

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОТОТИПУ КОМП'ЮТЕРНОЇ 3D ГРИ ЗА
ДОПОМОГОЮ СЕРЕДОВИЩА UNITY
DEVELOPMENT OF A 3D COMPUTER GAME PROTOTYPE USING THE
UNITY ENVIRONMENT

Виконав: студент(ка) 4 курсу, групи КНТ-128сп
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування

спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Зирянов Ю.О.

(прізвище та ініціали)

Керівник Каплієнко Т.І.

(прізвище та ініціали)

Рецензент Шитікова О.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Зирянова Юрія Олександровича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка прототипу комп'ютерної 3D гри за допомогою середовища Unity. Development of a 3D Computer Game Prototype Using the Unity Environment

керівник проєкту (роботи) Каплієнко Тетяна Ігорівна, к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) 31 травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Аналіз інструментів. 3. Опис проєкту. 4. Тестування. 5. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Каплієнко Т.І., к.н.т, доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання “20” лютого 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання
2	Аналіз предметної області.	2 тиждень	Розділ 1
3	Аналіз інструментів.	3 тиждень	Розділ 2
4	Опис програми.	4–6 тижні	Розділ 3
5	Тестування.	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї.	8-9 тиждень	Додатки
7	Нормоконтроль та рецензування.	10–11 тижні	
8	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Зирянов Ю.О.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Каплієнко Т.І.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
101 с., 3 табл., 50 рис., 2 дод., 12 джерел.

ІНТЕРАКТИВНА ГРА, КОМП'ЮТЕРНА ПРОГРАМА, ПРОТОТИП,
UNITY, C#, GAME.

Об'єкт дослідження – внутрішній устрій ігрової індустрії та принцип її роботи, різновид ігор та їх жанрова приналежність.

Предмет дослідження – популярні комп'ютерні ігри у жанрі виживання-пісочниці, їх особливості та недоліки.

Мета роботи – створення прототипу комп'ютерної 3D гри у жанрі виживання-пісочниці, що містить базові механіки та ознайомчий рівень гри.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мова програмування C#, середовище розробки Unity, персональний комп'ютер під управлінням операційної системи Microsoft Windows.

Результати. В процесі роботи над проєктом було розроблено основні механіки взаємодії з ігровим світом, а також систему розвитку персонажу. Проєкт включає в себе безліч розроблених методів, завдяки яким здійснюється взаємодія між ігровими елементами та об'єктами. Також була створена певна кількість необхідних ігрових моделей та іконок, 5 різних інтерфейсів, кожен з яких виконує свою роль.

Висновки. Розроблено прототип комп'ютерної гри, що включає до себе основні базові елементи взаємодії з ігровим світом, містить необхідні інтерфейси та ігрові елементи, а також створений ознайомчий рівень.

Галузь використання – різновид розваг та відпочинку.

ABSTRACT

Explanatory note to the bachelor's thesis: 101 pages, 3 tables, 50 figures, 2 appendices, 12 sources.

INTERACTIVE GAME, COMPUTER PROGRAM, PROTOTYPE, UNITY, C #, GAME.

The object of research is the internal structure of the gaming industry and the principle of its operation. Variety of games and their genre affiliation.

The subject of the research is popular computer games in the genre of sandbox survival, their features and shortcomings.

The purpose of the work is to create a prototype of a computer 3D game in the genre of sandbox survival, which contains basic mechanics and an introductory level.

Materials, methods and technical means: structured and object-oriented programming, C # programming language, Unity development environment, personal computer running Microsoft Windows operating system.

Results. In the process of working on the project, the basic mechanics of interaction with the game world were developed, as well as the system of character development. The project includes many developed methods through which the interaction between game elements and objects. A number of required game models and icons were also created. As well as 5 different interfaces, each of which performs its role.

Conclusions. A prototype of a computer game has been developed, which includes the basic elements of interaction with the game world, contains the necessary interfaces and game elements, as well as created an introductory level.

Field of application – variety of entertainment and recreation.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1. Аналіз предметної області.....	10
1.1. Опис об'єкта дослідження	10
1.2. Опис предмета дослідження	14
1.2.1. Minecraft.....	14
1.2.2. ARK: Survival Evolved	15
1.2.3. Rust	18
1.3. Постановка завдання роботи	18
2. Аналіз інструментів	20
2.1. Порівняльний аналіз середовища розробки	20
2.2. Порівняльний аналіз мов програмування.....	24
2.3. Порівняння платформ.....	26
2.4. Висновки за розділом 2	26
3. Опис проекту	28
3.1. Інформаційна частина.....	28
3.2. Функціональна частина	28
3.2.1. Підбір предметів	31
3.2.2. Сповіщення	32
3.2.3. Інвентар	32
3.2.4. Tooltip	35
3.2.5. Drag & Drop	36
3.2.6. Виготовлення предметів.....	37
3.2.7. Будівництво	40
3.2.8. Інструменти	41
3.2.9. Ресурси	42
3.2.10 База даних	46
3.2.11 Досвід	47

3.2.12 Здоров'я.....	47
3.3. Інтерфейс	48
3.3.1. Головне меню	49
3.3.2. Ігрове меню.....	50
3.3.3. Інвентар.....	51
3.3.4. Меню створення.....	52
3.3.5. Дерево розвитку	53
3.4. Візуальна частина	54
3.5. Висновки за розділом 3	54
4. Тестування	56
4.1. Вимоги до технічних та програмних засобів	56
4.2. Посібник користувача	56
4.3. Висновки за розділом 4	59
Висновки	60
Перелік джерел посилань	61
Додаток А Код програми.....	62
Додаток Б Презентація.....	94

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

DLC – Downloadable content;

Internet – Всесвітня система об'єднаних комп'ютерних мереж для зберігання і передачі інформації;

ММО, ММОГ – Massively Multiplayer Online Game;

.NET – .NET Framework;

PVP – Player Versus Player;

PVE – Player Versus Environment;

UI – User Interface;

Ассет – Ігровий ресурс, цифровий об'єкт, який представляє собою частину ігрового контенту і має якісь властивостями;

Біом – Природні зони зі своїми особливостями;

Крафт – Виготовлення будь-чого;

ООП – Об'єктно-орієнтоване програмування;

ОС – Операційна система;

ПК – Персональний комп'ютер;

Фарм – Довге повторення однієї дії або процесу для досягнення певної цілі.

ВСТУП

Індустрія комп'ютерних ігор – це сектор економіки, пов'язаний з розробкою, просуванням і продажем комп'ютерних ігор. У неї входить велика кількість спеціальностей, за якими працюють десятки тисяч людей по всьому світу [1].

Перші ігри з'явилися ще в 1960-х роках. Це були текстові пригодницькі ігри, в яких спілкування між гравцем і комп'ютером здійснювалося за допомогою введення команд через клавіатуру. Подальша еволюція ігор тісно пов'язана з ростом продуктивності і можливостей комп'ютерного обладнання. Це і популяризація комп'ютерних мишей в якості інтерфейсу взаємодії людина-комп'ютер, і поява звукових карт, що дають можливість відтворювати повноцінні звуки, і щорічне поліпшення якості зображення, що виводиться, і звичайно розвиток комп'ютерних мереж, що переросли в результаті в появу Інтернету. З еволюцією мереж на зміну звичним всім однокористувацьким клієнтським іграм, приходить нове покоління ігор, які дають можливість грати разом з іншими гравцями [2].

Індустрія комп'ютерних ігор зародилася в середині 1970-х років як рух ентузіастів і за кілька десятиліть виросла з невеликого ринку в переважний напрямок, з річним прибутком, що вираховується в мільярдах доларів США. На ринку працюють як великі гравці, так і невеликі фірми і стартапи, а також незалежні розробники і спільноти [1].

Сучасні ігри – це один з найбільш вимогливих додатків на персональних комп'ютерах (ПК). Багато потужні комп'ютери купуються гравцями, які потрібні для запуску новітніх ігор, в яких використовуються самі передові технології. Таким чином, ігрова індустрія тісно пов'язана з індустрією виробництва центральних процесорів і інших компонентів ПК, так як ігри часто вимагають більш високих апаратних потужностей, ніж бізнес-застосунки [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис об'єкта дослідження

Комп'ютерна гра – це комп'ютерна програма, створена з метою розваги користувача платформи, будь-якого віку, а також може надавати розвиваючий або оздоровчий ефект.

Створенням комп'ютерних ігор може займатися як окрема студія з великим штатом співробітників, або навіть декілька студій, так і окрема людина або група людей. Зазвичай ігри поділяються на: високобюджетні, середньобюджетні та низькобюджетні [3].

Високобюджетні ігри – це ігри, які розроблюються однією або декількома студіями з великим бюджетом і виходять із-під крила видавця, який бере на себе маркетинг та розповсюдження гри, а також може частково чи повністю фінансувати саме розроблення [3].

Середньобюджетні ігри – це ігри, які розробляються невеликою студією з невеликим бюджетом, яка має видавця, що розповсюджує їх продукт та може частково фінансувати розроблення [3].

Низькобюджетні ігри – це ігри, які створюються однією людиною або невеликими командами людей, мають дуже обмежений бюджет, зазвичай вони не мають видавця, а фінансування надходить з різних джерел, у тому числі й пожертвувань [3].

Перед початком основної розробки, створюється прототип майбутнього продукту, який демонструється інвесторам та гравцям. Це необхідно щоб оцінити зацікавленість людей у проекті, після чого керівництво вирішує чи варто продовжувати розробку або краще відмінити проект.

Прототип гри – це модель майбутньої гри, в яку можна пограти і відчувати ігровий процес. На підставі прототипу можна сказати чи є майбутнє у проекту. Прототип дає можливість дизайнеру гри відразу почати маніпулювати з ігровим світом, фізикою всередині гри, економікою і процесами. Основний сенс

прототипу – це перевірити ідею на практиці і, зменшивши ризики, продовжити розроблення проєкту [3].

Самі ігри класифікуються за різними параметрами, наприклад за: жанрами, стилістикою, або за кількістю гравців.

Класифікація за кількістю гравців: одиночні, багатокористувацькі, багатокористувацькі на одному комп'ютері, багатокористувацькі офлайн-ігри, масові онлайнові.

Класифікація за стилістикою: вестерн, стімпанк, кіберпанк, космічні, постапокаліптичні, сучасні, фентезі.

Класифікація за жанрами: екшн, аркада, симулятори, стратегії, пригоди, музичні ігри, рольові ігри, логічні, традиційні та настільні, текстові, та інші.

Існують ігри, в яких присутні елементи декількох жанрів. У цьому випадку гру зараховують або до одного з жанрів, який є основним, або до кількох, що найбільше виділяються в грі.

Симулятор виживання – жанр комп'ютерних ігор, різновид симуляторів життя, в яких основною метою гравця є збереження життя віртуального персонажа на тлі безлічі загрозливих йому небезпек. Елементи виживання містяться практично у всіх комп'ютерних іграх, але в симуляторах виживання ця задача висунута на передній план і є головною в грі. Широке визначення жанру дозволяє включити в нього багато ігор в жанрах roguelike, survival horror, навіть деякі комп'ютерні рольові ігри, такі, як Dark Souls [4].

«Вживання» відноситься до складових жанрів, які можуть базуватися на всіх основних жанрах і самі можуть бути центральним жанром гри. Складовими елементарними жанрами для «виживання» є:

- хоррор – ухилення від ворогів, несприятливих умов, погоди та іншого. Правила жанру припускають, що герой гри не володіє бойовими і багатьма іншими навичками (можливості створення об'єктів, будівель і т.п.), тому йому доводиться уникати зустрічі з ворогами, ховатися від них, тікати, обманювати, влаштовувати їм пастки і збивати зі сліду. Якщо взяти аналогію з природи, користувач грає жертвою, за якою полюють хижаки [5];

- аркада – збиральництво (мається на увазі перш за все пошук ресурсів) – будь-який процес, пов'язаний зі збором і накопиченням грошей, ресурсів, матеріалів, бонусів і т.п. Отримання переваг в поточний момент або на майбутнє [5];

- пісочниця – будівництво (створення, споруда бази) – розподіл доступних ресурсів для розвитку. У підпорядкуванні гравця знаходяться ресурси (гроші, будматеріали, будівельники, вільна для забудови земля), їх потрібно з розумом використовувати для виживання. Ресурси можуть бути у гравця з самого початку, або йому доводиться їх заробляти і добувати під час гри. Складність полягає в правильному розподілі і накопиченні ресурсів. Крім основної ідеї будівництва бази, у виживанні на передній план виступає створення предметів і витратних матеріалів, що відноситься до елементарного жанру "створення" [5];

- екшн – знищення ворогів. Так як ворогів багато, а у гравця персонаж всього один, в грі передбачається можливість компенсувати це за допомогою виготовлення предметів, хитрості, уникнення прямого бою або пасток. Розвиток жанру йде таким чином, що все більше і більше ігор стають кооперативними, більший пріоритет йде на командну роботу [5].

В симуляторах виживання гравець повинен досліджувати світ гри, добувати ресурси, необхідні йому для виживання, і уникати небезпек. Симулятори виживання часто звертаються до теми робінзонади, поміщаючи ігрового персонажа на безлюдний острів або в дику місцевість і надаючи йому самостійно знаходити їжу і воду, облаштовувати житло і захищатися від небезпек, будь то негода, дикі звірі, бандити чи такі фантастичні вороги, як мутанти і зомбі [4].

Загибель персонажа в симуляторах виживання не завжди є кінцем, це може бути лише ще один етап у розвитку, але це правило працює не завжди, у деяких випадках смерть ігрового персонажа призводить за програшу.

Чому і в яких моментах жанр "виживання" популярний і цікавий:

- крафт (виробництво предметів). Людина мало знайома з тим, що створено і чого досягли попередні покоління. Виражається це в незнанні більшої частини світу і речей повсякденного життя, виробничих циклів, пристрої та конфігурації предметів, і багато іншого [5]. Чим більше предметів в грі, і чим більше вони використовуються і включені в крафт, тим цікавіше гравцеві, навіть далекому від усього творчого. Незнання простих творчих моментів обмежує і збіднює людину, в житті він не контролює процеси навколо, від чого його сфера комфорту сильно звужується і він змушений замикає її на виключно емоційні події і явища, а в грі він може долучитися до процесу створення. Збиральництво або навіть фарм (монотонне бездумне збиральництво) хоч і відбуваються несвідомо, але навіть в цьому є плюс – гравець знайомиться з новими невідомими об'єктами [5];

- збирання – прагнення людини займатися бездумним накопиченням. У грі кожен інгредієнт і предмет може реально стати в нагоді або бере участь в ланцюжку крафта, що тільки підштовхує до подальшого збирання. Кожен окремий об'єкт що збирається здається корисним, і хоча пізніше він може взагалі ніде не знадобитися, безпосередньо в момент збирання він виступає в ролі своєрідного бонусу [5];

- індивідуалізм. Все виживання в іграх зводиться до захисту якогось центру, вузької групи людей (зграя, стадо), де кожен розуміє своє місце, роль і рід занять. При цьому немає звичних регламентів і відносин, гравці самі між собою домовляються, решта – це потенційні вороги. Та й навіть невеликі команди у виживанні – рідкість, зазвичай, всі незнайомі гравці поведуться агресивно один з одним. Ось і ще один алгоритм, який розвиває виживання – воно знищує можливість співпраці і співтворчості, розвитку і звільнення один одного від рутинних справ [5];

- зона комфорту (середина, яку людина контролює, розуміє і усвідомлює). Людина на несвідомому рівні завжди намагається захистити свою зону комфорту, не допускає втручання в неї. Але довге перебування в над захищеному середовищі і під потужною опікою з боку інших людей викликає у

людини процеси спрощення і дегенерації. Тому зону комфорту потрібно періодично залишати, переживати стреси, незвичайні ситуації, пригоди [5].

Розвиток віртуальних середовищ дозволив створити замкнуті-захищені-умовні системи, які вже свідомо контролюються людиною зовні, але коли він дозволяє собі грати і зануритися всередину них, він сприймає їх серйозно. За рахунок цього відбувається парадоксальна ситуація: з одного боку він абсолютно захищений, з іншого – він переживає нові незвичайні ситуації [5].

1.2 Опис предмета дослідження

Розглянемо існуючі аналоги у жанрі виживання-пісочниці, які є популярними, та займають велику долю ринку, а саме: «Minecraft», «ARK: Survival Evolved» і «Rust».

1.2.1 Minecraft

Minecraft (від англ. mine – «шахта», «добувати» і англ. craft – «ремесло») – комп'ютерна інді-гра в жанрі пісочниці з елементами симулятора виживання і відкритим світом, розроблена шведським програмістом Маркусом Перссоном, відомим також як «Notch», і пізніше випускається заснованої ним компанією Mojang [6].

Minecraft надає у розпорядження гравця тривимірний процедурно створюваний світ, що повністю складається з кубічних блоків, і дає можливість вільно перебудовувати його, створюючи з блоків складні споруди – ця особливість гри робить Minecraft схожою з конструктором LEGO. Гра не ставить перед гравцем якихось однозначних цілей, але пропонує йому безліч можливостей і занять: так, гравець може досліджувати світ, створювати різноманітні споруди і предмети, битися з різними супротивниками [6].

Ця гра є однією з найпопулярніших і впізнаваних ігор, мало хто хоча б раз не чув про Minecraft. Вона популярна не тільки серед молоді, а і серед

старшої аудиторії. Це гра яка доказала усім що найголовніше в іграх не графіка, а геймплей. В той час як майже уся ігрова індустрія прагнула зробити гру як більш реалістичною у плані графіки, Minecraft проігнорував це, і зробив упор в ігровий процес, і не прогадав, ця гра дуже сильно виділялася на фоні інших, а її свобода и необмежені можливості привернули увагу маси гравців з усього світу. У даний час Minecraft продовжує розвиватися та розширятися, на сьогодні останньою версією є «Minecraft 1.16.5».

Хоча Minecraft була задумана як розважальна комп'ютерна гра, по мірі зростання її популярності багаторазово обговорювалася і можливість неігрового застосування, зокрема, у сферах автоматизованого проєктування і освіти: у процесі гри діти отримують навички програмування, інженерної справи, архітектури і математики.

З 2011 року діє освітня організація MinecraftEdu — спільнота вчителів і програмістів, яке поставило своїм завданням використання Minecraft в шкільному освітньому процесі. У 2015 році копії адаптованої для освітнього процесу версії Minecraft були надані всім середнім школам Північної Ірландії, охопивши близько 200 шкіл, 30 бібліотек та понад 50 тисяч учнів, в подальшому вона набула поширення в Великобританії, США, Австралії та Китаї [6].

Недоліком, як для розважальної комп'ютерної гри, є дуже примітивні механіки і дуже мало об'єктів для підтримання довгого інтересу до гри. Хоча ця проблема вирішується дуже просто за допомогою модів, вони розширюють ігрову механіку, можливості та кількість об'єктів тим самим збільшуючи інтерес до гри. А завдяки тому що модів є безмежна кількість, гра кожного разу виглядає по-новому, тому майже ніколи не набридне.

1.2.2 ARK: Survival Evolved

ARK: Survival Evolved – комп'ютерна багатокористувацька гра в жанрі виживання, розроблена Studio Wildcard, Instinct Games, Effecto Studios, і Virtual

Basement. Розробка почалася в жовтні 2014. Команда розробників вивчила фізичні обличчя динозаврів спеціально для гри, проте в кінцевому підсумку вони привнесли свої зміни в їх зовнішність і поведінку [7].

ARK: Survival Evolved є симулятором виживання, в якому використовується вид від першої особи, з можливістю перейти на вигляд від третьої особи. Щоб вижити, гравці повинні побудувати базу з багаттям і зброєю. Такі можливості, як приручення представників тваринного світу, вимагають більшої кількості ресурсів і часу. Карта гри становить близько 48 квадратних кілометрів, тобто близько 36 квадратних кілометрів суші і 12 квадратних кілометрів океану [7].

У грі більше 60 видів істот, а також древніх членистоногих і ссавців. Динозаври можуть бути приручені за допомогою транквілізатора і подальшого годування, або спеціального предмета. Для кожного виду існує свій спосіб приручення, хижаки воліють м'ясо, трав'яні – ягоди. Після приручення гравці отримують можливість польоту або переміщення верхи на динозаврі. Деякі динозаври мають особливі здібності, наприклад ютараптор може захищати базу, а трицератопс атакувати базу суперників, і навпаки. У грі є й інші тварини, такі як додо, меганеври, мамонти, титаномирми, шаблезубі кішки, жаби, павуки, мавпи, черепахи, скорпіони і навіть гігантопитеки. Крім цього, в грі є можливість створити плем'я, яке дозволяє гравцям сформувати групу і працювати в напрямку досягнення загальних цілей [7].

У грі є підтримка модифікацій яких на даний момент дуже багато, вони значно розширюють можливості основної гри та доповнюють її.

Також у гри є 5 офіційні доповнення (DLC): Scorched Earth, Aberration, Extinction та Genesis Part 1, Part 2. Вони додають нові карти, а також купу нових істот, речей та механік.

Кожне доповнення унікальне, наприклад:

- в оригінальній грі карта складається з одного величезного острова поділеного на біоми (природні зони з різним особливостями), такі як: зимовий біом, лісовий, болотний, луговий та інші;

- у DLC Scorched Earth ми знаходимося на повністю пустинній карті, де на гравців чекає боротьба з екологічними особливостями: сильною жарою, піщаними та грозовими бурями;
- у DLC Aberration уся карта це мережа печер з дуже рідкими походами на поверхню, де загрозу представляють: землетруси, радіаційні зони та екстремальні температури на поверхні;
- у DLC Extinction карта представляє собою заражену землю з невеликими живими ділянками та вимерлим високорозвиненим містом, загрозою тут виступають: метеоритні дощі та скажені заражені звірі.
- і нарешті DLC Genesis Part 1 кидає гравцям серозний виклик, змушуючи виживати у підніжжя активного вулкану, у глибокому океані та навіть у космосі серед астероїдів. Part 2 в свою чергу завершує історію всієї гри, на величезному космічному кораблі, що перевозить виживших на іншу планету, де у якості карти виступає внутрішня сторона кілець космічного кораблю, наповнена різними біомами.

