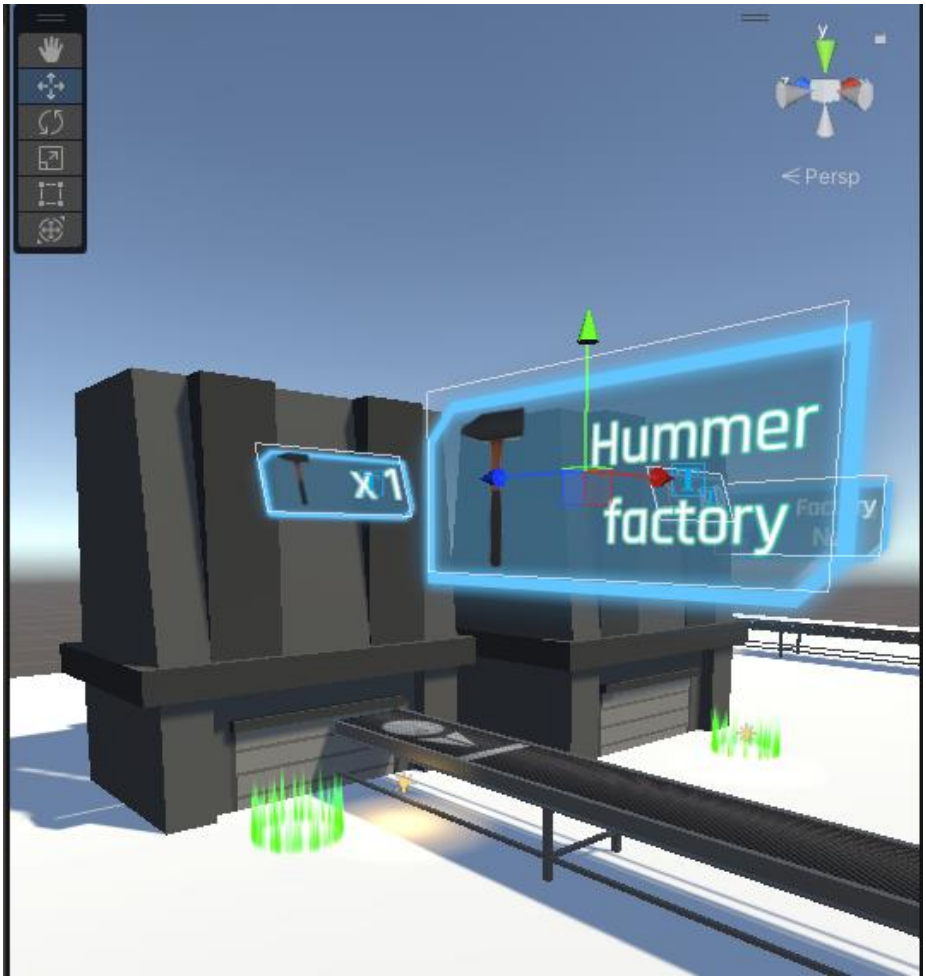


Рисунок 1.57. Тестування динамічних змінних заводу

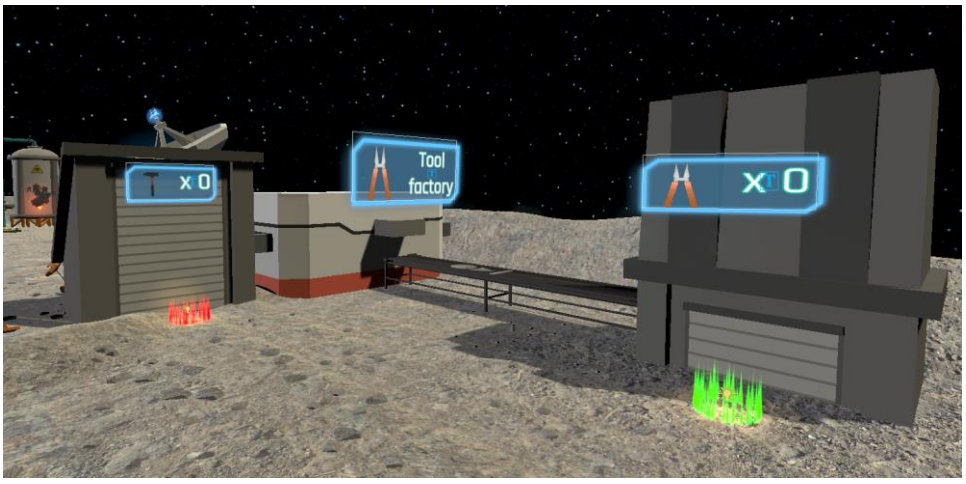


Рисунок 1.58. Тестування динамічних змінних заводу

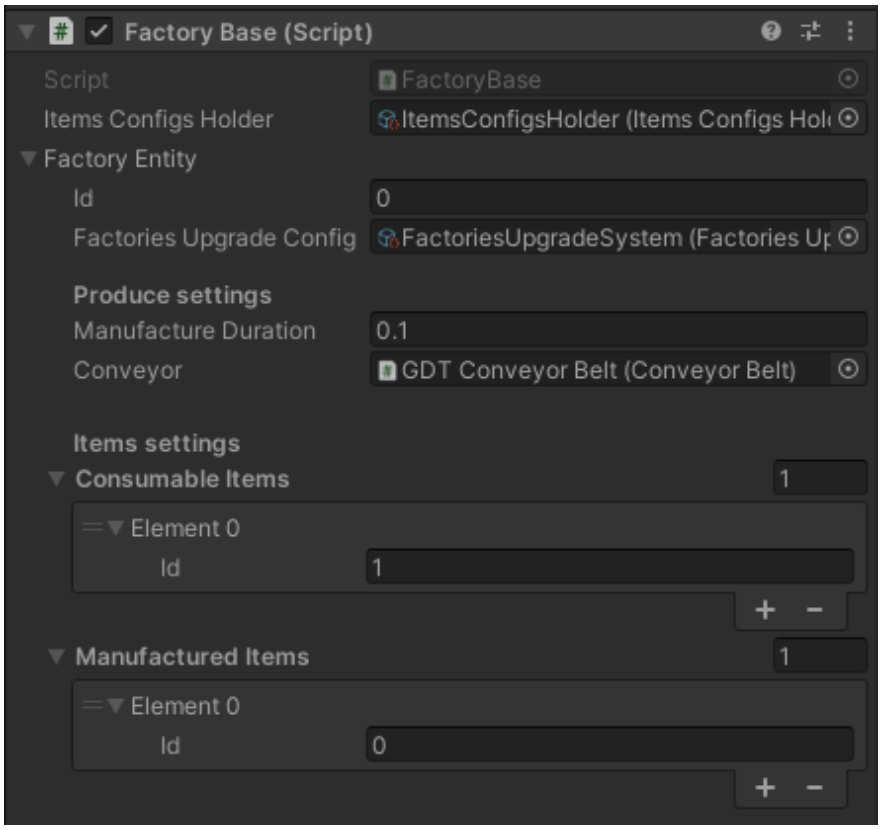


Рисунок 1.59. Тестування динамічних змінних заводу (тестові дані)

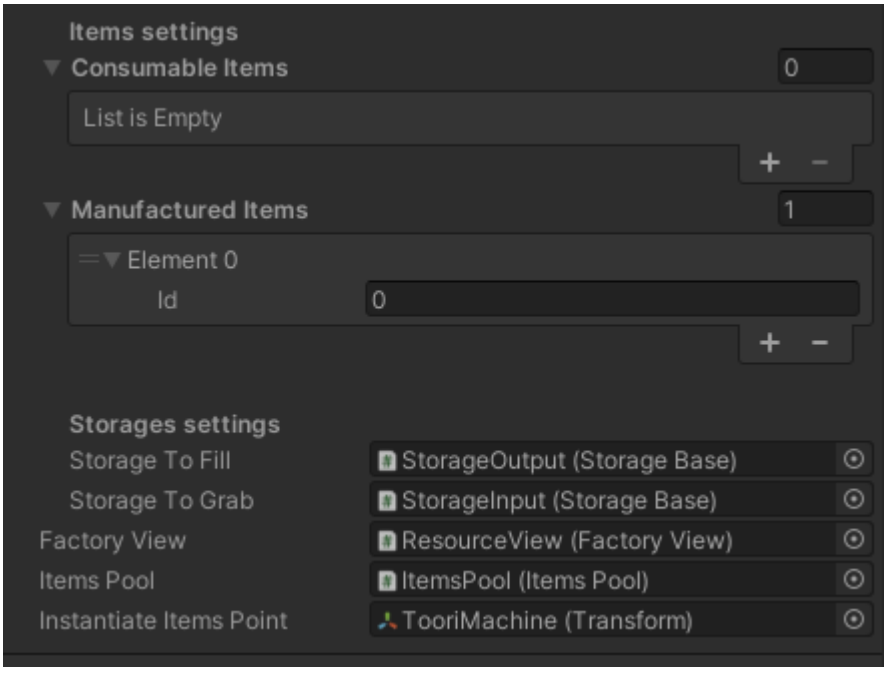


Рисунок 1.60. Тестування динамічних змінних заводу (тестові дані)

Тестування підтримки динамічних рецептів для предметів:

