Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Шевченка Кирила Миколайовича*
	(ПІБ)
академічної групи	*122М-22-3*
	(шифр)
спеціальності	*122 Комп'ютерні науки*
	(код і назва спеціальності)
освітньої програми	*«Комп'ютерні науки»*
	(назва освітньої програми)
на тему:	*Дослідження ефективності застосування штучного інтелекту для розробки комп'ютерних програм на основі грального двигуна Unity 3D*

К.М. Шевченко

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	*Проф. Алексєєв М.О.*			
економічний				

Рецензент				

Нормоконтролер				

Дніпро
2023

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

Проф. Алексєєв М.О.
(підпис) (прізвище, ініціали)

« » 20 23 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи
ступеня магістра

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

студенту *122м-22-3* (група) *Шевченко Кирилу Миколайовичу* (прізвище та ініціали)

Тема дипломного проекту *Дослідження ефективності застосування штучного інтелекту для розробки комп'ютерних програм на основі грального двигуна Unity 3D*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 09.10.2023 р. № 1277-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – Процесс розробки комп'ютерних програм на основі грального двигуна Unity 3D.

Предмет досліджень – методи та алгоритми штучного інтелекту, які використовуються у середовищі Unity 3D.

Мета НДР – оцінка ефективності штучного інтелекту у прискоренні та покращенні процесу розробки комп'ютерних програм та мобільних додатків.

Вихідні дані для проведення роботи – дослідження, що включає теорію, та експериментування створення комп'ютерних програм за допомогою алгоритмів штучного інтелекту.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Новизна запропонованих рішень полягає в інноваційній інтеграції алгоритмів штучного інтелекту в середовище Unity 3D для розробки комп'ютерних програм та нові підходи, засновані на розширенні меж традиційної розробки ігор та додатків.

Практична цінність дослідження полягає в поліпшенні розробки комп'ютерних програм шляхом впровадження ефективного інструменту у вигляді сучасного штучного інтелекту та надання розробникам практичні рекомендації, даючи можливість створювати додатки більш швидко та якісно.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

В результаті роботи для оцінки ефективності використання алгоритмів штучного інтелекту, повинна бути розроблена комп'ютерна програма чи мобільний додаток у середовищі Unity 3D, що дозволяє побачити та оцінити ефективність використання сучасного штучного інтелекту.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі	01.09.2023-31.09.2023
Дослідження методів, алгоритмів та інструментів штучного інтелекту для розробки додатків	01.09.2023-30.11.2023
Розробка мобільного додатку на основі отриманих даних та аналіз ефективності використання штучного інтелекту	01.09.2023-30.11.2023

6 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Дослідження спрямоване на отримання теоретичних та практичних навичок зі спрощення та прискорення розробки комп'ютерних програм у середовищі Unity за допомогою використання сучасних технологій штучного інтелекту.

РЕФЕРАТ

Пояснювальна записка: 86 стор., 28 рис., 2 таблиці, 2 додатка, 20 джерел.

Об'єкт дослідження – практичне застосування методів та алгоритмів штучного інтелекту з метою прискорення та покращення розробки комп'ютерних програму середовищі Unity 3D.

Предмет досліджень – методології, алгоритми та структури штучного інтелекту, які використовуються у середовищі Unity 3D.

Мета кваліфікаційної роботи - оцінка ефективності штучного інтелекту у прискоренні та покращенні процесу розробки комп'ютерних програм та мобільних додатків.

Новизна запропонованих рішень полягає в інноваційній інтеграції алгоритмів штучного інтелекту в середовище Unity 3D для розробки комп'ютерних програм та нові підходи, засновані на розширенні меж традиційної розробки ігор та додатків.

Практична цінність дослідження полягає в поліпшенні розробки комп'ютерних програм шляхом впровадження ефективного інструменту у вигляді сучасного штучного інтелекту та надання розробникам практичні рекомендації, даючи можливість створювати додатки більш швидко та якісно.

Область застосування охоплює широкий спектр галузей, включаючи ігрову індустрію, віртуальну реальність, симуляції та навчання. Результати дослідження можуть мати практичний вплив на розробників комп'ютерних програм, сприяючи створенню інтелектуальних та адаптивних програм. Крім того, вони можуть виявити значимість в сферах віртуального навчання, медичних симуляціях, архітектурного моделювання та інших областях, де застосування штучного інтелекту в Unity 3D може забезпечити вдосконалення функціональності та пришвидшення розробки.

Список ключових слів: ШТУЧНИЙ ІНТЕЛЕКТ, КОМП'ЮТЕРНА ПРОГРАМА, ІГРОВИЙ ДВИГУН, UNITY3D, VISUAL STUDIO, C#.

ABSTRACT

Explanatory Note: 86 pages, 28 figures, 2 tables, 2 appendices, 20 sources.

The object of the research is the practical application of artificial intelligence methods and algorithms to accelerate and enhance the development of computer programs in the Unity 3D environment.

The subject of the study includes methodologies, algorithms, and structures of artificial intelligence used in the Unity 3D environment.

The aim of the qualification work is to evaluate the effectiveness of artificial intelligence in accelerating and improving the process of developing computer programs and mobile applications.

The novelty of the proposed solutions lies in the innovative integration of artificial intelligence algorithms into the Unity 3D environment for program development and new approaches based on expanding the boundaries of traditional game and application development.

The practical value of the research lies in improving computer program development by introducing an efficient tool in the form of modern artificial intelligence and providing developers with practical recommendations, enabling the creation of applications more quickly and qualitatively.

The scope of application covers a wide range of industries, including the gaming industry, virtual reality, simulations, and education. The research results can have a practical impact on computer program developers, facilitating the creation of intelligent and adaptive programs. Additionally, they may prove significant in areas such as virtual learning, medical simulations, architectural modeling, and other fields where the application of artificial intelligence in Unity 3D can enhance functionality and expedite development.

Keyword list: ARTIFICIAL INTELLIGENCE, COMPUTER PROGRAM, GAMING ENGINE, UNITY 3D, VISUAL STUDIO, C#.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІС – інформаційна система;

СШІ – системи штучного інтелекту;

ШІ – штучний інтелект;

ЕОМ – електронно-обчислювальна машина;

ОС – операційна система;

ПЗ – програмне забезпечення;

ML – машинне навчання;

DL – глибинне навчання;

NLP – Natural Language Processing;

UL – самонавчання.

ЗМІСТ

ВСТУП

Актуальність роботи. Сучасна динаміка розробки комп'ютерних програм істотно змінюється під впливом інтеграції штучного інтелекту (ШІ). Це дослідження спрямоване на вивчення взаємодії ШІ та розробки програм, з фокусом на популярному движку Unity 3D. Зі збільшенням складності цифрових вражень росте потреба в розробці програм, які обладнані інтелектом, адаптивністю та динамічністю. Дослідження спрямоване на оцінку ефективності використання методів ШІ в Unity 3D з метою виявлення нюансів цієї взаємодії та її перспектив для майбутнього програмного інженерного забезпечення.

Unity 3D, як універсальний гральний двигун, стає ключовим елементом у створенні інтерактивних цифрових сценаріїв. За мірою розширення традиційної галузі розробки ігор, впровадження ШІ в Unity обіцяє відкривати нові грані функціональності, реакції та взаємодії з користувачем. Дослідження розташовує себе в центрі цього технологічного взаємодії, прагнучи звузити розрив між теоретичними аспектами ШІ та їхнім конкретним використанням у середовищі Unity 3D.

В умовах зростання використання ШІ-орієнтованих додатків у різних галузях, розуміння тонкощів інтеграції ШІ в Unity 3D має важливе практичне значення. Результати дослідження виходять за рамки розробки ігор, впливаючи на такі галузі, як віртуальне навчання, медичні симуляції та архітектурне моделювання. Шляхом покращення функціональності програм та прискорення процесу розробки дослідження робить свій внесок в еволюцію практик програмної інженерії, надаючи рецепт для створення інтелектуальних та адаптивних додатків.

Мета дослідження полягає в оцінці ефективності використання ШІ при розробці комп'ютерних програм в Unity 3D. Аналіз методологій, алгоритмів та структур ШІ в середовищі Unity спрямований на виявлення інноваційних рішень, які прискорюють процес розробки та підвищують якість і адаптивність програм. Дослідження також ставить за мету надання практичних рекомендацій для розробників, які працюють в складному світі проектів на основі ШІ в Unity 3D.

Об'єктом дослідження є взаємодія алгоритмів штучного інтелекту та ігрового двигуна Unity 3D. Зокрема, ми концентруємось на розумінні того, як методології штучного інтелекту, починаючи від машинного навчання та закінчуючи процедурною генерацією, впливають на розробку комп'ютерних програм у рамках Unity. Дослідження охоплює безліч сценаріїв: від поведінки персонажів та прийняття рішень в ігровій динаміці до оптимізації управління ресурсами та процедурної генерації контенту. Уважно вивчаючи ці аспекти, ми прагнемо запропонувати цілісне уявлення про вплив ІІ на різні аспекти розробки ігор.

Методи дослідження. Для досягнення цілей даного дослідження було здійснено ретельний вибір методів дослідження. У цьому дослідженні застосовується змішаний підхід, що поєднує якісний та кількісний аналіз, щоб забезпечити повне розуміння інтеграції штучного інтелекту у розробку комп'ютерних програм Unity 3D. Опитування та інтерв'ю з професіоналами галузі та розробниками, які використовують ШІ в Unity, дадуть якісну інформацію, а кількісні дані будуть зібрані за допомогою показників продуктивності, показників ефективності та порівняльного аналізу процесів розробки з використанням штучного інтелекту.

У наступних розділах кожен із цих аспектів буде розглянутий докладно, пропонуючи структуровану та глибоку подорож до галузі розробки Unity 3D з використанням штучного інтелекту. У міру того, як ми розкриваємо складності та потенціал цих симбіотичних відносин, дослідження прагне зробити внесок не тільки в академічну науку, а й у практичний прогрес у сфері розробки ігор, що постійно змінюється.

Особистий внесок здобувача.

Мій особистий внесок в роботу полягає в комплексному підході до дослідження, спрямованому на отримання практичних результатів та реальні вдосконалення в галузі застосування алгоритмів штучного інтелекту в середовищі Unity 3D.

Структура та обсяг роботи. Робота складається з введення, трьох розділів і висновків. Містить 86 сторінок друкарського тексту, у тому числі 49 сторінки тексту основної частини з 28 малюнками, 2 таблицями, списку використаних джерел з 20 найменуваннями та 2 додатками.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Довідкова інформація про Unity 3D та огляд середовища

Unity 3D, потужний та універсальний інструмент для розробки ігор, став синонімом інновацій в галузі інтерактивних цифрових ворожень. Цей розділ надає глибоке дослідження історії Unity 3D, вказуючи на її еволюцію, виділяючи ключові функції та надаючи комплексний огляд середовища розробки. Розуміння цієї основи є критичним для контекстуалізації подальших досліджень ефективності інтеграції штучного інтелекту (ШІ) у середовище Unity 3D для розробки комп'ютерних програм.

Еволюція Unity 3D

Unity 3D була представлена у 2005 році, спочатку орієнтована на розробників macOS. Протягом років вона пройшла помітну еволюцію, розширивши свою доступність на різні платформи, включаючи Windows, Linux, iOS, Android та інші. Успіх Unity полягає у його інтуїтивному візуальному редакторі (Рис. 1.1.), який полегшує створення як 2D, так і 3D середовища. Unity 3D гарантує реалістичні взаємодії, а графічний конвеєр дозволяє розробникам отримувати зображення високої якості. Інструменти управління ресурсами Unity оптимізують робочі процеси, підтримуючи співпрацю між художниками та програмістами. Зокрема, адаптивність Unity дозволяє розробникам створювати різноманітні проекти, від мобільних ігор до віртуальних реальностей.

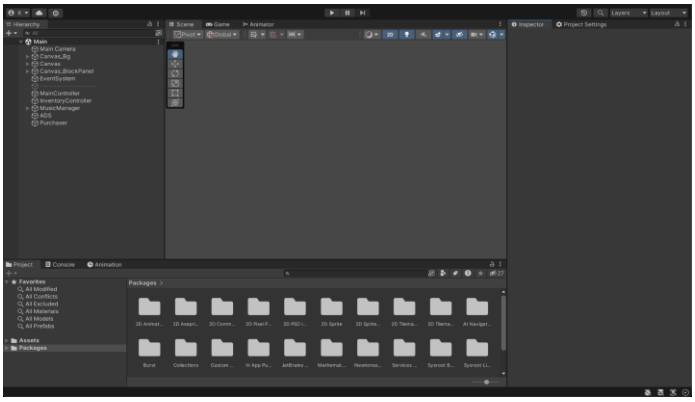

Рис. 1.1. Інтерфейс редактора Unity 3D

Екосистема Unity - це динамічний центр для розробників, що пропонує широкий вибір плагінів, ресурсів та інструментів через магазин Asset Store. Цей ринок покращує робочі процеси розробки, дозволяючи розробникам використовувати готові компоненти та прискорювати час створення проектів. Процвітаюча спільнота Unity відіграє важливу роль, з форумами, уроками та спільними просторами сприяє обміну знаннями та вирішенню проблем.

Крос-платформеність

Відзначеною рисою Unity є її вміння ефективно працювати на різних платформах. Розробники можуть створювати вміст для різних пристроїв та операційних систем, уникнувши необхідності в розширених листуваннях коду. Можливість експорту проектів на платформи, такі як консолі, мобільні пристрої та нові технології, такі як розширена реальність, підкреслює його універсальність.

1.2 Важливість штучного інтелекту у сучасній розробці програмного забезпечення

У технологічному ландшафті, що постійно розвивається, штучний інтелект (ШІ) став трансформаційною силою, яка революціонізувала різні галузі, а розробка програмного забезпечення стоїть на передньому краї цієї зміни парадигми. Інтеграція штучного інтелекту в процеси розробки програмного забезпечення стає все більш необхідною, пропонуючи розробникам потужні інструменти та методології для оптимізації робочих процесів, підвищення ефективності та створення більш інтелектуальних та адаптивних програм. У цьому есе досліджується величезна важливість штучного інтелекту в сучасній розробці програмного забезпечення, проливаючи світло на його вплив на інновації, продуктивність і загальний досвід користувача.

Одна з головних переваг впровадження штучного інтелекту в розробку програмного забезпечення полягає в його здатності значно прискорити життєвий цикл розробки. Інструменти на основі штучного інтелекту, такі як автоматичне створення коду, можуть значно скоротити час і зусилля, необхідні для рутинних

завдань кодування. Алгоритми машинного навчання можуть аналізувати величезні набори даних і виявляти шаблони, допомагаючи розробникам приймати зважені рішення та писати оптимізований код. Автоматизоване тестування з використанням штучного інтелекту забезпечує всебічне охоплення тестуванням і швидке виявлення помилок, забезпечуючи вищу якість програмного забезпечення.

Крім того, середовища розробки, керовані штучним інтелектом, можуть розуміти та адаптуватися до вподобань розробників, пропонуючи персоналізовані пропозиції, завершення коду та навіть прогнозуючи можливі проблеми. Це не тільки прискорює процес кодування, але й підвищує загальну продуктивність розробників, дозволяючи їм зосередитися на більш складних і творчих аспектах розробки програмного забезпечення.

Дані відіграють ключову роль у сучасній розробці програмного забезпечення, а штучний інтелект дає розробникам змогу повністю використовувати їхній потенціал. Алгоритми машинного навчання можуть аналізувати масивні набори даних, щоб отримати корисну інформацію, що полегшує прийняття рішень на основі даних. Ці статистичні дані інформують розробників про поведінку користувачів, продуктивність додатків і потенційні області для вдосконалення. Розуміючи переваги та шаблони користувачів, розробники можуть створювати більш персоналізовані та адаптивні програми.

Інструменти аналітики на основі штучного інтелекту також можуть передбачати потенційні проблеми та вузькі місця в процесі розробки, забезпечуючи проактивне вирішення проблем. Ця можливість прогнозування не тільки запобігає майбутнім викликам, але й дозволяє краще розподіляти ресурси та планувати, що призводить до більш ефективних циклів розробки програмного забезпечення.

Штучний інтелект зробив революцію у взаємодії з користувачем, дозволивши розробляти інтелектуальні й адаптивні програми. Обробка природної мови (NLP) і алгоритми машинного навчання сприяють створенню розмовних інтерфейсів, чат-ботів і віртуальних помічників, які покращують

залучення користувачів. Ці програми можуть розуміти та реагувати на введення користувача, забезпечуючи більш інтуїтивно зрозумілий і зручний досвід.

Крім того, ШІ дозволяє впроваджувати системи рекомендацій, які аналізують поведінку та вподобання користувачів, щоб пропонувати персоналізований контент, продукти або послуги. Це не тільки підвищує задоволеність користувачів, але й сприяє вищому рівню утримання користувачів. По суті, інтеграція штучного інтелекту в розробку програмного забезпечення має потенціал змінити те, як користувачі взаємодіють із програмними додатками та отримують від них вигоду.

Підсумовуючи, слід сказати, що впровадження штучного інтелекту в сучасну розробку програмного забезпечення є не просто тенденцією, а необхідністю в сучасному технологічному ландшафті, що швидко розвивається. Переваг багато, починаючи від підвищеної швидкості та ефективності розробки до прийняття рішень на основі даних і створення інноваційного досвіду користувача. Оскільки технології штучного інтелекту продовжують розвиватися, їхня роль у розробці програмного забезпечення, безсумнівно, стане ще більш помітною, прокладаючи шлях до майбутнього, де інтелектуальні, адаптивні та орієнтовані на користувача програми стануть нормою. Розробники та організації, які охоплюють і використовують потужність штучного інтелекту, готові стати лідерами у формуванні майбутнього розробки програмного забезпечення.

1.3 Приклади використання ШІ в середовищі Unity 3D

Інтеграція Unity 3D і штучного інтелекту (ШІ) стала революційною синергією, що відкриває нові виміри в інтерактивному та інтелектуальному віртуальному досвіді. У цьому есеї розглядаються варті уваги приклади такої інтеграції, демонструючи, як Unity 3D, потужний рушій для розробки ігор, у поєднанні з технологіями штучного інтелекту, трансформував різноманітні додатки. Від ігор до симуляцій і не тільки, ці приклади підкреслюють універсальність і трансформаційний потенціал поєднання Unity 3D зі штучним інтелектом.

Яскравим прикладом інтеграції Unity 3D і ШІ є розробка неігрових персонажів (NPC) з просунутим штучним інтелектом. Такі ігри, як "The Elder Scrolls V: Skyrim" та "Assassin's Creed", використовують Unity 3D для створення захоплюючих світів, де NPC, керовані ШІ, демонструють реалістичну поведінку. Ці персонажі адаптують свої дії відповідно до вибору гравця, пропонуючи динамічну та реалістичну взаємодію в ігровому середовищі. Універсальність Unity 3D дозволяє розробникам впроваджувати різні алгоритми ШІ для прийняття рішень, пошуку шляхів та адаптивної поведінки NPC, покращуючи загальний ігровий досвід.