Незважаючи на те, що для більшості ARK є лише виживанням з елементами ММО, крім цього він ще має дуже цікавий прихований сюжет, про який не кожен знає, він подається у вигляді записок невідомих людей, які жили на цих територіях до нас, а також роликів які демонструються після вбивства босів.

Недоліки:

- при стандартних настройках гра вимагає дуже багато часу, вона дуже повільна, дуже повільна прокачка рівня, приручення істот и дуже багато фарму ресурсів. Хоча дану проблему можна вирішити за допомогою гнучкої системи налаштування ігрового процесу, але це тільки у одиночному режимі, або власному сервері;
- відносно невелика кількість речей, які вивчаються в процесі прокачки рівня, хоча на перший погляд їх більше ніж потрібно, але у процесі гри велика кількість речей не буде використовуватися, хоча тут потрібно враховувати де буде проходити гра на PVP або PVE серверах, чи навіть Соло.

Дана проблема є незначною, але якщо хочеться більше то завжди можна додати кілька модифікацій.

1.2.3 Rust

Rust – комп'ютерна гра в жанрі симулятора виживання, була створена незалежної британською студією Facerpunch, на чолі якої з 2014 року стоїть Гаррі Ньюмен. Гра була розроблена за допомогою середовища Unity. Випуск гри відбувся 8 лютого 2013 року у ранньому доступі, а 8 лютого 2018 року відбувся повноцінний реліз [8].

Гравець починає досліджувати відкритий світ Rust з примітивними знаряддям і зброєю, але з часом доберетеся до гармат і повноцінних поселень. Гравцеві доведеться вбивати диких тварин, боротися з голодом і спрагою. Втім, головна загроза буде виходити від десятків інших гравців на сервері.

Недоліки:

- гра повністю зосередження на онлайні та протистоянні іншим гравцям, це водночас як спасає гру так і вбиває її, тому що більшість механік дуже скудна та нерозрахована на довгу перспективу що в свою чергу зменшує реіграбельність.
- у гри дуже слабкий захист, що призводить до великої кількості нечесних гравців, які використовують сторонні програми для отримання переваг.

1.3 Постановка завдання роботи

У рамках дипломного проєкту потрібно розробити прототип гри в жанрі «Виживання».

Найменування продукту – «Відродження: Початок».

Прототип повинен містити в собі базові механіки жанру «Виживання», а саме: добування та обробку ресурсів, виготовлення предметів призначених для

різних видів діяльності. А також системи розвитку та здоров'я, з певною загрозою для останньої.

Повинен бути розроблений навчальний рівень, що буде містити в собі всі необхідні ресурси для ознайомлення базовими принципами гри, та опробуванням на практиці всіх механік.

2 АНАЛІЗ ІНСТРУМЕНТІВ

2.1 Порівняльний аналіз середовища розробки

На сьогоднішній день існує багато різних середовищ для розробки, більшість яких зосереджена на конкретних напрямках, жанрах, але є й середовища призначені для більш широкого використання. Найпопулярніші середовища для розробки ігор на сьогоднішній день є Unity та Unreal Engine. Порівняння середовищ розробки, наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння середовищ розробки Unity та Unreal Engine

Середовище розробки	Вивчення	Зручність	Направлення	Оптимізація продукту, що розроблюється
Unity	Легше вчити	Зручний	Невеликі проекти	Легше оптимізувати
Unreal Engine	Складніше вчити	Розроблений для професіоналів	Великі проекти	Складніше оптимізувати

Unreal Engine – один з лідерів серед ігрових движків розроблений компанією Epic Games. Починаючи з 25 ігор, розроблених на першій версії движка, останні його ітерації налічують багато сотень проектів, причому абсолютно різних жанрів – від платформерів до MMORPG і шутерів [9].

Переваги Unreal Engine:

- Unreal Engine дуже гнучкий і універсальний. На відміну від Unity, який вимагає установки безлічі плагінів (часто – платних), UE4 вже «з коробки» забезпечений всіма необхідними інструментами розробки. Користувачі, які знають C++, освоють движок швидше, адже він використовує саме цю мову програмування. Втім, для тих, хто не знайомий з C++, Unreal Engine теж піддасться без проблем – все завдяки візуальному редактору Blueprints, який дозволяє створювати скрипти і розміщувати об'єкти, не написавши жодного рядка коду [9];

- до того ж, движок містить безліч інструментів, які полегшують роботу з ним. Наприклад, підтримує безліч форматів текстур, точно передає фізичні властивості матеріалів, дозволяє змінювати об'єкти в реальному часі, задавати для них функції і коментарі, автоматично вибрати джерела освітлення, додати туман та інші ефекти, і так далі. Велику колекцію ассетів (платних і безкоштовних) можна використовувати при розробці ігор, а відкритий вихідний код движка дає можливість вносити в нього зміни при необхідності. Движок гнучко підлаштовується під платформу розробки, що дозволяє оптимізувати ігри під консолі, мобільні гаджети та ПК [9];

- Epic Games стабільно тримає лідерські позиції в області передових технологій, тому Unreal Engine 4 наповнений ними під зав'язку, забезпечуючи, при належній майстерності ігоробів, вражаючу візуальну складову з якісним освітленням, м'якими тінями, чесними відображеннями, достовірної анімацією персонажів і іншими ефектами. За рахунок цього, движок також використовується в створенні комп'ютерної графіки в кіноіндустрії – наприклад, його силами було створено дроїд K-2SO для «Ізгоя-1», сцени з «У пошуках Дорі» і «Мандалорця» [9].

Недоліки Unreal Engine:

- на Unreal Engine 4 проблематично створювати великі безшовні світи, розраховані на безліч гравців, що робить скрутним розробку MMORPG і інших ММО-ігор на движку [9];

- аналогічні проблеми виникають з штучним інтелектом: якщо додати на локацію занадто багато розумних істот, спроби движка обробити поведінку всіх одночасно викличуть падіння fps, тому розробникам доведеться придумувати способи обмеження діяльності монстрів, які перебувають за межами взаємодії від гравців [9];

- зручність використання – не першорядне, але досить важливий параметр движка. І в цьому плані UE4 програє все тому ж Unity: якщо другий розрахований на ігоробів-новачків, то перший розроблений для професіоналів, що позначилося на ергономічності інтерфейсу [9];

- нарешті, Unreal Engine змушує більше працювати над оптимізацією ігор. Деякі проєкти, страждають від слабкої продуктивності навіть на потужних ігрових ПК. Думки з цього приводу різняться, але факт залишається фактом – необхідно ретельно працювати над оптимізацією на UE, щоб домогтися плавності картинки [9].

Unity – це більше, ніж движок, це середовище для розробки комп'ютерних ігор, в якій об'єднані різні програмні засоби, що використовуються при створенні програмного забезпечення (ПЗ) – текстовий редактор, компілятор, відладчик і так далі. При цьому, завдяки зручності використання, Unity робить створення ігор максимально простим і комфортним, а мультиплатформеність движка дозволяє ігорробам охопити якомога більшу кількість ігрових платформ і операційних систем. Серед ігрових движків Unity займає далеко не останнє місце. Його використовують як великі розробники, так і (набагато частіше) невеликі незалежні студії [10].

Перевагами Unity, в першу чергу, є те, що середовище Unity3D дає можливість розробляти ігри, не вимагаючи для цього якихось особливих знань. Тут використовується компонентно-орієнтований підхід, в рамках якого розробник створює об'єкти (наприклад, головного героя) і до них додає різні компоненти (наприклад, візуальне відображення персонажа і способи управління ним). Завдяки зручному «Drag & Drop» інтерфейсу і функціональним графічному редактору движок дозволяє малювати карти і розставляти об'єкти в реальному часі і відразу ж тестувати результат [10].

Друга перевага движка – наявність величезної бібліотеки ассетів і плагінів, за допомогою яких можна значно прискорити процес розробки гри. Їх можна імпортувати і експортувати, додавати в гру цілі заготовки – рівні, ворогів, патерни поведінки штучного інтелекту і так далі. Ніякої метушні з програмуванням. Багато ассетів надаються безкоштовно, інші пропонуються за невелику суму, і при бажанні можна створювати власний контент, публікувати його в Unity Asset Store і отримувати від цього прибуток [10].

Третя сильна сторона Unity 3D – підтримка величезної кількості платформ, технологій, API. Створені на движку ігри можна легко перенести між ОС Windows, Linux, OS X, Android, iOS, на консолі сімейств PlayStation, Xbox, Nintendo, на VR- і AR-пристрої. Unity підтримує DirectX і OpenGL, працює з усіма сучасними ефектами рендеринга, включаючи новітню технологію трасування променів в реальному часі [10].

Фізика твердих тіл, ragdoll і тканин, система Level of Detail, колізії між об'єктами, складні анімації – все це можна реалізувати силами движка. Стереотипна думка про те, що движок придатний тільки для невеликих інді-ігор і нездатний видавати красиву картинку, давно вже не актуально: досить подивитися технодемо ADAM, The Blacksmith і Book of the Dead від творців середовища Unity, щоб переконатися в її видатних здібностях [10].

Нарешті, Unity доступний безкоштовно, що відкриває перед незалежними розробниками двері в ігрову індустрію. Звичайно, існують обмеження: безкоштовна версія движка демонструє логотип Unity перед запуском гри, а проєкт, створений з її допомогою, не повинен приносити розробнику більше \$ 100 тисяч в рік. Про-версія коштує \$ 125 на місяць, що не так вже й багато в порівнянні з іншими середовищами розробки, причому базова версія містить рівно той же функціонал, що і професійна [10].

При всіх своїх перевагах, движок має і свої недоліки. Так, якщо команда захоче розробити що-небудь складніше простого клікера або платформера, то їй доведеться шукати хорошого програміста на C#, який напише скрипти і компоненти, запровадить їх у гру і змусить працювати [10].

З цього випливає інша проблема движка Unity – повільність. Створення масштабних, складних сцен з великою кількістю компонентів може негативно вплинути на продуктивність гри, в результаті чого розробникам доведеться витратити додатковий час і ресурси на оптимізацію, а можливо – і видалення деяких елементів з проєкту [10].

Крім того, застосунки, створені на Unity, досить «великовагові»: навіть найпростіша піксельна гра може займати кілька сотень мегабайт на ПК. Так,

для жорсткого диска комп'ютерів це невеликий обсяг, але, якщо проєкт розробляється і для мобільних платформ, слід задуматися про оптимізацію його розміру [10].

Для розробки проєкту було вибрано середовище Unity, тому що воно спрощує процес розробки невеликих ігор, процес розробки здійснюється за допомогою C# що спрощує написання коду порівняно з C++, а також завдяки цьому спрощується процес оптимізації що дуже важливо для невеликих ігор. У той час як в Unreal Engine використовується C++, що декілька ускладнює процес розробки, а також вимагає оптимізувати проєкт вручну що сильно гальмує і ускладнює процес розробки, це невигідно для невеликих ігор які створюються маленькою командою.

2.2 Порівняльний аналіз мов програмування

Виходячи з вибраної платформи для розробки, на вибір є такі мови програмування як JavaScript і C#. Порівняння характеристик мов програмування, наведено у таблиці 2.2.

Таблиця 2.2 – Порівняння мов JavaScript і C#

	JavaScript	C#
Популярність	Дуже популярний	Дуже популярний
Підтримка ООП	Не підтримує	Підтримує
Можливості	Потрібні додаткові бібліотеки і фреймворки	Має багато можливостей із коробки
Вивчення	Легше вчити	Складніше вчити

Об'єктно-орієнтоване програмування (ООП) – це підхід до моделювання та організації коду. Якщо методики ООП застосовуються правильно, вони допомагають програмісту створювати простий і добре організований код. Крім

того, ООП спрощує повторне використання важливих функціональних можливостей програм [11].

Недоліки JavaScript:

- JavaScript сумно відомий відсутністю підтримки об'єктно-орієнтованого програмування. Насправді, JavaScript-розробники традиційно обходять цей недолік, використовуючи всілякі дивні конструкції. Але якщо розробник абсолютно нова людина в програмуванні, використання подібних сумнівних конструкцій, що реалізують базові концепції програмування – це абсолютно неправильно [11].

- для JavaScript-розробки потрібні додаткові бібліотеки і фреймворки. Для того щоб отримати доступ до більш широкого, ніж є в мові, набору функціональних можливостей, JavaScript-програмістам потрібно використовувати бібліотеки і фреймворки сторонніх розробників. Вибір правильних інгредієнтів, які використовуються при створенні якогось проєкту – це не так просто, як може здатися на перший погляд. Мова йде, зокрема, про те, що обрані додаткові інструменти повинні правильно вирішувати поставлені перед ними завдання, потрібно, щоб у програміста була упевненість в тому, що вони ще довго будуть користуватися підтримкою своїх розробників, потрібно, щоб вони не конфліктували один з одним. З якимось різновидом цієї проблеми стикаються і розробники, що використовують інші мови. Однак треба зазначити, що не всі мови страждають від цієї проблеми так само сильно, як JavaScript [11].

C# – це мова з C-подібним синтаксисом. В цьому відношенні він близький до C ++ і Java. Будучи об'єктно-орієнтованою мовою, він багато перейняв у Java і C ++.

Недоліки C#:

- пріоритетна орієнтованість на платформу Windows.
- .NET використовує концепцію JIT-компіляції. Це означає, що програма буде скомпільована в машинні коди в міру необхідності прямо під час

роботи програми. З одного боку, це звичайно непогано, але при першому запуску можливі дуже серйозні гальма [12].

- C# дуже легко дизасембелюється (спосіб отримання вихідного тексту програми на асемблері з програми в машинних кодах). Це означає, що з великою ймовірністю код буде отримано і вивчений конкурентами. Звичайно є спеціальні інструменти, які можуть ускладнити цей процес, але на 100% захиститися від цього практично неможливо[12].

З огляду на порівняльну характеристику мов програмування та переваги розробки мовою C#, прийнято рішення про використання цієї мови у розробці.

2.3 Порівняння платформ

Існують різні види платформ, такі як: комп'ютери, консолі та мобільні пристрої.

Розробка під кожен із платформ доволі унікальна та по своєму складна.

Раніше розробка для декількох платформ була дуже затратною та складною із-за специфіки кожної із платформ, особливостей архітектури та унікальної системи. Тому при розробці однієї гри чи портуванні на іншу платформу гра розроблялася з самого початку і навіть могла дуже сильно відрізнятись на різних платформах.

На сьогоднішній день розробка ігор на декілька платформ чи портування – це звичайна справа, яку можуть дозволити собі навіть незалежні або починаючі розробники, а все завдяки схожій архітектурі. Винятком є хіба що мобільні пристрої.

2.4 Висновки за розділом 2

У даному розділі проведено аналіз середовищ розробки, мов програмування доступних в цих середовищах, їх переваги та недоліки. У

результаті аналізу та порівняння, було обрано середовище Unity, так як воно більш підходить для розробки невеликих проєктів з невеликою командою.

Стосовно мов програмування, на вибір було надано дві мови, а саме JavaScript і C#. В ході аналізу було обрано C#, тому що він більш самобутній та має вагомні переваги.

Стосовно платформ вибір пав на персональні комп'ютери, хоча це ніяк не обмежує подальше портування проєкта на консолі, тому що технічна частина дуже схожа і не потребує серйозної переробки.

3 ОПИС ПРОЄКТУ

3.1 Інформаційна частина

Розроблений проєкт представляє собою комп'ютерну гру у жанрі Виживання-Пісочниця, розроблену під операційну систему Windows. Проєкт розроблено в середовищі у Unity на мові програмування C#.

Виживання-Пісочниця передбачає: збирання ресурсів, створення нових предметів, будівництво та певний розвиток.

На даному етапі розробки проєкт має в собі один рівень, що містить всі необхідні базові механіки, та виконує ознайомчу функцію гравця з механіками гри.

3.2 Функціональна частина

Проєкт містить в собі наступні методи та функції, що наведені нижче у таблиці 3.1.

Таблиця 3.1 – Функції та методи

Функція/метод	Опис / Застосування
Control()	Здійснює пересування гравця по ігровому полю, за допомогою клавіш WASD.
MouseControl()	Надає гравцю можливість керувати ігровою камерою.
Inventory.Update()	Здійснює взаємодію з ігровими об'єктами на короткій відстані, за допомогою невидимого променя.
Message()	Здійснює виведення повідомлення про підібраний предмет.
DestroyMessages()	Видаляє повідомлення через певний час.

Продовження таблиці 3.1

Функція/метод	Опис / Застосування
AddItem()	Виконує додавання предмета до інвентарю. Перевіряє предмет на кучність, та викликає один з методів, AddUnstackableItem або AddStackableItem, в залежності від результату.
AddUnstackableItem()	Шукає пусту комірку, та додає новий предмет до інвентарю.
AddStackableItem()	Шукає комірку зі схожим предметом, та збільшує кількість предмету у комірці. Якщо схожий предмет не знайдено, виконується метод AddUnstackableItem.
ToggleInventory()	Здійснює відкриття/закриття одного з інтерфейсів, таких як: інвентар, вікно виготовлення або дерево розвитку.
DisplayItems()	Здійснює оновлення та відображення предметів в інвентарі.
DamadeHPItem()	Зменшує міцність інструменту при використанні.
PauseMenu()	Зупиняє гру, та відкриває інтерфейс ігрового меню.
Death()	Зупиняє гру, та відкриває інтерфейс загибелі ігрового персонажу.
Slot()	Здійснює заміну комірки на активну, при наведенні курсору та на стандартну при виведенні курсору з комірки.
CurrentItem.OnPointerClick()	Відстежує натиснення правої та лівої кнопок миші на предметі в інвентарі, та здійснює викидання або використання предмета.

Продовження таблиці 3.1

Функція/метод	Опис / Застосування
Remove()	Виконує зменшення/видалення предмета з інвентарю.
Drop()	Виконує створення предмета в сцені перед гравцем, при викиданні предмета.
OnDrop()	Виконує переміщення предмета між комірок інвентарю.
Tooltip()	Здійснює відображення додаткової інформації про предмет, при наведенні курсору на комірку з предметом.
SwitchFlash()	Вмикає/вимикає відображення вибраного інструменту в руці гравця.
Instrument.Update()	Здійснює добуток ресурсів за допомогою інструментів.
TypeInstrument()	Перевіряє відповідність інструменту до ресурсу що добувається.
Bonus()	Нараховує додаткові ресурси при добутку.
Damage()	Зменшує міцність добуваного ресурсу. А також звертається до методу DamadeHPItem.
Experience()	Здійснює накопичення досвіду та підвищення рівня.
LevelUp()	Здійснює відображення досягнень у дереві розвитку.
CraftItem()	Відкриває нові рецепти по мірі підвищення рівня персонажу.

Продовження таблиці 3.1

Функція/метод	Опис / Застосування
CraftItemDatabase.OnPointerClick()	При натисканні на рецепт, здійснює відображення додаткової інформації про предмет, а також необхідні ресурси для виготовлення, та ресурси в наявності.
NewItem()	Додаємо створений предмет до інвентарю та видаляємо або зменшуємо, використані при створенні ресурси.
Helthbar()	Впливає на здоров'я ігрового персонажу.

3.2.1 Підбір предметів

Скрипт «Item» зберігає в собі інформацію про предмет, а саме: назву, id, опис, кількість, і т.д. Даний скрипт міститься на кожному предметі, з яким є взаємодія, окрім ресурсів добування та структур.

Підбір предмета здійснюється наступним чином – при натисканні на певну кнопку, що відповідає за підбір, пускається промінь з центру екрана на 2 метри вперед, якщо промінь стикається з об'єктом на якому висить скрипт «Item», звертаємося до методу «AddItem», де визначаємо чи є предмет кучним, в залежності від результату виконується один з наступних методів:

- предмет кучний – шукаємо відповідний предмет в інвентарі, якщо такий є то збільшуємо значення кількості на 1, оновлюємо інвентар, видаляємо об'єкт зі сцени. Якщо відповідного предмета немає то перенаправляємо на другий метод.
- предмет одинарний – шукаємо вільну позицію в інвентарі, додаємо туди предмет, оновлюємо інвентар, видаляємо об'єкт зі сцени.

3.2.2 Сповіщення

При підборі предмета викликається метод «Message», спочатку створюється блок сповіщення, що містить в собі зображення і текст, блок в дочеряється до об'єкту «MessageManager», який вибудовує блоки сповіщень в список що відображається на екрані. Потім блок сповіщення зчитує інформацію з підбраного об'єкту, а саме назву та іконку предмета. Приклад виведення сповіщення наведено на рисунку 3.1.

Після появи повідомлення про підбір, запускається таймер, який видаляє повідомлення через певний час.

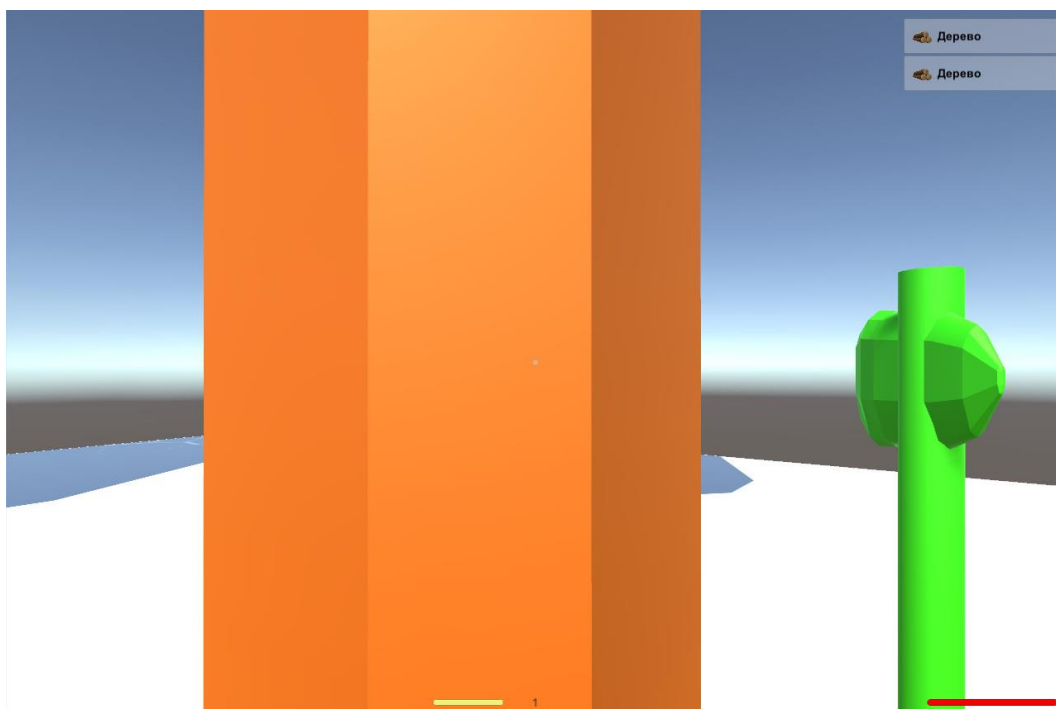


Рисунок 3.1 – Виведення сповіщення про підбір предмета

3.2.3 Інвентар

Панель інвентарю складається з певної кількості комірок, кожна з яких містить в собі: зображення, позначення кількості предмета та міцності. На рисунку 3.2 наведено приклад панелі інвентарю.