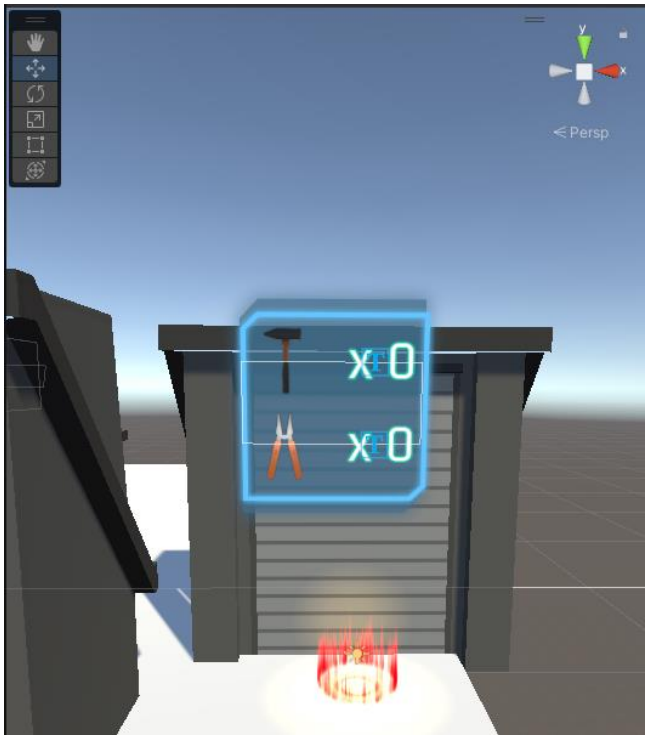


Рисунок 1.61. Тестування динамічних рецептів заводу (вид на сцені)

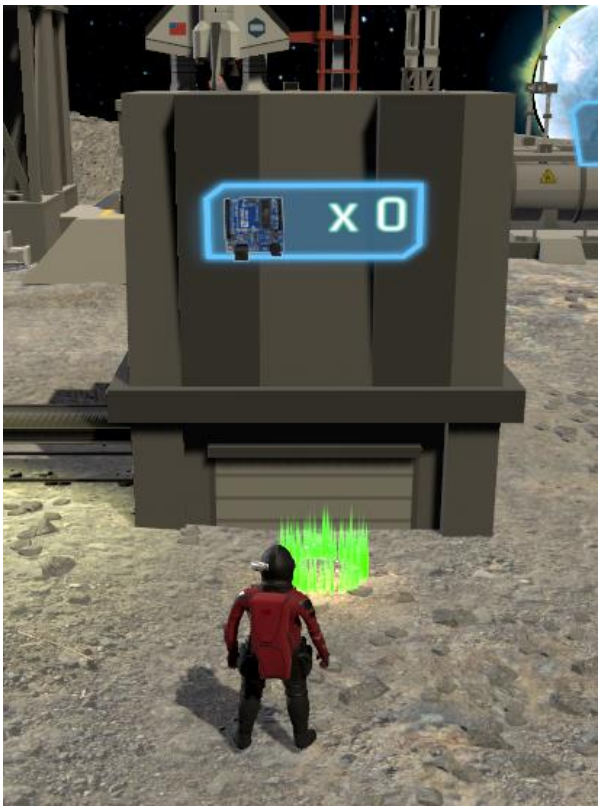


Рисунок 1.62. Тестування динамічних рецептів заводу (вид на сцені)

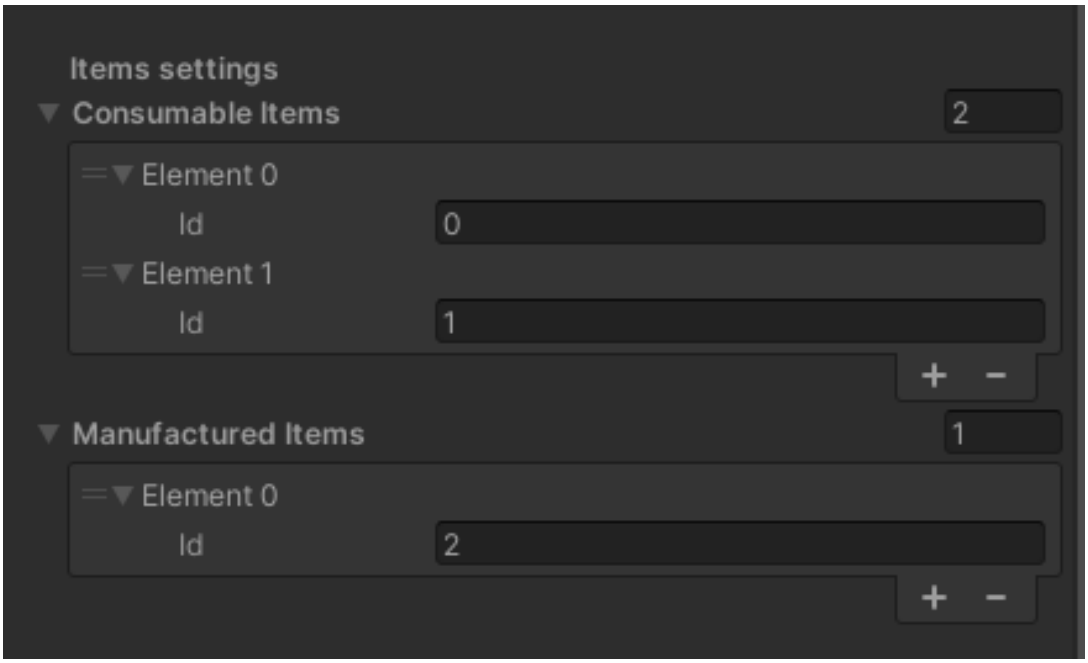


Рисунок 1.63. Тестування динамічних рецептів заводу (конфігурація)

Тестування взаємодії головного героя зі складом:

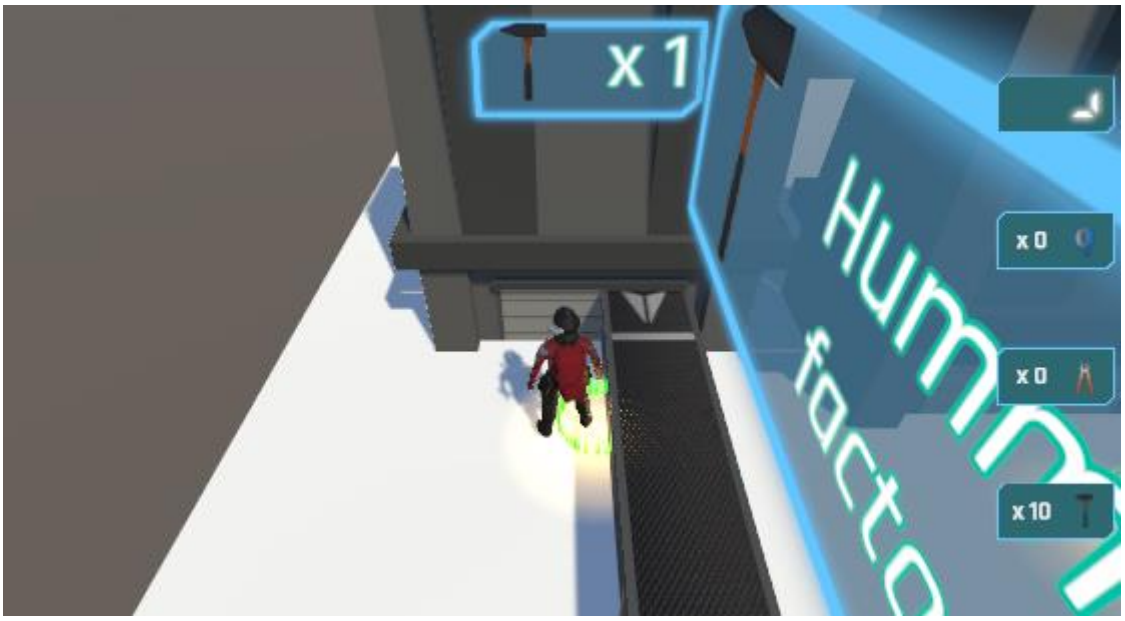


Рисунок. 1.64. – тестування тригерів складу

Головний функціонал гри є надійним та готовим до динамічних подій, які можуть бути відтворені під час гри. Змінні можуть приймати різні значення в процесі гри. Це не провокує критичні помилки, які можуть заважати проходженню гри.

2 ОХОРОНА ПРАЦІ

2.1 Аналіз та безпека умов праці працівника на робочому місці

Охорона праці грає ключову роль сучасному виробництві, оскільки її метою є забезпечення безпеки та здоров'я працівників. Розробка заходів з охорони праці в рамках дипломного проекту є необхідною, оскільки вона спрямована на покращення умов праці та запобігання можливим ризикам та нещасним випадкам.

Основні завдання охорони праці ґрунтуються на Конституції України, Законі України про охорону праці, засадах трудового законодавства та інших законодавчих актах. Вони включають забезпечення безпеки та здоров'я працівників, запобігання професійним захворюванням, визначення норм та вимог щодо умов праці, регулювання організаційних питань та здійснення контролю за дотриманням правил охорони праці.

Для аналізу умов праці в рамках бакалаврської роботи було обрано звичайне робоче місце в офісі за комп'ютером. Вибір актуальний, оскільки робота за комп'ютером, часто, має на увазі перебування в офісі2.1 Аналіз та безпека умов праці працівника на робочому місці

2.1.1 Організація робочого місця

Для аналізу умов праці та оцінки потенційно небезпечних та шкідливих факторів, які можуть виникати при використанні обладнання, необхідно враховувати різні аспекти, включаючи фізичні, хімічні, біологічні та психофізіологічні.