Інструментарій Unity ML-Agents для навчання з підкріпленням:Агенти машинного навчання Unity (ML-Agents) - це потужний інструмент для інтеграції Unity 3D з навчанням з підкріпленням, підмножиною штучного інтелекту, орієнтованою на навчання агентів за допомогою позитивних і негативних заохочень. Дослідники та розробники використовують цей інструментарій для навчання інтелектуальних агентів у середовищах Unity. Прикладом може слугувати навчання віртуальних агентів навігації складними лабіринтами або розв'язування головоломок за допомогою алгоритмів навчання з підкріпленням. Ця інтеграція демонструє, як Unity 3D слугує основою для розвитку досліджень ШІ, особливо у сфері автономного прийняття рішень та навчання.

Інтеграція Unity 3D зі штучним інтелектом знайшла застосування за межами ігрової індустрії, зокрема в розробці реалістичних симуляторів і навчальних середовищ. Наприклад, у військових симуляторах Unity 3D у поєднанні зі штучним інтелектом дозволяє створювати розумних супротивників, які адаптуються до різних сценаріїв, забезпечуючи більш динамічний і реалістичний тренувальний процес для особового складу. Це застосування поширюється на такі сфери, як охорона здоров'я, де медичні працівники можуть практикувати процедури у віртуальних середовищах, населених об'єктами, керованими штучним інтелектом, які реалістично реагують на втручання.

Інтеграція Unity 3D з NLP демонструє потенціал для створення віртуальних розмовних агентів у додатках. Чат-боти та віртуальні персонажі,

розроблені за допомогою Unity 3D та інтегровані з NLP, можуть розуміти природну мову та реагувати на неї, підвищуючи залученість користувачів. Ця інтеграція виходить за межі ігор, знаходячи застосування в інтерактивних освітніх середовищах, віртуальних представниках служби підтримки клієнтів та інших сценаріях, де взаємодія людини і комп'ютера залежить від розуміння природної мови.

Представлені приклади підкреслюють трансформаційний вплив використання штучного інтелекту та середовища Unity 3D. Від покращення ігрового досвіду з інтелектуальними NPC до використання навчання з підкріпленням для автономних агентів - співпраця Unity 3D з технологіями штучного інтелекту відкрила нові горизонти у віртуальному середовищі. Застосування цієї інтеграції виходить за рамки розваг і охоплює симуляції, тренінги та інтерактивну освіту.

Оскільки технології продовжують розвиватися, інтеграція Unity 3D та ШІ обіцяє ще більш складні та захоплюючі додатки. Ця спільна синергія ілюструє постійно зростаючий потенціал об'єднання передових інструментів розробки ігор з можливостями штучного інтелекту, прокладаючи шлях у майбутнє, де віртуальні середовища будуть не тільки візуально приголомшливими, але й динамічно інтелектуальними та реагуючими на дії користувача.

1.4. Історичний контекст використання ШІ у розробці ігор

Інтеграція штучного інтелекту (ШІ) в розробку ігор і додатків - це трансформаційний шлях, який суттєво вплинув на те, як ми взаємодіємо з цифровим середовищем. У цьому есе досліджується історичний контекст використання штучного інтелекту в цих галузях, простежується його еволюція від ранніх експериментів до складних додатків, які ми бачимо сьогодні.

Використання ШІ в розробці ігор можна простежити ще з середини 20-го століття. Ранні експерименти, такі як створення першої комп'ютерної програми для гри в шахи в 1950-х роках, заклали основу для вивчення

потенціалу машин для імітації людського інтелекту. З плином десятиліть штучний інтелект проникав в ігри через системи, засновані на правилах, які керували поведінкою NPC (неігрових персонажів).

У 1980-х роках такі ігри, як "Космічні загарбники" та "Пакман", мали спрощені алгоритми ШІ, які контролювали рухи та поведінку ворога. Ці перші спроби, хоч і примітивні за сьогоднішніми мірками, стали першими кроками у використанні штучного інтелекту для покращення ігрового досвіду.

Розвиток експертних систем і штучного інтелекту

У 1990-х роках спостерігався сплеск використання експертних систем і штучного інтелекту, заснованого на правилах, у розробці ігор. Розробники ігор почали використовувати дерева рішень і скриптові моделі поведінки для створення більш динамічних і чуйних NPC. Такі ігри, як "F.E.A.R." і "Black & White", продемонстрували досягнення в галузі штучного інтелекту, представивши персонажів з еволюціонуючими характерами та адаптивним прийняттям рішень.

Однак залежність від заздалегідь визначених правил і дерев рішень мала свої обмеження. Ігри намагалися забезпечити по-справжньому непередбачувану та реалістичну поведінку NPC, що спонукало розробників шукати більш складні підходи.

Поява машинного навчання та нейронних мереж

21 століття ознаменувалося зміною парадигми розробки ігор та додатків з відродженням машинного навчання та нейронних мереж. Розробники почали використовувати такі методи, як навчання з підкріпленням, для навчання NPC, що дозволяє їм адаптуватися і вчитися на взаємодії з гравцями. Цей перехід від систем, заснованих на правилах, до систем, заснованих на навчанні, приніс новий рівень складності та реалізму у віртуальні середовища.

Такі ігри, як "Half-Life 2" та "F.E.A.R.", використовували нейронні мережі для створення NPC, які навчалися та адаптувалися до стратегій гравця, забезпечуючи більш захоплюючий та складний досвід. Здатність штучного інтелекту розвиватися з часом на основі даних у реальному часі відкрила двері

до динамічних ігрових світів, що еволюціонують.

Unity 3D і демократизація ШІ в розробці ігор

В останні роки поява потужних рушіїв для розробки ігор, таких як Unity 3D, демократизувала інтеграцію ШІ в розробку ігор. Інструментарій ML-Agents для Unity, представлений у 2017 році, дозволив розробникам впроваджувати алгоритми навчання з підкріпленням для навчання інтелектуальних агентів у середовищах Unity. Цей прорив відкрив двері для ширшого кола розробників, які можуть експериментувати зі штучним інтелектом та впроваджувати його у свої ігри.

У сучасному світі штучний інтелект - це не просто інструмент для покращення поведінки NPC; він став творчим партнером у процесі розробки. Процедурна генерація контенту на основі алгоритмів ШІ дозволяє розробникам створювати величезні, динамічні та персоналізовані ігрові світи. Такі ігри, як "No Man's Sky", використовують ШІ для створення різноманітних планет, істот та екосистем, надаючи гравцям безпрецедентний досвід дослідження.

Крім ігор, ШІ став невід'ємною частиною розробки додатків. Обробка природної мови (NLP) та алгоритми машинного навчання використовуються для створення інтелектуальних віртуальних помічників, чат-ботів і рекомендаційних систем, покращуючи досвід у різних додатках.

Виклики та етичні міркування

Хоча розвиток штучного інтелекту в розробці ігор і додатків був вражаючим, він не позбавлений проблем. Етичні міркування, пов'язані з використанням ШІ, від конфіденційності даних до упередженості алгоритмів, вимагають ретельної уваги. Дотримання балансу між інноваціями та відповідальним використанням ШІ має вирішальне значення для забезпечення подальшого позитивного розвитку цих технологій.

Історичний контекст застосування штучного інтелекту в розробці ігор і додатків відображає шлях, позначений інноваціями, експериментами і безперервною еволюцією. Від ранніх систем, заснованих на правилах, до сучасної ери машинного навчання і нейронних мереж, ШІ став невід'ємною силою у формуванні віртуального досвіду. Оскільки технології продовжують

розвиватися, співпраця між людською творчістю та штучним інтелектом обіцяє ще більше захоплюючих можливостей.

1.5 Ключові методології та алгоритми, що використовують в штучному інтелекті для Unity

1. Системи, засновані на правилах:

Однією з основних методологій в AI для Unity є використання систем, заснованих на правилах. У цьому підході розробники визначають набір правил, які регулюють поведінку неігрових персонажів (NPC) та ігрових об'єктів. Ці правила диктують, як NPC реагують на певні стимули або дії гравця, створюючи структуроване і передбачуване середовище. Хоча системи, засновані на правилах, забезпечують міцну основу, вони мають обмеження в адаптації до складних і динамічних сценаріїв.

2. Кінцеві автомати (FSM):

Кінцеві автомати широко використовуються в Unity для проектування ШІ. Ця методологія організовує поведінку NPC у дискретні стани, а переходи відбуваються на основі заздалегідь визначених умов. FSM ефективні для моделювання персонажів з чіткими поведінковими станами, такими як бездіяльність, патрулювання, атака та відступ. Однак, FSM можуть стати громіздкими, коли мають справу з великою кількістю станів або складними процесами прийняття рішень.

3. Алгоритми пошуку шляху:

Пошук шляху є критично важливим аспектом навігації NPC у іграх, і Unity включає різні алгоритми пошуку шляху для оптимізації руху. A* (A-зірка) - популярний алгоритм, який ефективно знаходить найкоротший шлях між двома точками на сітці. Система NavMesh (Рис. 1.2.), у Unity використовує для пошуку шляхів у реальному часі, дозволяючи NPC безперешкодно орієнтуватися у динамічному середовищі.

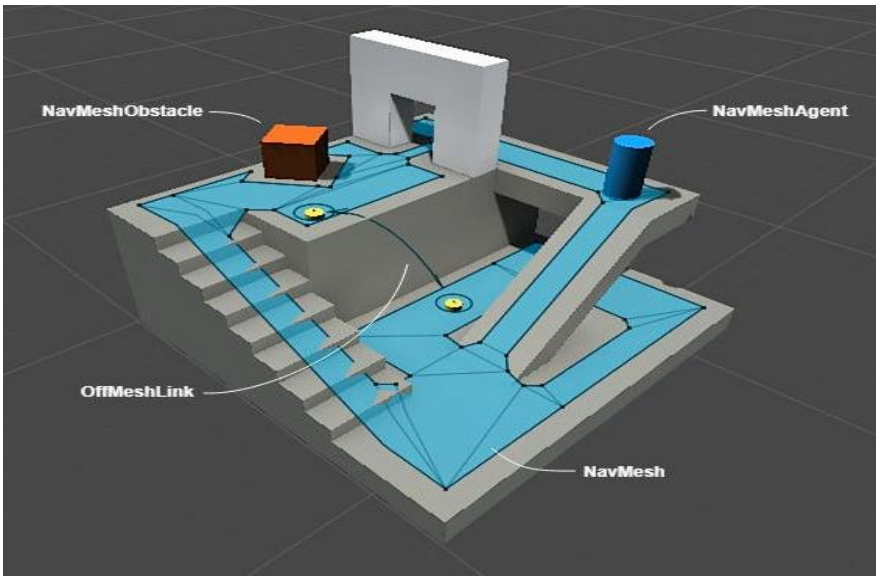

Рис. 1.2. Система NavMesh у редакторі Unity 3D

4. Дерева поведінки:

Дерева поведінки забезпечують ієрархічний та модульний підхід до проектування ШІ в Unity. Ця методологія структурує поведінку NPC у вигляді дерева завдань та умов. Кожен вузол у дереві представляє певну дію або рішення, а структура дерева дозволяє легко модифікувати та розширювати його. Дерева поведінки чудово справляються зі складними процесами прийняття рішень і широко використовуються для проектування адаптивної та динамічної поведінки NPC.

5. Машинне навчання (ML) та ML-агенти Unity:

Машинне навчання відкрило нову еру ШІ в Unity, особливо з появою інструментарію Unity ML-Agents. Навчання з підкріпленням (RL) - це підмножина ML, яка фокусується на навчанні агентів приймати послідовні рішення шляхом проб і помилок. Unity ML-Agents використовує глибоке навчання з підкріпленням для створення інтелектуальних агентів, здатних навчатися на основі взаємодії з навколишнім середовищем. Це відкриває шлях до складної та адаптивної поведінки NPC, створюючи віртуальні сутності, які еволюціонують з часом.

6. Обробка природної мови (NLP):

Окрім традиційних ігрових додатків, можливості штучного інтелекту Unity поширюються на обробку природної мови. NLP дозволяє розробникам

створювати віртуальних персонажів, здатних розуміти і реагувати на природну мову користувачів. Це відкриває можливості для інтерактивного оповідання історій, освітніх сценаріїв та більш захоплюючого користувацького досвіду.

7. Генетичні алгоритми:

Генетичні алгоритми використовуються в Unity для еволюційних процесів. У контексті ШІ генетичні алгоритми можна застосовувати для еволюції поведінки NPC протягом кількох поколінь. За допомогою процесу відбору, кросинговеру та мутацій можна вивести віртуальні сутності з успішною поведінкою, що призведе до появи більш розумних та адаптивних NPC.

8. Нейронні мережі:

Нейронні мережі, натхненні будовою людського мозку, набули популярності в Unity для просунутих додатків ШІ. Глибоке навчання, підмножина машинного навчання, передбачає використання нейронних мереж з декількома шарами (глибокі нейронні мережі) для моделювання складних взаємозв'язків і закономірностей. Розробники Unity можуть інтегрувати нейронні мережі для таких завдань, як розпізнавання зображень, обробка мови та створення ШІ-агентів з більш складними можливостями прийняття рішень.

Хоча ці методології та алгоритми значно просунули ШІ в Unity, проблеми залишаються. Етичні міркування, конфіденційність даних та інтерпретованість складних моделей - це сфери, які потребують постійної уваги. Крім того, майбутнє ШІ в Unity може передбачати подальшу інтеграцію навчання з підкріпленням, вдосконалення архітектури нейронних мереж і посилення співпраці між ШІ та розробниками-людьми для підвищення креативності та інновацій.

1.6. Комплексний огляд інструментів та технологій, що використовуються для розробки Unity 3D

Unity 3D став потужним рушієм у сфері розробки ігор, надаючи

розробникам можливість створювати захоплюючі та візуально приголомшливі ігри на різних платформах. Ефективність Unity 3D ґрунтується на складному наборі інструментів і технологій, які спрощують процес розробки. Цей есе має на меті надати всебічний огляд цих інструментів і технологій, пропонуючи уявлення про їхні функціональні можливості, значення та вплив на розробку Unity 3D.

В основі розробки Unity 3D лежить його комплексне середовище, яке забезпечує уніфіковану платформу для проектування, створення та розгортання додатків. Редактор Unity Editor є основним інтерфейсом, що дозволяє розробникам створювати та маніпулювати ігровими ресурсами, проектувати сцени та реалізовувати ігрову логіку. Це середовище розробки підтримує як 2D, так і 3D розробку ігор, сприяючи універсальності процесу створення.

Unity 3D підтримує декілька мов програмування, основною мовою для написання сценаріїв є C#. C# обрано через її читабельність, ефективність та бездоганну інтеграцію з середовищем розробки Unity. Використання C# забезпечує стандартизований та ефективний досвід кодування, дозволяючи розробникам створювати масштабований та підтримуваний ігровий код.

Хоча Unity 3D сам слугує середовищем розробки, зовнішні інтегровані середовища розробки покращують досвід кодування. Visual Studio, широко розповсюджене середовище розробки, легко інтегрується з Unity 3D, пропонуючи розширені можливості редагування, налагодження та профілювання коду. Синергія між Unity 3D та Visual Studio надає розробникам потужний набір інструментів для ефективного створення та оптимізації коду.

Unity 3D включає в себе ряд інструментів для управління ресурсами та їх створення, необхідних для проектування захоплюючих ігрових середовищ. Магазин ресурсів слугує централізованим ринком, що дозволяє розробникам отримати доступ до безлічі готових ресурсів, плагінів та інструментів. Крім того, система Unity's Terrain (Рис. 1.3.) полегшує створення реалістичних ландшафтів, а такі інструменти, як ProBuilder, дозволяють моделювати та створювати прототипи в редакторі.

Рис. 1.3. Інтерфейс Unity під час редагування ландшафту

Фізика та анімація відіграють ключову роль у створенні реалістичного та динамічного ігрового досвіду. Unity 3D пропонує вбудований фізичний рушій, який забезпечує реалістичну взаємодію між ігровими об'єктами. Інструмент Animator дозволяє розробникам створювати складні анімації за допомогою візуального інтерфейсу, спрощуючи процес анімації та підвищуючи загальну візуальну привабливість ігор.

Unity Collaborate та Unity Teams надають командам розробників важливі інструменти для співпраці та контролю версій. Ці функції сприяють безперешкодній співпраці, дозволяючи членам команди одночасно працювати над одним проектом. Контроль версій гарантує, що зміни відстежуються, що полегшує управління ітераціями та відкотами, сприяючи спільному та організованому процесу розробки.

Unity 3D позиціонує себе на передньому краї розробки VR і AR. Такі інструменти, як Unity XR та XR Interaction Toolkit, дають розробникам можливість створювати захоплюючі програми віртуальної та доповненої реальності. Включення цих технологій розширює сферу застосування Unity 3D за межі традиційних ігор, відкриваючи шляхи для інноваційних додатків у різних галузях.

Unity 3D надає набір інструментів для оптимізації та підвищення продуктивності, щоб забезпечити безперебійну роботу ігор на різних

платформах. Інструмент Profiler дозволяє розробникам аналізувати вузькі місця в продуктивності, що дає змогу проводити цілеспрямовану оптимізацію. Крім того, компілятор Burst Compiler та система завдань Unity підвищують продуктивність, використовуючи багатоядерні процесори та оптимізуючи виконання коду.

Кінцевою метою розробки ігор часто є комерційний успіх, а Unity 3D полегшує стратегії розгортання та монетизації. Unity Cloud надає платформу для розповсюдження ігор, а Unity Ads та Unity IAP (In-App Purchases) дозволяють розробникам безперешкодно впроваджувати функції реклами та монетизації.

Unity 3D надає пріоритет безпеці та відповідності вимогам, що є важливими в сучасному цифровому ландшафті. Рушій включає в себе такі функції, як Unity Analytics Privacy Dashboard, що забезпечують відповідність нормам захисту даних. Заходи безпеки Unity, включаючи обфускацію коду та шифрування, сприяють захисту інтелектуальної власності та даних користувачів.

Таким чином, Unity 3D є свідченням бездоганної інтеграції інструментів і технологій, які в сукупності утворюють комплексну екосистему для розробки ігор. Середовище розробки, мови програмування, IDE, інструменти для створення ресурсів та безліч додаткових функцій об'єднуються, щоб надати розробникам можливість створювати інноваційні та захоплюючі ігри. Прихильність Unity 3D до універсальності, співпраці та оптимізації продуктивності зміцнює його позиції як провідного рушія для розробки ігор, що формує ландшафт інтерактивного цифрового досвіду в сучасному бізнес-середовищі. З розвитком технологій Unity 3D залишається на передовій, адаптуючись і розвиваючись, щоб відповідати вимогам динамічного і конкурентного світу розробки ігор.