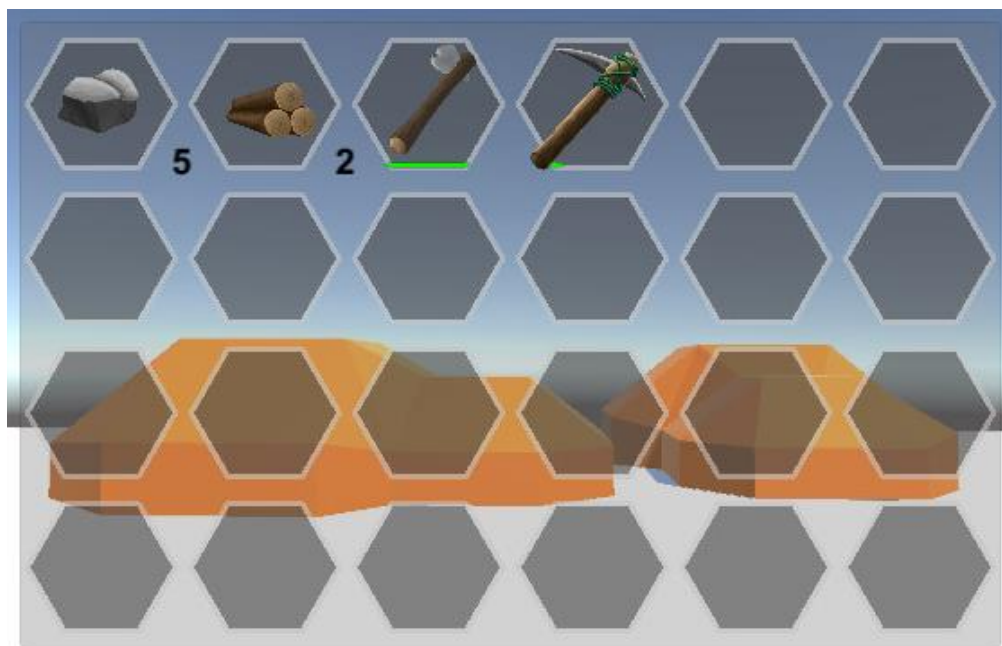


Рисунок 3.2 – Панель інвентарю

При старті гри створюється колекція «item», яка заповнюється порожніми об'єктами, тобто кожна комірка, що є порожньою, насправді являє собою нульовий об'єкт.

При старті гри інвентар вимкнений, після натискання на певну кнопку він відображається, а при повторному натисканні зникає. Курсор миші ховається, коли інвентар закритий, і відображається, коли інвентар відкривається. Коли інвентар відкритий, курсор миші відображається, а точка, що відображає центр екрану вимикається, також вимикається управління персонажем. Коли інвентар закритий, управління персонажем відновлюється, а також відновлюється відображення точки що демонструє центр екрана, відображення курсору миші вимикається.

Розглянемо відображення предметів у інвентарі. Для початку отримуємо доступ до комірки. Якщо комірка не є нульовим об'єктом, то завантажується необхідна іконка ресурсу, яка зберігає в інформації про предмет, вмикаємо відображення іконки. Теж саме робимо і з текстом, якщо в інвентарі є кучний предмет, у якого значення більше 1, то вмикаємо відображення тексту. Якщо

таких комірок немає, то вимикаємо відображення картинки і тексту, а також обнуляємо дані, якщо такі були.

Комірка змінюється при наведенні на неї курсору. Відстежуємо наведення курсору на комірку і прибирання курсору з комірки. Якщо курсор наводиться на комірку то він змінює своє зображення на активне, а коли курсор виводиться з комірки вона повертає своє початкове зображення. При закритті інвентарю всі комірки повертають собі початковий вид, тим самим уникаючи багу з активною коміркою при повторному включенні інвентарю. На рисунку 3.3 наведено приклад з активною коміркою.



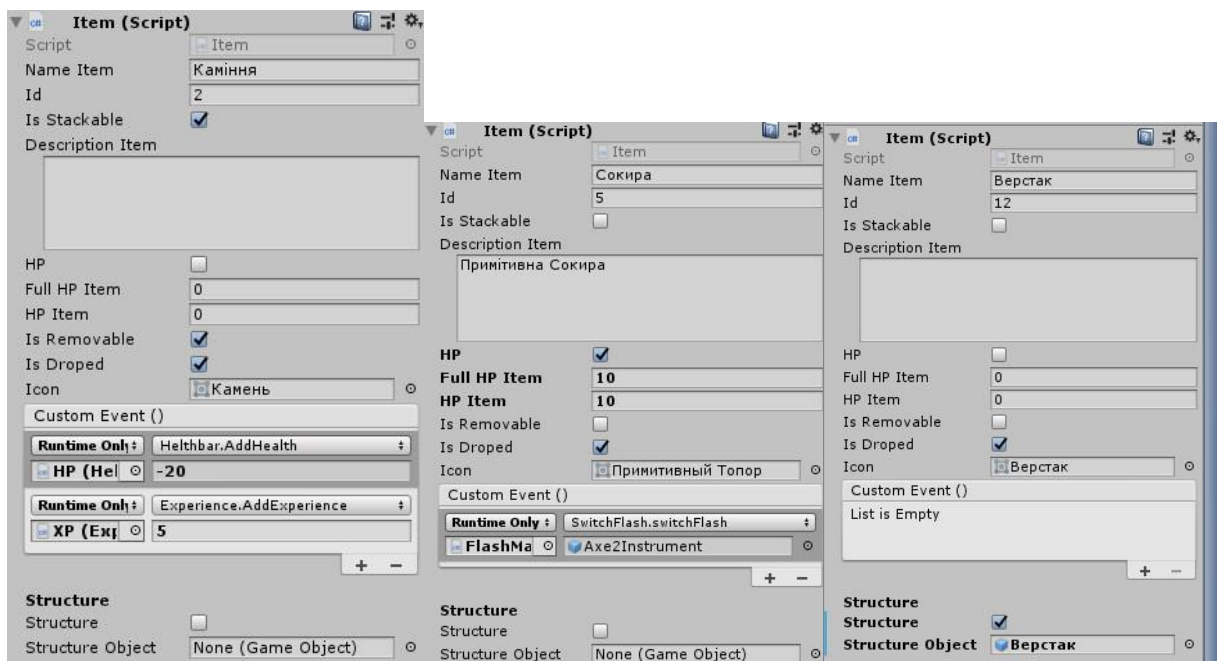
Рисунок 3.3 – Приклад активної та неактивних комірок

Розглянемо видалення предметів з інвентарю. Відстежуємо натискання правої кнопки миші на комірці, якщо комірка не порожня, перевіряється чи можна викинути предмет, якщо так, то створюємо в сцені потрібний об'єкт перед гравцем, довантажуючи його з бази даних і зменшуємо кількість предмета, що викидається, якщо кількість предмета менше одиниці то комірка стає порожньою. Оновлюємо відображення інвентарю.

Розглянемо використання предметів. У грі присутні різні типи використовуваних предметів, такі як: інструменти, структури та предмети що вживаються. При натисканні на об'єкт лівою кнопкою перевіряється чи є предмет використовуваним, якщо так, то виконується подія, закладена в предметі. Якщо це вживаємий об'єкт, то при використанні зменшується кількість предмета, а також виконується подія, яка впливає на здоров'я гравця, зменшуючи або відновлюючи його. Якщо це інструмент, то при використанні

предмет не видаляється, а також виконується подія, яка відображає інструмент в руці, який користувач надалі може використовувати за призначенням, при повторному натисканні інструмент прибирається. При цьому виконується перевірка на тип інструменту який активується, що дозволяє не тільки включати і відключати відображення потрібного інструменту, але і перемикатися між інструментами. Якщо це структура, то при використанні відповідний об'єкт буде розміщено на карті перед гравцем.

На рисунку 3.4а, 3.4б та 3.4в наведено приклад кожного з трьох типів предметів.



а

б

в

а – ресурс; б – інструмент; в – структура

Рисунок 3.4 – Приклад різних типів предметів

3.2.4 Tooltip

Створюємо UI елемент «Tooltip», який буде відображатися при наведенні на предмет в інвентарі, що зберігає в собі: іконку, назву предмета і

його опис. Якщо комірка, на яку навели курсор, не є пустою, то вмикається відображення «Tooltip» і передаються дані про предмет. «Tooltip» слідкує за курсором миші з невеликим відступом. Як тільки курсор виходить за рамки комірки або інвентар закривається, відображення «Tooltip» вимикається. На рисунку 3.5 наведено приклад з відображенням додаткової інформації про предмет.

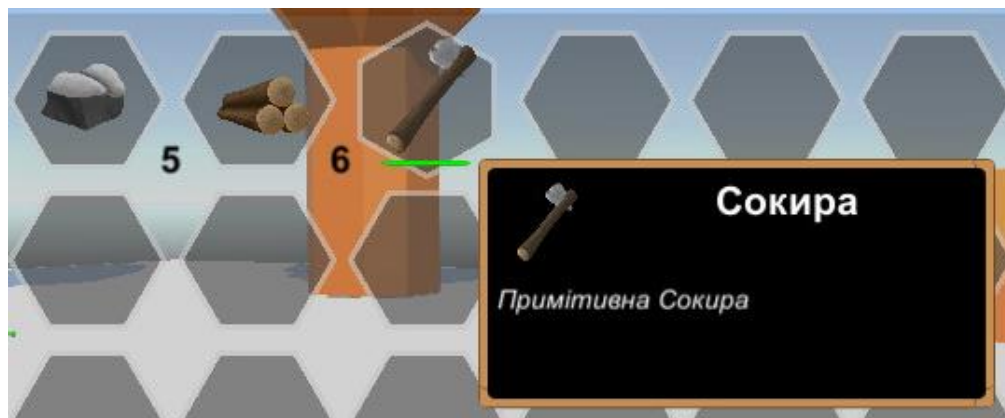


Рисунок 3.5 – Приклад відображення додаткової інформації про предмет

3.2.5 Drag & Drop

Створюємо UI елемент, тимчасову комірку, яка зберігає в собі іконку і кількість предмета. Якщо комірка, на якій затисли курсор не є пустою, то вмикається відображення UI, в тимчасову комірку записуються дані (іконка та кількість предмета). У момент перетягування об'єкта з однієї комірки в іншу, тимчасова комірка слідкує за курсором. Саме переміщення предмета здійснюється записуванням предмета, що перетягується, в тимчасову змінну, далі з комірки куди користувач хоче помістити предмет, значення присвоюється в комірку, яка є відправною точкою предмета, що перетягається, після чого ця комірка приймає на себе значення з тимчасової змінної. Оновлюємо відображення інвентарю. На рисунку 3.6 наведено приклад перетягування предмету.



Рисунок 3.6 – Приклад перетягування предмету

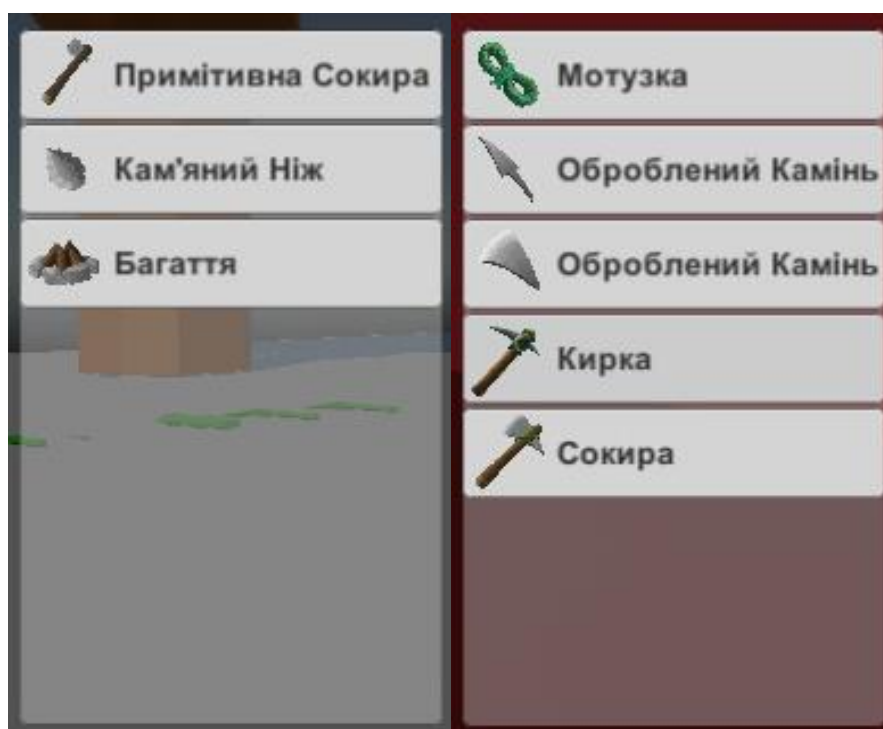
Викидання предмета за допомогою перетягування. Перевіряємо чи є під курсором будь-який UI елемент, якщо немає, то викликаємо процес викидання з інвентарю.

3.2.6 Виготовлення предметів

При старті гри панель виготовлення вимкнена, при натисканні на певну кнопку панель вмикається, а при повторному натисканні вимикається. Для взаємодії з верстатом необхідно направити центральну точку екрану на верстак і натиснути кнопку взаємодії.

Панель виготовлення складається з декількох блоків, таких як: список доступних рецептів, інформація про рецепт та необхідні ресурси для виготовлення.

Список доступних рецептів, складається з вертикального списку наповненого різними рецептами, що в свою чергу складаються з іконки і назви, що наведено на рисунку 3.7а та 3.7б.



а

б

а – базові; б – за допомогою верстаку

Рисунок 3.7 – Список доступних рецептів

Інформація про рецепт відображається після натискання на будь-який з рецептів в списку, сюди входять: іконка предмета, назва і опис (рис. 3.8).

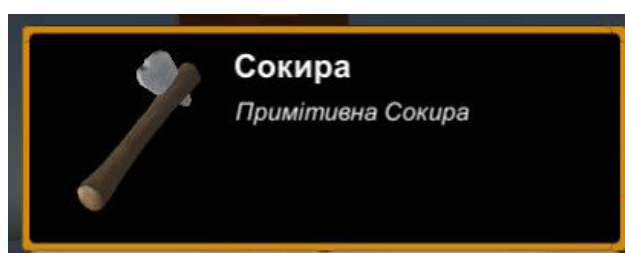


Рисунок 3.8 – Відображення інформації про вибраний рецепт

Необхідні ресурси для виготовлення відображаються у вигляді блоку, що включає в себе: назву необхідного ресурсу, кількість необхідного ресурсу, а також наявну кількість ресурсу. При виготовленні можуть вимагатися до 6

різних видів ресурсів. На рисунку 3.9 наведено приклад відображення потрібних ресурсів для виготовлення.

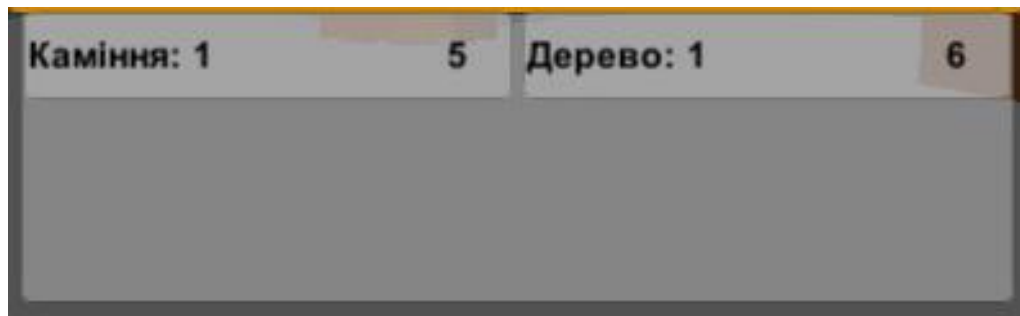
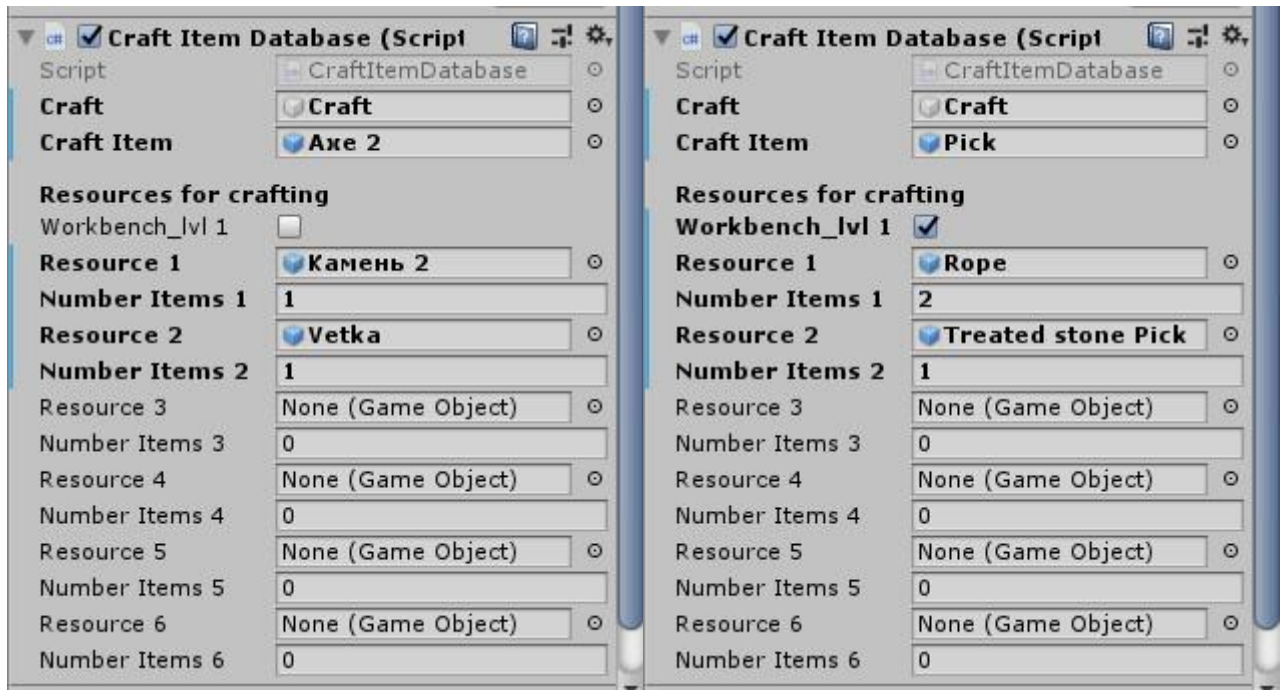


Рисунок 3.9 – Відображення потрібних для виготовлення ресурсів

В процесі гри, по мірі підвищення рівня, відкриваються нові рецепти. При досягненні першого рівня, відкриваються рецепти: примітивної сокири, кам'яного ножа і багаття. При досягненні другого рівня, відкривається рецепт верстака, а також рецепти кам'яних інструментів, які вже створюються за допомогою верстака і відображаються тільки там.

Кожен рецепт зберігає в собі інформацію про вимоги для створення, а саме: предмет що створюється (посилання на предмет в базі даних), місце виготовлення (базова панель створення або верстак), ресурси необхідні для виготовлення (до 6 різних видів ресурсів) та їх кількість. При натисканні на рецепт, відображається інформація про предмет створення, а саме: іконка предмету, назва та опис. А також необхідні ресурси: назва ресурсу, необхідна кількість, наявна кількість. Якщо необхідних ресурсів для обраного рецепта недостатньо, кнопка виготовлення стає неактивною. Приклади рецептів наведено на рисунку 3.10а та 3.10б.



а

б

а – ручне виготовлення; б – виготовлення за допомогою верстаку

Рисунок 3.10 – Приклади рецептів

При створенні, новий предмет додається в інвентар, а використані ресурси при створенні зменшуються. У разі, якщо одним з необхідних ресурсів є інструмент, то у нього лише зменшується міцність.

3.2.7 Будівництво

При використанні предмета йде перевірка, яка визначає чи є об'єкт структурою. Будівництво здійснюється шляхом розміщення об'єкта на карті, в місці куди направлена камера гравця, на певній відстані, та подальшого його видалення з інвентарю. Модель, що розміщується при використанні, зберігається в інформації предмета. На рисунку 3.11 наведено приклад предмета що являється структурою.