Фізичні фактори:

- Неправильна організація робочого місця може призвести до незручних положень тіла та неправильної постави, що може спричинити проблеми з м'язами та скелетною системою.

- Недостатнє освітлення може призвести до напруження очей та погіршення зору.
- Недостатня вентиляція та провітрювання приміщення можуть призвести до накопичення шкідливих речовин у повітрі та погіршення його якості.

Психофізіологічні фактори:

- Тривала робота перед комп'ютером може призвести до втоми очей та розвитку синдрому комп'ютерного зору.
- Відсутність адекватних перерв та відпочинку може викликати психологічну напругу, стрес та виснаження.

Усі елементи робочого місця та їхнє взаємне розташування повинні відповідати ергономічним вимогам, що враховують характер та особливості трудової діяльності. Ці вимоги визначаються стандартами, як-от ГОСТ 12.2.032-78, ГОСТ 22.269-76 і ГОСТ 21.889-76.

Рекомендується розміщувати робочі місця таким чином, щоб природне світло падало збоку, переважно зліва. Для оптимальної відстані від очей користувача екран має бути розташований на відстані 650-750 мм, але не менше 600 мм, з урахуванням розміру символів на екрані. Монітор слід розміщувати прямо перед працівником на рівні очей або трохи нижче.

Для забезпечення комфортного робочого простору ширина робочого столу повинна бути не менше 120 см, а глибина – не менше 80 см. Висота столу повинна бути в діапазоні від 70 до 75 см, а висота стільця – від 40 до 50 см, щоб підтримувати пряму спину . працівника, а підлокітники стільця надавали підтримку для плечей та зап'ясть.

Клавіатуру слід розміщувати на поверхні столу на відстані 100-300 мм від працівника, який звернувся до неї. Конструкція клавіатури повинна передбачати підставку, що дозволяє регулювати кут нахилу поверхні клавіатури в діапазоні від 5 до 15 градусів.

Комп'ютерна миша повинна бути розташована поряд з клавіатурою на такій висоті, щоб плечі були розслаблені, а зап'ястки не були під тиском або в неприродному положенні. Рекомендується кут нахилу миші близько 20-30 градусів.

Працівник повинен мати підтримку для нижньої частини спини та підтримувати правильну поставу. Рекомендується робити перерви кожні 30-60 хвилин роботи за комп'ютером для розтяжки та фізичних вправ. При перервах рекомендується виконувати прості фізичні вправи, розтяжку м'язів та відпочинок для очей.

Для одного комп'ютера рекомендується виділяти не менше 6 м2 площі, а об'єм приміщення повинен бути не менше 24 м3. Фактичні значення цього робочого місця: площа робочого місця 7 м², обсяг приміщення 120 м³.

2.1.2 Вимоги безпеки до мікроклімату виробничих приміщень, освітлення та шуму

Нормативні документи, які визначають параметри мікроклімату виробничих приміщень, включають ДСН 3.3.6.042-99 та ГОСТ 12.1.005-88. Ці параметри регулюються для робочої зони, яка є визначеною областю, де розташовані робочі місця. Згідно з нормативним документом ДСН 3.3.6.042-99 "Санітарні норми мікроклімату виробничих приміщень", температура повітря в приміщенні повинна бути у діапазоні 21-26 °С, а рекомендована вологість повітря становить 40-60%.

Робочі приміщення мають бути належним чином оснащені природним і штучним освітленням, щоб забезпечити достатню якість і кількість світла на робочих місцях.

Природне освітлення здійснюється через вікна, переважно орієнтовані на схід. Штучне освітлення в приміщенні забезпечується системою загального рівномірного освітлення. Рівень освітлення на робочому місці за домашнім комп'ютером повинен бути в діапазоні приблизно від 300 до 500 люкс, щоб забезпечити комфортну і ефективну робочу атмосферу. Розташування освітлення повинно бути таким, щоб уникнути бліків на робочій поверхні та утворення непотрібних тіней.

З метою запобігання виникненню шумів відповідно до ГОСТ 12.1.029-80 "Захист від шуму. Загальні вимоги безпеки", в приміщенні передбачається звукоізоляція вікон і дверей для зниження шуму і вібрації.

2.2 Пожежна безпека

У робочому приміщенні та на робочих місцях присутні речовини та матеріали, які можуть бути небезпечними через свою вибухонебезпечність або здатність спричинити пожежу. Серед таких речовин можуть бути електричні розетки, перемикачі світла, комп'ютер та його компоненти, кондиціонер, роутер, маршрутизатор, електричні пристрої та зарядні пристрої.

Ці речовини та матеріали мають певні характеристики, які створюють вибухонебезпеку або можуть спричинити пожежу. Наприклад, електричні розетки та перемикачі світла можуть викликати коротке замикання або загоряння через пошкоджену проводку, вологу або перевантаження мережі. Комп'ютер та його компоненти можуть перегріватися через несправну систему охолодження, пил або перевантаження компонентів, що може призвести до загоряння або короткого замикання.

Такі ж проблеми можуть виникнути із кондиціонером, роутером, різними електричними пристроями та зарядними пристроями. Аналізуючи можливі місця та причини загорянь та вибухів у приміщенні, виявлено, що такі проблеми можуть виникнути через неправильне використання або несправність

електроприладів, перевантаження електричної мережі, несправність або пошкодження електропроводки, використання неякісних або несумісних зарядних пристроїв та недотримання правил експлуатації побутової техніки. Приміщення віднесено до вибухонебезпечної зони В-IIа.

З метою запобігання пожежам та вибухам рекомендується мати наявність наступних засобів пожежогасіння: порошкові вогнегасники, які підходять для гасіння пожеж класу А, В і С, включаючи пожежі від електроприладів, комп'ютерів, кондиціонерів та інших електроустановок; вуглекислотні вогнегасники, які ефективні для гасіння пожеж класу В та С, включаючи пожежі від електроустановок, без пошкодження обладнання.

Для зниження ризику виникнення пожеж рекомендується підтримувати порядок та чистоту на робочих місцях, регулярно очищати їх від сміття та легкозаймистих матеріалів, а також правильно використовувати електроустаткування, уникати перевантажень та правильно підключати його.

ВИСНОВКИ

Розробка кваліфікаційної роботи на основі ігрового двигуна Unity є вдалим рішенням. На це вказують наступні фактори :

- Широкі можливості: Unity надає багатий набір інструментів та функціональності для створення різноманітних типів ігор, включаючи 2D та 3D проекти.
- Простота використання: Unity має дружній інтерфейс користувача і просту навчання, що робить його доступним для розробників з різним рівнем досвіду.

В ході виконання роботи було визначено основні етапи розробки, такі як проектування гри, створення ігрових механік, управління ресурсами та інтерфейс користувача, а також тестування та налагодження. Важливим аспектом при реалізації програмного продукту є застосування патерну проектування Factory Pattern, який надав можливості проекту бути гнучким для модернізації. На етап розробки архітектурної бази проекту було витрачено більше всього часу.

На розробку частини левелдизайну було також впроваджено певний інструментарій для полегшення роботи з редактором Unity, та прискоренням процесу відтворення ігрового світу. Були створені необхідні матеріали та тестури. Ця частина проекту була найбільш впливовою на продуктивність продукту, тому було впроваджено певні заходи щодо оптимізації 3D та 2D зіставної проекту.

Розробка ігрового інтерфейсу було проведено згідно зі стандартними дозволами екранів для PC, що дозволяє всьому інтерфейсу в грі бути гнучким для різних форматів екрану.

Увесь проект з самого початку був підключений до системи управління версіями GitHub, що дозволило вести проект декільком програмістів одночасно, не заважаючи та не збиваючи прогрес один одного.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Офіційний сайт Unity [Електронне джерело]: https://unity.com/
2. Unity Learn [Електронне джерело]: https://learn.unity.com/
3. Unity Asset Store [Електронне джерело]: https://assetstore.unity.com/
4. UnityForum[Електронне джерело]– форум
   URL https://forum.unity.com/
5. Metanit [Електронне джерело] – література
   URL - https://metanit.com/sharp/
6. Unreal Engine [Електронне джерело] – Ігровий двигун Unreal Engine
   URL – https://www.unrealengine.com/en-US
7. Godot [Електронне джерело] – Ігровий двигун Godot:
   URL https://godotengine.org/
8. Cocos2D [Електронне джерело]: https://www.cocos.com/en
9. Harrison, Ferrone. “Learning C# by Developing Games”
   Publishing; 7th ed. edition (November 29, 2022)
10. Лебланк, Хаppic. "Learning C# by Developing Games with Unity".
8. Jesse Schell: «The Art of Game Design: A Book of Lenses».
9. Kevin Saunders: «Game Development Essentials: Game Interface Design».
10. Joe Hocking: «Unity in action: Multiplatform game development with C#».
11. Alan Thorn: Unity Animation Essential