1.6.1. Аналіз їхньої ролі в інтеграції ШІ

Процедурна генерація контенту зі штучним інтелектом:

Процедурна генерація контенту (PCG) є критично важливим аспектом розробки ігор, що впливає на дизайн рівнів, створення ландшафту і генерацію ресурсів. Інтегруючи алгоритми ШІ в Unity 3D, розробники можуть автоматизувати та оптимізувати процес PCG. У цьому розділі розглядається, як AI-керований PCG може призвести до створення динамічного та нескінченно різноманітного ігрового середовища, скорочуючи час і ресурси, традиційно необхідні для створення контенту.

Також штучний інтелект можна використовувати для створення нескінченних варіацій ігрових рівнів разом з інструментами Unity. Замість того, щоб створювати кожен рівень вручну, розробники можуть впроваджувати системи ШІ, які генерують нові завдання та оточення "на льоту". Така адаптивність гарантує гравцям унікальний ігровий процес у кожній сесії, що сприяє реграбельності та подовжує життя гри.

Інтелектуальна поведінка NPC:

Одним із викликів у розробці ігор є створення неігрових персонажів (NPC) з реалістичною та динамічною поведінкою. ШІ можна використовувати для того, щоб наділити NPC адаптивними можливостями і здатністю до навчання, зробити їх чутливими до дій гравця і створити більш захоплюючий ігровий досвід. Ми розглянемо, як інтеграція ШІ в Unity 3D дозволяє розробникам створювати NPC, які навчаються, еволюціонують і дають гравцям унікальні завдання.

Покращене тестування ігор та забезпечення якості:

Забезпечення якості - важливий етап у розробці ігор, і ШІ може відігравати ключову роль в автоматизації та оптимізації процесів тестування. У цьому розділі розглядається, як Unity 3D з інтегрованими інструментами штучного інтелекту може забезпечити більш ефективне і всебічне тестування, зменшити ймовірність помилок і збоїв, а також забезпечити більш плавний цикл розробки.

Використовуючи алгоритми машинного навчання, Unity 3D може передбачати потенційні помилки або збої, аналізуючи патерни з попередніх етапів тестування. Ця можливість прогнозування допомагає розробникам вирішувати проблеми до того, як вони стануть критичними, мінімізуючи час і

ресурси, витрачені на виправлення після релізу. Як результат, ігри можна запускати з більшою стабільністю та надійністю.

Реалістична анімація та рух:

Реалістична анімація персонажів і об'єктів - складне і трудомістке завдання. Технології штучного інтелекту, такі як інструменти анімації на основі машинного навчання, можуть значно прискорити цей процес. Ми обговоримо, як інтеграція ШІ в Unity 3D може призвести до більш реалістичної та чутливої анімації, сприяючи більш захоплюючому та візуально привабливому ігровому досвіду.

Адаптивний досвід гравця:

Персоналізація все більше стає ключовим елементом в іграх. ШІ можна використовувати в Unity 3D для створення адаптивного ігрового досвіду, підлаштовуючи геймплей, рівні складності та елементи розповіді під індивідуальні вподобання та поведінку гравця. У цьому розділі досліджується, як адаптивність, керована штучним інтелектом, може підвищити залученість і задоволеність гравців.

Обробка природної мови для ігрових діалогів:

Включення обробки природної мови (NLP) в Unity 3D може докорінно змінити спосіб обробки ігрових діалогів. NLP на основі ШІ може забезпечити більш реалістичну та контекстну взаємодію між гравцями та персонажами, покращуючи розповідь історії та розвиток персонажів. Ми розглянемо, як ця інтеграція може сприяти створенню більш захопливого та емоційно насиченого ігрового досвіду.

1.7. Виявлення проблемних областей використання ШІ у розробці

Етичні міркування при розробці ШІ:

Інтеграція технології розпізнавання облич в іграх може викликати етичні проблеми, пов'язані з приватним життям. Розробники повинні дотримуватися

балансу між покращенням користувацького досвіду та дотриманням прав на приватність. Випадки, коли дані про обличчя збираються без явної згоди користувача або зберігаються ненадійно, можуть призвести до серйозних етичних наслідків.

Програми, які використовують ШІ для аналізу даних і прийняття рішень, можуть бути вразливими до витоку даних. Особливо це стосується обробки конфіденційної інформації, такої як медичні записи або фінансові дані. Розробники повинні впроваджувати надійні заходи безпеки для захисту від несанкціонованого доступу та потенційного зловживання даними, згенерованими штучним інтелектом.

Упередженість і справедливість алгоритмів ШІ:

В іграх і програмах, що використовують системи рекомендацій, можуть виникати упередження, засновані на історичних даних користувачів. Якщо навчальні дані не є різноманітними, алгоритм ШІ може увічнити стереотипи та обмежити доступ користувачів до вузького кола контенту. Цей приклад підкреслює важливість активного усунення упереджень в алгоритмах для забезпечення справедливих та інклюзивних рекомендацій.

Упередженість, притаманна обробці природної мови (NLP):

Обробка природної мови, ключовий компонент багатьох додатків зі штучним інтелектом, може демонструвати упередженість, присутню в навчальних даних. Наприклад, мовні моделі можуть ненавмисно засвоювати і відтворювати соціальні упередження, що призводить до дискримінаційних результатів. Розробники повинні активно працювати над виявленням і пом'якшенням цих упереджень, щоб забезпечити справедливість та інклюзивність у програмній та ігровій взаємодії.

Пояснюваність та прозорість:

В іграх з персонажами, керованими ШІ, непрозорі процеси прийняття рішень можуть розчаровувати гравців. Якщо дії ШІ недостатньо прозорі, гравцям може бути складно зрозуміти, чому відбуваються певні події в грі, що призводить до менш захоплюючого досвіду. Розробники повинні зосередитися на створенні

систем штучного інтелекту, які пропонують чіткі пояснення своїх рішень, щоб підвищити рівень розуміння та задоволення гравців.

У фінансових програмах, що використовують ШІ для алгоритмічної торгівлі, брак прозорості у прийнятті рішень створює значні проблеми. Трейдерам і регуляторним органам може бути складно зрозуміти обґрунтування торгових рішень, що викликає занепокоєння щодо маніпулювання ринком і системних ризиків. Прозорі алгоритми ШІ мають вирішальне значення для збереження довіри та дотримання нормативних вимог.

Технічні обмеження і складність:

Ігри, які потребують ресурсномістких процесів навчання ШІ, можуть зіткнутися з проблемами, пов'язаними з термінами розробки. Тривалий час навчання складних моделей ШІ може перешкоджати ітеративній розробці, ускладнюючи розробникам можливість експериментувати з функціоналом ШІ та вчасно його вдосконалювати.

Інтеграція ШІ в існуючі програмні архітектури, особливо в застарілі системи, може бути технічно складною. Старі системи можуть бути не пристосовані до обчислювальних вимог алгоритмів ШІ, що вимагає значних модифікацій. Це створює дилему для розробників, які прагнуть використовувати досягнення ШІ, зберігаючи при цьому сумісність з існуючими системами.

Проблеми співпраці між людиною і ШІ:

У програмах, де ШІ впроваджується для автоматизації певних завдань, опір працівників-людей може перешкоджати успішній співпраці. Працівники можуть боятися втратити роботу або не мати необхідних навичок для роботи зі штучним інтелектом. Розробники повинні вирішувати ці проблеми за допомогою ефективної комунікації, програм підвищення кваліфікації та формування культури співпраці.

У командах розробників ігор інтеграція ШІ може вимагати нових навичок. Розробники, які звикли до традиційних методологій, можуть зіткнутися з труднощами під час адаптації до робочих процесів, керованих штучним інтелектом. Подолання цього розриву в навичках вимагає стратегічних навчальних ініціатив і сприятливого середовища, яке заохочує обмін знаннями між членами

команди.

Віртуальні асистенти в програмах часто покладаються на безперервне навчання, щоб адаптуватися до уподобань користувачів. Однак проблеми виникають, коли застарілі моделі не здатні врахувати поведінку користувачів, що змінюється, або не враховують нові тенденції. Розробники повинні впроваджувати ефективні механізми оновлення та вдосконалення моделей ШІ, щоб забезпечити постійну актуальність віртуальних асистентів.

Проблеми адаптації в іграх зі штучним інтелектом:

Ігри, які використовують адаптивну поведінку ШІ, можуть зіткнутися з труднощами у реагуванні на швидку зміну стратегій гравців. Якщо ШІ не здатний швидко адаптуватися, ігровий процес може стати передбачуваним і менш захопливим. Розробники повинні впроваджувати механізми навчання та адаптації в режимі реального часу, щоб забезпечити динамічне та складне ігрове середовище.

Занепокоєння щодо безпеки:

Зловмисні атаки на кібербезпеку зі штучним інтелектом:

Програми кібербезпеки зі штучним інтелектом можуть бути вразливими до ворожих атак. Зловмисники можуть використовувати слабкі місця в алгоритмах ШІ, обманюючи системи та компрометуючи заходи безпеки. Розробники повинні постійно вдосконалювати моделі ШІ для захисту від нових загроз, щоб забезпечити надійність додатків для кібербезпеки.

Використання вразливостей ШІ в іграх:

В онлайн-іграх зловмисники можуть спробувати використати вразливості алгоритмів ШІ, щоб отримати несправедливу перевагу. Шахрайські програми, що використовують ШІ, можуть поставити під загрозу цілісність багатокористувацького досвіду. Розробники повинні впроваджувати надійні механізми боротьби з шахрайством і регулярно оновлювати моделі ШІ, щоб випереджати потенційні зловживання.

Відповідність регуляторним нормам і правові питання:

Програми зі штучним інтелектом повинні відповідати нормам захисту даних, таким як Загальний регламент про захист даних (GDPR). Програми, що

використовують ШІ для обробки даних користувачів, повинні вирішувати проблеми забезпечення прозорості, отримання інформованої згоди та впровадження механізмів, що дозволяють користувачам контролювати свої дані. Недотримання цих вимог може призвести до юридичних наслідків і репутаційних втрат.

Ігри, які використовують ШІ для створення контенту, можуть зіткнутися з проблемами інтелектуальної власності. Визначення прав власності та авторських прав, пов'язаних із контентом, створеним за допомогою ШІ, викликає юридичні складнощі. Розробники повинні орієнтуватися в цих юридичних тонкощах, щоб забезпечити належну атрибуцію та захист прав інтелектуальної власності.

Сприйняття і опір користувачів:

У програмах, де ШІ впливає на процеси прийняття рішень, скептицизм користувачів може перешкоджати прийняттю. Користувачі можуть не довіряти рекомендаціям або рішенням, згенерованим ШІ, через брак довіри або розуміння. Розробники повинні використовувати зручні інтерфейси, чітку комунікацію та освітні ініціативи, щоб подолати опір і сприяти прийняттю.

Ігри, в яких реалізовані функції зі штучним інтелектом, можуть ненавмисно створювати негативний користувацький досвід, якщо штучний інтелект поводиться непередбачувано або деструктивно. Випадки, коли ШІ приймає рішення, які сприймаються як несправедливі або неприємні, можуть призвести до незадоволення гравців. Розробники повинні дотримуватися балансу між складним ігровим процесом і забезпеченням позитивного загального досвіду, щоб забезпечити прийняття користувачами.

Проблеми, пов'язані з інтеграцією ШІ в розробку програм та ігор, різноманітні та багатогранні. Врахування етичних міркувань, зменшення упередженості, забезпечення прозорості, подолання технічних обмежень, сприяння ефективній співпраці між людиною і ШІ, а також подолання правових складнощів мають вирішальне значення для розкриття справжнього потенціалу ШІ. Розробники повинні підходити до цих викликів з проактивним мисленням, використовуючи інноваційні рішення для створення етичних, інклюзивних і технологічно просунутих додатків, які визначають майбутнє розробки програм та

ігор.

1.7.1 Пропоновані рішення та стратегії для ефективної інтеграції ШІ

Впровадження штучного інтелекту в дизайн:

Розробка ігор і програмного забезпечення, які безперешкодно інтегрують ШІ, вимагає вдумливого підходу з самого початку. Дизайнери повинні враховувати користувацький досвід, динаміку гри та можливості ШІ. Одна зі стратегій полягає в тому, щоб прийняти підхід до дизайну, орієнтований на користувача, коли ШІ покращує, а не домінує над користувацьким досвідом. Наприклад, в ігровому дизайні персонажі, керовані штучним інтелектом, можуть адаптуватися до поведінки гравця, створюючи більш персоналізований і цікавий досвід.

Крім того, використання генеративного дизайну - методу, коли алгоритми ШІ допомагають у створенні та розвитку дизайну - може сприяти підвищенню креативності та ефективності. Цей підхід дозволяє розробникам досліджувати безліч можливостей дизайну, вдосконалюючи та оптимізуючи його на основі інсайтів, згенерованих штучним інтелектом (Рис. 1.3.).

Рис. 1.3. Згенероване зображення 2д персонажа

ШІ в процесах розробки:

На етапі розробки штучний інтелект може значно спростити процеси, скоротивши час і ресурси. Впровадження автоматизованих інструментів генерації коду на основі ШІ може прискорити цикли розробки. Ці інструменти, такі як системи завершення коду та підказок, допомагають розробникам писати чистіший та

ефективніший код. Крім того, інструменти налагодження на основі ШІ можуть виявляти і виправляти проблеми, підвищуючи загальну стабільність програмного забезпечення.

Платформи для спільної розробки, оснащені інструментами управління проектами на основі ШІ, можуть покращити командну роботу, передбачаючи потенційні вузькі місця, оптимізуючи розподіл завдань і надаючи зворотний зв'язок у реальному часі. Такий підхід до спільної роботи на основі штучного інтелекту сприяє створенню більш гнучкого та ефективного середовища розробки.

ШІ в тестуванні та забезпеченні якості:

Тестування - важливий аспект розробки програмного забезпечення та ігор, який гарантує, що кінцевий продукт відповідає стандартам якості. Інтеграція штучного інтелекту в процеси тестування може розширити покриття тестів і виявити потенційні проблеми більш комплексно. Інструменти тестування на основі ШІ можуть імітувати взаємодію з користувачем, виявляти вузькі місця в продуктивності та автоматизувати регресійне тестування, дозволяючи розробникам зосередитися на більш складних і творчих аспектах проекту.

Алгоритми машинного навчання також можуть аналізувати відгуки та поведінку користувачів для прогнозування потенційних проблем, що дає змогу проактивно усувати помилки. Використовуючи ШІ в тестуванні, розробники можуть створювати більш надійне програмне забезпечення зі скороченими термінами тестування.

Розгортання додатків зі штучним інтелектом:

Розгортання ігор і програмного забезпечення зі штучним інтелектом вимагає ретельного аналізу масштабованості, продуктивності та адаптивності для користувача. Розробники повинні оптимізувати свої програми для розгортання в хмарі, використовуючи ШІ для динамічного масштабування ресурсів на основі попиту користувачів. Крім того, включення рекомендаційних систем на основі ШІ може підвищити залученість користувачів, надаючи їм персоналізований контент і функції.

Забезпечення етичного розгортання ШІ має вирішальне значення, особливо в додатках, які взаємодіють з користувачами. Розробники повинні забезпечити прозорість і справедливість алгоритмів ШІ, уникаючи упередженості та забезпечуючи позитивний користувацький досвід. Регулярні оновлення та технічне обслуговування на основі аналітики ШІ допоможуть зберегти актуальність і безпеку програми з плином часу.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ЗА ДОПОМОГОЮ ШТУЧНОГО ІНТЕЛЕКТУ

2.1. Функціональне призначення програми

Завданням кваліфікаційної роботи є спроектувати та розробити відео гру в жанрі Action/RPG на платформі Unity, метою якої є створення оптимальних умов праці, які створюють у людини відчуття задоволення виконаної роботи.

- переміщення головного героя відбувається в двомірному просторі, можна рухатися вліво та вправо.
- на полі бою головному герою потрібно воювати з різними юнітами та добувати предмети які гравець зможе використати для відновлення своїх характеристик (Кількість життів, Витривалість, Голод, Спрагу).
- при закінчені гри відображається панель перемоги або програшу та статистику проведеного гравцем часу в бою. В будь-який момент гра може бути призупинена для редагування користувачем управління переміщення та гучності звуків.
- головне меню містить кнопки виходу з гри та ігрових налаштувань. Додатково, для ознайомлення з правилами гри та особливостями додатку, в головному меню міститься кнопка інструкції, яка повинна включити підказки для гравця.

2.2. Опис застосованих математичних методів

Під час проектування та розробки додатку ми використовували математичний метод Дейкстри для визначення найкоротшого шляху. Цей підхід був доповнений методами оптимізації з використанням штучного інтелекту. Системи пошуку шляхів на основі штучного інтелекту часто потребують оптимізації для забезпечення ефективної роботи в реальному часі. Ця оптимізація може включати методи просторового розбиття, ієрархічні

представлення та різні стратегії, спрямовані на зменшення обчислювальної складності при збереженні точності.

Алгоритм Дейкстри будує найкоротші шляхи з початкової вершини графа до інших вершин графа, якщо такі існують. Алгоритм працює, послідовно позначаючи розглянуті вершини графа. Остання позначена вершина знаходиться ближче до початкової вершини, ніж всі непозначені вершини, але далі, ніж всі попередні позначені вершини.

Спочатку алгоритм позначає початкову вершину. Наступною позначається та вершина, яка є найближчою до початкової і суміжною з нею.

У випадках, коли на певному кроці вже позначено декілька вершин, алгоритм використовує відомі найкоротші шляхи від початкової вершини до цих позначених вершин. Для кожної непозначеної вершини виконуються наступні кроки:

Оцінюються всі дуги, що ведуть з позначених вершин до непозначеної вершини. Кожна така дуга є кінцевою ланкою на шляху від початкової вершини до непозначеної.

Виберіть найкоротший шлях серед цих дуг.

Виберіть найкоротший шлях серед усіх непозначених вершин і позначте відповідну вершину.

Алгоритм завершується, коли всі досяжні вершини позначені. Результатом роботи алгоритму Дейкстри є побудова дерева, що складається з найкоротших шляхів.

2.3. Опис використаних технологій та мов програмування

2.3.1. Опис використаних засобів програмування

Для запуску гри на пк необхідна версія ОС Windows не нижче 7 SP1, але для кращої стабільності необхідно використовувати більш сучасні версії.