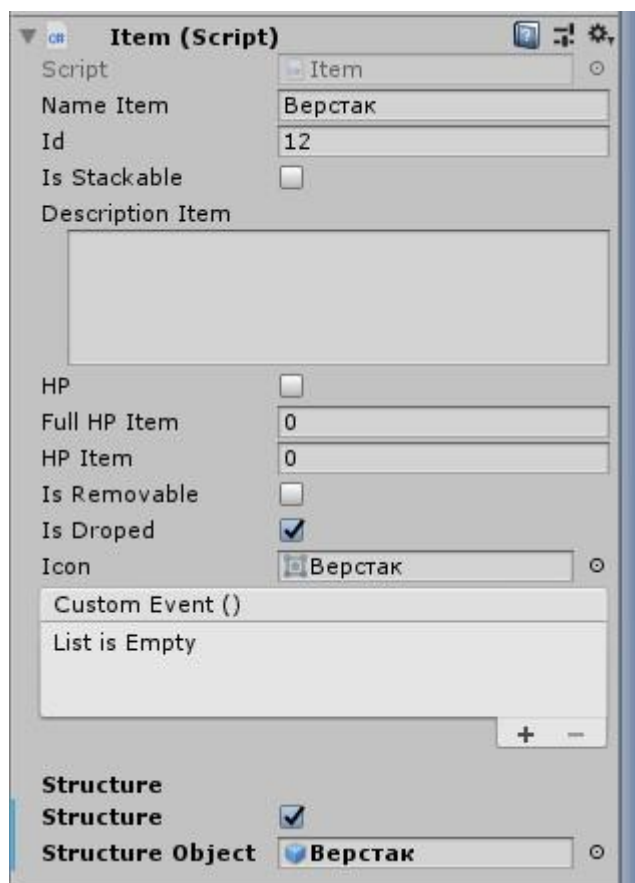


Рисунок 3.11 – Приклад структури

3.2.8 Інструменти

У грі присутні різного роду інструменти, які необхідні для виживання. На даний момент в грі є 4 види інструментів, такі як: примітивна сокира, кам'яний ніж, кам'яна сокира і кам'яна кирка. Рецепти інструментів відкриваються по мірі підвищення рівня, відрізняються складністю виготовлення, а також ефективністю видобутку. По досягненню першого рівня для виготовлення стане доступна примітивна сокира, що дозволить добувати ресурси, які не можна зібрати руками, а також стане доступний рецепт ножа, що необхідний як для більш ефективного видобутку деяких ресурсів, так і для обробки цих ресурсів. Після того як гравець отримає необхідний рівень, а також обзаведеться верстаком, гравцеві зможе зробити більш кращі інструменти, виготовлення яких більш складне, але і ефективність видобутку яких набагато вище першого примітивного інструменту.

У кожного з інструментів є своя міцність, яка витрачається при використанні, якщо міцність інструменту впаде до 0 то інструмент зламається, поточну міцність інструменту можна відстежувати за допомогою відповідної смужки міцності на комірці з інструментом в інвентарі.

На рисунку 3.12 зображені різні види інструментів, а також поточна міцність кожного з них.



Рисунок 3.12 – Різновид інструментів та їх поточна міцність

3.2.9 Ресурси

Також в грі існують різні види ресурсів, які можуть відрізнятися за методом збору. На даний момент існують 4 методи збору: ручний збір, змішаний збір, збір за допомогою інструментів, а також об'єкти які не збираються (структури). Для зручності всі об'єкти пофарбовані в свій колір: зелений, жовтий, помаранчевий і червоний відповідно.

На рисунку 3.13 зображено всі види ресурсів, а також візуальне позначення їх методу збору.

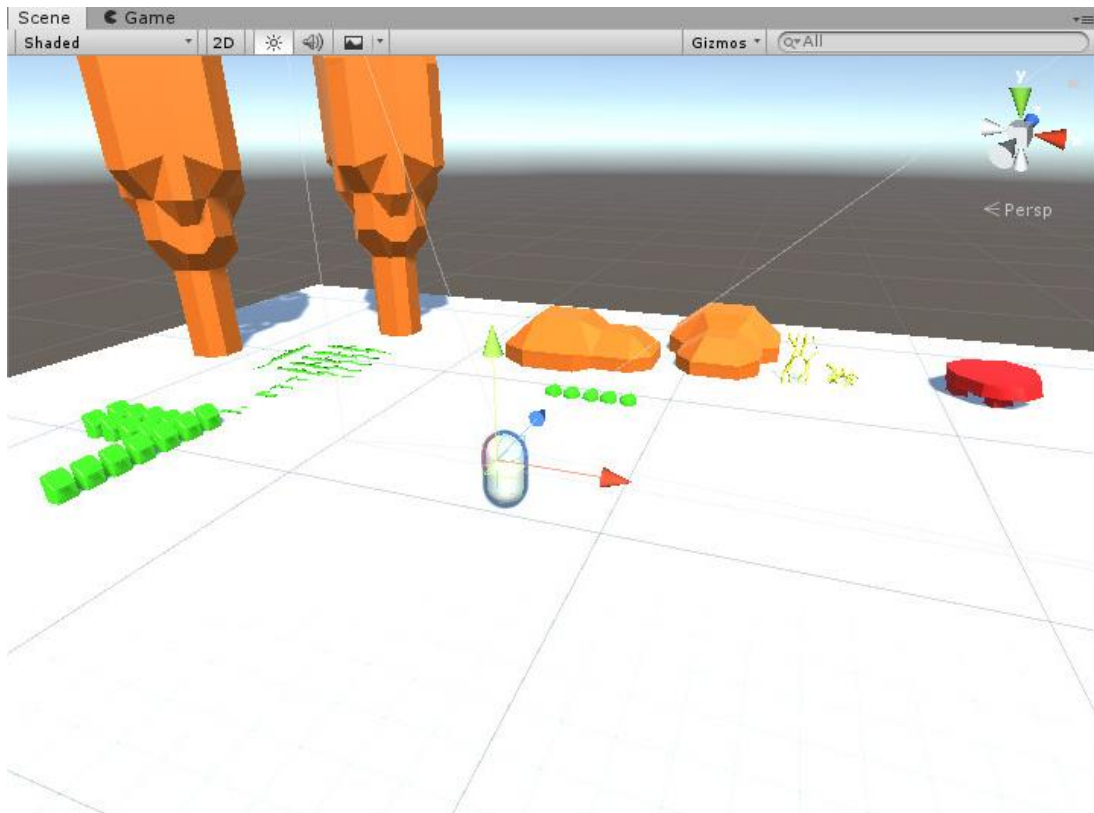


Рисунок 3.13 – Види ресурсів та візуальне відображення їх методу збору

Ручний збір – гравець може збирати даний вид ресурсу тільки за допомогою рук, без необхідності використання будь-яких інструментів. Позначається зеленим кольором.

Змішаний збір – гравець може добувати даний вид ресурсу як руками, так і за допомогою інструменту. Варіант збору за допомогою інструменту більш ефективний, і принесе більше ресурсів при зборі ніж за допомогою рук. Позначається жовтим кольором.

Збір за допомогою інструментів – ресурс можна добути тільки за допомогою потрібного інструменту. Такі ресурси як велике каміння і дерева, видобуваються за допомогою кирки і сокири відповідно, при цьому дерево також можна добувати киркою, а камінь сокирою, але у цьому випадку ефективність видобутку буде сильно нижче, і ресурсів збереться менше (рис. 3.14). Позначається помаранчевим кольором.

Приклади збору різних ресурсів за допомогою різних інструментів, з демонстрацією їх ефективності, наведені на рисунках 3.15а, 3.15б, 3.16а, 3.16б та 3.17а, 3.17б.

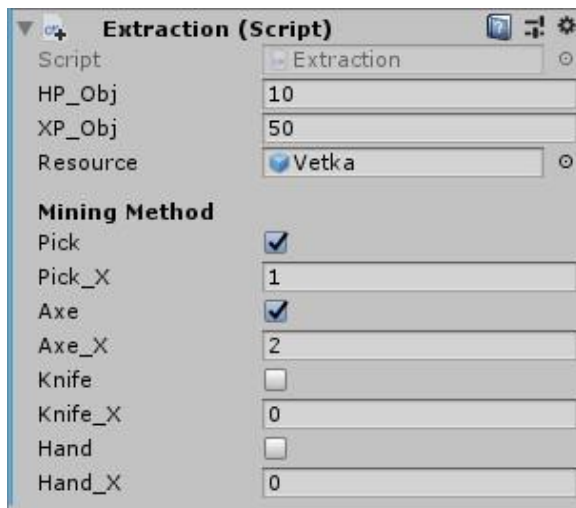
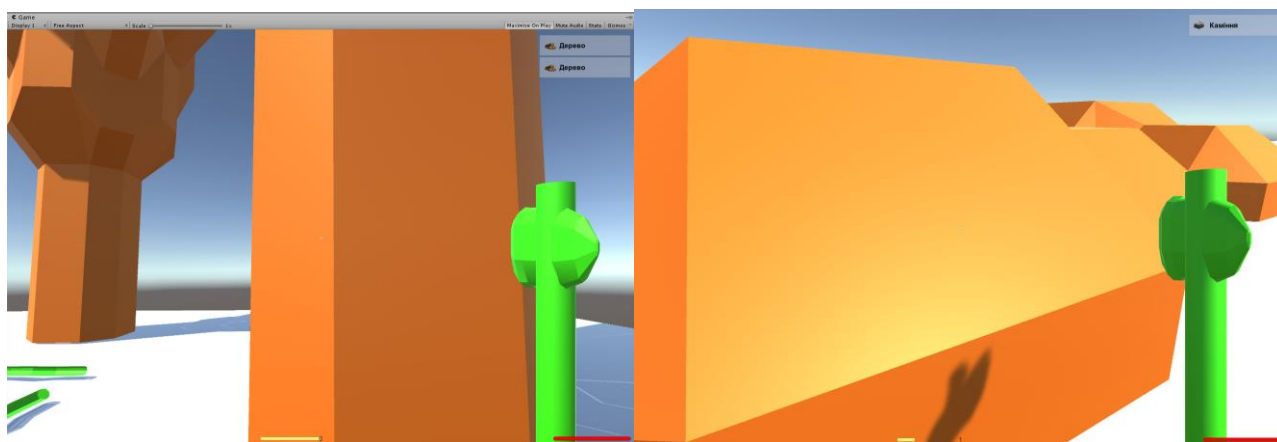


Рисунок 3.14 – Приклад ресурсу що добувається за допомогою інструменту

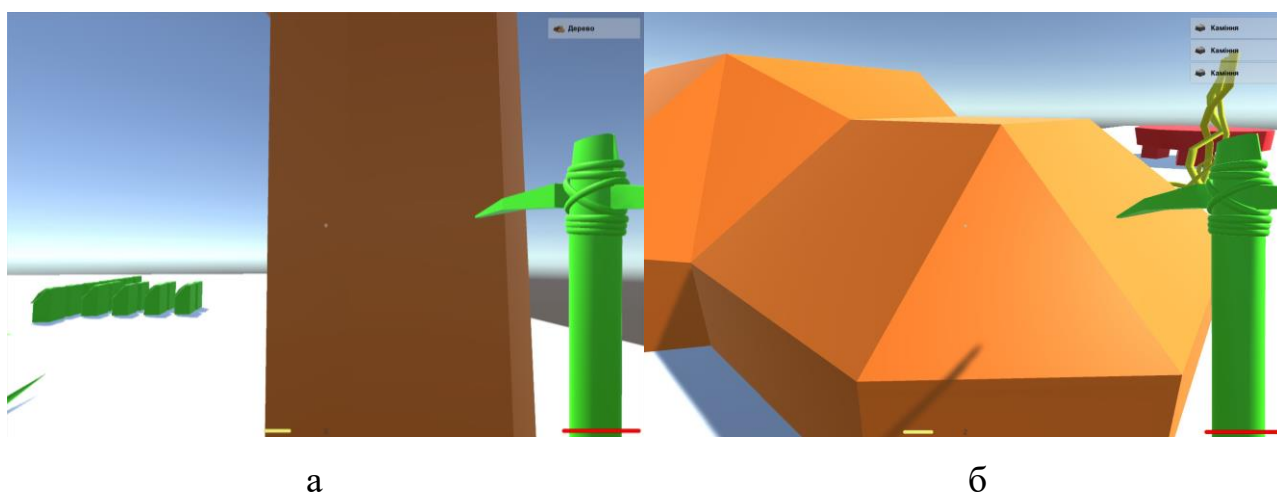


а

б

а – добування дерева; б – добування каменю

Рисунки 3.15 – Приклад збору ресурсів за допомогою примітивної сокири

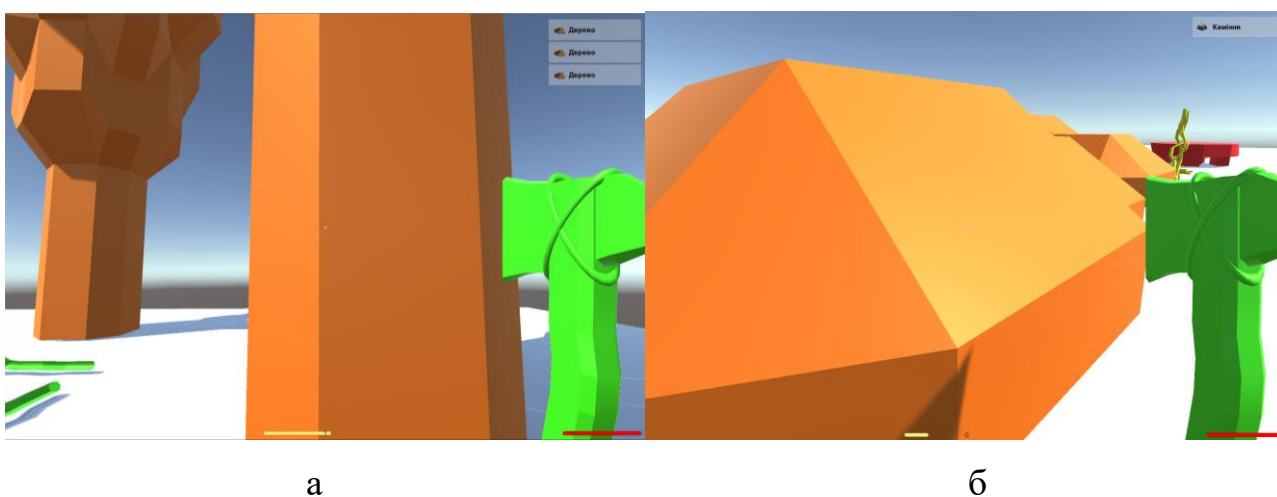


а

б

а – добування дерева; б – добування каменю

Рисунки 3.16 – Приклад збору за допомогою кирки



а

б

а – добування дерева; б – добування каменю

Рисунки 3.17 – Приклад збору за допомогою сокири

Об'єкти, які не збираються – об'єкти, які не можна добути, найчастіше в якості таких об'єктів виступають різного роду структури, що можна розміщувати, але не можна зібрати. Позначається червоним кольором.

3.2.10 База даних

База даних реалізована у вигляді розміщених в сцені об'єктів, що знаходяться поза основною картою, звідки буде здійснюватися підзавантаження об'єктів на основну карту (рис. 3.18).

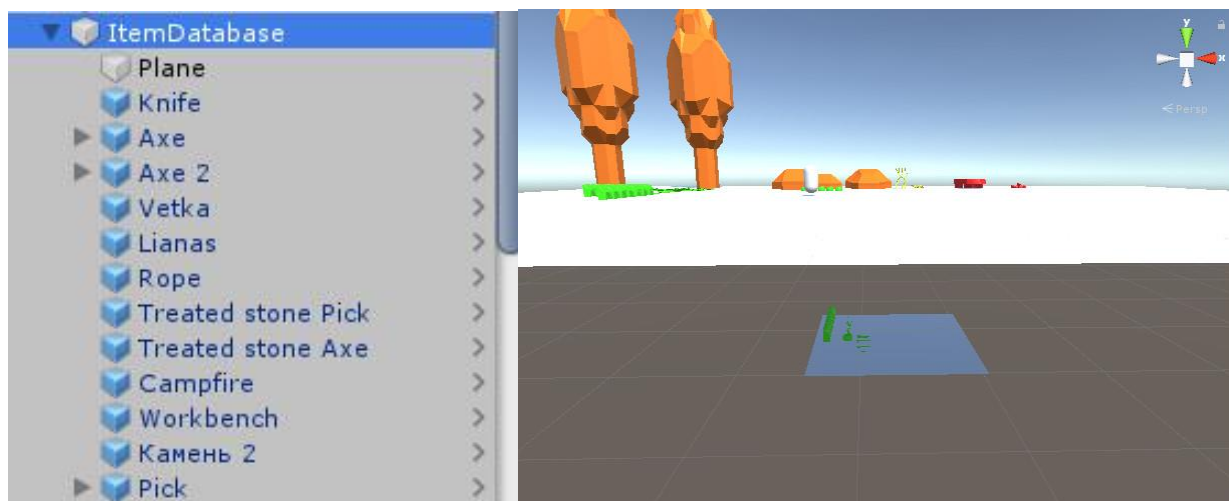


Рисунок 3.18 – Розташування бази даних у сцені

База даних реалізована таким чином, оскільки у предметів можуть бути унікальні події, які відрізняються в залежності від предмета і можуть мати кілька таких подій. Тому предмет повинен знаходитися на сцені, де вже будуть збудовані і налаштовані всі події, в іншому випадку доведеться під кожен предмет прописувати окремий схожий скрипт, але з унікальними властивостями для конкретного предмета. Система унікальних подій спрощує цей момент і робить його більш гнучким і налаштовуваним в самому редакторі.

Тут зберігаються всі ігрові об'єкти та вся інформація про них. При необхідності звідси здійснюється підзавантаження як конкретної інформації про об'єкт, так і сам об'єкт.

3.2.11 Досвід

У грі також реалізована система розвитку, по ходу гри при різних взаємодіях з ігровим світом, гравець буде отримувати досвід, який при накопиченні певної кількості переростає в новий рівень розвитку, який відкриває нові рецепти для створення. Досвід можна отримати за: видобуток ресурсів, збиральництво, створення та за використання певних предметів. Кількість одержуваного досвіду різниться в залежності від методу його отримання. Поточну кількість досвіду візуально відображено у якості жовтої смужки на екрані (рис. 3.19).



Рисунок 3.19 – Візуальне відображення поточного досвіду та рівня

3.2.12 Здоров'я

При використанні певних предметів, вони можуть впливати на здоров'я персонажа, зменшуючи або відновлюючи його. Якщо значення здоров'я впаде до нуля, персонаж загине. Поточне здоров'я візуально відображено у якості червоної смужки на екрані (рис. 3.20).

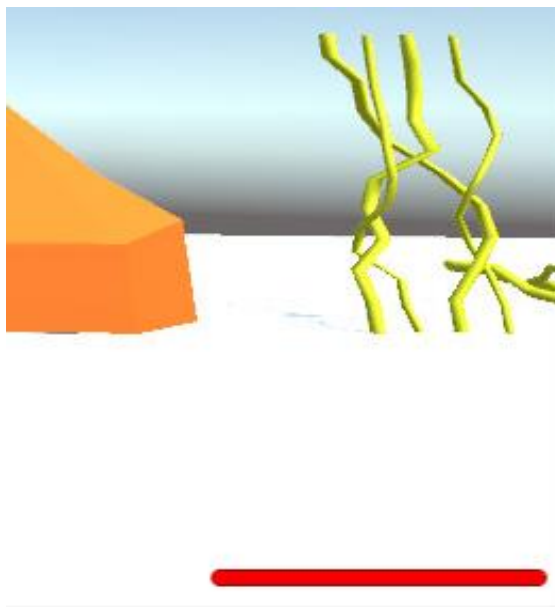


Рисунок 3.20 – Візуальне відображення поточного здоров'я

3.3 Інтерфейс

На даному етапі розробки, в проєкті існує 5 видів інтерфейсу: головне меню, ігрове меню, інвентар, меню створення та дерево розвитку.

Управління інтерфейсами здійснюється за допомогою центра управління (рис. 3.21), через який і викликаються різні інтерфейси. А також в центрі управління здійснюється призначення відповідних кнопок, що будуть викликати ці інтерфейси.

Центр управління також є ядром усіх взаємозв'язків у грі, між інтерфейсами та їхніми окремими елементами, через яке проходять усі ігрові події.

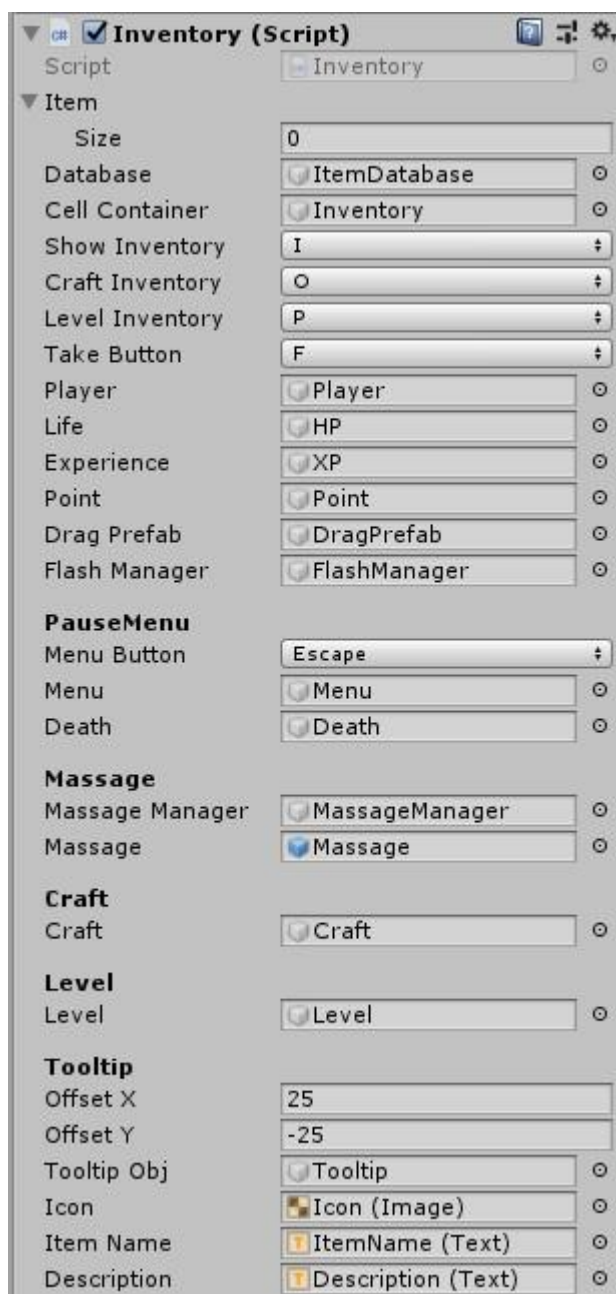


Рисунок 3.21 – Центр управління інтерфейсами

3.3.1 Головне меню

При старті гри, нас у першу чергу зустрічає головне меню, через яке гравець потрапляє до ігрового світу (рис. 3.22). Щоб розпочати гру, потрібно натиснути на кнопку «Грати», після чого запуситься ознайомчий рівень. При натисканні кнопки «Вихід», гра завершиться.