ДОДАТОК А

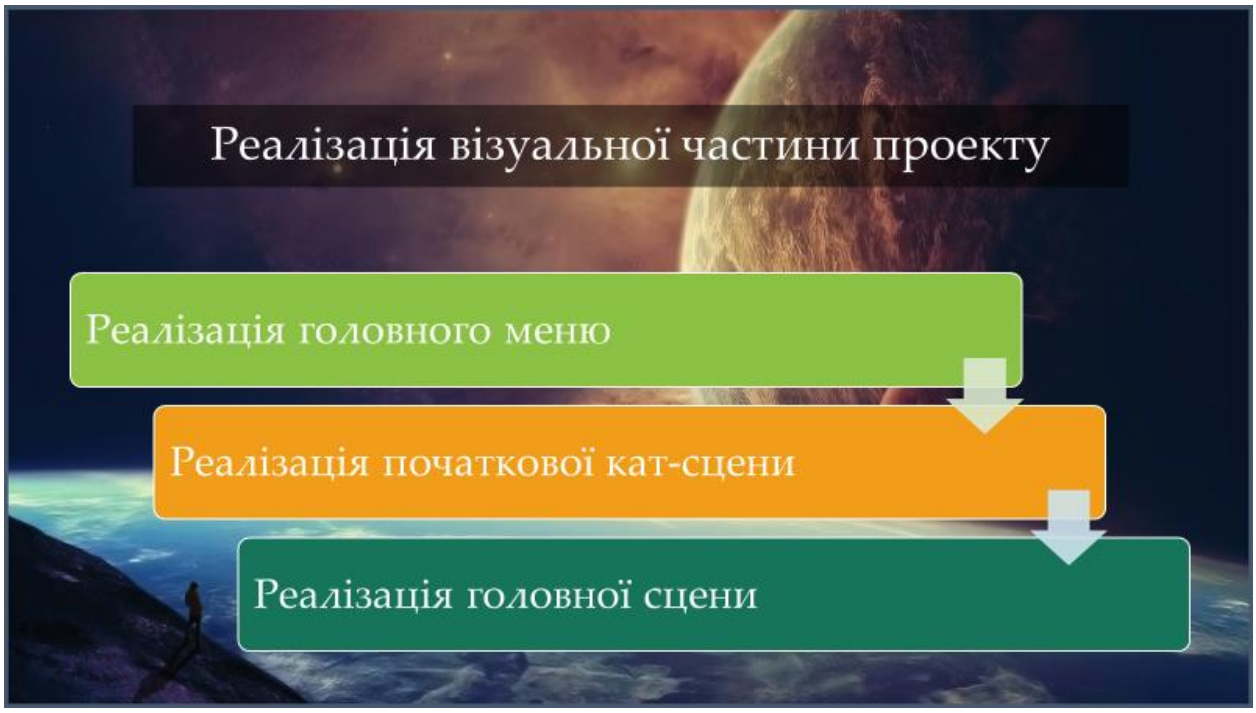
Скрипт створення системи гаманцю WalletController для головного героя.

```
public class WalletController : MonoBehaviour
    {
        [SerializeField] private ItemsConfigsHolder itemsConfigsHolder;
        [SerializeField] private List<WalletView> walletViews;
        [SerializeField] private WalletView balanceWalletView;
        [SerializeField] private WalletEntity walletEntity;
        [SerializeField] private Sprite moneyIcon;

        public int Balance
        {
            get => walletEntity.moneyBalance;
        }

        private void Start()
        {
            InitializeWallet();
        }

        private void InitializeWallet()
        {
            balanceWalletView.SetIcon(moneyIcon);
            balanceWalletView.SetText(walletEntity.moneyBalance);
            balanceWalletView.Activate();
            walletEntity.items.ForEach(i =>
            {
                var currentView = walletViews.Find(v => !v.IsActive());
                var  currentConfig  =  itemsConfigsHolder.itemsConfigs.Find(c  =>  c.id  ==
i.itemId);
                currentView.SetIcon(currentConfig.sprite);
                currentView.SetText(i.itemsCapacity);
                currentView.Id = i.itemId;
                currentView.Activate();
            });
        }

        public void IncreaseBalance(int value)
        {
            walletEntity.moneyBalance += value;
            balanceWalletView.SetText(walletEntity.moneyBalance);
        }

        public void DecreaseBalance(int value)
        {
            if (walletEntity.moneyBalance - value < 0)
            {
                return;
            }

            walletEntity.moneyBalance -= value;
            balanceWalletView.SetText(walletEntity.moneyBalance);
        }

        public void IncreaseWallet(int id, int value)
        {
            var currentItem = walletEntity.items.Find(i => i.itemId == id);
            currentItem.itemsCapacity += value;
```

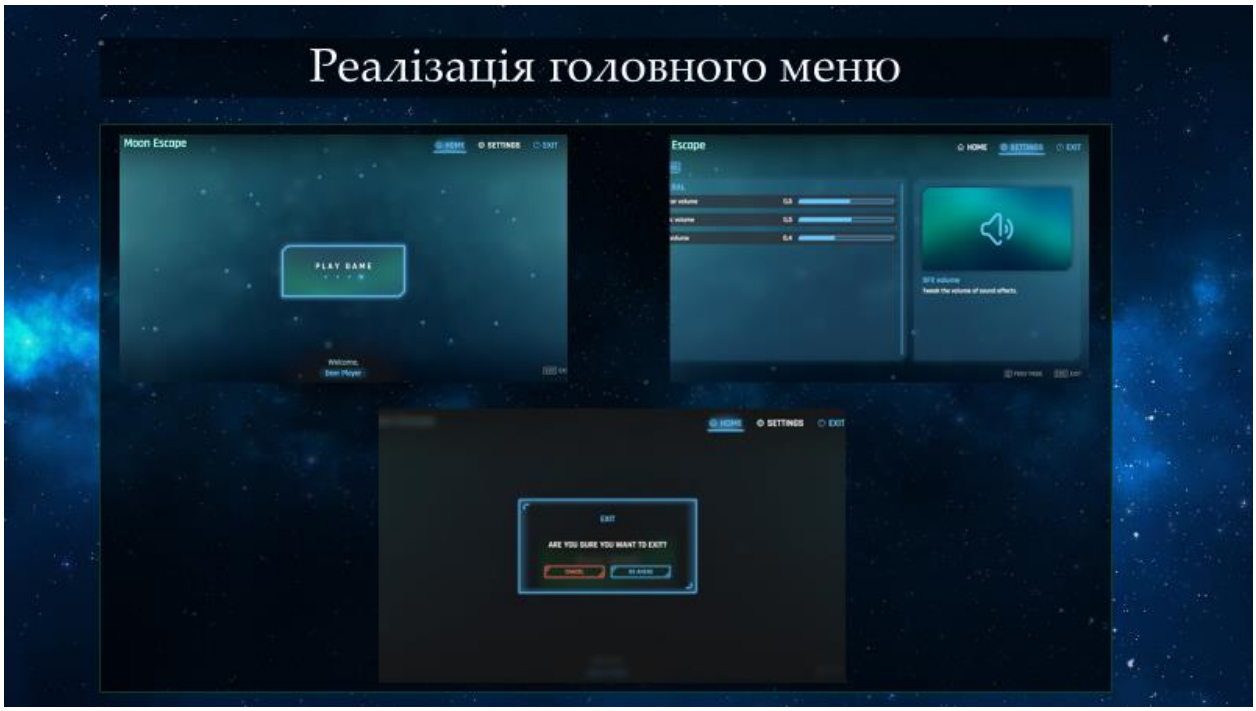
```
            walletViews.Find(v => v.Id == id).SetText(currentItem.itemsCapacity);
        }

        public void DecreaseWallet(int id, int value)
        {
            var currentItem = walletEntity.items.Find(i => i.itemId == id);
            currentItem.itemsCapacity -= value;
            walletViews.Find(v => v.Id == id).SetText(currentItem.itemsCapacity);
        }

        public bool HaveNoItems(int id)
        {
           return walletEntity.items.Find(i => i.itemId == id).itemsCapacity == 0;
        }

        public bool HaveNoMoney()
        {
            return walletEntity.moneyBalance == 0;
        }

        public List<int> GetExistingItemsIds()
        {
            List<int> ids = new List<int>();
            walletEntity.items.ForEach(i => ids.Add(i.itemId));
            return ids;
        }

        public bool StackIsFull(int id)
        {
            return   walletEntity.items.Find(i   =>   i.itemId   ==   id).itemsCapacity   ==
walletEntity.maxItemCapacity;
        }
    }
```