Unity3d є сучасним крос-платформним движком для створення ігор і

двигуна можна розробляти не тільки додатки для комп'ютерів, але і для мобільних пристроїв, ігрових приставок і інших девайсів. Інтерфейс платформи зображено на рис. 2.1.

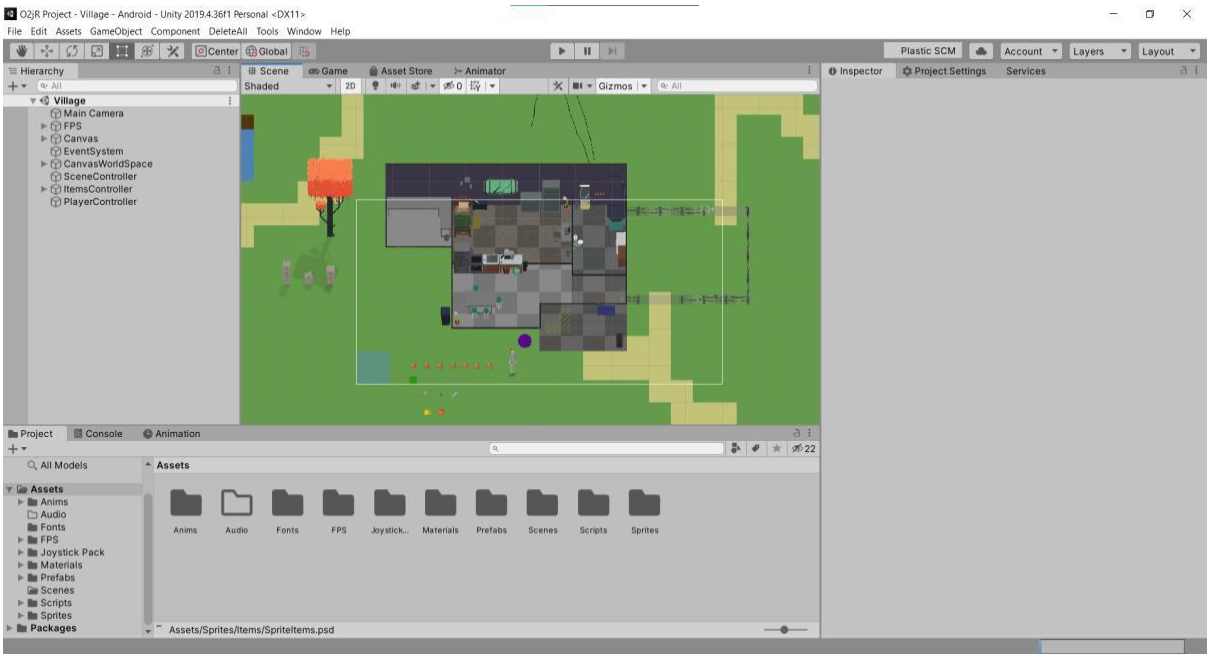

Рис. 2.1. Інтерфейс платформи Unity

Характеристики платформи:

- по-перше, в середовище розробки Unity інтегрований ігровий движок, за допомогою якого можна протестувати свою гру не виходячи з редактора;

- по-друге, Unity підтримує імпорт величезної кількості різних форматів, що дозволяє розробнику гри конструювати самі моделі в більш зручному додатку, а Unity використовувати за прямим призначенням - розробки продукту;

- по-третє, написання сценаріїв (скриптів) здійснюється на найбільш популярних мовах програмування - C# і JavaScript.

Таким чином, Unity3d є актуальною платформою, за допомогою якої можна створювати свої власні додатки і експортувати їх на різні пристрої, будь то мобільний телефон або приставка Nintendo Wii.

Щоб розробити власну гру в Unity, вам знадобиться знання однієї з мов програмування - на вибір: C#, JavaScript або Boo. Компоненти в Unity мають різні властивості або змінні, які можна налаштувати або через вікно інспектора в редакторі Unity, або за допомогою скрипту.

Хоча Unity пропонує ряд готових компонентів, ви також маєте можливість створювати власні для реалізації певних алгоритмів. Для цього потрібно створити власний скрипт і прикріпити його до потрібного об'єкта. Кожен скрипт взаємодіє з внутрішніми механізмами Unity, реалізуючи клас, похідний від вбудованого класу MonoBehavior. Компонент сценарію полегшує ініціювання ігрових подій, таких як тестування зіткнень між об'єктами, використання фізичних властивостей та реагування на дії користувача.

C# - це об'єктно-орієнтована мова, яка підтримує компонентно-орієнтоване програмування. Сучасна тенденція в розробці додатків схиляється до створення програмних компонентів як самодостатніх і самоописуваних пакетів, які втілюють окремі функціональні можливості. Ці компоненти функціонують як програмна модель з властивостями, методами та подіями, що супроводжуються атрибутами, які містять декларативну інформацію та документацію. C# включає мовні конструкції, які безпосередньо підтримують цю робочу концепцію, що робить її добре придатною як для створення, так і для використання програмних компонентів.

Декілька особливостей C# сприяють надійності та стабільності додатків. Збір сміття автоматично звільняє пам'ять, зайняту недоступними, невикористаними об'єктами. Обробка винятків забезпечує структурований та розширюваний підхід до виявлення та відновлення після помилок. Безпечна структура мови запобігає читанню з неініціалізованих змінних та індексації масивів за їх межами.

У C# всі типи даних, включаючи базові, такі як int та double, мають спільний кореневий тип, який називається "об'єкт". Це означає, що вони успадковують стандартизований набір операцій, що дозволяє зберігати, передавати та обробляти значення будь-якого типу в однаковий спосіб. C# також підтримує користувацькі типи посилань і значень, що дозволяє динамічно виділяти пам'ять для об'єктів і зберігати спрощені структури в стеку.

Мова забезпечує дотримання правила доступу до неініціалізованих змінних, запобігаючи неконтрольованому приведенню типів та виходу за межі масиву даних.

Запуск додатків на C# вимагає встановлення та налаштування платформи .NET Framework, яка зазвичай входить до інсталяційного пакету Windows. Крім того, її можна завантажити та встановити окремо. Також доступні версії для Linux та macOS.

В рамках платформи середовище Common Language Runtime (CLR) керує виконанням виконуваного коду. Воно складається з уніфікованого набору бібліотек і класів, розроблених Microsoft відповідно до глобального стандарту Common Language Infrastructure (CLI). Основним механізмом CLR є бібліотека mscoree.dll, яка також називається Common Object Runtime Execution Engine. При зверненні до збірки для використання, mscoree.dll автоматично завантажується і, в свою чергу, завантажує необхідну збірку в пам'ять.

Щоб забезпечити сумісність у майбутніх розробках, під час розробки C# особливу увагу було приділено контролю версій. Ця увага до контролю версій відрізняє її від багатьох інших мов програмування, які часто ігнорують цей важливий аспект.

Як наслідок, мови, яким бракує ефективного контролю версій, часто зазнають частих збоїв у роботі програм, коли з'являються нові версії залежних бібліотек. У сфері розробки C# проблеми контролю версій суттєво вплинули на різні аспекти, такі як введення окремих віртуальних та перевизначальних модифікаторів, рекомендації щодо усунення перевантаження методів та схвалення явного оголошення членів інтерфейсу.

Останні ітерації C# охоплюють різноманітні парадигми програмування. Мова тепер включає функції, що підтримують функціональні методи програмування, зокрема лямбда-вирази. Крім того, нові функціональні можливості полегшують розділення даних та алгоритмів, прикладом чого може слугувати введення шаблонної відповідності.

C# базується на таких фундаментальних поняттях, як типи та змінні:

Класи, наріжний тип у C#, породжують об'єкти, які є екземплярами класів. Створення класів передбачає визначення їх членів.

Масиви, як структури даних у C#, містять декілька змінних, доступних через обчислювані індекси.

Інтерфейси встановлюють договірні рамки, що реалізуються класами та структурами. Вони можуть містити методи, властивості, події та індексатори, визначаючи необхідні члени без надання їхньої реалізації.

Делегати в C# - це посилання на методи з певними параметрами та типами значень, що повертаються. Делегати уможливлюють використання методів як сутностей, дозволяючи зберігати їх у змінних та передавати як параметри. На відміну від вказівників на функції в деяких мовах, делегати в C# є об'єктно-орієнтованими і строго типізованими. Атрибути дозволяють програмам передавати додаткові описові дані щодо типів, членів та інших сутностей.

Інтегроване середовище розробки Visual Studio [11] - це стартовий майданчик для написання, налагодження і складання коду, а також подальшої публікації додатків. Інтегроване середовище розробки (IDE) являє собою багатофункціональну програму, яку можна використовувати для різних аспектів розробки програмного забезпечення. Крім стандартного редактора і відладчика, які існують в більшості середовищ IDE, Visual Studio включає в себе компілятори, засоби авто-завершення коду, графічні конструктори і багато інших функцій для спрощення процесу розробки. Інтерфейс середи (рис.2.2).

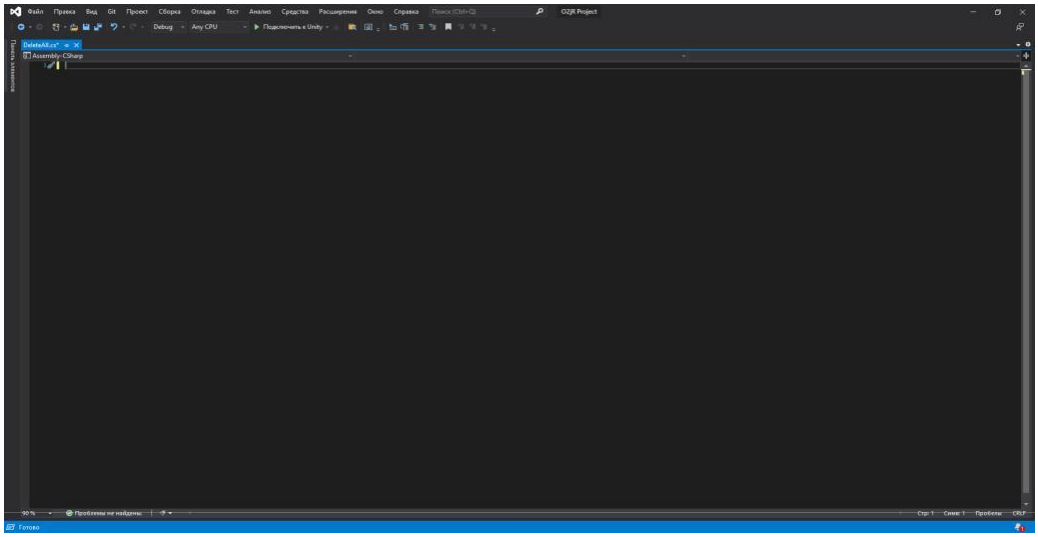

Рис. 2.2. Інтерфейс середовище Visual Studio

На рисунку показано середовище Visual Studio з відкритим проектом і декількома вікнами основних інструментів.

Оглядач рішень (вгорі праворуч) дозволяє переглядати файли коду, пересуватися по ньому і управляти ними. Оглядач рішень дозволяє впорядкувати код шляхом об'єднання файлів в рішення і проекти.

У вікні редактора (центр), відображається вміст файлу. Тут можна редагувати код або розробляти призначений для користувача інтерфейс, наприклад вікно з кнопками або текстові поля.

Провідник команд, розташований у правому нижньому куті, дозволяє відстежувати робочі елементи та полегшує спільне використання коду за допомогою технологій контролю версій, таких як Git та Team Foundation Version Control System (TFVC). Можливості Visual Studio охоплюють весь процес розробки програмного забезпечення, пропонуючи сучасні інструменти для кодування, проектування графічних інтерфейсів, створення, налагодження та тестування програм. Розширюваність Visual Studio дозволяє користувачам розширювати її функціональність шляхом інтеграції необхідних розширень.

Редактор коду у Visual Studio використовує візуальні індикатори, такі як хвилясті лінії, щоб виділити помилки або потенційні проблеми під час введення тексту. Ці візуальні підказки дозволяють негайно вирішити проблему, запобігаючи необхідності чекати, поки помилка випливе на поверхню під час збірки або виконання програми. Якщо навести вказівник миші на хвилясту лінію, ви побачите додаткову інформацію про помилку, а піктограма лампочки запропонує швидкі дії для її усунення. Доступне форматування коду в один клік і застосування виправлень на основі параметрів стилю коду, вказаних у файлі EditorConfig. Очищення коду допомагає вирішити численні проблеми з кодом ще до формальних перевірок.

Операції рефакторингу у Visual Studio включають інтелектуальне перейменування змінних, витягування коду в нові методи, зміну порядку параметрів методу тощо.

Редактор коду Visual Studio має функції підсвічування синтаксису, вставки фрагментів коду, відображення структури коду та пов'язаних з ним функцій. Технологія IntelliSense прискорює роботу, забезпечуючи автозавершення коду під час введення.

Для налагодження вбудований відладчик Visual Studio виявляє та виправляє помилки у вихідному коді навіть на низькому апаратному рівні. Засоби діагностики оцінюють якість коду з точки зору продуктивності та

використання пам'яті. Visual Studio підтримує автоматизоване тестування додатків, включаючи перевірку інтерфейсу, модульне і навантажувальне тестування.

Система налагодження дозволяє перевіряти код з кроком в одну інструкцію, з можливістю встановлювати точки зупинки і спостерігати за значеннями змінних. Додатки на основі ШІ можна легко інтегрувати в стандартне середовище, пропонуючи такі можливості, як автоматична генерація коду для фрагментів або функцій на основі описів природної мови, інтелектуальна генерація тестів і розстановка пріоритетів, а також рефакторинг коду на основі ШІ для підвищення продуктивності, читабельності та дотримання кращих практик.

Adobe Photoshop — багатофункціональний графічний редактор, що розробляється та розповсюджується компанією Adobe Systems. Здебільшого працює з растровими зображеннями, однак має деякі векторні інструменти. Продукт є лідером ринку в області комерційних засобів редагування растрових зображень та найвідомішою програмою розробника.

В даний час Photoshop (рис. 2.3) доступний на платформах macOS, Windows та iPadOS. Для Windows Phone та Android доступна спрощена версія програми під назвою Adobe Photoshop Touch.

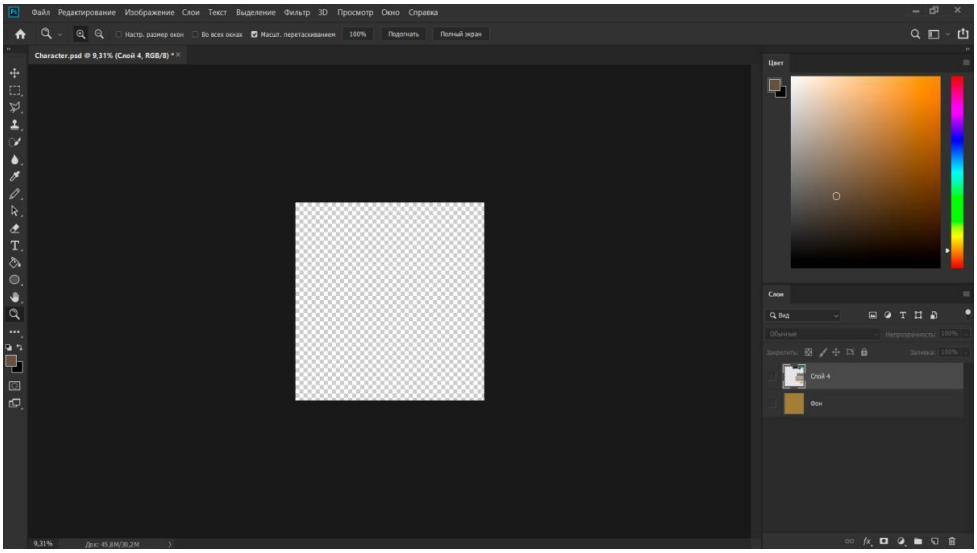

Рис. 2.3. Вікно програми Photoshop

Також існує версія Photoshop Express для Windows Phone 8 та 8.1. У 2014 році у США проходило бета-тестування потокової версії продукту для Chrome OS. Ранні версії редактора були портовані під SGI IRIX, але офіційна підтримка припинена з третьої версії продукту. Для версій 8.0 та CS6 можливий запуск під Linux за допомогою альтернативи Windows API – Wine.

У 2023 році з'явилася можливість генерувати зображення у новому редакторі AI Photoshop. Штучний інтелект уможливив автоматизоване редагування та вдосконалення, що дозволяє користувачам досягати професійних результатів з мінімальними зусиллями. Такі функції, як "Автоматичний тон" і "Автоматичний контраст" на базі Adobe Sensei, використовують алгоритми машинного навчання для аналізу вмісту зображення та інтелектуального застосування коригувань. Це не лише економить час, але й дає змогу користувачам, від початківців до професіоналів, створювати візуально приголомшливі результати.

2.4. Опис структури системи та алгоритмів її функціонування

2.4.1. Загальний опис структури програми

Гра розрахована на одного гравця. Гравець грає за добру сторону, ворог за ворожу. Є карта на якій гравець та ворог [12] вільно переміщаються. Основна мета гри полягає в тому, щоб перемогти всіх запропонованих ворогів.