Рисунок 3.22 – Стартове меню

3.3.2 Ігрове меню

Ігрове меню викликається при натисненні на кнопку «Esc», а при повторному натисненні меню закривається (рис. 3.23). Звідси можна перейти до головного меню. Поки ігрове меню відкрите усі ігрові процеси зупиняються.



Рисунок 3.23 – Меню гри

3.3.3 Інвентар

Інвентар викликається при натисненні кнопки «І». Складається з 24 комірок, кожна з яких містить в собі: зображення, позначення кількості та смужку міцності. Приклад візуального зображення інвентарю наведено на рисунку 3.24.

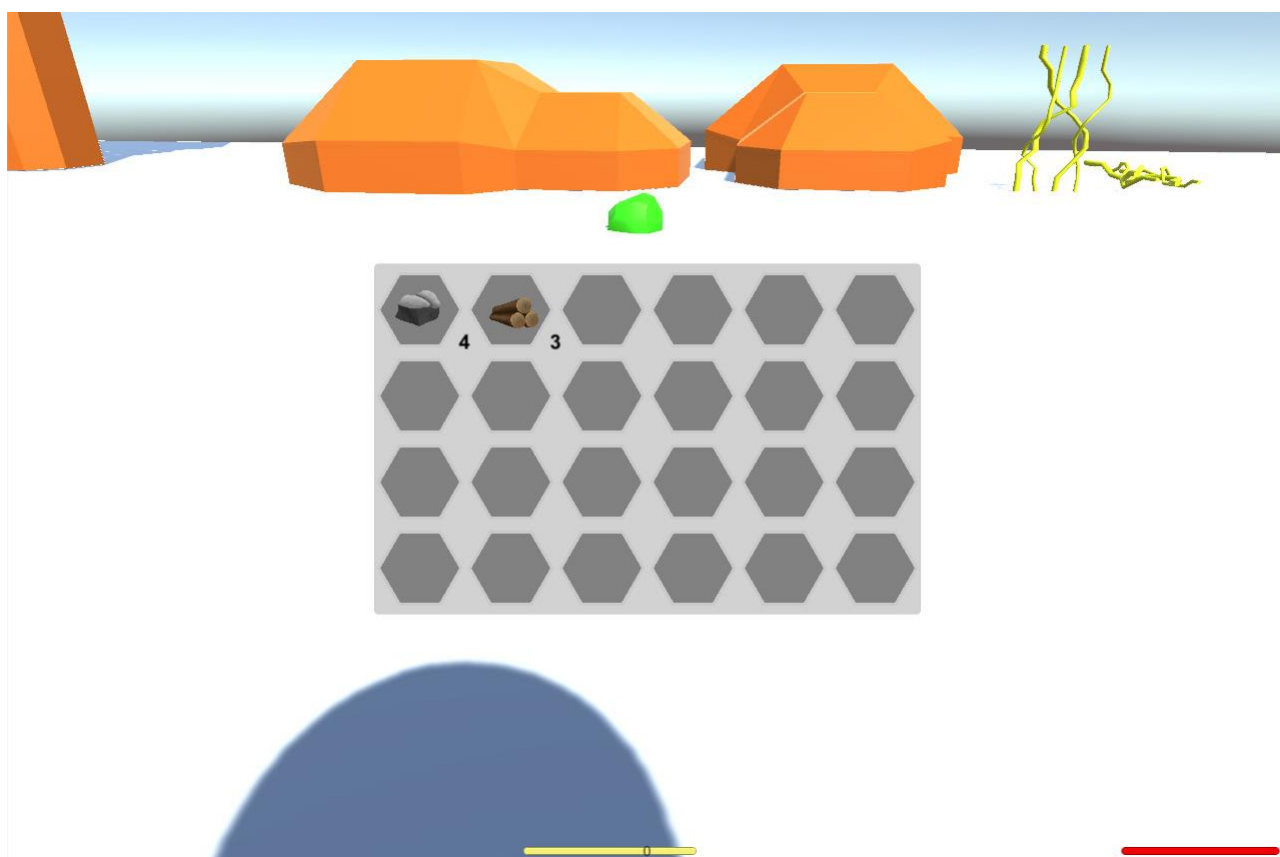


Рисунок 3.24 – Інвентар

3.3.4 Меню створення

Меню створення викликається при натисненні кнопки «О». Складається з трьох основних блоків: списку рецептів, додаткової інформації, та списком необхідних для створення ресурсів. На рисунку 3.25 зображено приклад меню створення.

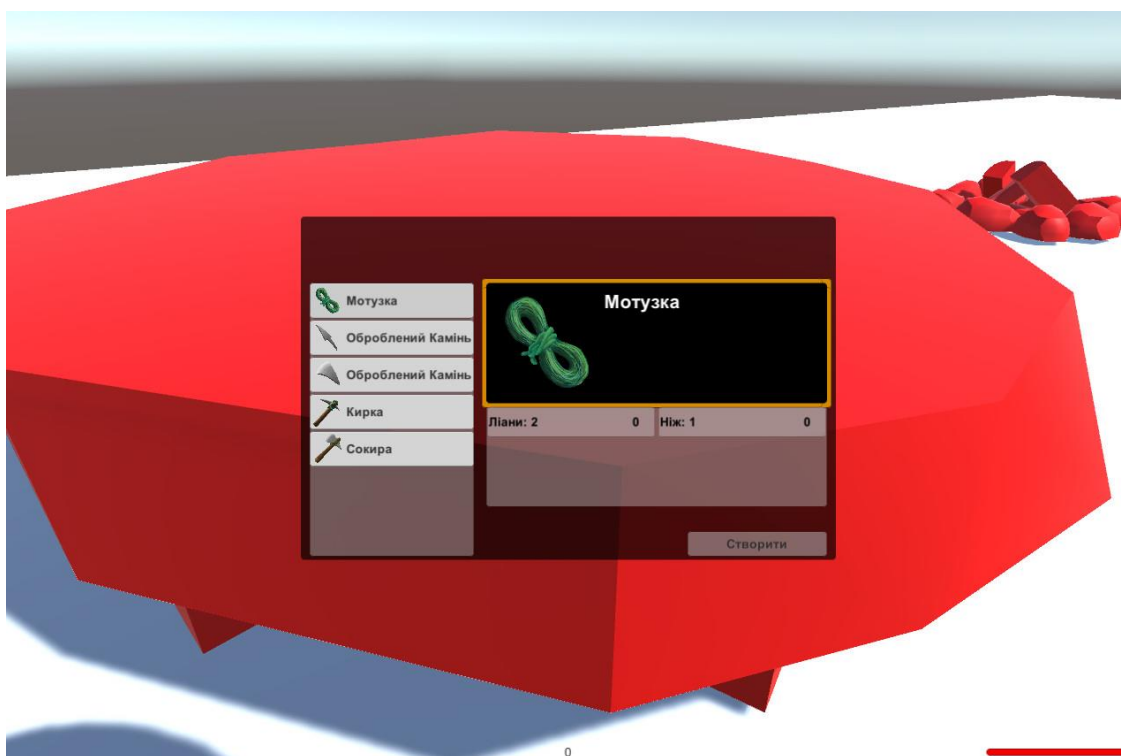
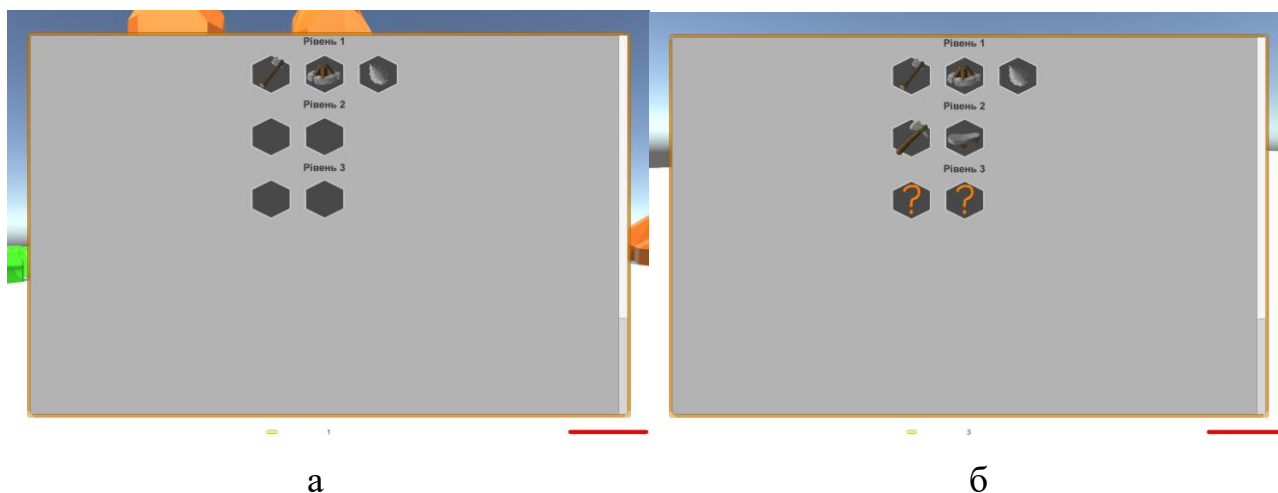


Рисунок 3.25 – Меню створення

3.3.5 Дерево розвитку

Дерево розвитку викликається при натисненні кнопки «Р». Дерево розвитку розділено по рівням доступу. На початку гри всі рецепти заблоковані, та відкриваються по мірі зростання рівня персонажу (рис 3.26а, 3.26б).



а

б

а – частково заблоковане; б – розблоковане

Рисунок 3.26 – Дерево розвитку

3.4 Візуальна частина

Усі ігрові об'єкти були створені самостійно, спеціально для цього проєкту. Проєкт нараховує 12 унікальних 3D моделей, а також 16 унікальних зображень.

Приклад ігрової графіки зображено на рисунку 3.27.

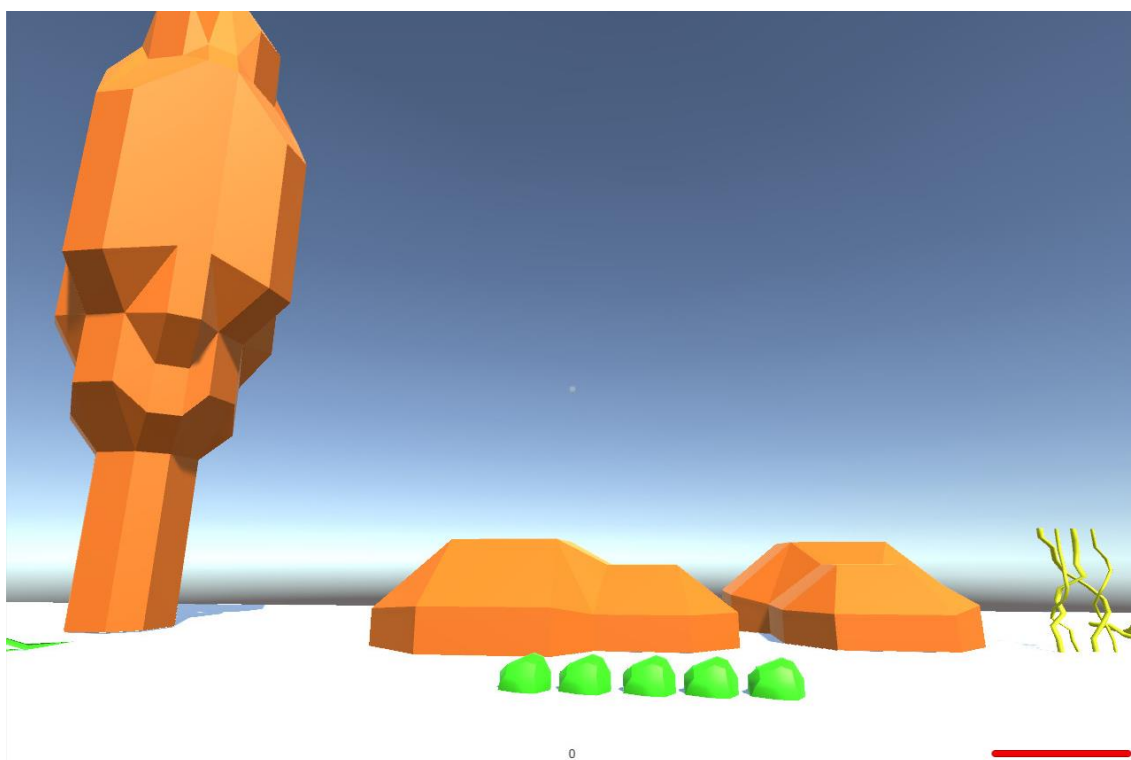


Рисунок 3.27 – Ігрова графіка

3.5 Висновки за розділом 3

В процесі роботи над проєктом було розроблено основні механіки взаємодії з ігровим світом, а також систему розвитку персонажу.

Проєкт включає в себе безліч розроблених методів, що були розглянуті у цьому розділі, завдяки яким здійснюється взаємодія між ігровими елементами та об'єктами.

Також була створена певна кількість необхідних ігрових моделей та іконок. А також 5 різних інтерфейсів, кожен з яких виконує свою роль.

4 ТЕСТУВАННЯ

4.1 Вимоги до технічних та програмних засобів

Мінімальні системні вимоги:

- операційна система (ОС): Windows XP/Vista/7/8/10;
- процесор: Intel Dual Core 2.0 GHz / AMD Athlon X2 5200+ 2.6 GHz;
- оперативна пам'ять: 1 Гб;
- відеопам'ять: 512 Мб;
- місце на жорсткому диску: 130 Мб;
- DirectX 9.0 и вище.

4.2 Посібник користувача

Для запуску гри «Відродження: Початок» необхідно запустити виконавчий файл «Відродження. Початок.exe». На рисунку 4.1 зображено вікно запуску, у якому користувач може настроїти параметри запуску гри.

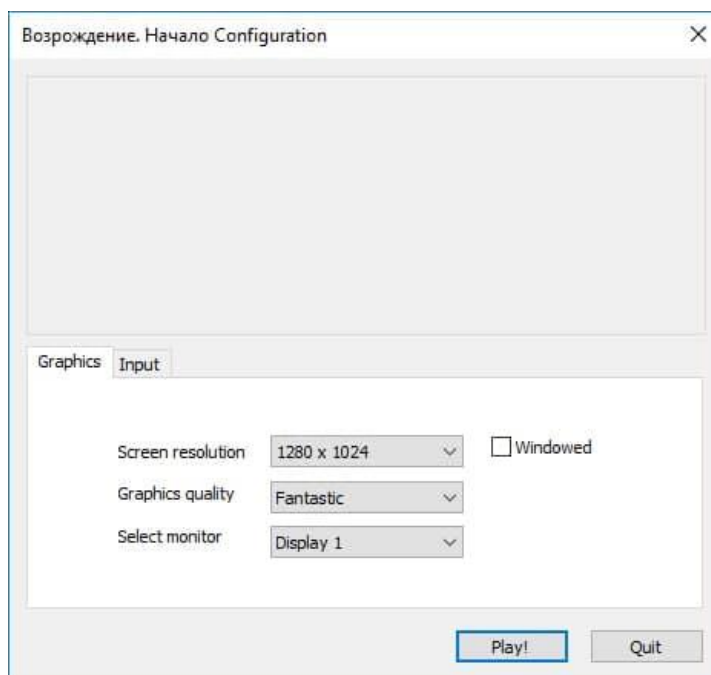


Рисунок 4.1 – Вікно запуску гри

Після вибору оптимальних параметрів, необхідно натиснути кнопку «Play!», після чого запуститься гра.

Після запуску гри, в головному меню потрібно натиснути кнопку «Грати», після чого розпочнеться гра.

Коли гравець потрапляє до ігрового світу, при собі він немає нічого, а його рівень розвитку дорівнює нулю. Керування ігровим аватаром здійснюється за допомогою клавіш «W», «A», «S», «D», розташованих на лівій частині клавіатури, а керування ігровою камерою за допомогою переміщення миші.

З цього моменту починається виживання. Для початку гравцю необхідно підвищити рівень персонажу, щоб відкрити перші рецепти створення примітивних інструментів, для цього необхідно знаходити та збирати невеликі ресурси що розташовані на карті, такі як: мале каміння та дерев'яні гілки. Щоб зібрати ресурс необхідно підійти до нього, навести центр екрану таким чином, щоб крапка у центрі екрану знаходилася на предметі, після чого натиснути кнопку «F». Усі зібрані ресурси будуть відображатися у інвентарі, який відкривається при натисненні на кнопку «I».

Коли гравець назбирає необхідну кількість ресурсів для підвищення рівня, йому відкриваються перші рецепти, які відобразяться у дереві розвитку, яке відкривається при натисненні на кнопку «P». А також рецепти з'являться в меню створення, що відкривається при натисненні на кнопку «O». Рівень гравця також відображено у нижній частині екрану, у вигляді жовтої смужки, що відображає поточну кількість досвіду, а також цифрове відображення рівня персонажу.

Після того, як було розблоковано перші рецепти, гравець повинен відкрити меню створення, у якому вибрати рецепт примітивного інструменту та створити його, для цього потрібно мати при собі необхідні ресурси, які вже збиралися раніше.

Коли перший примітивний інструмент було створено, гравець отримує можливість добувати інші види ресурсів, що до цього моменту було неможливо. Щоб використати створений інструмент, потрібно перейти до

інвентарю, де буде зберігатися інструмент і натиснути на нього лівою клавішею миші, після чого інструмент відобразиться у руках персонажу. Збір за допомогою інструмента здійснюється при наведенні на певний тип ресурсу, та натисканні лівої клавіші миші. Типи ресурсів, що збираються за допомогою інструментів: дерева, велике каміння, ліани.

Як тільки необхідна кількість ресурсів, для підвищення наступного рівня, була зібрана, гравцеві відкривається новий рецепт, що являє собою верстак, який необхідний для створення більш складних речей.

Потрібно перейти до меню створення, вибрати рецепт верстаку, та створити його. Після чого потрібно розмістити цю структуру на карті, для цього потрібно перейти до інвентарю і натиснути на верстак лівою клавішею миші. Верстак буде розташовано перед гравцем.

Після розміщення верстака на карті, потрібно їм скористатися, для цього наводимося на верстак і натискаємо на кнопку взаємодії («F»), після чого відкриється меню створення, яке буде містити в собі нові рецепти, недоступні без верстаку.

Створюємо нові, більш кращі, інструменти. Для цього потрібно підготувати необхідні компоненти, а саме:

- мотузку — яка створюється при обробці ліан ножем, рецепт доступний в верстаку;
- кам'яну основу для інструменту — яка створюється при обробці каменю в верстаку, є два рецепта основи: для кирки та сокири;
- також нам знадобиться одне дерево.

Як тільки все необхідні компоненти будуть у наявності, виготовляємо кам'яні інструменти, які є більш міцними, та ефективними.

Як тільки всі ресурси на карті будуть зібрані, знайомство з механіками гри буде завершено.

4.3 Висновки за розділом 4

При роботі над четвертим розділом було виведено мінімальні вимоги до технічного та програмного забезпечення.

А також складено керівництво користувача, що описує принцип роботи гри та поетапно проводить користувача від початку користування до завершення знайомства з основними механіками гри та ознайомчим рівнем.

ВИСНОВКИ

Під час розробки прототипу комп'ютерної гри в середовищі «Unity3D» було проаналізовано відповідну предметну область та принципи її роботи, а також розглянуто існуючі аналоги.

Для реалізації проєкту було обрано середовище розробки Unity, а у якості мови програмування C#. Розробка здійснюється для платформи персональних комп'ютерів.

Створений проєкт відповідає меті роботи та всім вимогам, що були пред'явлені до проєкту. Даний продукт є унікальним, тому що всі системи, які містить проєкт, були реалізовані без використання готових рішень.

Для прототипу було створено 12 унікальних моделей, 16 зображень, 5 інтерфейсів та безліч методів роботи програми.

Повний код проєкту наведено у додатку А.

За темою роботи є 1 наукова публікація, а саме тези доповіді на науково-практичній конференції «Тиждень Науки», яка дистанційно проводилася у Національному Університеті «Запорізька Політехніка» у квітні 2021 року [3].