ДОДАТОК Б

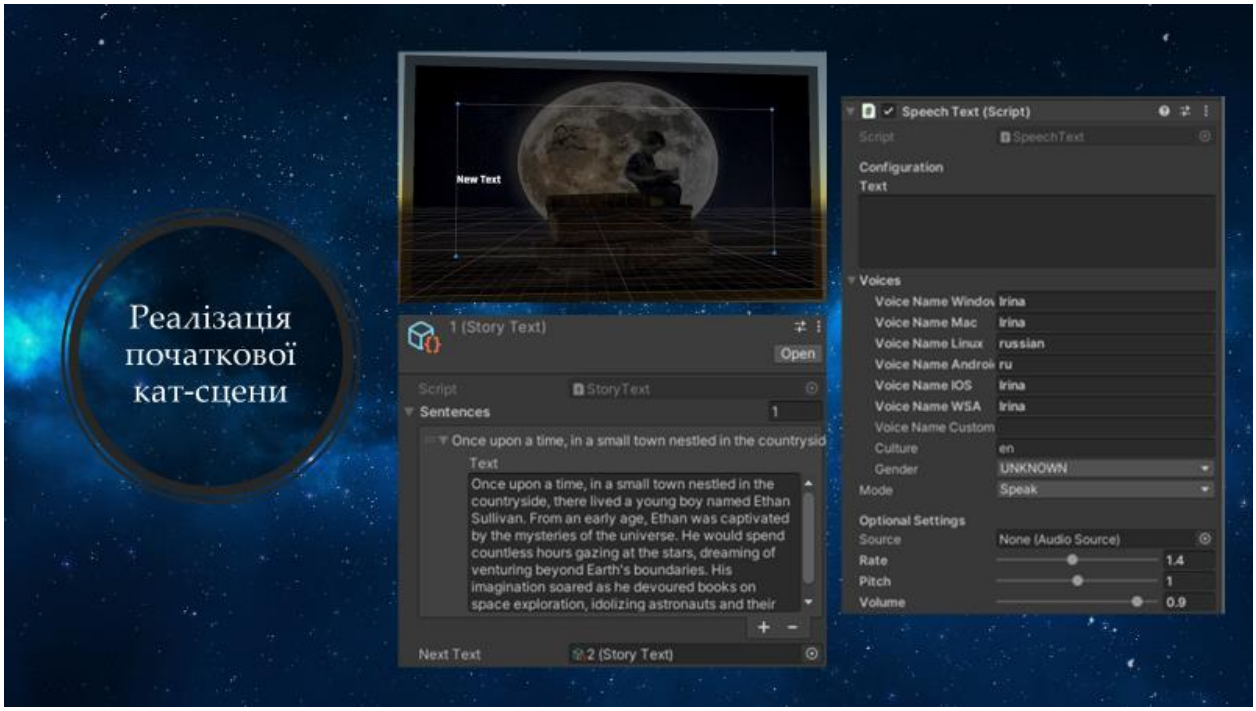
Слайди мультимедійної презентації

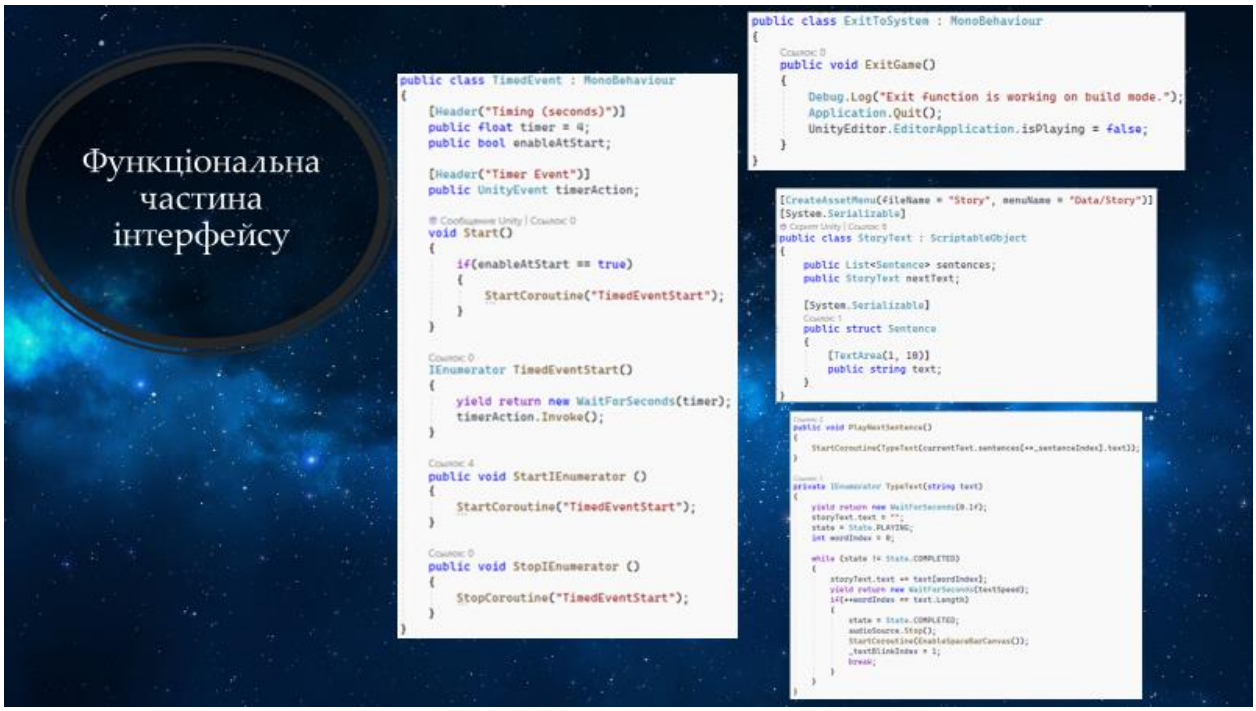


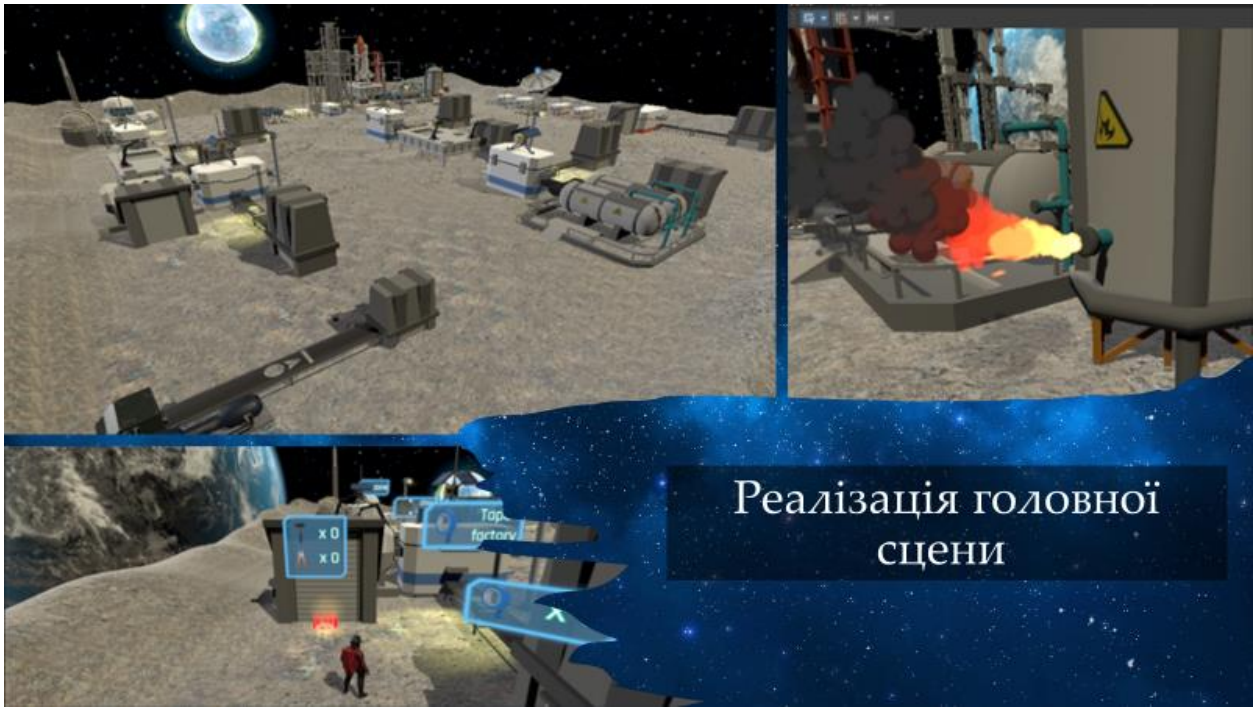
Реалізація головного меню

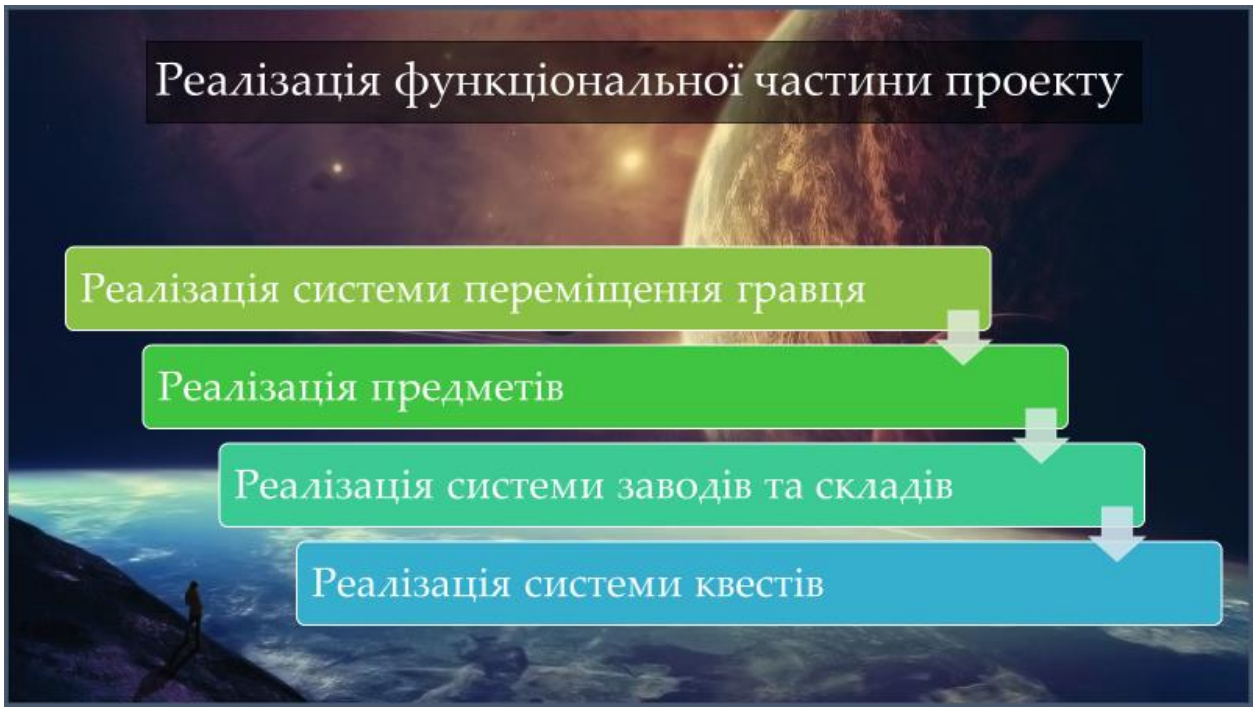


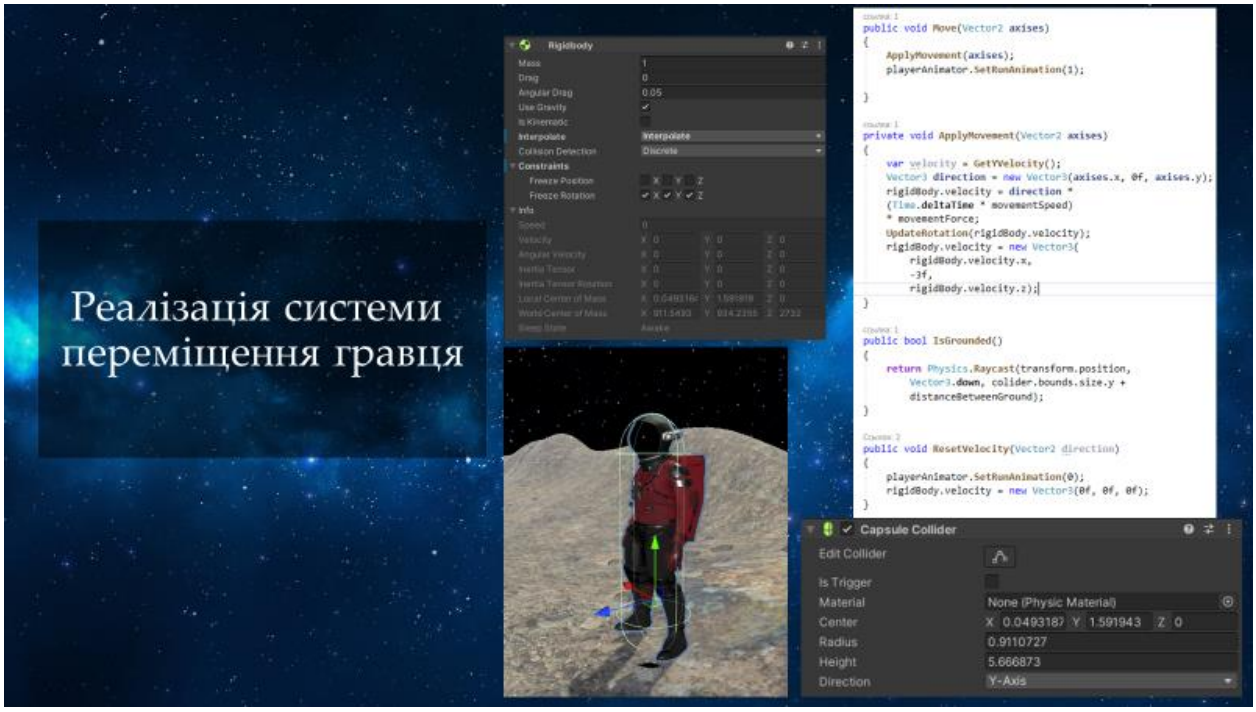
Реалізація початкової кат-сцени




Функціональна частина інтерфейсу
public class TimedEvent : MonoBehaviour
{
[Header("Timing (seconds)")]
public float timer = 4;
public bool enableAtStart;
[Header("Timer Event")]
public UnityEvent timerAction;
void Start()
{
if(enableAtStart == true)
{
StartCoroutine("TimedEventStart");
}
}
IEnumerator TimedEventStart()
{
yield return new WaitForSeconds(timer);
timerAction.Invoke();
}
public void StartIEnumerator ()
{
StartCoroutine("TimedEventStart");
}
public void StopIEnumerator ()
{
StopCoroutine("TimedEventStart");
}
}
public class ExitToSystem : MonoBehaviour
{
public void ExitGame()
{
Debug.Log("Exit function is working on build mode.");
Application.Quit();
UnityEditor.EditorApplication.isPlaying = false;
}
}
[CreateAssetMenu(fileName = "Story", menuName = "Data/Story")]
[System.Serializable]
public class StoryText : ScriptableObject
{
public List<Sentence> sentences;
public StoryText nextText;
[System.Serializable]
public struct Sentence
{
[TextArea(1, 10)]
public string text;
}
}
public void PlayNextSentence()
{
StartCoroutine(TypeText(currentText.sentences[++_sentenceIndex].text));
}
private IEnumerator TypeText(string text)
{
yield return new WaitForSeconds(0.1f);
storyText.text = "";
state = State.PLAYING;
int wordIndex = 0;
while (state != State.COMPLETED)
{
storyText.text += text[wordIndex];