Взаємодію користувача з системою графічно відображено на діаграмі використання (рис.2.4 та 2.5)

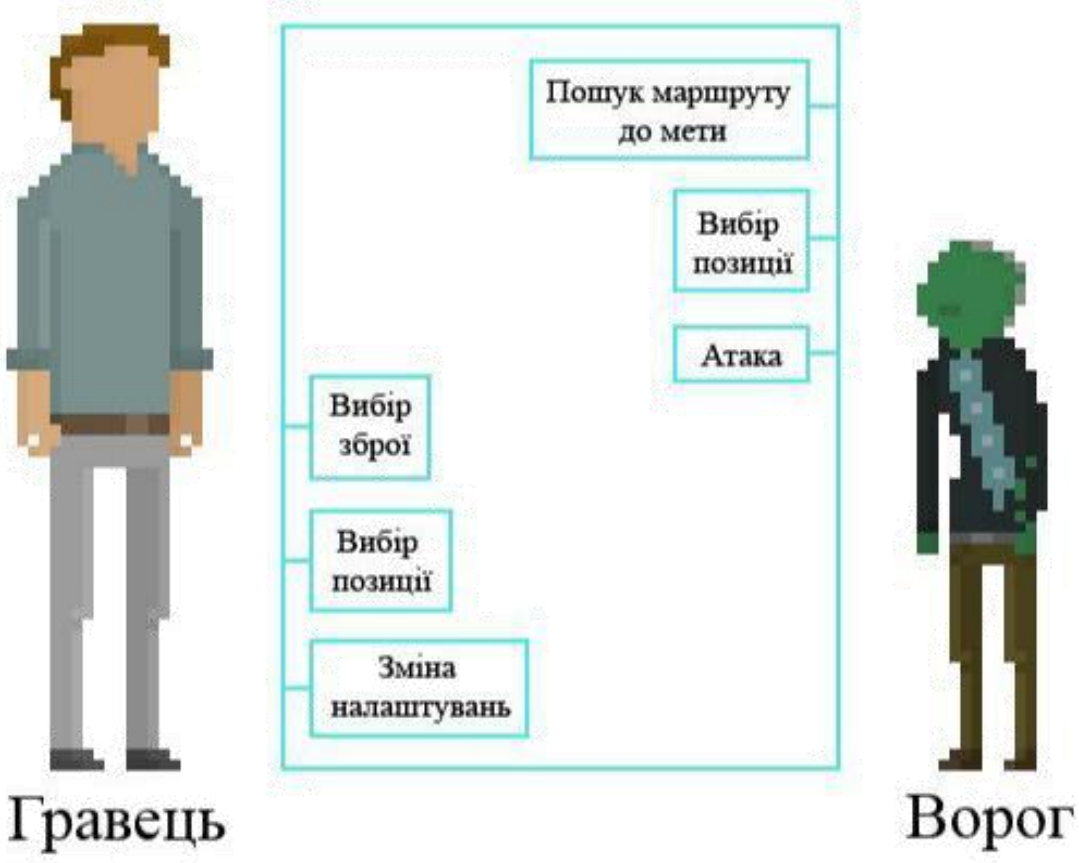

Рис. 2.4. Діаграма використання

Рис. 2.5. Діаграма діяльності

Склад файлової системи розробленого проекту зображено на рис. 2.6:

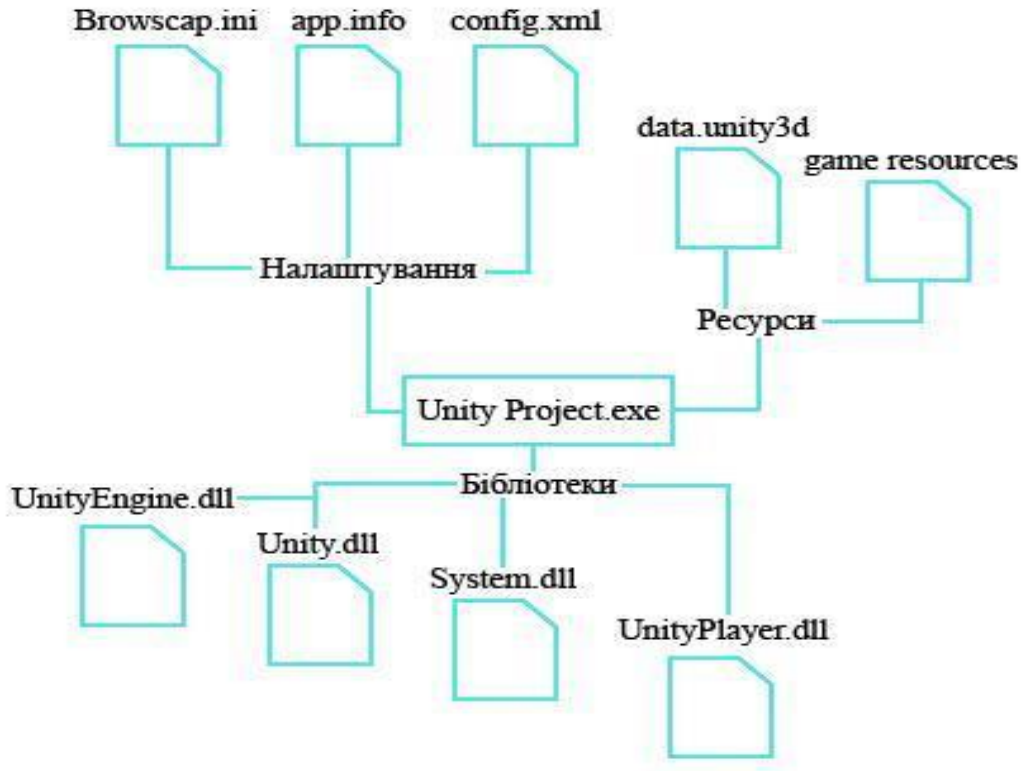

Рис. 2.6. Файлова система проекту

2.4.2. Опис основних компонентів

В платформі Unity кожен об’єкт представляє собою GameObject, який зберігає атрибути користувацьких та влаштованих компонентів. Так, наприклад, стандартним компонентом GameObject, який не можливо видалити є «Transform» (рис. 2.7). Він зберігає параметри Position, Rotation, та Scale відповідно позиція, поворот і розмір за трьома осями (X,Y,Z) відносно початку координат.

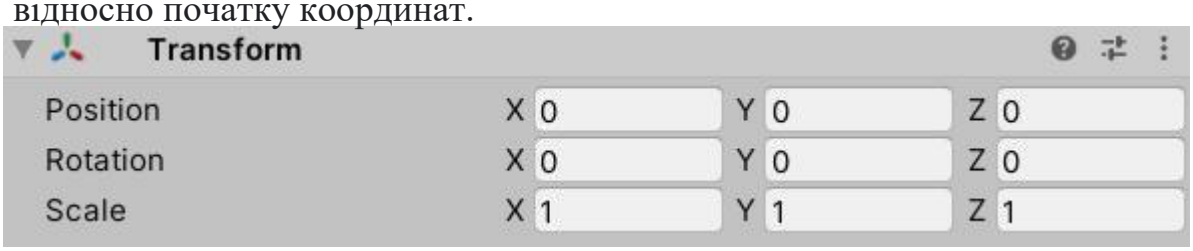

Рис. 2.7. Компонент «Transform»

Щоб відобразити двомірні зображення в тривимірному ігровому просторі використовується компонент «Sprite renderer» (рис. 2.8) Після вказівки у властивості Sprite посилання на імпортоване в проект гри над зображення можна проводити різні маніпуляції, такі як обертання, переміщення, зміна кольору тощо.

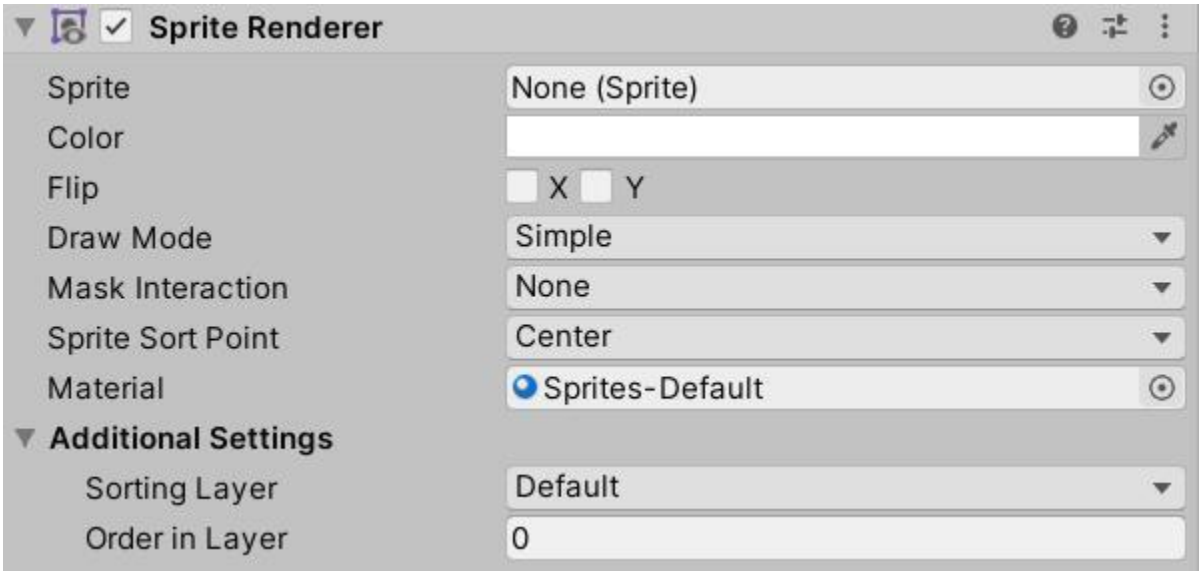

Рис. 2.8. Компонент «Sprite renderer»

Спроектовану анімацію гри розглянемо на прикладі фігури, для якої в проекті застосовано компонент «Animator» (рис. 2.9). Він зв’язує об’єкт з файлами анімацій (.anim), що створюються у вікні «Animation».

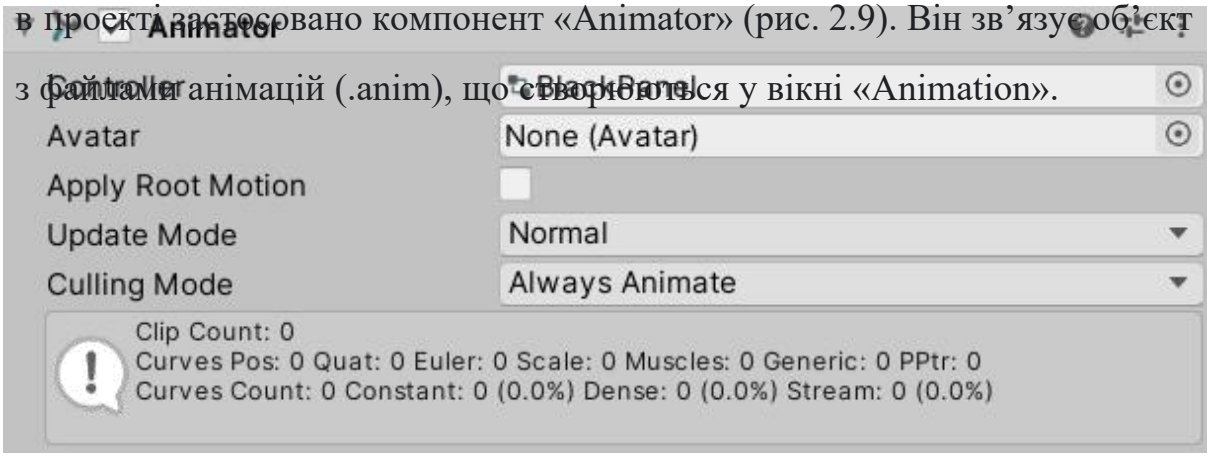

Рис. 2.9. Компонент «Animator»

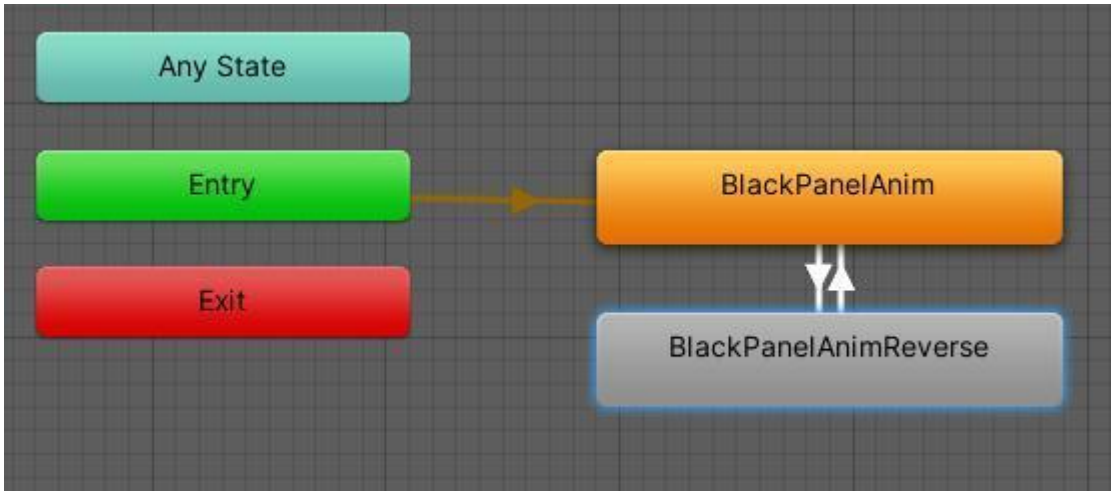

Рис. 2.10. Вікно «Animator»

Група компонентів «Collider» додає колізію, тобто фізичну взаємодію з іншими об’єктами. Розглянемо компонент «Capsule Collider» (рис. 2.11) – він не дає головному героя або іншому об'єкту пройти крізь інший твердий об'єкт.

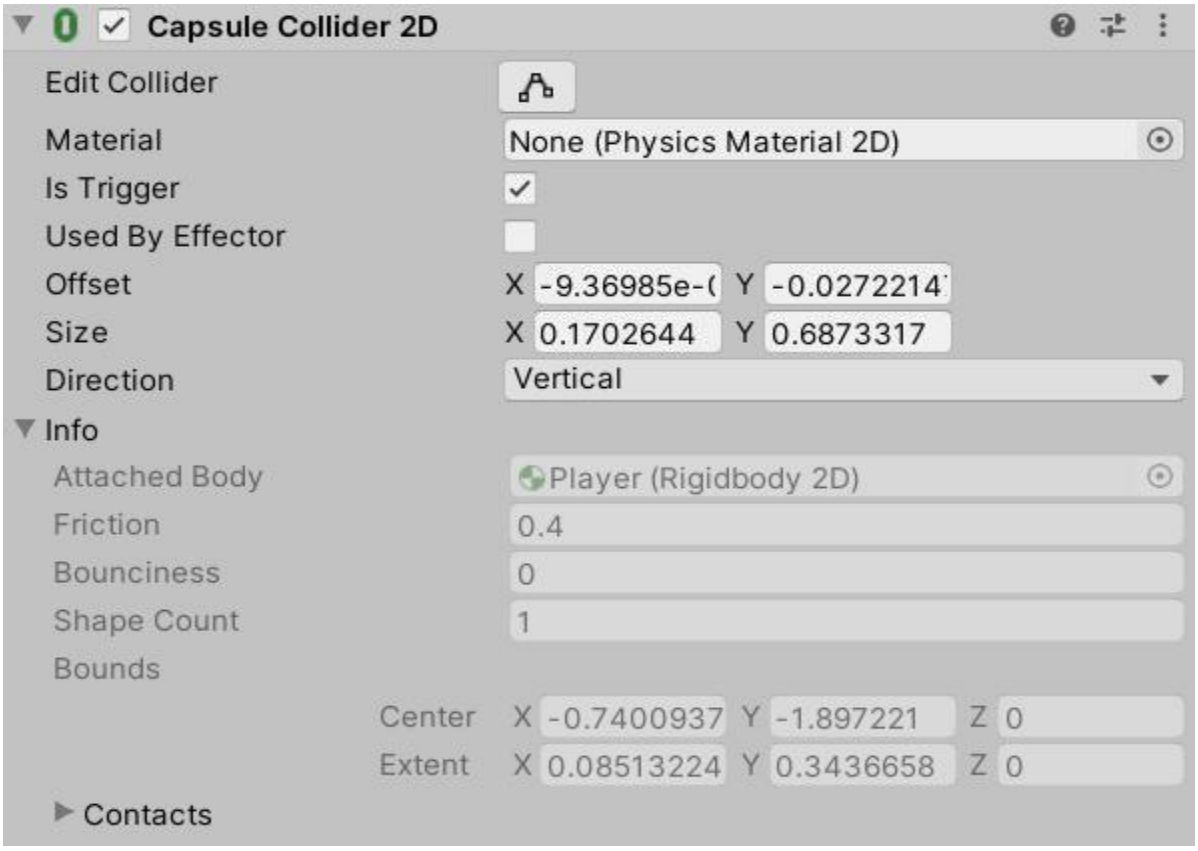

Рис. 2.11. Компонент «Capsule Collider»

Створений користувачем компонент може керувати властивостями інших компонентів, додавати нові, змінювати ігрову сцену та використовувати всі можливості мови C#. Лістинг всіх користувацьких скриптів знаходиться у додатку А.

2.4.3. Реалізація ігрових елементів

Дизайн гри створений у стилі pixel-art [10], що набирає чималої популярності у сучасній ігровій індустрії. Серед використаних засобів обрамлення містяться:

- привабливі декорації жанру sci-fi;

- цікаві та стилізовані персонажі.

Зовнішній вигляд інтерфейсу тісно пов’язаний із самою грою та сутністю ігрового процесу. В основному він асоціюється з графікою відеоігор 80-х та 90-х років. Тоді художникам доводилося враховувати обмеження пам'яті та низького дозволу. Персонажі, що оточують гравця, створені та стилізовані відповідно до жанру. Приклади персонажів наведено на рис. 2.12.

Рис. 2.12. Персонажі гри

Саме ігрове поле, що буде знаходитись перед гравцем більшу частину ігрового процесу, також стилізоване під заданий стиль. Ліворуч та праворуч від поля наявні панелі з основним функціоналом. Усе графічне оформлення було створене спеціально для гри.

Для локації було створено наступні об’єкти і компоненти (табл. 2.1). Суть цієї локації полягає в тому, щоб гравець на ній воював з ворогами. Вигляд локації відображено на рис. 2.13.

Таблиця 2.1

Компоненти сцени

Об’єкт	Компонент	Призначення
Земля	Sprite Renderer	Візуальна картинка
	Box Collider 2D	Колізія
	Script (Ground)	Контроль усіма об’єктами у домі
Фон	Sprite Renderer	Візуальна картинка
	Script (Parallax)	Контроль взаємодії з гравцем
Coor X		Позиція по координаті X
Coor Y	Transform	Позиція по координаті Y
Coor Z		Позиція по координаті Z

Рис. 2.13. Концепт сцени

Для елементу камера був створений скрипт «SlowCamera». Суть камери полягає у тому, щоб постійно слідкувати за головним героєм, відображати ігрову сцену і зміни які на ній відбуваються.

Скрипт «SlowCamera» дає можливість плавного стеження за гравцем, обмеження за вказаними координатами щоб камера не вийшла за умовні межі та зміна розміру камери для повного захоплення різних об'єктів у полі зору гравця. Камера та її кордони на сцені відображені на рис. 2.14.

Рис. 2.14. Камера на сцені

Для головного героя було створено наступні об’єкти і компоненти (табл. 2.2). Вигляд і дія головного героя відображено на рис. 2.15.