План майбутнього розвитку:

- розширення існуючих механік, та додавання нових;
- індивідуалізація проєкту, шляхом додавання основної особливості гри, що буде виділяти продукт на фоні інших;
- впровадження сюжету;
- заміна всіх ігрових моделей та зображень на високоякісні;
- додавання нових унікальних рівнів зі своїми особливостями.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Краткая история игровой индустрии в разбивке по платформам [Электрон. ресурс] / – Режим доступа: <https://habr.com/ru/company/miip/blog/312884/>
2. Индустрия компьютерных игр [Электрон. ресурс] / – Режим доступа: https://ru.wikipedia.org/wiki/Индустрия_компьютерных_игр
3. Зирянов, Ю. Особливості розроблення сучасних ігор [Текст] / Ю. Зирянов, Т. Каплієнко // Тиждень науки-2021 : щорічна науково-практична конференція викладачів, науковців, молодих учених, аспірантів та студентів, Запоріжжя, 19-23 квітня 2021 р : proceedings. – Запоріжжя : подано до друку.
4. Симулятор выживания [Электрон. ресурс] / – Режим доступа: https://ru.wikipedia.org/wiki/Симулятор_выживания
5. Жанр «Выживание» в компьютерных играх [Электрон. ресурс] / – Режим доступа: https://gamesisart.ru/theory_genre_survival_kk.html
6. Minecraft [Электрон. ресурс] / – Режим доступа: <https://ru.wikipedia.org/wiki/Minecraft>
7. Ark: Survival Evolved [Электрон. ресурс] / – Режим доступа: https://ru.wikipedia.org/wiki/Ark:_Survival_Evolved
8. Rust [Электрон. ресурс] / – Режим доступа: [https://ru.wikipedia.org/wiki/Rust_\(игра\)](https://ru.wikipedia.org/wiki/Rust_(игра))
9. Движок Unreal Engine – особенности, преимущества и недостатки [Электрон. ресурс] / – Режим доступа: <https://cubiq.ru/dvizhok-unreal-engine/>
10. Движок Unity – особенности, преимущества и недостатки [Электрон. ресурс] / – Режим доступа: <https://cubiq.ru/dvizhok-unity/>
11. JavaScript – лучший язык программирования для начинающих. Так это или нет? [Электрон. ресурс] / – Режим доступа: <https://habr.com/ru/company/ruvds/blog/445272/>
12. C# – Преимущества и недостатки [Электрон. ресурс] / – Режим доступа: <https://shwanoff.ru/plus-minus-c-sharp/>

ДОДАТОК А**Код програми**

Control.cs

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

public class Control : MonoBehaviour
{
    public float moveSpeed = 10f;
    //public float turnSpeed = 52f;

    void Start()
    {

    }

    void Update()
    {
        if (Input.GetKey(KeyCode.W))
            transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime);
        if (Input.GetKey(KeyCode.S))
            transform.Translate(Vector3.back * moveSpeed * Time.deltaTime);
        if (Input.GetKey(KeyCode.A))
            transform.Translate(Vector3.left * moveSpeed * Time.deltaTime);
        if (Input.GetKey(KeyCode.D))
            transform.Translate(Vector3.right * moveSpeed * Time.deltaTime);
    }
}
```

Craft.cs

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

public class Craft : MonoBehaviour
{
    [HideInInspector]
    public Item currentItem;

    GameObject inventoryObj;
    Inventory inventory;
    CraftItem craft;

    void Start()
    {
        inventoryObj = GameObject.FindGameObjectWithTag("Global_Interface");
        inventory = inventoryObj.GetComponent<Inventory>();
        craft = inventory.craft.GetComponent<CraftItem>();
    }

    public void NewItem()
    {
        inventory.AddItem(currentItem);
        for (int i = 0; i < craft.craftManager.transform.childCount; i++)
        {
            if
(craft.craftManager.transform.GetChild(i).GetComponent<CraftItemDatabase>().craftItem.gameObject.GetComponent
<Item>() == currentItem)
            {
                Debug.Log("1");
            }
        }
    }
}
```

```

CraftItemDatabase craftItemDatabase =
craft.craftManager.transform.GetChild(i).GetComponent<CraftItemDatabase>();
for (int j = 0; j < inventory.cellContainer.transform.childCount; j++)
{
    if (craftItemDatabase.resource1 != null)
    {
        Debug.Log("2");
        if (inventory.item[j].id == craftItemDatabase.resource1.gameObject.GetComponent<Item>().id)
        {
            Debug.Log("3");
            if (inventory.item[j].hP)
            {
                Debug.Log("минус ХП инструмента");
                inventory.item[j].hPItem--;
            }
            else
            {
                Debug.Log("минус ресурс");
                inventory.item[j].countItem -= craftItemDatabase.numberItems1;
            }
        }
    }
}
if (craftItemDatabase.resource2 != null)
{
    if (inventory.item[j].id == craftItemDatabase.resource2.gameObject.GetComponent<Item>().id)
    {
        if (inventory.item[j].hP)
        {
            Debug.Log("минус ХП инструмента");
            inventory.item[j].hPItem--;
        }
        else
        {
            Debug.Log("минус ресурс");
            if (inventory.item[j].countItem > craftItemDatabase.numberItems2)
            {
                inventory.item[j].countItem -= craftItemDatabase.numberItems2;
            }
            else
            {
                inventory.item[j] = new Item();
                inventory.DisplayItems();
            }
        }
    }
}
if (craftItemDatabase.resource3 != null)
{
    if (inventory.item[j].id == craftItemDatabase.resource3.gameObject.GetComponent<Item>().id)
    {
        if (inventory.item[j].hP)
        {
            Debug.Log("минус ХП инструмента");
            inventory.item[j].hPItem--;
        }
        else
        {
            Debug.Log("минус ресурс");
            if (inventory.item[j].countItem > craftItemDatabase.numberItems3)
            {
                inventory.item[j].countItem -= craftItemDatabase.numberItems3;
            }
        }
    }
}

```

```

        else
        {
            inventory.item[j] = new Item();
            inventory.DisplayItems();
        }
    }
}

}
if (craftItemDatabase.resource4 != null)
{
    if (inventory.item[j].id == craftItemDatabase.resource4.gameObject.GetComponent<Item>().id)
    {
        if (inventory.item[j].hP)
        {
            Debug.Log("минус ХП инструмента");
            inventory.item[j].hPItem--;
        }
        else
        {
            Debug.Log("минус ресурс");
            if (inventory.item[j].countItem > craftItemDatabase.numberItems4)
            {
                inventory.item[j].countItem -= craftItemDatabase.numberItems4;
            }
            else
            {
                inventory.item[j] = new Item();
                inventory.DisplayItems();
            }
        }
    }
}
if (craftItemDatabase.resource5 != null)
{
    if (inventory.item[j].id == craftItemDatabase.resource5.gameObject.GetComponent<Item>().id)
    {
        if (inventory.item[j].hP)
        {
            Debug.Log("минус ХП инструмента");
            inventory.item[j].hPItem--;
        }
        else
        {
            Debug.Log("минус ресурс");
            if (inventory.item[j].countItem > craftItemDatabase.numberItems5)
            {
                inventory.item[j].countItem -= craftItemDatabase.numberItems5;
            }
            else
            {
                inventory.item[j] = new Item();
                inventory.DisplayItems();
            }
        }
    }
}
if (craftItemDatabase.resource6 != null)
{
    if (inventory.item[j].id == craftItemDatabase.resource6.gameObject.GetComponent<Item>().id)
    {
        if (inventory.item[j].hP)

```



```

void Update()
{
    Text txt = xP.GetComponent<Text>();
    int level = int.Parse(txt.text);

    if (level > 0)
    {
        for (int i = 0; i < 3; i++)
        {
            craftManager.transform.GetChild(i).gameObject.SetActive(true);
        }
    }
    if (level > 1)
    {
        craftManager.transform.GetChild(3).gameObject.SetActive(true);
    }
}
}

```

CraftItemDatabase.cs

```

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

public class CraftItemDatabase : MonoBehaviour, IPointerClickHandler
{
    public GameObject craft;
    public GameObject craftItem;

    [Header("Resources for crafting")]
    //public bool isCrafting; //имеет ли предмет рецепт крафта
    //уровень доступа рецепта
    public bool workbench_lvl1; //место крафта
    //инструмент
    public GameObject resource1;
    public int numberItems1;
    public GameObject resource2;
    public int numberItems2;
    public GameObject resource3;
    public int numberItems3;
    public GameObject resource4;
    public int numberItems4;
    public GameObject resource5;
    public int numberItems5;
    public GameObject resource6;
    public int numberItems6;

    GameObject inventoryObj;
    Inventory inventory;

    void Start()
    {
        inventoryObj = GameObject.FindGameObjectWithTag("Global_Interface");
        inventory = inventoryObj.GetComponent<Inventory>();
    }

    public void OnPointerClick(PointerEventData eventData)
    {

```

```

if (eventData.button == PointerEventData.InputButton.Left)
{
    craft.GetComponent<CraftItem>().buttonCraft.gameObject.GetComponent<Button>().interactable = true;

    for (int i = 0; i < 3; i++)
    {
        craft.GetComponent<CraftItem>().tooltipPanel.transform.GetChild(i).gameObject.SetActive(true);
    }
    Image icon = craft.GetComponent<CraftItem>().tooltipPanel.transform.GetChild(0).GetComponent<Image>();
    icon.sprite = craftItem.GetComponent<Item>().icon;
    Text nameItem =
craft.GetComponent<CraftItem>().tooltipPanel.transform.GetChild(1).GetComponent<Text>();
    nameItem.text = craftItem.GetComponent<Item>().nameItem;
    Text descriptionItem =
craft.GetComponent<CraftItem>().tooltipPanel.transform.GetChild(2).GetComponent<Text>();
    descriptionItem.text = craftItem.GetComponent<Item>().descriptionItem;

    for (int i = 0; i < 6; i++)
    {
        craft.GetComponent<CraftItem>().resourcesGroup.transform.GetChild(i).gameObject.SetActive(false);
    }

    if (resource1 != null)
    {
        Transform resource = craft.GetComponent<CraftItem>().resourcesGroup.transform.GetChild(0);
        resource.gameObject.SetActive(true);
        Text nameResource = resource.GetChild(0).GetComponent<Text>();
        nameResource.text = resource1.GetComponent<Item>().nameItem + ": " + numberItems1;
        Text availableResources = resource.GetChild(1).GetComponent<Text>();
        for (int i = 0; i < inventory.cellContainer.transform.childCount; i++)
        {
            if (inventory.item[i].id == resource1.gameObject.GetComponent<Item>().id)
            {
                availableResources.text = inventory.item[i].countItem.ToString();
            }
        }
        int countItem = int.Parse(availableResources.text);
        //Debug.Log(countItem + " pecypcov");
        if (countItem < numberItems1)
        {
            craft.GetComponent<CraftItem>().buttonCraft.gameObject.GetComponent<Button>().interactable = false;
        }
    }
    if (resource2 != null)
    {
        Transform resource = craft.GetComponent<CraftItem>().resourcesGroup.transform.GetChild(1);
        resource.gameObject.SetActive(true);
        Text nameResource = resource.GetChild(0).GetComponent<Text>();
        nameResource.text = resource2.GetComponent<Item>().nameItem + ": " + numberItems2;
        Text availableResources = resource.GetChild(1).GetComponent<Text>();
        for (int i = 0; i < inventory.cellContainer.transform.childCount; i++)
        {
            if (inventory.item[i].id == resource2.gameObject.GetComponent<Item>().id)
            {
                availableResources.text = inventory.item[i].countItem.ToString();
            }
        }
        int countItem = int.Parse(availableResources.text);
        //Debug.Log(countItem + " pecypcov");
        if (countItem < numberItems2)
        {
            craft.GetComponent<CraftItem>().buttonCraft.gameObject.GetComponent<Button>().interactable = false;
        }
    }
}

```

```

    }
}
if (resource3 != null)
{
    Transform resource = craft.GetComponent<CraftItem>().resourcesGroup.transform.GetChild(2);
    resource.gameObject.SetActive(true);
    Text nameResource = resource.GetChild(0).GetComponent<Text>();
    nameResource.text = resource3.GetComponent<Item>().nameItem + ": " + numberItems3;
    Text availableResources = resource.GetChild(1).GetComponent<Text>();
    for (int i = 0; i < inventory.cellContainer.transform.childCount; i++)
    {
        if (inventory.item[i].id == resource3.gameObject.GetComponent<Item>().id)
        {
            availableResources.text = inventory.item[i].countItem.ToString();
        }
    }
    int countItem = int.Parse(availableResources.text);
    //Debug.Log(countItem + " pecypcov");
    if (countItem < numberItems3)
    {
        craft.GetComponent<CraftItem>().buttonCraft.gameObject.GetComponent<Button>().interactable = false;
    }
}
if (resource4 != null)
{
    Transform resource = craft.GetComponent<CraftItem>().resourcesGroup.transform.GetChild(3);
    resource.gameObject.SetActive(true);
    Text nameResource = resource.GetChild(0).GetComponent<Text>();
    nameResource.text = resource4.GetComponent<Item>().nameItem + ": " + numberItems4;
    Text availableResources = resource.GetChild(1).GetComponent<Text>();
    for (int i = 0; i < inventory.cellContainer.transform.childCount; i++)
    {
        if (inventory.item[i].id == resource4.gameObject.GetComponent<Item>().id)
        {
            availableResources.text = inventory.item[i].countItem.ToString();
        }
    }
    int countItem = int.Parse(availableResources.text);
    //Debug.Log(countItem + " pecypcov");
    if (countItem < numberItems4)
    {
        craft.GetComponent<CraftItem>().buttonCraft.gameObject.GetComponent<Button>().interactable = false;
    }
}
if (resource5 != null)
{
    Transform resource = craft.GetComponent<CraftItem>().resourcesGroup.transform.GetChild(4);
    resource.gameObject.SetActive(true);
    Text nameResource = resource.GetChild(0).GetComponent<Text>();
    nameResource.text = resource5.GetComponent<Item>().nameItem + ": " + numberItems5;
    Text availableResources = resource.GetChild(1).GetComponent<Text>();
    for (int i = 0; i < inventory.cellContainer.transform.childCount; i++)
    {
        if (inventory.item[i].id == resource5.gameObject.GetComponent<Item>().id)
        {
            availableResources.text = inventory.item[i].countItem.ToString();
        }
    }
    int countItem = int.Parse(availableResources.text);
    //Debug.Log(countItem + " pecypcov");
    if (countItem < numberItems5)
    {

```

```

        craft.GetComponent<CraftItem>().buttonCraft.gameObject.GetComponent<Button>().interactable = false;
    }
}
if (resource6 != null)
{
    Transform resource = craft.GetComponent<CraftItem>().resourcesGroup.transform.GetChild(5);
    resource.gameObject.SetActive(true);
    Text nameResource = resource.GetChild(0).GetComponent<Text>();
    nameResource.text = resource6.GetComponent<Item>().nameItem + ": " + numberItems6;
    Text availableResources = resource.GetChild(1).GetComponent<Text>();
    for (int i = 0; i < inventory.cellContainer.transform.childCount; i++)
    {
        if (inventory.item[i].id == resource6.gameObject.GetComponent<Item>().id)
        {
            availableResources.text = inventory.item[i].countItem.ToString();
        }
    }
    int countItem = int.Parse(availableResources.text);
    //Debug.Log(countItem + " pecypcов");
    if (countItem < numberItems6)
    {
        craft.GetComponent<CraftItem>().buttonCraft.gameObject.GetComponent<Button>().interactable = false;
    }
}

craft.GetComponent<CraftItem>().buttonCraft.GetComponent<Craft>().currentItem =
craftItem.GetComponent<Item>();
}
}
}

```

CurrentItem.cs

```

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

public class CurrentItem : MonoBehaviour, IPointerClickHandler, IDropHandler
{
    [HideInInspector]
    public int index;

    GameObject inventoryObj;
    Inventory inventory;

    // Start is called before the first frame update
    void Start()
    {
        inventoryObj = GameObject.FindGameObjectWithTag("Global_Interface");
        inventory = inventoryObj.GetComponent<Inventory>();
    }

    public void OnPointerClick(PointerEventData eventData)
    {
        if (eventData.button == PointerEventData.InputButton.Left)
        {
            if (inventory.item[index].customEvent != null)
            {
                if (inventory.item[index].hp) //inventory.item[index].customEvent.Equals(inventory.flashManager)
                {

```

```

        //Debug.Log("Инструмент демедж");
        inventory.item[index].customEvent.Invoke();
        inventory.activeItem = index;
    }
    else
    {
        inventory.item[index].customEvent.Invoke();
    }
}
if (inventory.item[index].isRemovable)
{
    Remove();
}
if (inventory.item[index].structure) //Добавление объекта в сцену (к камере)
{

    GameObject structureObj = Instantiate(inventory.item[index].structureObject); //создание объекта в сцене
    structureObj.transform.position = Camera.main.transform.position + (Camera.main.transform.forward * 5.0f);
// размещение объекта на расстоянии перед камерой
    Remove();
}
}

if (eventData.button == PointerEventData.InputButton.Right)
{
    if (inventory.item[index].isDropped)
    {
        Drop();
        Remove();
    }
}
}

public void Remove()
{
    if (inventory.item[index].id != 0)
    {
        if (inventory.item[index].countItem > 1)
        {
            inventory.item[index].countItem--;
        }
        else
        {
            inventory.flashManager.GetComponent<SwitchFlash>().primitiveAxe.SetActive(false);
            inventory.flashManager.GetComponent<SwitchFlash>().pick.SetActive(false);
            inventory.flashManager.GetComponent<SwitchFlash>().axe.SetActive(false);
            inventory.flashManager.GetComponent<SwitchFlash>().knife.SetActive(false);
            inventory.item[index] = new Item();
        }
        inventory.DisplayItems();
    }
}

public void Drop()
{
    if (inventory.item[index].id != 0)
    {
        for (int i = 0; i < inventory.database.transform.childCount; i++)
        {
            Item item = inventory.database.transform.GetChild(i).GetComponent<Item>();
            if (item)
            {

```

```

        if (inventory.item[index].id == item.id)
        {
            GameObject droppedObj = Instantiate(item.gameObject);
            droppedObj.transform.position = Camera.main.transform.position + Camera.main.transform.forward;
            if(inventory.item[index].hP)
            {
                droppedObj.GetComponent<Item>().hPItem = inventory.item[index].hPItem;
            }
        }
    }
}

public void OnDrop(PointerEventData eventData)
{
    GameObject draggedObject = Drag.draggedObject;
    if(draggedObject == null)
    {
        return;
    }

    CurrentItem currentdragetItem = draggedObject.GetComponent<CurrentItem>();

    if (currentdragetItem)
    {
        Item currentItem = inventory.item[GetComponent<CurrentItem>().index];
        inventory.item[GetComponent<CurrentItem>().index] = inventory.item[currentdragetItem.index];
        inventory.item[currentdragetItem.index] = currentItem;
        inventory.DisplayItems();
    }
}
}

```

Death.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

public class Death : MonoBehaviour
{
    public GameObject buttonLoad;
    public GameObject player;
    public GameObject globalInterface;

    [System.Serializable]
    public class Position
    {
        public float x;
        public float y;
        public float z;
    }

    void Start()
    {
        buttonLoad.SetActive(false);
    }
}

```

```

void Update()
{
    if (File.Exists(Application.dataPath + "/saves/saveTest.al"))
    {
        buttonLoad.SetActive(true);
    }
}

public void ExitMenu()
{
    Application.LoadLevel(0);
}
}

```

DestroyMessages.cs

```

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

public class DestroyMessages : MonoBehaviour
{
    public float time = 1;

    // Update is called once per frame
    void Update()
    {
        Destroy(gameObject, time);
    }
}

```

Drag.cs

```

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

public class Drag : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler
{
    public static GameObject draggedObject;

    Inventory inventory;

    // Start is called before the first frame update
    void Start()
    {
        inventory = GameObject.FindGameObjectWithTag("Global_Interface").GetComponent<Inventory>();
    }

    // Update is called once per frame
    void Update()
    {
    }

    public void OnBeginDrag(PointerEventData eventData)
    {
        draggedObject = gameObject;
        inventory.dragPrefab.SetActive(true);
    }
}

```

```

inventory.dragPrefab.GetComponent<CanvasGroup>().blocksRaycasts = false;

if(dragedObject.GetComponent<CurrentItem>())
{
    int index = draggedObject.GetComponent<CurrentItem>().index;
    inventory.dragPrefab.GetComponent<Image>().sprite = inventory.item[index].icon;
    if(inventory.item[index].countItem > 1)
    {
        inventory.dragPrefab.transform.GetChild(0).GetComponent<Text>().text =
inventory.item[index].countItem.ToString();
    }
    else
    {
        inventory.dragPrefab.transform.GetChild(0).GetComponent<Text>().text = null;
    }
    if(inventory.dragPrefab.GetComponent<Image>().sprite == null)
    {
        draggedObject = null;
        inventory.dragPrefab.SetActive(false);
    }
}
inventory.flashManager.GetComponent<SwitchFlash>().primitiveAxe.SetActive(false);
inventory.flashManager.GetComponent<SwitchFlash>().pick.SetActive(false);
inventory.flashManager.GetComponent<SwitchFlash>().axe.SetActive(false);
inventory.flashManager.GetComponent<SwitchFlash>().knife.SetActive(false);
}

public void OnDrag(PointerEventData eventData)
{
    inventory.dragPrefab.transform.position = Input.mousePosition;
}

public void OnEndDrag(PointerEventData eventData)
{
    if(!EventSystem.current.IsPointerOverGameObject())
    {
        draggedObject.GetComponent<CurrentItem>().Drop();
        draggedObject.GetComponent<CurrentItem>().Remove();
    }

    draggedObject = null;
    inventory.dragPrefab.SetActive(false);
    inventory.dragPrefab.GetComponent<CanvasGroup>().blocksRaycasts = true;
}
}

```

Experience.cs

```

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

public class Experience : MonoBehaviour
{
    void Update()
    {
        if (GetComponent<Slider>().value == GetComponent<Slider>().maxValue)
        {
            Transform level = transform.GetChild(2);
            Text txt = level.GetComponent<Text>();
            int i = int.Parse(txt.text);

```

```

        i++;
        txt.text = i.ToString();
        GetComponent<Slider>().value = 0;
        GetComponent<Slider>().maxValue *= 2;
    }
}

public void AddExperience(int value)
{
    GetComponent<Slider>().value += value;
}
}

```

Extraction.cs

```

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

public class Extraction : MonoBehaviour
{
    public int hP_Obj;
    public int xP_Obj;

    public GameObject resource;

    [Header("Mining Method")]
    public bool pick;
    public double pick_X;
    public bool axe;
    public double axe_X;
    public bool knife;
    public double knife_X;
    public bool hand;
    public double hand_X;
}

```

Helthbar.cs

```

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

public class Helthbar : MonoBehaviour
{
    public void AddHealth(int value)
    {
        GetComponent<Slider>().value += value;
    }
}

```

Instrument.cs

```

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

public class Instrument : MonoBehaviour
{
    public GameObject interfaceObj;
}

```

```

public GameObject experience;

public KeyCode keyCode;

public int speed;
public double multiplier;

[Header("Type of instrument")]
public bool pick;
public bool axe;
public bool knife;

[Header("Bonus")]
public int percent;

Inventory inventory;

int I = 0;

// Start is called before the first frame update
void Start()
{
    inventory = interfaceObj.GetComponent<Inventory>();
}

// Update is called once per frame
void Update()
{
    if (Input.GetKeyDown(keyCode))
    {
        if (inventory.cellContainer.activeSelf == false && inventory.craft.activeSelf == false &&
inventory.level.activeSelf == false)
        {
            Ray ray = Camera.main.ScreenPointToRay(new Vector2(Screen.width / 2, Screen.height / 2));
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit, 2f))
            {
                if (hit.collider.GetComponent<Extraction>())
                {
                    Extraction extraction = hit.collider.GetComponent<Extraction>();
                    Item currentItem = hit.collider.GetComponent<Extraction>().resource.GetComponent<Item>();
                    if (speed < 2)
                    {
                        if (I == 0)
                        {
                            I++;
                        }
                        else if (I == 1)
                        {
                            TypeInstrument(extraction, currentItem);
                            I--;
                        }
                    }
                }
                else if (speed > 2)
                {
                    TypeInstrument(extraction, currentItem);
                    TypeInstrument(extraction, currentItem);
                }
                else if (speed == 2)
                {
                    TypeInstrument(extraction, currentItem);
                }
            }
        }
    }
}

```