yield return new WaitForSeconds(TextSpeed);
if(++wordIndex == text.Length)
{
state = State.COMPLETED;
audioSource.Stop();
StartCoroutine(EnableSpaceBarCanvas());
_textBlinkIndex = 1;
break;
}
}
}

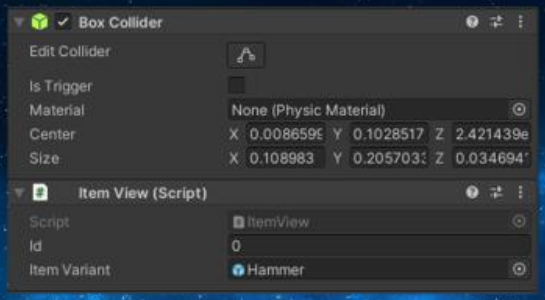

Реалізація головної сцени


Реалізація функціональної частини проекту
Реалізація системи переміщення гравця
Реалізація предметів
Реалізація системи заводів та складів
Реалізація системи квестів

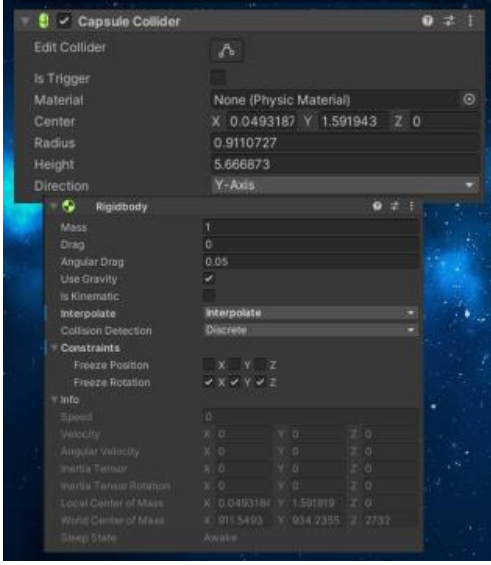

Реалізація системи переміщення гравця
RigidBody
Mass
Drag
Angular Drag
Use Gravity
Is Kinematic
Interpolate
Collision Detection
Constraints
Freeze Position
Freeze Rotation
Info
Speed
Velocity
Angular Velocity
Inertia Tensor
Inertia Tensor Rotation
Local Center of Mass
World Center of Mass
Sleep State
public void Move(Vector2 axises)
{
ApplyMovement(axises);
playerAnimator.SetRunAnimation(1);
}
private void ApplyMovement(Vector2 axises)
{
var velocity = GetYVelocity();
Vector3 direction = new Vector3(axises.x, 0f, axises.y);
rigidBody.velocity = direction *
(Time.deltaTime * movementSpeed)
* movementForce;
UpdateRotation(rigidBody.velocity);
rigidBody.velocity = new Vector3(
rigidBody.velocity.x,
-3f,
rigidBody.velocity.z);
}
public bool IsGrounded()
{
return Physics.Raycast(transform.position,
Vector3.down, collider.bounds.size.y +
distanceBetweenGround);
}
public void ResetVelocity(Vector2 direction)
{
playerAnimator.SetRunAnimation(0);
rigidBody.velocity = new Vector3(0f, 0f, 0f);
}
Capsule Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0.0493187 Y 1.591943 Z 0
Radius
0.9110727
Height
5.666873
Direction
Y-Axis