Таблиця 2.2

Компоненти головного героя

Об’єкт	Компонент	Призначення
Player	Sprite Renderer	Візуальна картинка
	Capsule Collider 2D	Колізія
	Trigger	Колізія з об'єктами
	Script (Player)	Контроль головним героєм
	Shadow	Тінь
	PlayerHighlight	Підсвічування
	Rigidbody 2D	Контроль фізики
	Animator	Контроллер анімаціями

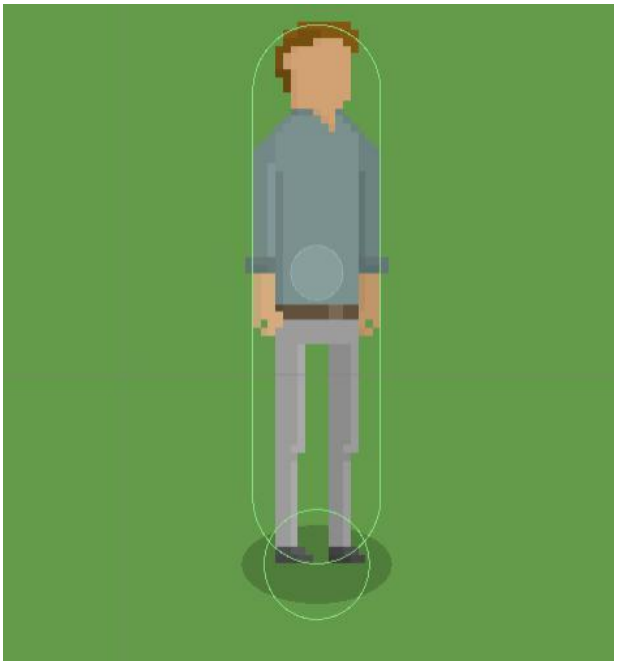

Рис. 2.15. Концепт головного героя

Скрипт «Player» призначений для переміщення головного героя гравцем, взаємодію з предметами та контролю всіх характеристик гравця.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Згідно з постановкою завдання у якості вхідних даних програми виступають:

- переміщення персонажу;
- взаємодія з різними об'єктами;
- налаштування гри.
- Вихідними даними

є:

- відображення положення головного героя;
- програвання звуків;
- відображення анімацій;
- вивід результату гри.

2.6. Опис роботи розробленого програмного продукту

2.6.1. Використані технічні засоби

Для розробки даної гри використовувалися наступні технічні засоби:

- Процесор: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz 2.30 GHz;
- Відеокарта: Nvidia GeForce 1080 Ti;
- Оперативна пам’ять: 8 гб;
- ОС: Windows 10.

2.6.2. Використані програмні засоби

Реалізація виконана на платформі Unity, мова програмування для написання скриптів та ігрової логіки – C#. Об’єктно-орієнтована концепція мови програмування C# найліпше підходить для описання ігрової логіки об’єктів за допомогою системи класів у редакторі Visual Studio.

Вибране програмне забезпечення Ai Photoshop [6] підійшло для створення та генерування двомірних об’єктів.

Штучний інтелект у вигляді плагіну для оптимізації пошуку найкоротшого шляху для NPC на базі NavMesh.

Штучний інтелект у вигляді додатку для Visual Studio.

2.6.3. Виклик та завантаження програми

На початку роботи з грою, необхідно завантажити її у вигляді rar/zip архіву. Після завантаження файлу, необхідно розпакувати весь його зміст в комфортне для користувача місце в документах пристрою. Після розпакування архіву, відкрити .exe файл (рис. 2.16). Після цього відбудеться запуск гри. Для андроїд [9] потрібно лише встановити на смартфон файл .apk та встановити гру (рис. 2.17).

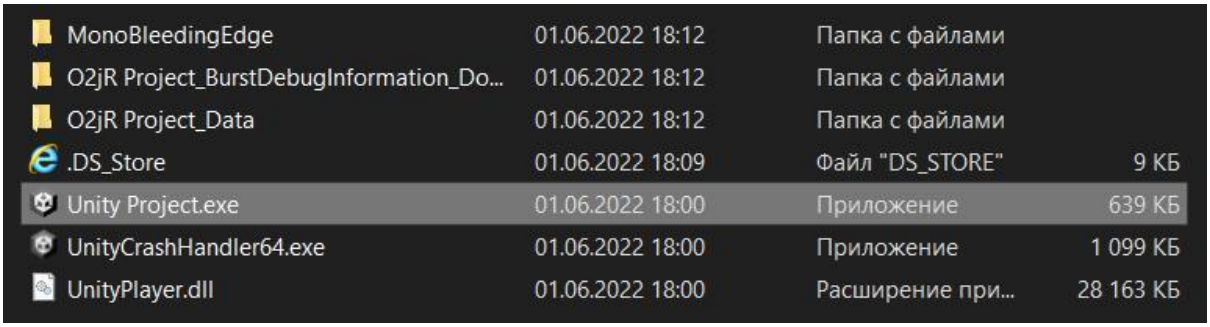

Рис. 2.16. Встановлення гри на пк

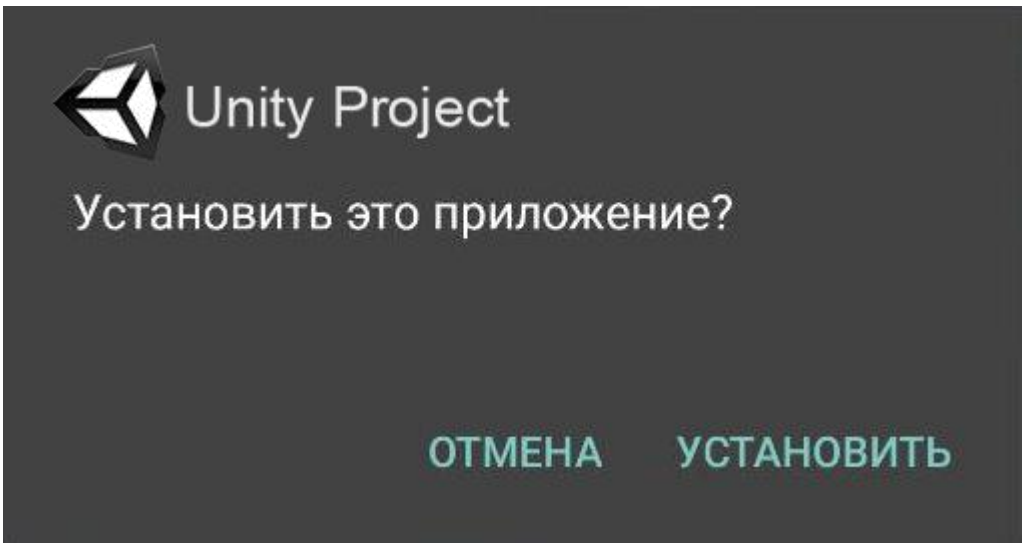

Рис. 2.17. Встановлення гри на андроїд

2.7. Опис інтерфейсу користувача

Після запуску гри, перед користувачем з’являється Головне меню (рис. 2.18), яке представлене декількома кнопками взаємодії з проектом: грати, додаткові опції та вихід.

Рис. 2.18. Головне меню

При виборі кнопки Options користувач потрапляє в вікно редагування графіки та звуку. Усі зміни зберігаються автоматично методом Player Prefs [4] (рис. 2.19).

Рис. 2.19. Вікно налаштування музики та графіки

При завантаженні головної сцени, за допомогою написаних скриптів, виконується скрипт віддалення камери від сцени. Камера стабілізується на головному герої (рис. 2.20). Завантаження всіх об'єктів на сцені відбувалося в перші секунди після запуску гри під час відтворення анімації згасання екрану.

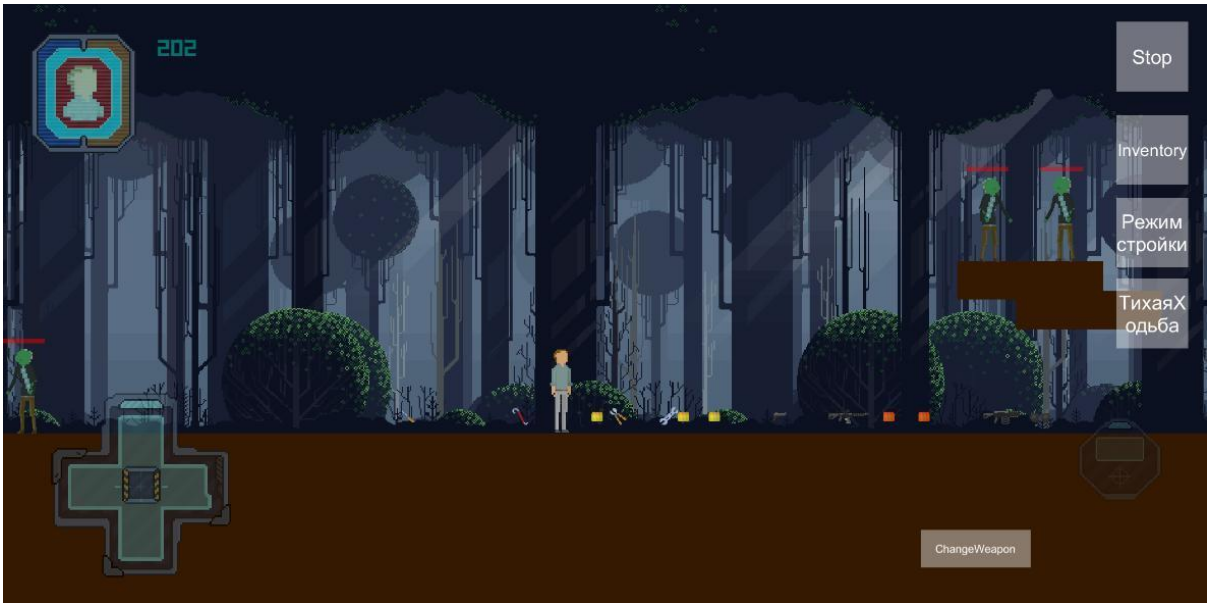

Рис. 2.20. Положення камери після завантаження сцени

На сцені у лівому верхньому куті розташована панель з характеристиками головного героя (рис. 2.21) Ця панель містить у собі такі показники: кількість життів, витривалість головного героя, шкала спраги, шкала голоду.

Рис. 2.21. Панель з характеристиками головного героя

У правому верхньому куті розташовані кнопки: зупинити гру, відкрити інвентар, режим огляду локації та тиха ходьба. Натиснувши на кнопку - зупиниту гру, відкривається панель (рис. 2.22) із кнопками: продовжити гру, налаштування, вийти з гри.

Рис. 2.22. Панель паузи гри

При натисканні на кнопку інвентарю, відкривається панель з інструментами та предметами гравця (рис. 2.23). Усі об'єкти в інвентарі інтерактивні. Їх можна перетягувати та викидати із інвентарю.

Рис. 2.23. Панель інвентаря гравця

У нижній частині екрану, розташовуються елементи управління головним героєм. Зліва - джойстик управління ходьбою. Справа - приціл та кнопка зміни зброї в руках. При натисканні на кнопку прицілу гравець вистрілює (рис. 2.24) у бік напряму зброї за умови, що в руках у гравця є зброя, а в інвентарі є патрони для діючої зброї.

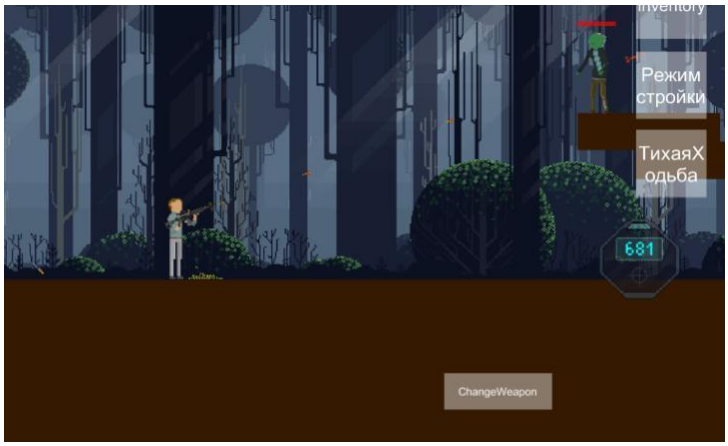

Рис. 2.24. Прицілювання та стрільба

РОЗДІЛ 3
ЗАГАЛЬНИЙ АНАЛІЗ ЕФЕКТИВНОСТІ РОЗРОБКИ ГРИ В UNITY 3D З ВИКОРИСТАННЯ ШІ

3.1 Покращена динаміка процесу розробки

Під час розробки штучний інтелект відіграв ключову роль у формуванні різних аспектів проекту. Впровадження складних алгоритмів ШІ полегшило нюансування поведінки персонажів, сприяючи підвищенню відчуття реалізму в ігровому світі. Завдяки використанню ШІ проект досяг безпрецедентного рівня динамічності, а неігрові персонажі (NPC) розумно реагували на зміну ігрових стимулів. Интеграція технологій штучного інтелекту та середовища Unity дала можливість максимально швидко находити рішення проблем які виникали через людський фактор.

3.2 Ефективний пошук шляхів та навігація

Однією з головних переваг інтеграції ШІ в Unity 3D є значне зниження обчислювального навантаження на пристрій, що тестується. Традиційна розробка ігор часто вимагала складних сценаріїв і систем, заснованих на правилах, що накладало значне навантаження на обчислювальні можливості цільового пристрою. З появою рішень на основі ШІ в Unity 3D розробники можуть перекласти певні процеси прийняття рішень на інтелектуальні алгоритми, тим самим полегшуючи обчислювальне навантаження на DUT.

3.3 Оптимізоване використання ресурсів

Інтеграція ШІ в Unity 3D полегшує динамічне управління ресурсами і стратегіями оптимізації, додатково сприяючи зниженню навантаження на пристрій. Адаптивні алгоритми ШІ можуть аналізувати дані про поведінку гравців у реальному часі та відповідно коригувати розподіл ресурсів. Наприклад, другорядні ресурси або процеси можуть бути тимчасово депріоритезовані в періоди низької активності

користувачів, зберігаючи обчислювальні ресурси для більш важливих завдань. Такий динамічний підхід до управління ресурсами підвищує загальну продуктивність і швидкість реагування, особливо на пристроях з обмеженими обчислювальними можливостями.

3.4 Висновки 3 розділу

Отже, інтеграція штучного інтелекту в 3D-рушій Unity знаменує собою важливу віху в еволюції розробки ігор. Симбіоз ШІ та Unity 3D не тільки відкриває безпрецедентні творчі можливості, але й вирішує критичну проблему обчислювального навантаження на тестований пристрій. Завдяки динамічному управлінню ресурсами, ефективному рендерингу та стратегічній оптимізації, рішення на основі ШІ в Unity 3D підвищують загальну продуктивність і швидкість відгуку ігор навіть на обладнанні з обмеженими можливостями. Оскільки ігрова індустрія продовжує розвиватися, поєднання ШІ та ігрових рушіїв, таких як Unity 3D, обіцяє змінити ландшафт, надаючи розробникам потужні інструменти для створення більш захоплюючих, ефективних і приємних цифрових вражень.

ВИСНОВКИ

Було досліджено ефективність застосування методів штучного інтелекту (ШІ) для вдосконалення комп'ютерних програм у контексті ігрового рушія Unity 3D. Завдяки всебічному дослідженню та експериментам ми отримали наступні ключові висновки та ідеї:

Покращена функціональність та складність:

Інтеграція штучного інтелекту продемонструвала суттєве покращення функціональності та складності комп'ютерних програм, розроблених на ігровому рушію Unity 3D. Алгоритми штучного інтелекту довели свою ефективність у наданні віртуальним об'єктам інтелектуальної поведінки, сприяючи тим самим більш захоплюючому та динамічному ігровому досвіду.

Ефективність і автоматизація:

Використання штучного інтелекту в процесі розробки показало багатообіцяючі результати з точки зору ефективності та автоматизації. Завдяки алгоритмам машинного навчання система продемонструвала здатність адаптуватися та оптимізуватися, зменшуючи потребу в ручному втручанні та прискорюючи життєвий цикл розробки.

Адаптивне навчання та прийняття рішень у режимі реального часу:

Дослідження виявило потенціал ШІ для полегшення адаптивного навчання у віртуальних середовищах. Включення моделей машинного навчання дозволяє приймати рішення в реальному часі ігровими об'єктами, що робить ігровий процес більш гнучким і цікавим для користувачів.

Виклики та міркування:

Незважаючи на позитивні результати, слід визнати, що існують такі проблеми, як інтерпретованість моделей ШІ, етичні міркування та потенційні упередження в наборах даних для навчання. Досягнення балансу між складністю алгоритмів ШІ та їхньою практичною реалізацією залишається критично важливим аспектом у забезпеченні етичного та відповідального використання ШІ в ігровій індустрії.

Майбутні напрямки:

Проведене дослідження відкрило шляхи для подальшого вивчення та вдосконалення. Подальші дослідження нових методологій ШІ, інтеграція етичних рамок і вирішення проблем, пов'язаних із застосуванням ШІ в розробці ігор, є життєво важливими сферами для майбутніх досліджень. Крім того, варто дослідити потенціал ШІ для оптимізації використання ресурсів і посилення аспектів співпраці в розробці ігор.