```

        //Bonus(extraction, currentItem);
        //Damage(extraction);
    }
}

}

}

void TypeInstrument(Extraction extraction, Item currentItem)
{
    if (pick == true && extraction.pick == true)
    {
        for (int i = 0; i < (extraction.pick_X * multiplier); i++)
        {
            inventory.Message(currentItem);
            inventory.AddItem(currentItem);
        }
        Bonus(extraction, currentItem);
        Damage(extraction);
    }

    if (axe == true && extraction.axe == true)
    {
        for (int i = 0; i < (extraction.axe_X * multiplier); i++)
        {
            inventory.Message(currentItem);
            inventory.AddItem(currentItem);
        }
        Bonus(extraction, currentItem);
        Damage(extraction);
    }

    if (knife == true && extraction.knife == true)
    {
        for (int i = 0; i < (extraction.knife_X * multiplier); i++)
        {
            inventory.Message(currentItem);
            inventory.AddItem(currentItem);
        }
        Bonus(extraction, currentItem);
        Damage(extraction);
    }
}

void Bonus(Extraction extraction, Item currentItem)
{
    //Получить случайное число (в диапазоне от А до В)
    int value = Random.Range(0, 100);

    if (value <= percent)
    {
        inventory.Message(currentItem);
        inventory.AddItem(currentItem);
    }
}

void Damage(Extraction extraction)
{
    inventory.DamadeHPItem();
    extraction.hP_Obj = extraction.hP_Obj - speed;
    if (extraction.hP_Obj <= 0)

```

```

    {
        experience.GetComponent<Slider>().value += extraction.xP_Obj;
        Destroy(extraction.gameObject);
    }
}

```

Inventory.cs

```

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

public class Inventory : MonoBehaviour
{
    [HideInInspector]
    public List<Item> item;

    public GameObject database;

    public GameObject cellContainer;
    public KeyCode showInventory;
    public KeyCode craftInventory;
    public KeyCode levelInventory;
    public KeyCode takeButton;

    public GameObject player;
    public GameObject life;
    public GameObject experience;
    public GameObject point;

    public GameObject dragPrefab;

    public GameObject flashManager;

    [Header("PauseMenu")]
    public KeyCode menuButton;
    public GameObject menu;
    public GameObject death;

    [Header("Message")]
    public GameObject messageManager;
    public GameObject message;

    [Header("Craft")]
    public GameObject craft;

    [Header("Level")]
    public GameObject level;

    [Header("Tooltip")]
    public float offsetX;
    public float offsetY;

    public GameObject tooltipObj;
    public Image icon;
    public Text itemName;
    public Text description;

    // Start is called before the first frame update
    void Start()

```

```

{
    menu.SetActive(false);
    death.SetActive(false);

    Control controlScript = player.GetComponent<Control>();
    controlScript.enabled = true;
    MouseControl mouseControlScript = player.GetComponent<MouseControl>();
    mouseControlScript.enabled = true;
    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;
    point.SetActive(true); //показать поинт (центр экрана)
    Time.timeScale = 1; //возвращение времени

    tooltipObj.SetActive(false);

    item = new List<Item>();

    cellContainer.SetActive(false);
    for (int i = 0; i < cellContainer.transform.childCount; i++)
    {
        cellContainer.transform.GetChild(i).GetComponent<CurrentItem>().index = i;
    }

    for (int i = 0; i < cellContainer.transform.childCount; i++)
    {
        item.Add(new Item());
    }

    craft.SetActive(false);
    level.SetActive(false);
}

// Update is called once per frame
void Update()
{
    ToggleInventory();
    PauseMenu();
    Death();

    if (Input.GetKeyDown(takeButton))
    {
        if (cellContainer.activeSelf == false && craft.activeSelf == false && level.activeSelf == false)
        {
            Ray ray = Camera.main.ScreenPointToRay(new Vector2(Screen.width / 2, Screen.height / 2));
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit, 2f))
            {
                if (hit.collider.GetComponent<Item>())
                {
                    Item currentItem = hit.collider.GetComponent<Item>();
                    Message(currentItem);
                    AddItem(hit.collider.GetComponent<Item>());
                }
                if (hit.collider.GetComponent<Extraction>())
                {
                    Extraction extraction = hit.collider.GetComponent<Extraction>();
                    Item currentItem = hit.collider.GetComponent<Extraction>().resource.GetComponent<Item>();
                    if (extraction.hand == true)
                    {
                        Message(currentItem);
                        AddItem(currentItem);
                        extraction.hP_Obj = extraction.hP_Obj - 1;
                    }
                }
            }
        }
    }
}

```

```

        if (extraction.hP_Obj <= 0)
        {
            experience.GetComponent<Slider>().value += extraction.xP_Obj;
            Destroy(extraction.gameObject);
        }
    }
}
if (hit.collider.GetComponent<Workbench>())
{
    craft.SetActive(true);
    for (int i = 0; i < craft.GetComponent<CraftItem>().craftManager.transform.childCount; i++)
    {
        CraftItemDatabase itemDatabase =
craft.GetComponent<CraftItem>().craftManager.transform.GetChild(i).GetComponent<CraftItemDatabase>();
        if (itemDatabase.workbench_lv11)
        {
craft.GetComponent<CraftItem>().craftManager.transform.GetChild(i).gameObject.SetActive(true);
        }
    }

    Control controlScript = player.GetComponent<Control>();
    controlScript.enabled = false;
    MouseControl mouseControlScript = player.GetComponent<MouseControl>();
    mouseControlScript.enabled = false;
    Cursor.visible = true;
    Cursor.lockState = CursorLockMode.None;
    point.SetActive(false); //скрыть поинт (центр экрана)
    cellContainer.SetActive(false);
    level.SetActive(false);
    }
}
}
}
}

public void Massage(Item currentItem) //Сообщения о подборе
{
    GameObject msgObj = Instantiate(message);
    msgObj.transform.SetParent(messageManager.transform);

    Image msgImg = msgObj.transform.GetChild(0).GetComponent<Image>();
    msgImg.sprite = currentItem.icon;

    Text msgTxt = msgObj.transform.GetChild(1).GetComponent<Text>();
    msgTxt.text = currentItem.nameItem;
}

public void AddItem (Item currentItem) //Добавление предмета в инвентарь
{
    if (currentItem.isStackable)
    {
        AddStackableItem(currentItem);
    }
    else
    {
        AddUnstackableItem(currentItem);
    }
}

void AddUnstackableItem(Item currentItem)
{

```

```

for (int i = 0; i < item.Count; i++)
{
    if (item[i].id == 0)
    {
        item[i] = currentItem;
        item[i].countItem = 1;
        DisplayItems();
        experience.GetComponent<Slider>().value += 10;
        if (currentItem.transform.position.y > -1)//database.transform.position.y)
        {
            Destroy(currentItem.gameObject);
        }
        break;
    }
}
}

void AddStackableItem(Item currentItem)
{
    for (int i = 0; i < item.Count; i++)
    {
        if (item[i].id==currentItem.id)
        {
            item[i].countItem++;
            DisplayItems();
            experience.GetComponent<Slider>().value += 10;
            if (currentItem.transform.position.y > -1)//database.transform.position.y)
            {
                Destroy(currentItem.gameObject);
            }
            return;
        }
    }
    AddUnstackableItem(currentItem);
}

void ToggleInventory() //Открытие и закрытие инвентаря
{
    if (Input.GetKeyDown(showInventory))
    {
        if (cellContainer.activeSelf)
        {
            cellContainer.SetActive(false);

            Control controlScript = player.GetComponent<Control>();
            controlScript.enabled = true;
            MouseControl mouseControlScript = player.GetComponent<MouseControl>();
            mouseControlScript.enabled = true;
            Cursor.visible = false;
            Cursor.lockState = CursorLockMode.Locked;
            point.SetActive(true); //показать поинт (центр экрана)
        }
        else
        {
            cellContainer.SetActive(true);

            Control controlScript = player.GetComponent<Control>();
            controlScript.enabled = false;
            MouseControl mouseControlScript = player.GetComponent<MouseControl>();
            mouseControlScript.enabled = false;
            Cursor.visible = true;
            Cursor.lockState = CursorLockMode.None;
        }
    }
}

```

```

        point.SetActive(false); //скрыть поинт (центр экрана)
        craft.SetActive(false);
        level.SetActive(false);
    }
}

if (Input.GetKeyDown(craftInventory))
{
    if (craft.activeSelf)
    {
        for (int i = 0; i < craft.GetComponent<CraftItem>().craftManager.transform.childCount; i++)
        {
            CraftItemDatabase itemDatabase =
craft.GetComponent<CraftItem>().craftManager.transform.GetChild(i).GetComponent<CraftItemDatabase>();
            if (itemDatabase.workbench_lv11)
            {
                craft.GetComponent<CraftItem>().craftManager.transform.GetChild(i).gameObject.SetActive(false);
            }
        }
        craft.SetActive(false);

        Control controlScript = player.GetComponent<Control>();
        controlScript.enabled = true;
        MouseControl mouseControlScript = player.GetComponent<MouseControl>();
        mouseControlScript.enabled = true;
        Cursor.visible = false;
        Cursor.lockState = CursorLockMode.Locked;
        point.SetActive(true); //показать поинт (центр экрана)

        for (int i = 0; i < 3; i++)
        {
            craft.GetComponent<CraftItem>().tooltipPanel.transform.GetChild(i).gameObject.SetActive(false);
        }
        for (int i = 0; i < 6; i++)
        {
            craft.GetComponent<CraftItem>().resourcesGroup.transform.GetChild(i).gameObject.SetActive(false);
        }
craft.GetComponent<CraftItem>().resourcesGroup.transform.GetChild(i).GetChild(1).GetComponent<Text>().text =
"0";
    }

    craft.GetComponent<CraftItem>().buttonCraft.GetComponent<Craft>().currentItem = null;
    craft.GetComponent<CraftItem>().buttonCraft.gameObject.GetComponent<Button>().interactable = false;
}
else
{
    craft.SetActive(true);

    Control controlScript = player.GetComponent<Control>();
    controlScript.enabled = false;
    MouseControl mouseControlScript = player.GetComponent<MouseControl>();
    mouseControlScript.enabled = false;
    Cursor.visible = true;
    Cursor.lockState = CursorLockMode.None;
    point.SetActive(false); //скрыть поинт (центр экрана)
    cellContainer.SetActive(false);
    level.SetActive(false);
}
}

if (Input.GetKeyDown(levelInventory))
{

```

```

if (level.activeSelf)
{
    level.SetActive(false);

    Control controlScript = player.GetComponent<Control>();
    controlScript.enabled = true;
    MouseControl mouseControlScript = player.GetComponent<MouseControl>();
    mouseControlScript.enabled = true;
    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;
    point.SetActive(true); //показать поинт (центр экрана)
}
else
{
    level.SetActive(true);

    Control controlScript = player.GetComponent<Control>();
    controlScript.enabled = false;
    MouseControl mouseControlScript = player.GetComponent<MouseControl>();
    mouseControlScript.enabled = false;
    Cursor.visible = true;
    Cursor.lockState = CursorLockMode.None;
    point.SetActive(false); //скрыть поинт (центр экрана)
    cellContainer.SetActive(false);
    craft.SetActive(false);
}
}

public void PauseMenu()
{
    if (Input.GetKeyDown(menuButton))
    {
        if(menu.activeSelf)
        {
            menu.SetActive(false);

            Control controlScript = player.GetComponent<Control>();
            controlScript.enabled = true;
            MouseControl mouseControlScript = player.GetComponent<MouseControl>();
            mouseControlScript.enabled = true;
            Cursor.visible = false;
            Cursor.lockState = CursorLockMode.Locked;
            point.SetActive(true); //показать поинт (центр экрана)
            Time.timeScale = 1; //возвращение времени
        }
        else
        {
            menu.SetActive(true);

            Control controlScript = player.GetComponent<Control>();
            controlScript.enabled = false;
            MouseControl mouseControlScript = player.GetComponent<MouseControl>();
            mouseControlScript.enabled = false;
            Cursor.visible = true;
            Cursor.lockState = CursorLockMode.None;
            point.SetActive(false); //скрыть поинт (центр экрана)
            Time.timeScale = 0; //остановка времени
        }
    }
}

```

```

public void Death()
{
    if (life.GetComponent<Slider>().value <= 0)
    {
        death.SetActive(true);

        Control controlScript = player.GetComponent<Control>();
        controlScript.enabled = false;
        MouseControl mouseControlScript = player.GetComponent<MouseControl>();
        mouseControlScript.enabled = false;
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;
        point.SetActive(false); //скрыть поинт (центр экрана)
        Time.timeScale = 0; //остановка времени

        menu.SetActive(false);
        cellContainer.SetActive(false);
        craft.SetActive(false);
        level.SetActive(false);
    }
}

public void DisplayItems() //Отображение предметов и инвентаре
{
    for (int i = 0; i < item.Count; i++)
    {
        Transform cell = cellContainer.transform.GetChild(i);
        Transform icon = cell.GetChild(0);
        Transform count = icon.GetChild(0);
        Transform hPSlider = icon.GetChild(1);

        Slider slider = hPSlider.GetComponent<Slider>();
        GameObject sliderObj = slider.fillRect.gameObject;
        Text txt = count.GetComponent<Text>();
        Image img = icon.GetComponent<Image>();

        if (item[i].id != 0)
        {
            img.enabled = true;
            img.sprite = item[i].icon;

            if (item[i].countItem > 1)
            {
                txt.text = item[i].countItem.ToString();
            }
            else
            {
                txt.text = null;
            }

            if (item[i].hP)
            {
                slider.maxValue = item[i].fullHPItem;
                slider.value = item[i].hPItem;
                sliderObj.SetActive(true);
            }
            else
            {
                slider.value = 0;
                sliderObj.SetActive(false);
            }
        }
    }
}

```

```

        else
        {
            img.enabled = false;
            img.sprite = null;
            txt.text = null;

            slider.value = 0;
            sliderObj.SetActive(false);
        }
    }
}

[HideInInspector]
public int activeItem;

public void DamadeHPItem()
{
    if(item[activeItem].hP)
    {
        if (item[activeItem].hPItem > 0)
        {
            item[activeItem].hPItem--;
        }
        else
        {
            flashManager.GetComponent<SwitchFlash>().primitiveAxe.SetActive(false);
            flashManager.GetComponent<SwitchFlash>().pick.SetActive(false);
            flashManager.GetComponent<SwitchFlash>().axe.SetActive(false);
            flashManager.GetComponent<SwitchFlash>().knife.SetActive(false);
            item[activeItem] = new Item();
        }
        DisplayItems();
    }
}
}

```

Item.cs

```

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

public class Item : MonoBehaviour
{
    public string nameItem;
    public int id;
    [HideInInspector]
    public int countItem;
    public bool isStackable;
    [Multiline(5)]
    public string descriptionItem;
    public bool hP;
    public int fullHPItem;
    public int hPItem;
    public bool isRemovable;
    public bool isDropped;
    public Sprite icon;
    public UnityEvent customEvent;

    [Header("Structure")]
    public bool structure;
}

```

```

    public GameObject structureObject;
}

```

LevelUp.cs

```

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

public class LevelUp : MonoBehaviour
{
    public GameObject xP;
    public GameObject lvl_1;
    public GameObject lvl_2;
    public GameObject lvl_3;

    // Start is called before the first frame update
    void Start()
    {
        //lvl_1.transform.GetChild(1).GetChild(0).gameObject.SetActive(false);
        for (int i = 1; i <= 3; i++)
        {
            lvl_1.transform.GetChild(i).GetChild(0).gameObject.SetActive(false);
        }
        for (int i = 1; i <= 2; i++)
        {
            lvl_2.transform.GetChild(i).GetChild(0).gameObject.SetActive(false);
        }
        for (int i = 1; i <= 2; i++)
        {
            lvl_3.transform.GetChild(i).GetChild(0).gameObject.SetActive(false);
        }
    }

    // Update is called once per frame
    void Update()
    {
        Text txt = xP.GetComponent<Text>();
        int level = int.Parse(txt.text);
        if (level > 0)
        {
            for (int i = 1; i <= 3; i++)
            {
                lvl_1.transform.GetChild(i).GetChild(0).gameObject.SetActive(true);
            }
        }
        if (level > 1)
        {
            for (int i = 1; i <= 2; i++)
            {
                lvl_2.transform.GetChild(i).GetChild(0).gameObject.SetActive(true);
            }
        }
        if (level > 2)
        {
            for (int i = 1; i <= 2; i++)
            {
                lvl_3.transform.GetChild(i).GetChild(0).gameObject.SetActive(true);
            }
        }
    }
}

```

```
}
```

Menu.cs

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

public class Menu : MonoBehaviour
{
    public GameObject settings;

    public void Game()
    {
        Application.LoadLevel(1);
    }

    public void Settings()
    {
        settings.SetActive(!settings.activeSelf);
    }

    public void Exit()
    {
        Application.Quit();
    }

    public void setMusic(float value)
    {
        Global.music = value;
    }

    public void setSound(float value)
    {
        Global.sound = value;
    }
}
```

MouseControl.cs

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

public class MouseControl : MonoBehaviour
{
    public float sensitiveX = 3f;
    public float sensitiveY = 3f;

    public float minX = -360;
    public float maxX = 360;
    public float minY = -90;
    public float maxY = 90;

    private Quaternion originalRot;
    private float rotX = 0;
    private float rotY = 0;

    private void Start()
    {
        Cursor.visible = false; //скрытие курсора
        Cursor.lockState = CursorLockMode.Locked; //заложить курсор
    }
}
```

```

Rigidbody rb = GetComponent<Rigidbody>();

if (rb)
{
    rb.freezeRotation = true;
}

originalRot = transform.localRotation;
}

private void Update()
{
    rotX += Input.GetAxis("Mouse X") * sensitiveX;
    rotY += Input.GetAxis("Mouse Y") * sensitiveY;

    rotX = rotX % 360;
    rotY = rotY % 360;

    rotX = Mathf.Clamp(rotX, minX, maxX);
    rotY = Mathf.Clamp(rotY, minY, maxY);

    Quaternion xQuaternion = Quaternion.AngleAxis(rotX, Vector3.up);
    Quaternion yQuaternion = Quaternion.AngleAxis(rotY, Vector3.left);

    transform.localRotation = originalRot * xQuaternion * yQuaternion;
}
}

```

PauseMenu.cs

```

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

public class PauseMenu : MonoBehaviour
{
    public void Exit()
    {
        Application.LoadLevel(0);
    }
}

```

SaveLoad.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

public class SaveLoad : MonoBehaviour
{
    public GameObject player;
    public GameObject globalInterface;

    //public List<Item> item = new List<Item>();

    [System.Serializable]
    public class Position
    {

```

```

    public float x;
    public float y;
    public float z;
}

[System.Serializable]
public class ItemInfo
{
    //public GameObject cellContainer;
    public int index;
    public int id;
    public int countItem;
    public bool hP;
    public int hPItem;
}

// Start is called before the first frame update
public void Save()
{
    Position position = new Position();
    position.x = player.transform.position.x;
    position.y = player.transform.position.y;
    position.z = player.transform.position.z;

    float HP = globalInterface.GetComponent<Inventory>().life.GetComponent<Slider>().value;

    float maxXP = globalInterface.GetComponent<Inventory>().experience.GetComponent<Slider>().maxValue;
    float XP = globalInterface.GetComponent<Inventory>().experience.GetComponent<Slider>().value;
    string level =
globalInterface.GetComponent<Inventory>().experience.transform.GetChild(2).GetComponent<Text>().text;

    //5
    List<Item> currentItem = new List<Item>();
    for (int i = 0; i < globalInterface.GetComponent<Inventory>().cellContainer.transform.childCount; i++)
    {
        CurrentItem currentdragetItem =
globalInterface.GetComponent<Inventory>().cellContainer.transform.GetChild(i).GetComponent<CurrentItem>();
        if (currentdragetItem)
        {
            currentItem[i] = globalInterface.GetComponent<Inventory>().item[currentdragetItem.index];
        }
    }

    if (!Directory.Exists(Application.dataPath + "/saves"))
        Directory.CreateDirectory(Application.dataPath + "/saves");
    FileStream fs = new FileStream(Application.dataPath + "/saves/saveTest.al", FileMode.Create);
    BinaryFormatter formatter = new BinaryFormatter();
    formatter.Serialize(fs, position);
    formatter.Serialize(fs, HP);
    formatter.Serialize(fs, maxXP);
    formatter.Serialize(fs, XP);
    formatter.Serialize(fs, level);
    //formatter.Serialize(fs, item);
    //formatter.Serialize(fs, itemInfo);
    //formatter.Serialize(fs, gObject);
    formatter.Serialize(fs, currentItem);
    fs.Close();
}

// Update is called once per frame
public void Load()
{

```

```

if(File.Exists(Application.dataPath + "/saves/saveTest.al"))
{
    FileStream fs = new FileStream(Application.dataPath + "/saves/saveTest.al", FileMode.Open);
    BinaryFormatter formatter = new BinaryFormatter();
    try
    {
        Position pos = (Position)formatter.Deserialize(fs);
        player.transform.position = new Vector3(pos.x, pos.y, pos.z);

        float HP = (float)formatter.Deserialize(fs);
        globalInterface.GetComponent<Inventory>().life.GetComponent<Slider>().value = HP;

        float maxXP = (float)formatter.Deserialize(fs);
        globalInterface.GetComponent<Inventory>().experience.GetComponent<Slider>().maxValue = maxXP;
        float XP = (float)formatter.Deserialize(fs);
        globalInterface.GetComponent<Inventory>().experience.GetComponent<Slider>().value = XP;
        string level = (string)formatter.Deserialize(fs);
        globalInterface.GetComponent<Inventory>().experience.transform.GetChild(2).GetComponent<Text>().text
= level;

        //5
        List<Item> currentItem = (List<Item>)formatter.Deserialize(fs);
        for (int i = 0; i < globalInterface.GetComponent<Inventory>().cellContainer.transform.childCount; i++)
        {
            CurrentItem currentdragetItem =
globalInterface.GetComponent<Inventory>().cellContainer.transform.GetChild(i).GetComponent<CurrentItem>();
            if (currentdragetItem)
            {
                //currentItem[i] =
globalInterface.GetComponent<Inventory>().item[GetComponent<CurrentItem>().index];
                globalInterface.GetComponent<Inventory>().item[GetComponent<CurrentItem>().index] =
currentItem[i];
                //globalInterface.GetComponent<Inventory>().DisplayItems();
            }
        }
    }
    catch (System.Exception e)
    {
        Debug.Log(e.Message);
    }
    finally
    {
        fs.Close();
    }

    gameObject.SetActive(false);

    globalInterface.GetComponent<Inventory>().death.SetActive(false);
    Control controlScript = player.GetComponent<Control>();
    controlScript.enabled = true;
    MouseControl mouseControlScript = player.GetComponent<MouseControl>();
    mouseControlScript.enabled = true;
    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;
    globalInterface.GetComponent<Inventory>().point.SetActive(true); //показать поинт (центр экрана)
    Time.timeScale = 1; //возвращение времени
}
else
{
    Application.Quit();
}
}