Реалізація предметів

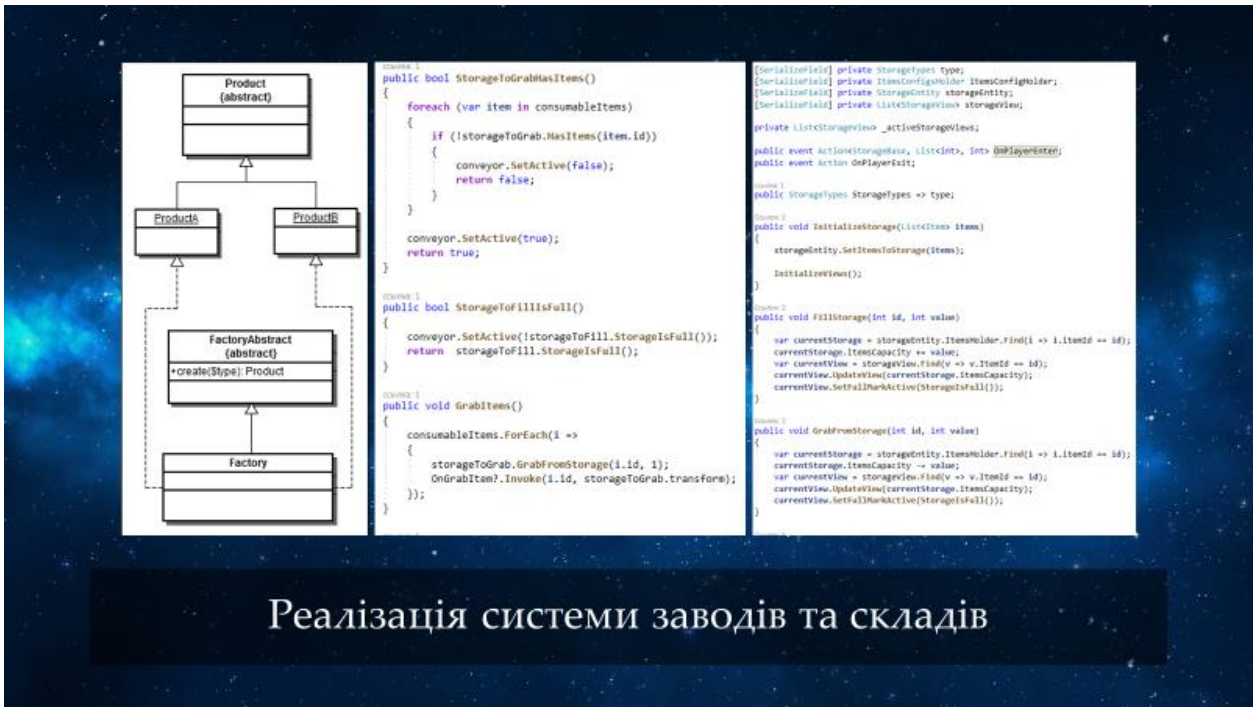
```
namespace Gameplay.FactorySystem.Configurations
{
    [Serializable]
    public class ItemConfigEntity
    {
        public int id;
        public int price;
        public Sprite sprite;
    }
}
```

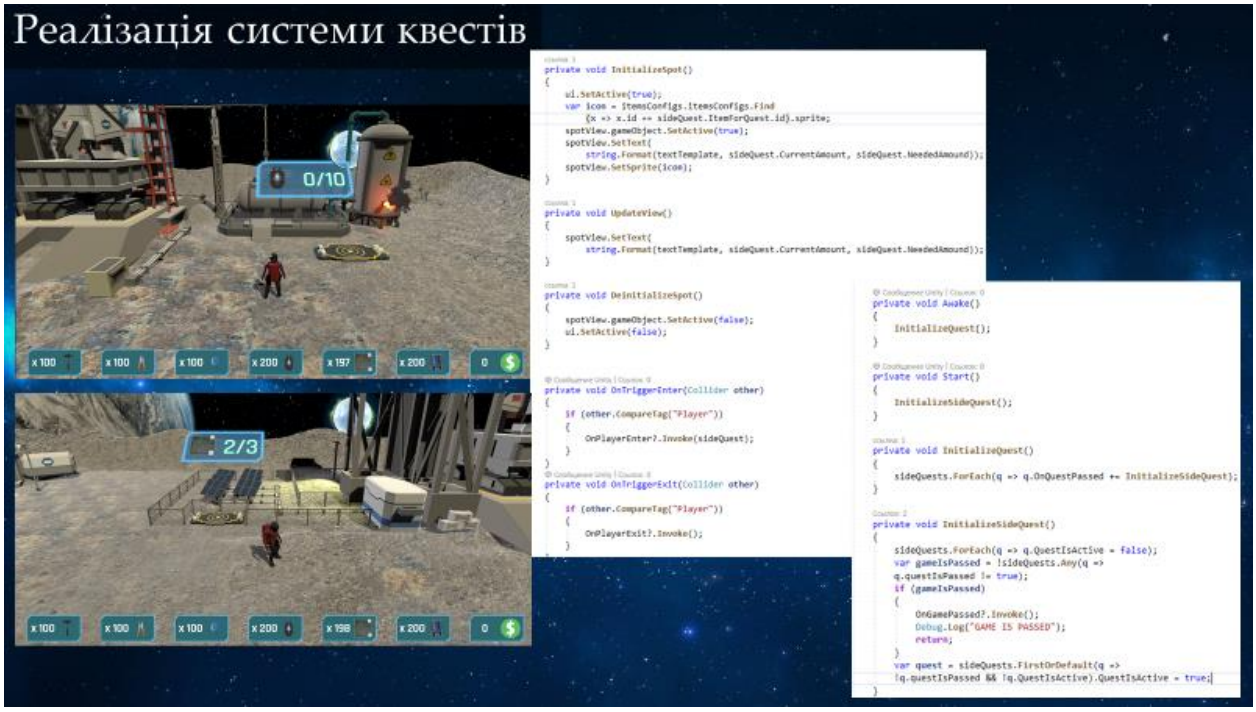


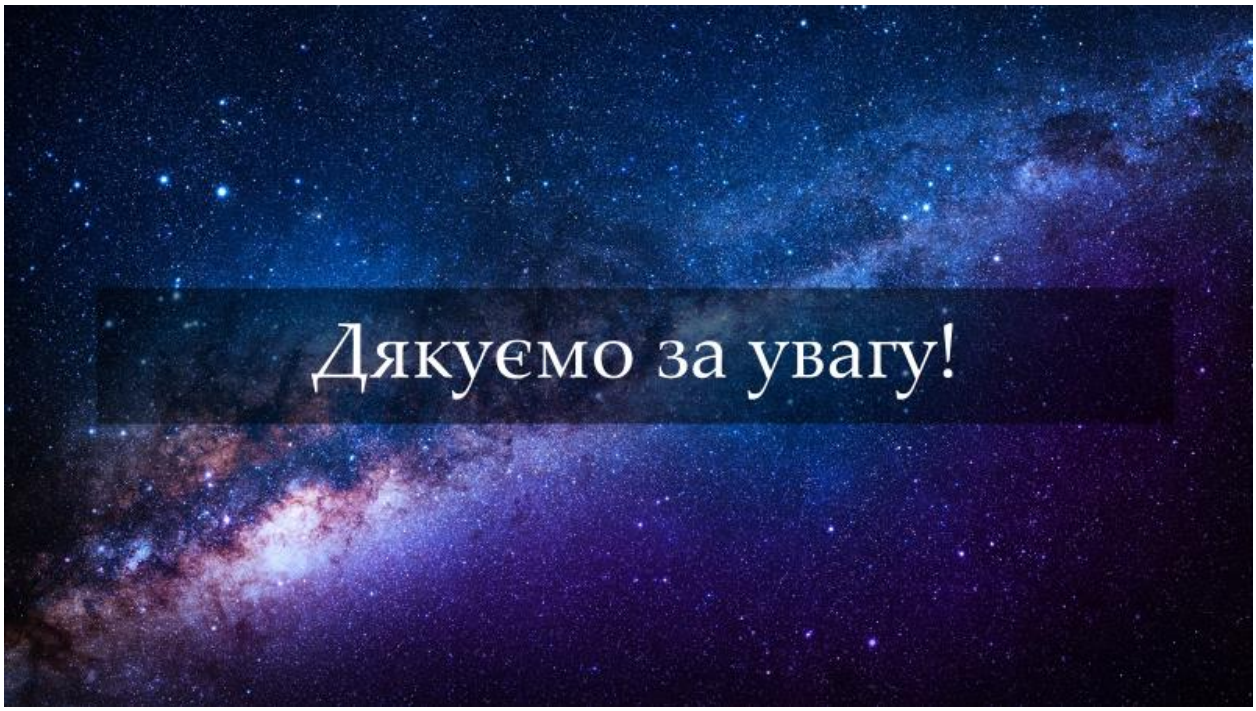
Реалізація системи переміщування гравця




```
[SerializeField] private float movementSpeed;
[SerializeField] private float movementForce;
[SerializeField] private float rotationSpeed;
[SerializeField] private float distanceBetweenGround;
[SerializeField] private Rigidbody rigidBody;
[SerializeField] private Collider colider;
[SerializeField] private PlayerAnimator playerAnimator;

public void Move(Vector2 axises)
{
    ApplyMovement(axises);
    playerAnimator.SetRunAnimation(1);

}

private void ApplyMovement(Vector2 axises)
{
    var velocity = GetYVelocity();
    Vector3 direction = new Vector3(axises.x, 0f, axises.y);
    rigidBody.velocity = direction * (Time.deltaTime * movementSpeed) * movementForce;
    UpdateRotation(rigidBody.velocity);
    rigidBody.velocity = new Vector3(rigidBody.velocity.x, -3f, rigidBody.velocity.z);
}

public bool IsGrounded()
{
    return Physics.Raycast(transform.position, Vector3.down, colider.bounds.size.y + distanceBetweenGround);
}

public void ResetVelocity(Vector2 direction)
{
    playerAnimator.SetRunAnimation(0);
    rigidBody.velocity = new Vector3(0f, 0f, 0f);
}
```