Підсумовуючи, результати цього дослідження підкреслюють трансформаційний потенціал штучного інтелекту у сфері розробки 3D-ігор Unity. Використовуючи можливості ШІ, розробники можуть не тільки створювати більш складні та інтерактивні ігри, але й сприяти постійному розвитку інтелектуальних систем у ширшій галузі комп'ютерних наук. Оскільки перетин ШІ та розробки ігор продовжує розвиватися, це дослідження слугує основою для подальших досліджень та інновацій у цій динамічній галузі, що стрімко розвивається.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація Unity 3D [Електронний ресурс] // URL: https://docs.unity3d.com/ru/530/Manual/ (дата звернення 17.05.2023)
2. Документація C# Docs [Електронний ресурс] // URL: https://docs.microsoft.com/en-us/dotnet/csharp/ (дата звернення 17.05.2023)
3. Жанр гри Hack'n'slash [Електронний ресурс] // URL: https://cubiq.ru/hack-and-slash/ (дата звернення 18.05.2023)
4. Збереження даних у Unity за допомогою Player Prefs [Електронний ресурс]//URL: https://docs.unity3d.com/ru/2019.4/ScriptReference/PlayerPrefs.html (дата звернення 26.05.2023)
5. Застосування алгоритму Дейкстри [Електронний ресурс] // URL: https://habr.com/ru/post/111361/ (дата звернення 01.06.2023)
6. Навчання роботи в Photoshop [Електронний ресурс] // URL: https://www.youtube.com/watch?v=KXOMHopxIl4 (дата звернення 01.06.2023)
7. Пояснення як робити управління 2D персонажем [Електронний ресурс] // URL: https://www.youtube.com/watch?v=tywt9tOubEY&ab_channel=Unity (дата звернення 18.05.2023)
8. Пояснення роботи алгоритму пошуку шляху юнітам [Електронний ресурс] // URL: https://www.red-gate.com/simple-talk/development/dotnet-development/pathfin ding-unity-c/ (дата звернення 01.06.2023)
9. Пояснення технічних особливостей двигуну Unity 3D для компіляції проекту під андроїд платформу [Електронний ресурс] // URL: https://www.youtube.com/watch?v=0NA6UAT-cAo&ab_channel=loftblog (дата звернення 24.05.2023)
10. Стиль Pixel Art [Електронний ресурс] // UR

https://diskettelounge.com/statie/istoriya-pixel-art/ (дата звернення 13.05.2023)

11. Що таке Visual Studio [Електронний ресурс] // URL: https://wiki.archlinux.org/title/Visual_Studio_Code_(%D0%A0%D1%83%D1%81%D1%81%D0%BA%D0%B8%D0%B9) (дата звернення 19.05.2022)

12. Що таке NPC у грі, та як його зробити у Unity [Електронний ресурс] // URL: https://hub.packtpub.com/how-to-create-non-player-characters-npc-with-unity-2 018/ (дата звернення 03.06.2023)

13. C# IntelliSense [Електронний ресурс] // URL: https://docs.microsoft.com/ru-ru/visualstudio/ide/visual-csharp-intellisense?vie w=vs-2023 (дата звернення 02.06.2023)

14. Github [Електронний ресурс] // URL: https://github.com/microsoft/vscode (дата звернення 25.05.2023)

15. Joe Hocking Unity in Action 2015. - 47с

16. Mark J. Price C# 7 and .NET Core 2017. - 33с

17. Mark Reed C#: The Ultimate Advanced Guide To Master C# Programming 2022. - 138с

18. Paul Schroeder Visual Studio 2019 Tricks and Techniques 2021. - 54с

19. Ui/Ux Unity 3d [Електронний ресурс] // URL: https://blog.unity.com/technology/evolving-the-unity-editor-ux (дата звернення 01.06.2023)

20. Will Goldstone Unity Game Development Essentials 2009. - 177с

ДОДАТОК А. Код програми

Item.cs - Контроль предметів на сцені

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

public class Item : Item, IPointerDownHandler, IPointerUpHandler, IDragHandler
{
    public MainController mainController;
    public Point_Starship point_Starship;
    [Space]
    public Image SpriteBlock;
    public Animator Anim_SpriteBlock;
    [Space]
    public bool Central_Block; //
    [Space]
    public int IndexBlock = -1; //
    public int Index_Category = 0; //
    [Space]
    public int Index_Point_Block = -1; //
    [Space]
    public bool move_Block = false; //
    public Btn_Item Btn_Spawner_This_Prefab_Block;

    Vector3 diff;


    void Start()
    {

    }

    public void OnPointerDown(PointerEventData eventData)
    {
        if (Central_Block)
        {
            diff = point_Starship.transform.position -
mainController.Cam.ScreenToWorldPoint(Input.mousePosition);

            point_Starship.Move_Starship = true;
        }
        else
        {
            Invoke("Start_Move_Block", 0.2f);
        }
    }

    public void OnDrag(PointerEventData eventData)
    {
        if (Central_Block)
            point_Starship.transform.position = mainController.Cam.ScreenToWorldPoint(Input.mousePosition) +
diff;
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        if (Central_Block)
            point_Starship.Move_Starship = false;
        else
```

```
        {
            if (!move_Block)
            {
                CancelInvoke("Start_Move_Block");
                CancelInvoke("MoveToPointBlock");
                CancelInvoke("Move_Block_Act1");

                mainController.Scroll_Items[mainController.Current_Panel_Scroll_Items].vertical = true;

                StopCoroutine(Move_Block(new Vector3(0, 0, 0)));

                Pos_PointBlock = point_Starship.point_Block[Index_Point_Block].transform.position;
                Invoke("MoveToPointBlock", 0.35f);

                mainController.Off_AreaToMove_ScrollItems();
            }
        }
    }

    void Start_Move_Block()
    {
        CancelInvoke("Start_Move_Block");
        CancelInvoke("MoveToPointBlock");
        Invoke("Move_Block_Act1", 0.25f);

        mainController.Scroll_Items[mainController.Current_Panel_Scroll_Items].vertical = false;

        StartCoroutine(Move_Block(mainController.Cam.ScreenToWorldPoint(Input.mousePosition) + new
Vector3(0, 0.7f, 0)));
    }

    IEnumerator Move_Block(Vector3 Pos)
    {
        float t = 0f;

        while (t <= 0.5f)
        {
            t += Time.deltaTime / 0.6f;

            transform.position = Vector3.Lerp(transform.position, Pos, 15f * Time.deltaTime);

            yield return null;
        }
    }

    void Move_Block_Act1()
    {
        CancelInvoke("Start_Move_Block");
        CancelInvoke("MoveToPointBlock");

        mainController.On_AreaToMove_ScrollItems();

        Point_Block point_Block = point_Starship.point_Block[Index_Point_Block];
        point_Block.Active_Point_Block = true;
        point_Block.Current_Index_Block = -2;
        point_Starship.Index_AllBlocks[Index_Point_Block] = -2;

        Check_Neighboring_Point_Blocks(point_Block);

        point_Starship.On_Active_Point_Blocks();

        move_Block = true;
    }
```

```
    void Check_Neighboring_Point_Blocks(Point_Block point_Block)
    {
        for (int i = 0; i < 4; i++)
        {
            if (point_Block.Neighboring_Point_Block[i] &&
point_Block.Neighboring_Point_Block[i].Active_Point_Block)
            {
                Point_Block neighboring_Point_Block = point_Block.Neighboring_Point_Block[i];

                bool Check = false;

                for (int j = 0; j < 4; j++)
                {
                    if (neighboring_Point_Block.Neighboring_Point_Block[j] &&
neighboring_Point_Block.Neighboring_Point_Block[j].Current_Index_Block >= 0)
                    {
                        Check = true;

                        break;
                    }
                }

                if (Check)
                    neighboring_Point_Block.Active_Point_Block = true;
                else
                    neighboring_Point_Block.Active_Point_Block = false;
            }
        }
    }


    RaycastHit2D Ray;
    Point_Block point_Block;

    void Update()
    {
        if (move_Block)
        {
            if (Input.touchCount == 1 || Input.GetMouseButton(0))
            {
                Vector3 MousePos = mainController.Cam.ScreenToWorldPoint(Input.mousePosition);
                Ray = Physics2D.Raycast(MousePos + new Vector3(0, 0.7f, 0), Vector2.zero, 1, 1 << 6);

                if(Ray.collider != null)
                {
                    if (point_Block != Ray.collider.gameObject.GetComponent<Point_Block>())
                    {
                        point_Block = Ray.collider.gameObject.GetComponent<Point_Block>();

                        transform.position = point_Block.transform.position
                    }
                }
                else
                {
                    if (point_Block)
                        point_Block = null;

                    transform.position = mainController.Cam.ScreenToWorldPoint(Input.mousePosition);
                    transform.position = new Vector3(transform.position.x, transform.position.y, 0) + new Vector3(0,
0.7f, 0);
                }
            }
```

```
        else
        {
            move_Block = false;

            if (Btn_Spawner_This_Prefab_Block)
            {
                if (point_Block)
                {
                    transform.parent = point_Starship.transform;
                    point_Block.Current_Index_Block = IndexBlock;
                    point_Block.Active_Point_Block = false;
                    point_Starship.Index_AllBlocks[point_Block.Index_Point_Block] = IndexBlock;
                    PlayerPrefs.SetInt("Index_AllBlocks_Key" + "/" + point_Block.Index_Point_Block.ToString(),
IndexBlock);

                    Index_Point_Block = point_Block.Index_Point_Block;

                    for (int i = 0; i < 4; i++)
                    {
                        if (point_Starship.point_Block[Index_Point_Block].Neighboring_Point_Block[i] &&
point_Starship.point_Block[Index_Point_Block].Neighboring_Point_Block[i].Current_Index_Block == -2)

point_Starship.point_Block[Index_Point_Block].Neighboring_Point_Block[i].Active_Point_Block = true;
                    }

                    Btn_Spawner_This_Prefab_Block.Destroy_Btn_Item();

                    point_Starship.Off_All_Point_Blocks();

                    point_Block = null;
                }
                else
                {
                    Btn_Spawner_This_Prefab_Block.StopAllInvoke();

                    Btn_Spawner_This_Prefab_Block = null;
                }
            }
            else //  Перетаскиваемый блок был взят из точки блока корабля
            {
                Vector3 MousePos = mainController.Cam.ScreenToWorldPoint(Input.mousePosition);
                Ray = Physics2D.Raycast(MousePos, Vector2.zero, 1, 1 << 7);

                bool destroy = false;

                if (Ray.collider != null)
                {
                    Btn_Item Prefab_Btn_Item = Instantiate(mainController.Prefab_Btn_Item, transform.position,
transform.rotation, mainController.Scroll_Items_Content[Index_Category].transform);
                    Prefab_Btn_Item.IndexItem = IndexBlock;
                    Prefab_Btn_Item.prefab_Block = GetComponent<Block>();
                    Prefab_Btn_Item.Image_Btn_Item.sprite = mainController.Sprite_Image_Btn_Item[IndexBlock];
                    mainController.All_Btn_Items.Add(Index_Category + "_" + IndexBlock);
                    Prefab_Btn_Item.IndexInListAllBtnItems = mainController.All_Btn_Items.Count - 1;
                    point_Starship.SaveAll_Btn_Items();
                    Prefab_Btn_Item.gameObject.SetActive(true);
                    if (mainController.Current_Panel_Scroll_Items == Index_Category)
                        Prefab_Btn_Item.Activate_Btn_Item();

                    point_Starship.point_Block[Index_Point_Block].Current_Index_Block = -2;
                    point_Starship.point_Block[Index_Point_Block].Active_Point_Block = true;
                    point_Starship.Index_AllBlocks[Index_Point_Block] = -2;
                    PlayerPrefs.SetInt("Index_AllBlocks_Key" + "/" + Index_Point_Block.ToString(), -2);
```

```
                Check_Neighboring_Point_Blocks(point_Starship.point_Block[Index_Point_Block]);

                Anim_SpriteBlock.Play("SpriteBlock_Anim");

                if(mainController.Current_Panel_Scroll_Items == Index_Category)
                    mainController.CheckNumItems();
                else

mainController.Btn_ChangePanel_CategoryItems[Index_Category].GetComponent<Animator>().Play("Btn_Ch
angePanel_CategoryItems_Anim");

                destroy = true;
            }
            else if(point_Block) // Перетаскиваемый блок примагничиваем к новой точке
            {
                PlayerPrefs.SetInt("Index_AllBlocks_Key" + "/" + Index_Point_Block.ToString(), -2);

                Index_Point_Block = point_Block.Index_Point_Block;

                point_Block.Current_Index_Block = IndexBlock;
                point_Block.Active_Point_Block = false;

                for (int i = 0; i < 4; i++)
                {
                    if (point_Starship.point_Block[Index_Point_Block].Neighboring_Point_Block[i] &&
point_Starship.point_Block[Index_Point_Block].Neighboring_Point_Block[i].Current_Index_Block == -2)

point_Starship.point_Block[Index_Point_Block].Neighboring_Point_Block[i].Active_Point_Block = true;
                }

                point_Starship.Index_AllBlocks[Index_Point_Block] = IndexBlock;
                PlayerPrefs.SetInt("Index_AllBlocks_Key" + "/" + Index_Point_Block.ToString(), IndexBlock);
            }
            else // Перетаскиваемый блок возвращаем в позицию откуда мы его взяли
            {

StartCoroutine(Move_Block(point_Starship.point_Block[Index_Point_Block].transform.position));

                Pos_PointBlock = point_Starship.point_Block[Index_Point_Block].transform.position;
                Invoke("MoveToPointBlock", 0.25f);

                point_Starship.point_Block[Index_Point_Block].Current_Index_Block = IndexBlock;
                point_Starship.point_Block[Index_Point_Block].Active_Point_Block = false;
                point_Starship.Index_AllBlocks[Index_Point_Block] = IndexBlock;

                for (int i = 0; i < 4; i++)
                {
                    if (point_Starship.point_Block[Index_Point_Block].Neighboring_Point_Block[i] &&
point_Starship.point_Block[Index_Point_Block].Neighboring_Point_Block[i].Current_Index_Block == -2)

point_Starship.point_Block[Index_Point_Block].Neighboring_Point_Block[i].Active_Point_Block = true;
                }
            }

            mainController.Scroll_Items[mainController.Current_Panel_Scroll_Items].vertical = true;

            mainController.Off_AreaToMove_ScrollItems();

            point_Starship.Off_All_Point_Blocks();

            if(destroy)
                Destroy(gameObject, 0.2f);
```

```
            }
        }
    }
}

Vector3 Pos_PointBlock;

void MoveToPointBlock()
{
    transform.position = Pos_PointBlock;
}
}
```

InventorySlot.cs - Контроль блоків інвентарю

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

public class Point_Starship : MonoBehaviour
{
    public MainController mainController;
    [Space]
    public int[] Index_AllBlocks; // Если -2, то блока нету. Если -1, то блок есть но соседние к нему нельзя
присоединить. Если 0, то место под блок занято частью большого блок. Если 1 то там находится блок
    public Point_Block[] point_Block; // Точки куда станет блок
    [Space]
    public Block[] Prefab_Blocks; // Префабы блоков
    [Space]
    public float Scale_Starship;
    [Space]
    public bool Move_Starship;


    public void Load_Starship()
    {
        Scale_Starship = transform.localScale.x;

        for (int i = 0; i < Index_AllBlocks.Length; i++)
        {
            Index_AllBlocks[i] = PlayerPrefs.GetInt("Index_AllBlocks_Key" + "/" + i.ToString(), -2);

            point_Block[i].Current_Index_Block = Index_AllBlocks[i];

            if (Index_AllBlocks[i] > 0)
            {
                Block prefab_Block = Instantiate(Prefab_Blocks[Index_AllBlocks[i]],
point_Block[i].transform.position, transform.rotation, transform);
                prefab_Block.transform.localScale = new Vector3(1, 1, 1);
                prefab_Block.Index_Point_Block = i;
                prefab_Block.gameObject.SetActive(true);
            }
        }

        for (int i = 0; i < Index_AllBlocks.Length; i++)
        {
            if (point_Block[i].Current_Index_Block == -2)
            {
                for (int j = 0; j < 4; j++)
                {
```

```
                if (point_Block[i].Neighboring_Point_Block[j] &&
point_Block[i].Neighboring_Point_Block[j].Current_Index_Block >= 0)
                {
                    point_Block[i].Active_Point_Block = true;
                    break;
                }
            }
        }
    }

    Load_All_Btn_Items();
  }

  void Load_All_Btn_Items()
  {
    if (!PlayerPrefs.HasKey("FirstLoad_Key"))
    {
        PlayerPrefs.SetInt("FirstLoad_Key", 0);

        mainController.IndexAll_Btn_Items = 1;
        PlayerPrefs.SetInt("IndexAll_Btn_Items_Key", mainController.IndexAll_Btn_Items);

        mainController.All_Btn_Items.Add("0_1"); // Категория_ИндексБлока

        for (int i = 0; i < mainController.IndexAll_Btn_Items; i++)
        {
            PlayerPrefs.SetString("IndexAll_Btn_Items_Key" + "/" + i.ToString(),
mainController.All_Btn_Items[i]);
        }

        mainController.All_Btn_Items.Clear();
    }

    mainController.IndexAll_Btn_Items = PlayerPrefs.GetInt("IndexAll_Btn_Items_Key");

    for (int i = 0; i < mainController.IndexAll_Btn_Items; i++)
    {
        mainController.All_Btn_Items.Add(PlayerPrefs.GetString("IndexAll_Btn_Items_Key" + "/" +
i.ToString()));

        string[] Current_Btn_Item = mainController.All_Btn_Items[i].Split(char.Parse("_"));

        Btn_Item Prefab_Btn_Item = Instantiate(mainController.Prefab_Btn_Item, transform.position,
transform.rotation, mainController.Scroll_Items_Content[int.Parse(Current_Btn_Item[0])].transform);
        Prefab_Btn_Item.IndexItem = int.Parse(Current_Btn_Item[1]);
        Prefab_Btn_Item.prefab_Block = GetComponent<Block>();
        Prefab_Btn_Item.Image_Btn_Item.sprite =
mainController.Sprite_Image_Btn_Item[int.Parse(Current_Btn_Item[1])];
        Prefab_Btn_Item.IndexInListAllBtnItems = i;
        Prefab_Btn_Item.gameObject.SetActive(true);
    }
  }

  public void SaveAll_Btn_Items()
  {
    mainController.All_Btn_Items.RemoveAll(string.IsNullOrEmpty);

    mainController.IndexAll_Btn_Items = mainController.All_Btn_Items.Count;
    PlayerPrefs.SetInt("IndexAll_Btn_Items_Key", mainController.IndexAll_Btn_Items);

    for (int i = 0; i < mainController.IndexAll_Btn_Items; i++)
    {
```

```
            PlayerPrefs.SetString("IndexAll_Btn_Items_Key" + "/" + i.ToString(),
mainController.All_Btn_Items[i]);
        }
    }


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////

    public void On_Active_Point_Blocks()
    {
        for (int i = 0; i < point_Block.Length; i++)
        {
            if (point_Block[i].Active_Point_Block)
            {
                point_Block[i].gameObject.SetActive(true);
                point_Block[i].Anim_Point_Block.Play("Point_Block_Anim");
            }
        }
    }

    public void Off_All_Point_Blocks()
    {
        for (int i = 0; i < point_Block.Length; i++)
        {
            if(point_Block[i].gameObject.activeSelf)
            {
                point_Block[i].Off_Point_Blocks();
            }
        }
    }
}
```

ItemsController.cs - Контроль інвентарю

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

public class MainController : MonoBehaviour
{
    public Camera Cam;
    public Canvas canvas;
    public ScrollPages ScrollPages;
    [Space]
    public GameObject Panel_Workshop;
    public Point_Starship Point_Starship;

    public GameObject Panel_Items;

    public int Current_Panel_Scroll_Items;
    public Button[] Btn_ChangePanel_CategoryItems;
    public ScrollRect[] Scroll_Items;
    public RectTransform[] Scroll_Items_Content;

    public GameObject AreaToMove_ScrollItems;
```

```
    public Button Btn_Hide_PanelItems;

    public bool Panel_Items_Hidden = false;

    public Btn_Item Prefab_Btn_Item; // Префаб кнопки предмета. Она универсальна для всех предметов
    public int IndexAll_Btn_Items;
    public List<string> All_Btn_Items;
    public Sprite[] Sprite_Image_Btn_Item;


    void Awake()
    {
        Point_Starship.Load_Starship();
    }


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////

    public void HideOrOpenPanelItems()
    {
        if (!Point_Starship.Move_Starship)
        {
            if (Panel_Items_Hidden)
            {
                Panel_Items_Hidden = false;