```

```
}
```

Slot.cs

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

public class Slot : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler
{
    public Sprite activeCell;

    Sprite cell;
    Image img;

    // Start is called before the first frame update
    void Start()
    {
        img = GetComponent<Image>();
        cell = img.sprite;
    }

    void OnDisable()
    {
        img.sprite = cell;
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        img.sprite = activeCell;
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        img.sprite = cell;
    }
}
```

SwitchFlash.cs

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

public class SwitchFlash : MonoBehaviour
{
    public GameObject primitiveAxe;
    public GameObject pick;
    public GameObject axe;
    public GameObject knife;

    public void switchFlash(GameObject flash)
    {
        if(flash == primitiveAxe)
        {
            pick.SetActive(false);
            axe.SetActive(false);
            knife.SetActive(false);
            flash.SetActive(!flash.activeSelf);
        }
    }
}
```

```

else if (flash == pick)
{
    primitiveAxe.SetActive(false);
    axe.SetActive(false);
    knife.SetActive(false);
    flash.SetActive(!flash.activeSelf);
}
else if (flash == axe)
{
    primitiveAxe.SetActive(false);
    pick.SetActive(false);
    knife.SetActive(false);
    flash.SetActive(!flash.activeSelf);
}
else if (flash == knife)
{
    primitiveAxe.SetActive(false);
    pick.SetActive(false);
    axe.SetActive(false);
    flash.SetActive(!flash.activeSelf);
}
}
}

```

Tooltip.cs

```

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

public class Tooltip : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler
{
    Inventory inventory;

    void OnDisable()
    {
        inventory.tooltipObj.SetActive(false);
    }

    // Start is called before the first frame update
    void Start()
    {
        inventory = GameObject.FindGameObjectWithTag("Global_Interface").GetComponent<Inventory>();
    }

    // Update is called once per frame
    void Update()
    {
        inventory.tooltipObj.transform.position = new Vector2(Input.mousePosition.x + inventory.offsetX,
Input.mousePosition.y + inventory.offsetY);
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        CurrentItem currentItem = GetComponent<CurrentItem>();
        Item item = inventory.item[currentItem.index];

        if(item.id != 0)
        {
            inventory.tooltipObj.SetActive(true);
            inventory.icon.sprite = item.icon;
        }
    }
}

```

```
        inventory.itemName.text = item.nameItem;  
        inventory.description.text = item.descriptionItem;  
    }  
}  
  
public void OnPointerExit(PointerEventData eventData)  
{  
    inventory.tooltipObj.SetActive(false);  
}  
}
```

ДОДАТОК Б

Презентація



Рисунок Б.1 – Слайд презентації 1

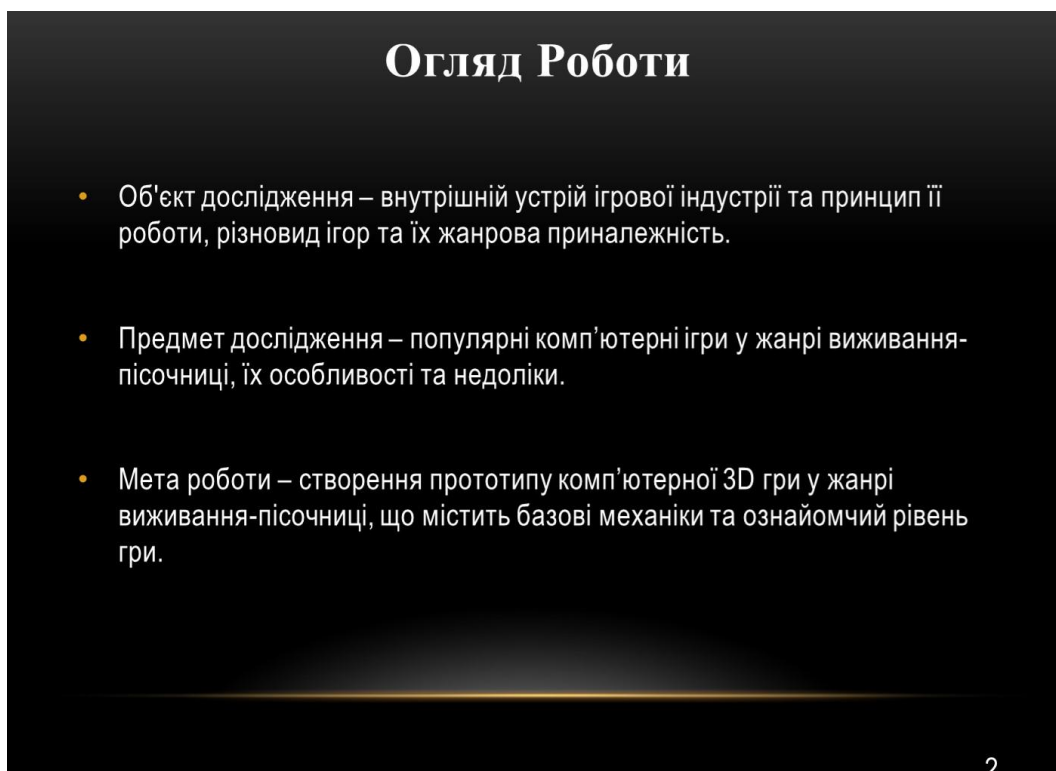
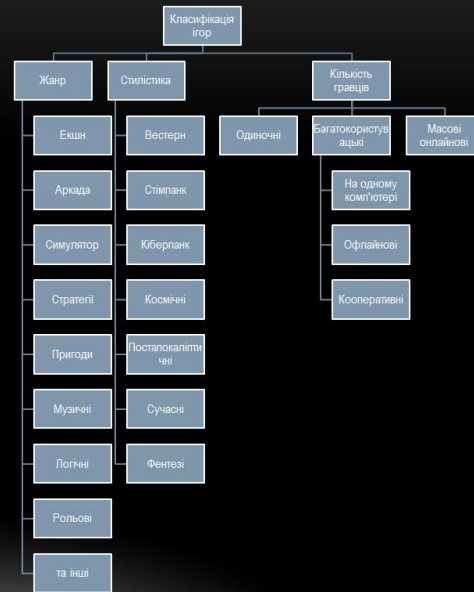


Рисунок Б.2 – Слайд презентації 2

Аналіз Предметної Області

- **Комп'ютерна гра** – це комп'ютерна програма, створена з метою розваги користувача платформи, будь-якого віку, а також може надавати розвиваючий або оздоровчий ефект.
 - **Прототип гри** – це модель майбутньої гри, в яку можна пограти і відчувати ігровий процес. На підставі прототипу можна сказати чи є майбутнє у проєкта. Прототип дає можливість дизайнеру гри відразу почати маніпулювати з ігровим світом, фізикою всередині гри, економікою і процесами. Основний сенс прототипу – це перевірити ідею на практиці і, зменшивши ризики, продовжити розроблення проєкту.
- Виходячи з обраного для розробки жанру виживання-пісочниці, розглянемо цей жанр.
- **Виживання** – жанр комп'ютерних ігор, різновид симуляторів життя, в яких основною метою гравця є збереження життя віртуального персонажа. «Виживання» відноситься до складових жанрів, які можуть базуватися на всіх основних жанрах і самі можуть бути центральним жанром гри.
 - **Пісочниця** – будівництво (створення, споруда бази) – розподіл доступних ресурсів для розвитку. У підпорядкуванні гравця знаходяться ресурси, які потрібно з розумом використовувати для виживання.



3

Рисунок Б.3 – Слайд презентації 3

Огляд Існуючих Аналогів

Розглянемо існуючі аналоги у жанрі виживання-пісочниці, які є популярними, та займають велику долю ринку, а саме: «Minecraft», «ARK: Survival Evolved» і «Rust».

- **Minecraft** – комп'ютерна інді-гра в жанрі пісочниці з елементами симулятора виживання і відкритим світом. Гра не ставить перед гравцем якихось однозначних цілей, але пропонує йому безліч можливостей і занять: так, гравець може досліджувати світ, створювати різноманітні споруди і предмети, битися з різними супротивниками.
- **ARK: Survival Evolved** – комп'ютерна багатокористувачка гра в жанрі виживання. Щоб вижити, гравці повинні побудувати базу з багаттям і зброєю, приручити диких представників тваринного світу, враховувати територіальні та природні особливості.
- **Rust** – комп'ютерна багатокористувачка гра в жанрі виживання. Розроблена за допомогою середовища Unity. Гравець починає досліджувати відкритий світ Rust з примітивними знаряддям і зброєю, але з часом доберетися до гармат і повноцінних поселень. Гравцеві доведеться вбивати диких тварин, боротися з голодом і спрагою. Втім, головна загроза буде виходити від десятків інших гравців на сервері.

4

Рисунок Б.4 – Слайд презентації 4

Обґрунтування Вибору Середовища Розробки

Порівняння середовищ розробки Unity та Unreal Engine

	Вивчення	Зручність	Направлення	Оптимізація
Unity	Легше вчити	Зручний	Невеликі проекти	Легше оптимізувати
Unreal Engine	Складніше вчити	Розроблений для професіоналів	Великі проекти	Складніше оптимізувати

Оскільки для реалізації було обрано середовище розробки Unity, розглянемо доступні мови програмування.

Порівняння мов програмування JavaScript і C#

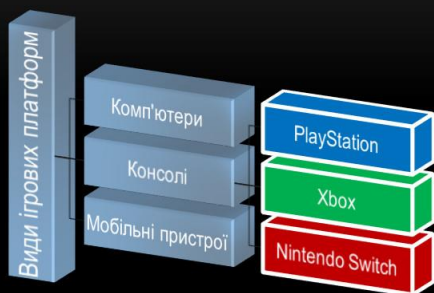
	Підтримка ООП	Можливості	Популярність	Вивчення
JavaScript	Не підтримує	Потрібні додаткові бібліотеки і фреймворки	Дуже популярний	Легше вчити
C#	Підтримує	Має багато можливостей із коробки	Дуже популярний	Складніше вчити

5

Рисунок Б.5 – Слайд презентації 5

Вибір Платформи

Популярність ігрових платформ



- Популярність ігрових платформ може сильно відрізнятися, в залежності від країни та окремого регіону



6

Рисунок Б.6 – Слайд презентації 6

Опис Проєкту

- Розроблений проєкт представляє собою комп'ютерну гру у жанрі Вживання-Пісочниця розроблену під операційну систему Windows. Проєкт розроблено в середовищі у Unity на мові програмування C#.
- На даному етапі розробки проєкт має в собі один рівень, що містить всі необхідні базові механіки, та виконує ознайомчу функцію, гравця з механіками гри.
- Для прототипу було створено 12 унікальних моделей, 16 зображень, 5 інтерфейсів та безліч методів роботи програми.
- Даний продукт є унікальним, тому що всі системи, які містить проєкт, були реалізовані без використання готових рішень.

7

Рисунок Б.7 – Слайд презентації 7

Функціональна Частина Проєкту



Схема взаємодії комп'ютерної гри

8

Рисунок Б.8 – Слайд презентації 8

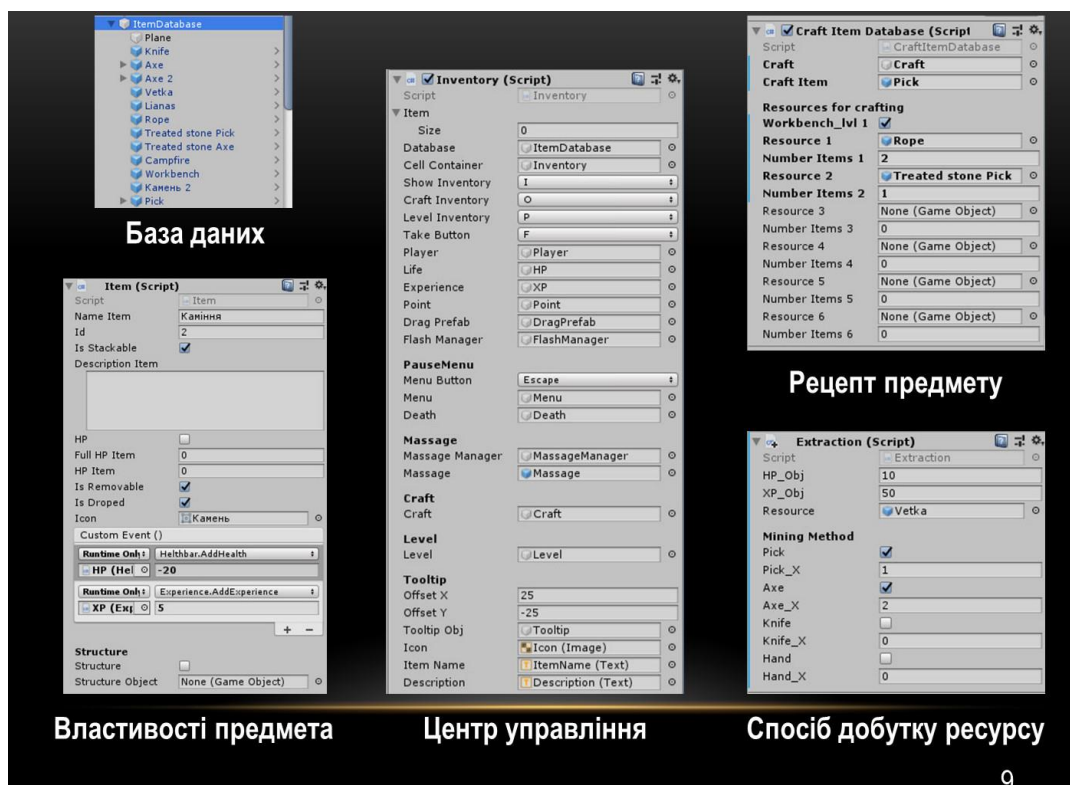


Рисунок Б.9 – Слайд презентації 9

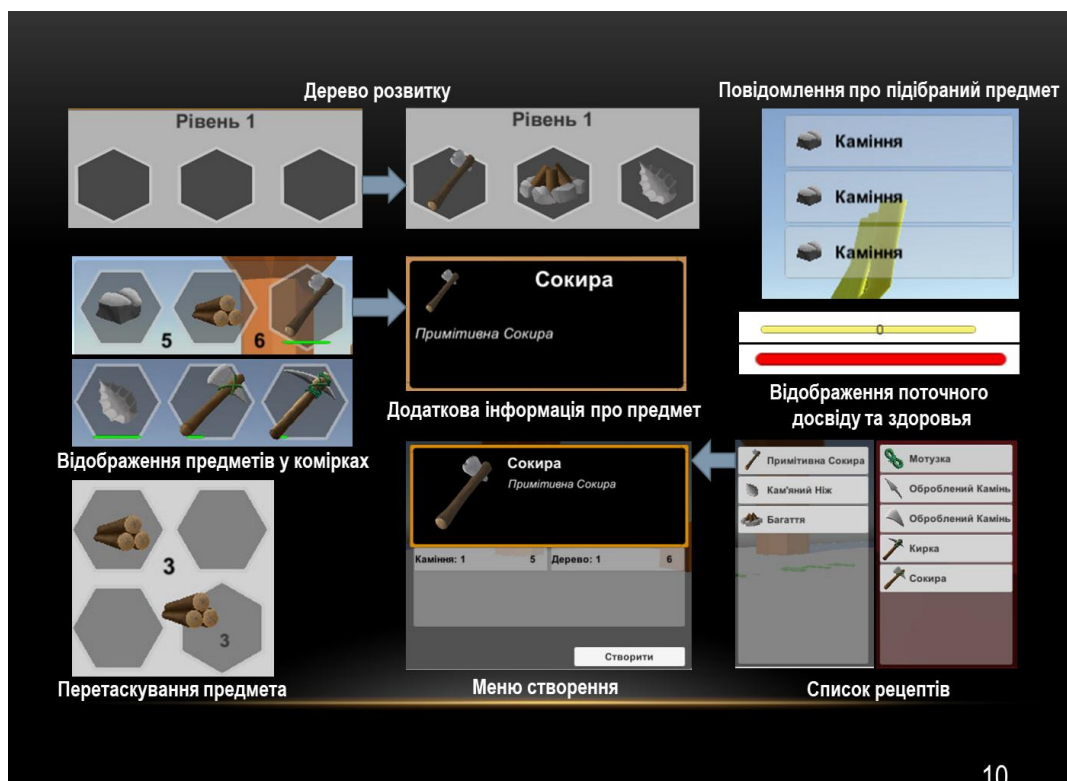


Рисунок Б.10 – Слайд презентації 10



Рисунок Б.11 – Слайд презентації 11

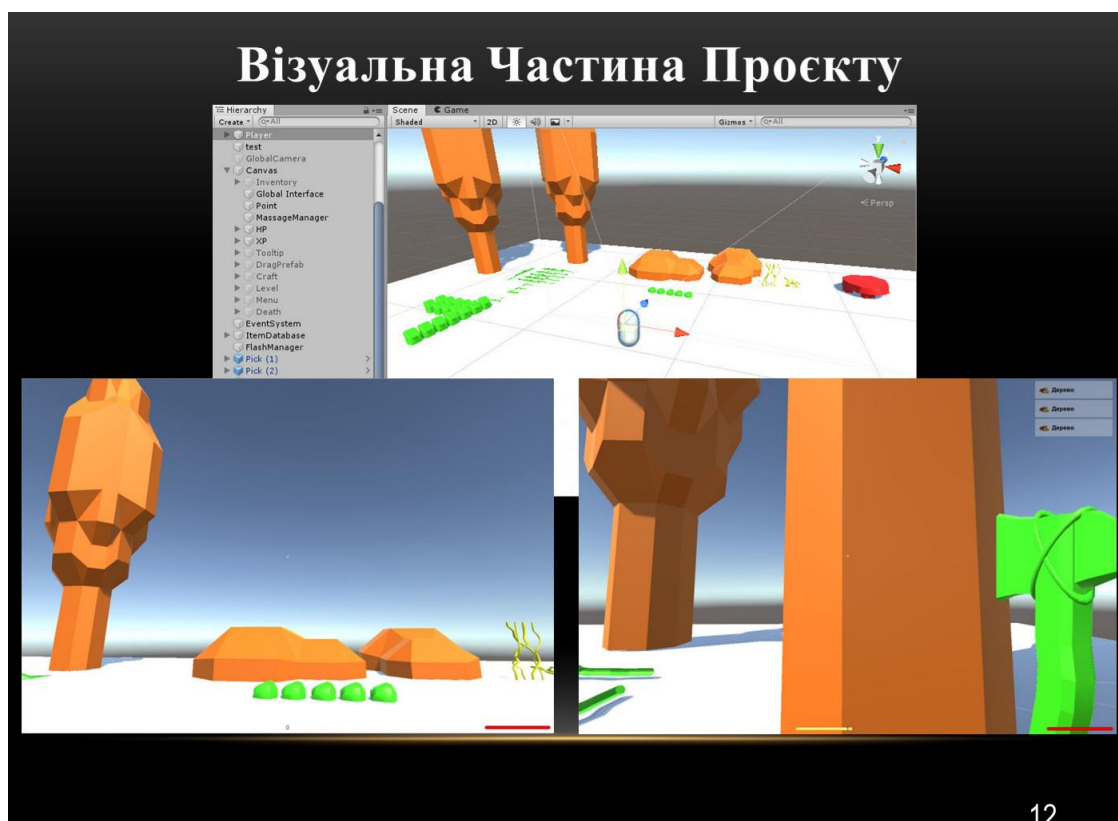


Рисунок Б.12 – Слайд презентації 12

Тестування

- При тестуванні розробленого продукту було виведено мінімальні вимоги до технічного та програмного забезпечення.
- Складено керівництво користувача, що описує принцип роботи гри, та поетапно проводить користувача від початку користування до завершення знайомства з основними механіками гри та ознайомчим рівнем.

Мінімальні системні вимоги	
операційна система (ОС):	Windows XP/Vista/7/8/10
процесор:	Intel Dual Core 2.0 GHz / AMD Athlon X2 5200+ 2.6 GHz
оперативна пам'ять:	1 Гб
відеопам'ять:	512 Мб
місце на жорсткому диску:	130 Мб
DirectX:	9.0 и вище

13

Рисунок Б.13 – Слайд презентації 13

Висновки

- В процесі роботи над проектом було розроблено основні механіки взаємодії з ігровим світом, а також систему розвитку персонажу.
- Проект включає в себе безліч розроблених методів, завдяки яким здійснюється взаємодія між ігровими елементами та об'єктами.
- Була створена певна кількість необхідних ігрових моделей та зображень.
- Створено 5 різних інтерфейсів, кожен з яких виконує свою роль.
- За темою роботи є 1 наукова публікація, а саме тези доповіді на науково-практичній конференції «Тижень Науки», яка дистанційно проводилася у Національному Університеті «Запорізька Політехніка» у квітні 2021 року.

План майбутнього розвитку:

- Розширення існуючих механік, та додавання нових;
- Індивідуалізація проекту, шляхом додавання основної особливості гри, що буде виділяти продукт на фоні інших;
- Впровадження сюжету;
- Заміна всіх ігрових моделей та зображень на високоякісні;
- Додавання нових унікальних рівнів зі своїми особливостями.

14

Рисунок Б.14 – Слайд презентації 14