Product
(abstract)
ProductA
ProductB
FactoryAbstract
(abstract)
+create($type): Product
Factory
public bool StorageToGrabHasItems()
{
foreach (var item in consumableItems)
{
if (!storageToGrab.HasItems(item.id))
{
conveyor.SetActive(false);
return false;
}
}
conveyor.SetActive(true);
return true;
}
public bool StorageToFillIsFull()
{
conveyor.SetActive(!storageToFill.StorageIsFull());
return storageToFill.StorageIsFull();
}
public void GrabItems()
{
consumableItems.ForEach(i =>
{
storageToGrab.GrabFromStorage(i.id, 1);
OnGrabItem?.Invoke(i.id, storageToGrab.transform);
});
}
[SerializeField] private StorageTypes type;
[SerializeField] private ItemsConfigsHolder itemsConfigHolder;
[SerializeField] private StorageEntity storageEntity;
[SerializeField] private List<StorageView> storageView;
private List<StorageView> _activeStorageViews;
public event Action<StorageBase, List<int>, int> OnPlayerEnter;
public event Action OnPlayerExit;
public StorageTypes StorageTypes => type;
public void InitializeStorage(List<Item> items)
{
storageEntity.SetItemsToStorage(items);
InitializeViews();
}
public void FillStorage(int id, int value)
{
var currentStorage = storageEntity.ItemsHolder.Find(i => i.itemId == id);
currentStorage.itemsCapacity += value;
var currentView = storageView.Find(v => v.ItemId == id);
currentView.UpdateView(currentStorage.itemsCapacity);
currentView.SetFullMarkActive(StorageIsFull());
}
public void GrabFromStorage(int id, int value)
{
var currentStorage = storageEntity.ItemsHolder.Find(i => i.itemId == id);
currentStorage.itemsCapacity -= value;
var currentView = storageView.Find(v => v.ItemId == id);
currentView.UpdateView(currentStorage.itemsCapacity);
currentView.SetFullMarkActive(StorageIsFull());
}
Реалізація системи заводів та складів


Реалізація системи квестів
0/10
2/3
x 100
x 100
x 100
x 200
x 197
x 200
0
x 100
x 100
x 100
x 200
x 198
x 200
0
private void InitializeSpot()
{
ui.SetActive(true);
var icon = itemsConfigs.itemsConfigs.Find
(x => x.id == sideQuest.ItemForQuest.id).sprite;
spotView.gameObject.SetActive(true);
spotView.SetText(
string.Format(textTemplate, sideQuest.CurrentAmount, sideQuest.NeededAmound));
spotView.SetSprite(icon);
}
private void UpdateView()
{
spotView.SetText(
string.Format(textTemplate, sideQuest.CurrentAmount, sideQuest.NeededAmound));
}
private void DeinitializeSpot()
{
spotView.gameObject.SetActive(false);
ui.SetActive(false);
}
private void OnTriggerEnter(Collider other)
{
if (other.CompareTag("Player"))
{
OnPlayerEnter?.Invoke(sideQuest);
}
}
private void OnTriggerExit(Collider other)
{
if (other.CompareTag("Player"))
{
OnPlayerExit?.Invoke();
}
}
private void Awake()
{
InitializeQuest();
}
private void Start()
{
InitializeSideQuest();
}
private void InitializeQuest()
{
sideQuests.ForEach(q => q.OnQuestPassed += InitializeSideQuest);
}
private void InitializeSideQuest()
{
sideQuests.ForEach(q => q.QuestIsActive = false);
var gameIsPassed = !sideQuests.Any(q =>
q.questIsPassed != true);
if (gameIsPassed)
{
OnGamePassed?.Invoke();
Debug.Log("GAME IS PASSED");
return;
}
var quest = sideQuests.FirstOrDefault(q =>
!q.questIsPassed && !q.QuestIsActive).QuestIsActive = true;
}


Дякуємо за увагу!

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу здобувача (здобувачки) освіти

відділення комп'ютерних систем

Ковальчука Костянтина Олексійовича

(прізвище, ім`я та по батькові)

Спеціальність 123 Комп'ютерна інженерія

Освітня програма Комп'ютерна інженерія

Керівник кваліфікаційної роботи Іванова Лілія Вікторівна

(прізвище, ім`я та по батькові)

Тема кваліфікаційної роботи *«Створення ігрового проекту - квесту на платформі Unity: розробка ігрової механіки та левелдизайну»*

Обсяг розрахунково-пояснювальної записки 61 сторінок

Обсяг графічної (презентаційної) частини 10 аркушів (слайдів)

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) заключення про ступінь відповідності виконаного кваліфікаційної роботи завданню

кваліфікаційна робота у повному обсязі відповідає темі та завданню

б) характеристика виконання кожного розділу кваліфікаційної роботи

Кваліфікаційна робота складається з розділів: Вступ. Аналітичний огляд ігрової індустрії Розробка концепт-документу для ігрового проекту Етапи створення ігрового додатку Аналіз існуючих платформ для розміщення ігрового додатку, та його публікація на itch.io Тестування, розробка подальшої підтримки, корекція балансу Розділ охорони праці. Висновок. Перелік використаних джерел інформації. Кожен розділ присвячено одному з етапів виконання завдання кваліфікаційної роботи та містить необхідну інформацію щодо результатів виконаної роботи.

в) оцінка якості виконання пояснювальної записки та графічної частини кваліфікаційної роботи

Пояснювальна записка виконана якісно, у достатньому обсязі, відповідно до індивідуального завдання та теми дипломного проекту, розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у дипломному проекті Презентація виконана якісно, у достатньому обсязі. Презентація наочно демонструє результати роботи.

г) перелік позитивних якостей кваліфікаційної роботи

1. Актуальна тематика
2. Сучасні технології реалізації програмного продукту
3. Якісне подання результатів роботи

д) основні недоліки кваліфікаційної роботи

Надмірна кількість зайвого теоретичного матеріалу. Етапи розробки варто було подати більш детально

Оцінка розрахункової частини відмінно

Оцінка графічної частини відмінно

Загальна оцінка відмінно

Прізвище, ім'я, по батькові рецензента Стайкуца Сергій Володимирович

Місце роботи і посада рецензента Державний університет інтелектуальних технологій і зв'язку, к.ф.н., доцент кафедри КБ та ТЗІ, пом.декану факультету інформаційних технологій та кібербезпеки

Підпис:

«16» червня 2023 р.

ПІДПИС ПОСВІДЧУЮ
НАЧАЛЬНИК ВІДДІЛУ КАДРІВ ДУІТЗ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ * ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ І ЗВ'ЯЗКУ * 43997335 *

ВІДДІЛ КАДРІВ

Відокремлений структурний підрозділ

Одеський технічний фаховий коледж ОНАХТ

ВІДГУК

Керівника про кваліфікаційну роботу бакалавра

Ковальчука Костянтина Олексійовича

(прізвище, ім`я та по батькові)

Освітньо-професійна програма *«Комп'ютерна інженерія»*

Спеціальність *123 «Комп'ютерна інженерія»*

Тема кваліфікаційної роботи

«Створення ігрового проекту - квесту на платформі Unity: розробка ігрової механіки та левелдизайну»

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) Обсяг і якість виконання роботи (розрахунково-пояснювальної записки)

Пояснювальна записка виконана якісно, у достатньому обсязі, відповідно до індивідуального завдання та теми дипломного проекту, розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у дипломному проекті

Презентація виконана якісно, у достатньому обсязі. Презентація наочно демонструє результати роботи.

б) Самостійність роботи над кваліфікаційною роботою

Студент самостійно обрала напрям та тематику кваліфікаційної роботи. Провів аналіз існуючих рішень і зробив необхідні висновки для реалізації проекту. Виявив навички самостійно опрацьовувати новий матеріал та виконувати пошук необхідної літератури та інших джерел інформації

в) Теоретична підготовка бакалавра

відповідає вимогам, що надаються до бакалавра зі спеціальності «Комп'ютерна інженерія»

г) Вміння розв`язувати виробничі і конструкторські питання на базі останніх досліджень науки і техніки, передових методів виробництва

У дипломному проекті розглянута та реалізована цікава тема створення ігрової платформи на Unity на основі якої розроблена демонстраційна казуальна гра у жанрі у жанрі квест, айдт, пригода, створений з використанням Unity та мови програмування C#, яка може зацікавити комерційні структури ринку комп'ютерних розваг. Застосовані сучасні програмні засоби для реалізації програмного забезпечення.

Загальна оцінка 5(відмінно)

Прізвище, ім`я, по батькові Іванова Лілія Вікторівна

Місце роботи і посада керівника проекту_ВСП «Одеський технічний фаховий коледж ОНТУ» к.т.н., зав. кафедрою Комп'ютерної інженерії

Підпис

«15» 06 2023р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Ковальчук Костянтин Олексійович,
здобувачка освіти гр. 2БКС-27, та

Іванова Лілія Вікторівна,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи молодшого спеціаліста на тему:

«Створення ігрового проекту - квесту на платформі Unity: розробка ігрової механіки та левелдизайну» (автор роботи – Ковальчук К.О., керівник роботи – Іванова Л.В..)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2023 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

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

Виконавець _____ / Ковальчук К.О. /

Керівник _____ / Іванова Л.В./

«15» 06 2023 р.

UNICHECK

Ім'я користувача:
Наталія Вікторівна Копусь

ID перевірки:
1015594828

Дата перевірки:
14.06.2023 09:23:54 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
14.06.2023 09:25:16 EEST

ID користувача:
100011688

Назва документа: **БКС-27 Ковальчук К. О**

Кількість сторінок: 61 Кількість слів: 5583 Кількість символів: 43204 Розмір файлу: 8.48 MB ID файлу: 1015243832

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

2.99%
Схожість

Найбільша схожість: 0.47% з Інтернет-джерелом (https://essuir.sumdu.edu.ua/handle/123456789/87116)

2.99% Джерела з Інтернету	426	Сторінка 63

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Замінені символи	3

Підозріле форматування	16 сторінок