                Scroll_Items[Current_Panel_Scroll_Items].vertical = true;

                Panel_Items.GetComponent<Animator>().Play("Panel_Items_Anim");

                Move_Point_Starship(new Vector2(0, 390));
            }
            else
            {
                Panel_Items_Hidden = true;

                Scroll_Items[Current_Panel_Scroll_Items].vertical = false;

                Panel_Items.GetComponent<Animator>().Play("Panel_Items_Anim_Reverse");

                Move_Point_Starship(new Vector2(0, 0));
            }

            // Сменить спрайт кнопке Btn_Hide_PanelItems
        }
    }

    public void Move_Point_Starship(Vector3 Pos)
    {
        if (Mathf.RoundToInt(Point_Starship.transform.localPosition.x) != Mathf.RoundToInt(Pos.x) ||
Mathf.RoundToInt(Point_Starship.transform.localPosition.y) != Mathf.RoundToInt(Pos.y))
```

```
        {
            StopCoroutine(move_Point_Starship(new Vector2(0, 0)));
            StartCoroutine(move_Point_Starship(Pos));
        }
    }

    IEnumerator move_Point_Starship(Vector2 Point)
    {
        float t = 0f;

        while (t <= 1.0f)
        {
            t += Time.deltaTime / 0.6f;
            Point_Starship.transform.localPosition = Vector3.Lerp(Point_Starship.transform.localPosition, Point,
15f * Time.deltaTime);

            yield return null;
        }
    }

    public void CheckNumItems() // Проверить количество обьектов и изменить размер
Scroll_Items_Content под это количество
    {
        if (Scroll_Items_Content[Current_Panel_Scroll_Items].gameObject.transform.childCount > 15)
        {
            int i = Scroll_Items_Content[Current_Panel_Scroll_Items].gameObject.transform.childCount - 15;

            int t = 1;

            if (i % 5 != 0)
                t = 2;

            Scroll_Items_Content[Current_Panel_Scroll_Items].sizeDelta = new Vector2(0, ((i / 5) + t) * 205 +
800);
        }
    }

    public void On_AreaToMove_ScrollItems()
    {
        AreaToMove_ScrollItems.SetActive(true);
    }

    public void Off_AreaToMove_ScrollItems()
    {

AreaToMove_ScrollItems.GetComponent<Animator>().Play("AreaToMove_ScrollItems_Anim_Reverse");

        Invoke("Off_AreaToMove_ScrollItems_Act1", 0.2f);
    }

    void Off_AreaToMove_ScrollItems_Act1()
    {
        AreaToMove_ScrollItems.SetActive(false);
```

```
    }

    public void Change_Panel_Scroll_Items(int Index)
    {
        if (Current_Panel_Scroll_Items != Index)
        {
            Current_Panel_Scroll_Items = Index;

            for (int i = 0; i < Scroll_Items.Length; i++)
            {
                Scroll_Items[i].gameObject.SetActive(false);
            }

            CheckNumItems();

            Scroll_Items_Content[Index].anchoredPosition = new
Vector3(Scroll_Items_Content[Index].anchoredPosition.x, 0, 0);
            Scroll_Items[Index].gameObject.SetActive(true);
        }
    }


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////
}
```

Player.cs - Скрипт управління головним героєм

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using UnityEngine.EventSystems;
using static UnityEngine.RuleTile.TilingRuleOutput;

public class Player : MonoBehaviour, IDragHandler, IPointerDownHandler, IPointerUpHandler
{
    public MainController mainController;
    public Point_Starship Point_Starship;
    public Canvas canvas;
    [Space]
    public GameObject[] Pages;
    [Space]
    public GameObject Panel_Account_Resources;
    bool Panel_Account_Resources_Up = false;
    bool Panel_Account_Resources_Hidden = false;
    [Space]
    public RectTransform[] Btn_Change_Pages;
    [Space]
    public int CurrentPage = 0; // Текущая страница в меню
    [Space]
    public Vector2[] ScalePage0;
    public Vector2[] PosPage0;
    [Space]
    public Vector2[] ScalePage1;
    public Vector2[] PosPage1;
    [Space]
```

```
    public Vector2[] ScalePage2;
    public Vector2[] PosPage2;
    [Space]
    public Vector2[] ScalePage3;
    public Vector2[] PosPage3;
    [Space]
    public Vector2[] ScalePage4;
    public Vector2[] PosPage4;
    [Space]
    public float[] Bias_CoorX;

    bool CanBeMoved = false;
    bool isDragging = false;

    Vector2 diff;
    Vector2 StartPos;


    void Start()
    {
        Btn_Change_Pages[CurrentPage].gameObject.GetComponent<Animator>().Play("Btn_PageAnim");
    }


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////

    public void ChangePage(int IndexPage) // Сменить страницу меню
    {
        if (!isDragging && !Point_Starship.Move_Starship)
        {
            if (CurrentPage != IndexPage)
            {

Btn_Change_Pages[CurrentPage].gameObject.GetComponent<Animator>().Play("Btn_PageAnim_Reverse");

                CurrentPage = IndexPage;

                //isDragging = true;
                //Invoke("IsDraggingFalse", 0.5f);

                Move(true);
            }
            else
            {

Btn_Change_Pages[CurrentPage].gameObject.GetComponent<Animator>().Play("Btn_PageAnim_Press");
            }
        }
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        if (!isDragging && !Point_Starship.Move_Starship)
        {
            RectTransformUtility.ScreenPointToLocalPointInRectangle(canvas.transform as RectTransform,
eventData.position, canvas.worldCamera, out Vector2 localPoint);
            diff = new Vector2(transform.position.x, 0) - new Vector2(localPoint.x, 0);

            StartPos = transform.localPosition;
        }
    }
```

```
    public void OnDrag(PointerEventData eventData)
    {
        if (!isDragging && CanBeMoved && !Point_Starship.Move_Starship)
        {
            RectTransformUtility.ScreenPointToLocalPointInRectangle(canvas.transform as RectTransform,
eventData.position, canvas.worldCamera, out Vector2 localPoint);

            transform.position = canvas.transform.TransformPoint(new Vector2(localPoint.x + diff.x +
Bias_CoorX[CurrentPage], 0));
        }
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        if (!isDragging && !Point_Starship.Move_Starship)
        {
            //isDragging = true;
            //Invoke("IsDraggingFalse", 0.5f);

            float dist = StartPos.x - transform.localPosition.x;
            StartPos = new Vector2(0, 0);
            bool MoveStarship = false;

            if (dist > 200)
            {
                if (CurrentPage < 4)
                {

Btn_Change_Pages[CurrentPage].gameObject.GetComponent<Animator>().Play("Btn_PageAnim_Reverse");

                    CurrentPage++;

                    MoveStarship = true;
                }
            }
            else if (dist < -200)
            {
                if (CurrentPage > 0)
                {

Btn_Change_Pages[CurrentPage].gameObject.GetComponent<Animator>().Play("Btn_PageAnim_Reverse");

                    CurrentPage--;

                    MoveStarship = true;
                }
            }

            if(MoveStarship)
                Move(true);
            else
                Move(false);
        }
    }

    public void PointerDown() // Проверка на нажатие именно на скролл панель
    {
        if(!Point_Starship.Move_Starship)
            CanBeMoved = true;
    }

    public void PointerUp() // Проверка на отпускание скролл панели
    {
```

```
        if(!Point_Starship.Move_Starship)
            CanBeMoved = false;
    }


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////
    /*
    void IsDraggingFalse()
    {
        isDragging = false;
    }
    */
    void Move(bool MoveStarship)
    {
        Btn_Change_Pages[CurrentPage].gameObject.GetComponent<Animator>().Play("Btn_PageAnim");

        if (MoveStarship)
        {
            if (mainController.Panel_Items_Hidden)
                mainController.Move_Point_Starship(new Vector2(0, 0));
            else
                mainController.Move_Point_Starship(new Vector2(0, 390));
        }

        if (CurrentPage <= 0)
        {
            CurrentPage = 0;

            for (int i = 0; i < Btn_Change_Pages.Length; i++)
            {
                Btn_Change_Pages[i].sizeDelta = ScalePage0[i];
                Btn_Change_Pages[i].anchoredPosition = PosPage0[i];
            }

            CancelInvoke("Panel_Account_Resources_Anim_Up");
            CancelInvoke("Panel_Account_Resources_Anim_Down");

            if (!Panel_Account_Resources_Hidden)
            {
                Panel_Account_Resources_Hidden = true;

                if(Panel_Account_Resources_Up)

Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Up_R");
                else

Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Down_R");
            }

            StartCoroutine(SmoothMoveTest(3000));
        }
        else if (CurrentPage == 1)
        {
            for (int i = 0; i < Btn_Change_Pages.Length; i++)
            {
                Btn_Change_Pages[i].sizeDelta = ScalePage1[i];
                Btn_Change_Pages[i].anchoredPosition = PosPage1[i];
            }

            CancelInvoke("Panel_Account_Resources_Anim_Up");
            CancelInvoke("Panel_Account_Resources_Anim_Down");
```

```
            if (Panel_Account_Resources_Hidden)
            {
                Panel_Account_Resources_Hidden = false;

                Panel_Account_Resources_Up = true;

Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Up");
            }
            else if (!Panel_Account_Resources_Hidden && !Panel_Account_Resources_Up)
            {
                Panel_Account_Resources_Up = true;


Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Down_R");

                Invoke("Panel_Account_Resources_Anim_Up", 0.2f);
            }

            StartCoroutine(SmoothMoveTest(1500));
        }
        else if (CurrentPage == 2)
        {
            for (int i = 0; i < Btn_Change_Pages.Length; i++)
            {
                Btn_Change_Pages[i].sizeDelta = ScalePage2[i];
                Btn_Change_Pages[i].anchoredPosition = PosPage2[i];
            }

            CancelInvoke("Panel_Account_Resources_Anim_Up");
            CancelInvoke("Panel_Account_Resources_Anim_Down");

            if (Panel_Account_Resources_Hidden)
            {
                Panel_Account_Resources_Hidden = false;

                Panel_Account_Resources_Up = false;

Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Down");
            }
            else if (!Panel_Account_Resources_Hidden && Panel_Account_Resources_Up)
            {
                Panel_Account_Resources_Up = false;


Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Up_R");

                Invoke("Panel_Account_Resources_Anim_Down", 0.2f);
            }

            StartCoroutine(SmoothMoveTest(0));
        }
        else if (CurrentPage == 3)
        {
            for (int i = 0; i < Btn_Change_Pages.Length; i++)
            {
                Btn_Change_Pages[i].sizeDelta = ScalePage3[i];
                Btn_Change_Pages[i].anchoredPosition = PosPage3[i];
            }

            CancelInvoke("Panel_Account_Resources_Anim_Up");
            CancelInvoke("Panel_Account_Resources_Anim_Down");

            if (Panel_Account_Resources_Hidden)
```

```
            {
                Panel_Account_Resources_Hidden = false;

                Panel_Account_Resources_Up = true;

Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Up");
            }
            else if (!Panel_Account_Resources_Hidden && !Panel_Account_Resources_Up)
            {
                Panel_Account_Resources_Up = true;


Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Down_R");

                Invoke("Panel_Account_Resources_Anim_Up", 0.2f);
            }

            StartCoroutine(SmoothMoveTest(-1500));
        }
        else if (CurrentPage >= 4)
        {
            CurrentPage = 4;

            for (int i = 0; i < Btn_Change_Pages.Length; i++)
            {
                Btn_Change_Pages[i].sizeDelta = ScalePage4[i];
                Btn_Change_Pages[i].anchoredPosition = PosPage4[i];
            }

            CancelInvoke("Panel_Account_Resources_Anim_Up");
            CancelInvoke("Panel_Account_Resources_Anim_Down");

            if (!Panel_Account_Resources_Hidden)
            {
                Panel_Account_Resources_Hidden = true;

                if (Panel_Account_Resources_Up)

Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Up_R");
                else

Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Down_R");
            }

            StartCoroutine(SmoothMoveTest(-3000));
        }
    }

    IEnumerator SmoothMoveTest(float PosX)
    {
        float t = 0f;

        while (t <= 1.0f)
        {
            t += Time.deltaTime / 0.6f;
            transform.localPosition = Vector3.Lerp(transform.localPosition, new Vector2(PosX, 0f), 15f *
Time.deltaTime);

            yield return null;
        }
    }
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////

    public void Panel_Account_Resources_Anim_Up()
    {
        Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Up");
    }

    public void Panel_Account_Resources_Anim_Down()
    {
        Panel_Account_Resources.GetComponent<Animator>().Play("Panel_Account_Resources_Anim_Down");
    }


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////
}
```

Btn_Item.cs - Скрипт управління предметами через кнопку

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

public class Btn_Item : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{

    public MainController mainController;
    public Point_Starship point_Starship;
    [Space]
    public Image Image_Btn_Item;
    public Animator Anim_Image_Btn_Item;

    public Animator Anim_Btn_Item;
    [Space]
    public int IndexItem = -1;
    public int IndexInListAllBtnItems = -1;
    [Space]
    public Block prefab_Block;
    bool BlockBtn = false;


    public void OnPointerDown(PointerEventData eventData)
    {
        if (!BlockBtn && !prefab_Block)
        {
            Invoke("Start_FillAnim", 0.2f);
        }
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        if (prefab_Block && !prefab_Block.move_Block)
        {
            StopAllInvoke();
        }
        else
        {
```

```
            CancelInvoke("Start_FillAnim");
        }
    }

    public void StopAllInvoke()
    {
        CancelInvoke("Start_FillAnim");
        CancelInvoke("StopMove_Prefab_Block");
        CancelInvoke("StopMoveBack_Prefab_Block");

        Anim_Image_Btn_Item.Play("Image_Btn_Item_Anim_Reverse");

        if(prefab_Block)
            prefab_Block.Anim_SpriteBlock.Play("SpriteBlock_Anim");

        point_Starship.Off_All_Point_Blocks();

        StopCoroutine(Move_Prefab_Block(new Vector3(0, 0, 0), 0));

        StartCoroutine(MoveBack_Prefab_Block(transform.position));

        mainController.Scroll_Items[mainController.Current_Panel_Scroll_Items].vertical = true;

        Invoke("StopMoveBack_Prefab_Block", 0.25f);
    }

    void Start_FillAnim()
    {
        BlockBtn = true;

        GetComponent<Button>().interactable = false;
        Anim_Image_Btn_Item.Play("Image_Btn_Item_Anim");
        //Anim_Btn_Item.Play("Btn_Item_Anim");

        mainController.Scroll_Items[mainController.Current_Panel_Scroll_Items].vertical = false;

        prefab_Block = Instantiate(point_Starship.Prefab_Blocks[IndexItem], transform.position,
transform.rotation, mainController.Panel_Workshop.transform);

        prefab_Block.transform.localScale = new Vector3(3, 3, 1);
        prefab_Block.Btn_Spawner_This_Prefab_Block = GetComponent<Btn_Item>();
        prefab_Block.transform.position = transform.position;
        prefab_Block.gameObject.SetActive(true);
        prefab_Block.Anim_SpriteBlock.Play("SpriteBlock_Anim_Reverse");

        StartCoroutine(Move_Prefab_Block(mainController.Cam.ScreenToWorldPoint(Input.mousePosition) +
new Vector3(0, 0.7f, 0), point_Starship.Scale_Starship));

        Invoke("StopMove_Prefab_Block", 0.25f);
    }

    IEnumerator Move_Prefab_Block(Vector3 Pos, float Scale)
    {
        float t = 0f;

        while (t <= 0.5f)
        {
            t += Time.deltaTime / 0.6f;

            if (prefab_Block)
            {
                prefab_Block.transform.position = Vector3.Lerp(prefab_Block.transform.position, Pos, 15f *
Time.deltaTime);
```

```
            prefab_Block.transform.localScale = Vector3.Lerp(prefab_Block.transform.localScale, new
Vector3(Scale, Scale, 1), 15f * Time.deltaTime);
        }

        yield return null;
    }
  }

  void StopMove_Prefab_Block()
  {
    StopCoroutine(Move_Prefab_Block(new Vector3(0, 0, 0), 0));

    point_Starship.On_Active_Point_Blocks();

    prefab_Block.move_Block = true;
  }

  IEnumerator MoveBack_Prefab_Block(Vector3 Pos)
  {
    float t = 0f;

    while (t <= 0.5f)
    {
      t += Time.deltaTime / 0.6f;

      if (prefab_Block)
      {
        prefab_Block.transform.position = Vector3.Lerp(prefab_Block.transform.position, Pos, 15f *
Time.deltaTime);
        prefab_Block.transform.localScale = Vector3.Lerp(prefab_Block.transform.localScale, new
Vector3(3, 3, 1), 15f * Time.deltaTime);
      }

      yield return null;
    }
  }

  void StopMoveBack_Prefab_Block()
  {
    StopCoroutine(MoveBack_Prefab_Block(new Vector3(0, 0, 0)));

    Destroy(prefab_Block.gameObject);

    BlockBtn = false;

    GetComponent<Button>().interactable = true;
    //Anim_Btn_Item.Play("None");

    mainController.Scroll_Items[mainController.Current_Panel_Scroll_Items].vertical = true;
  }


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////

  public void Activate_Btn_Item()
  {
    StartCoroutine(DecreaseBtn(0, 1));
  }

  public void Destroy_Btn_Item()
  {
    CancelInvoke("Start_FillAnim");
```

```
        CancelInvoke("StopMove_Prefab_Block");
        CancelInvoke("StopMoveBack_Prefab_Block");

        StopCoroutine(Move_Prefab_Block(new Vector3(0, 0, 0), 0));

        StopCoroutine(MoveBack_Prefab_Block(new Vector3(0, 0, 0)));

        GetComponent<Button>().interactable = false;

        StartCoroutine(DecreaseBtn(1, 0));

        mainController.CheckNumItems();

        mainController.All_Btn_Items[0] = "";
        point_Starship.SaveAll_Btn_Items();

        Destroy(gameObject, 0.4f);
    }

    IEnumerator DecreaseBtn(float StartScale, float FinalSize)
    {
        float t = 0f;

        transform.localScale = new Vector3(StartScale, StartScale, 1);

        while (t <= 0.4f)
        {
            t += Time.deltaTime / 5f;

            transform.localScale = Vector3.Lerp(transform.localScale, new Vector3(FinalSize, FinalSize, 1), 15f *
Time.deltaTime);

            yield return null;
        }
    }
}
```

Додаток Б
ПЕРЕЛІК ДОКУМЕНТІВ НА ДИСКУ

Ім’я файла	**Опис**
Пояснювальні документи	
Диплом_Шевченко.doc	Пояснювальна записка роботи. Документ Word.
Диплом_Шевченко.pdf	Пояснювальна записка роботи в форматі PDF
Програма	
Project.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_Шевченко.pptx	Презентація роботи