Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

СИМУЛЯТОР СИСТЕМНОГО АДМІНІСТРАТОРА В СЕРЕДОВИЩІ РОЗРОБКИ UNITY МОВОЮ ПРОГРАМУВАННЯ C#

SYSTEM ADMINISTRATOR SIMULATOR IN THE UNITY DEVELOPMENT ENVIRONMENT BY MEANS C# PROGRAMMING LANGUAGE

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІс-21
Бірук Богдан Валентинович

(підпис)

Керівник:
к.т.н., доцент
Христинець Наталія Анатоліївна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« 07 » червня 2024 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
« 10 » 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Біруку Богдану Валентиновичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Симулятор системного адміністратора в середовищі розробки Unity мовою програмування C#

Керівник роботи к.т.н., доцент Христинець Наталія Анатоліївна

затверджені наказом закладу вищої освіти від «30» грудня 2023 року № 459/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 11.06.2024р.

3. Вихідні дані до роботи Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування
3. Вихідні дані до роботи

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Аналіз предметної області
Проектування та моделювання додатку
Програмна реалізація та розробка додатку
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Аналітичний огляд
Проектування та моделювання додатку
Програмна реалізація та розробка додатку

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Христинець Н.А., доцент*		
Проектування та моделювання додатку	*Христинець Н.А., доцент*		
Програмна реалізація та розробка додатку	*Христинець Н.А., доцент*		
Нормоконтроль	*Багнюк Н.В., доцент*		
Гарант ОП	*Лавренчук С.В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О.І.,асистент*		

7. Дата видачі завдання 10.01.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Розділ 1 Аналіз предметної області*	до 15.02.2024 р.	Виконано
2.	*Розділ 2. Проектування та моделювання додатку*	до 15.03.2024 р.	Виконано
3.	*Розділ 3. Програмна реалізація та розробка додатку*	до 04.05.2024 р.	Виконано
4.	*Висновки та пропозиції*	до 07.05.2025 р.	Виконано
5.	*Формування списку використаних джерел*	до 10.05.2024 р.	Виконано
6.	*Формування додатків*	до 15.05.2024 р.	Виконано
7.	*Оформлення ілюстративного матеріалу*	до 20.05.2024 р.	Виконано
8.	*Нормоконтроль*	до 01.06.2024 р.	Виконано
9.	*Інструментальна перевірка на академічний плагіат*	до 04.06.2024 р	Виконано
10.	*Представлення кваліфікаційної роботи бакалавра до захисту*	до 11.06.2024 р.	Виконано

Здобувач вищої освіти _____ (підпис) Бірук Б.В. (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) Христинець Н.А. (прізвище, ініціали)

АНОТАЦІЯ

Бірук Б.В. Симулятор системного адміністратора в середовищі розробки Unity мовою програмування C#. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2024. 63 с.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, трьох додатків.

Перший розділ присвячено огляду предметної олбасті, тут розглядаються основні поняття про ігрові жанри, ігрові механіки притамані жанру 2D симулятору, основи піксельної графіки, порівняння популярних ігрових рушіїв, взаємодію обраного рушія з платформою Arduino. Також в цьому розділі здійснено огляд програм-аналогів (Симулятор системного адміністратора).

В другому розділі розглядаються основні алгоритми роботи додатку, способи інтеграції IoT, інструменти платформи Arduino.

Третій розділ присвячено опису створення додатку, реалізації головного меню, розробці графіки для гри, створеню ігрових механік, і тестування та запуск додатку.

Об'єкт дослідження – симулятор системного адміністратора в середовищі розробки Unity.

Предмет дослідження – технічні та інформаційні аспекти створення симулятора у середовищі розробки Unity з використанням мови програмування C#.

Метою роботи є створення інтерактивного та навчального віртуального середовища.

Ключові слова: симулятор, системний адміністратор, рушій, Unity, C#, 2D, піксельна графіка, arduino, графіка.

ANNOTATION

Biruk B.V. System administrator simulator in the Unity development environment by means C# programming language. Manuscript.

Qualifying work of a bachelor of EP «Computer Engineering» specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2024. 63 p.

The qualification work consists of an introduction, three chapters, conclusions, a list of references, and three appendices.

The first chapter is devoted to an overview of the subject area, it discusses the basic concepts of game genres, game mechanics inherent in the 2D simulator genre, the basics of pixel graphics, a comparison of popular game engines, the interaction of the selected engine with the Arduino platform. This section also provides an overview of analog programs (System Administrator Simulator).

The second section discusses the basic algorithms of the application, ways to integrate IoT, and tools of the Arduino platform.

The third section describes the creation of the application, the implementation of the main menu, the development of graphics for the game, the creation of game mechanics, and the testing and launch of the application.

The object of research is a system administrator simulator in the Unity development environment.

The subject of research is the technical and informational aspects of creating a simulator in the Unity development environment using the C# programming language.

The aim of the work is to create an interactive and educational virtual environment.

Key words: simulator, system administrator, engine, Unity, C#, 2D, pixel graphics, arduino, graphics.

ЗМІСТ

ВСТУП 7
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 9
1.1 Жанр симулятору 9
1.2 Механіки жанру 2D симулятора 10
1.3 Основи піксель-арту 11
1.4 Штучний інтелект в комп’ютерних застосунках 13
1.5 Вибір рушія 15
1.6 Взаємодія Unity з платою Arduino 16
РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ДОДАТКУ 18
2.1 Алгоритм роботи додатку 18
2.2 Способи інтеграції IoT та взаємодія платформи Arduino з симулятором 20
2.3 Інструменти платформи Arduino 22
РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА РОЗРОБКА ДОДАТКУ 25
3.1 Створення головного меню 25
3.2 Розробка графіки для додатку 29
3.3 Розробка основних механік 32
3.4 Тестування та запуск додатку 41
ВИСНОВКИ 44
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 46
ДОДАТКИ 47

ВСТУП

У сучасному інформаційному суспільстві, де технології стрімко розвиваються, важливість інформаційної безпеки та ефективного управління системами набуває критичного значення. Актуальність теми «Симулятор системного адміністратора в середовищі розробки Unity мовою програмування C#» очевидна, оскільки даний застосунок спрямований на надання нового підходу до навчання та тренування фахівців, а також на вдосконалення їхніх навичок у віртуальному середовищі.

Мета роботи – створення інтерактивного та навчального віртуального середовища.

Розробка має завдання емулювати складні ситуації, з якими стикаються системні адміністратори у реальному житті. Вона спрямована на розвиток навичок управління системами, вирішення проблем та прийняття рішень в умовах віртуального реалізму.

Об’єкт дослідження – симулятор системного адміністратора в середовищі Unity.

Предметом дослідження є технічні та інформаційні аспекти створення симулятора у середовищі розробки Unity з використанням мови програмування C#.

Для досягнення мети поставлено наступні завдання:

– провести аналітичний огляд питань формування мультимедійних додатків і визначити, які елементи професійної діяльності системного адміністратора будуть впроваджені в розробку;

– провести огляд популярних рішень з питань застосування елементів штучного інтелекту та визначити доцільність їх впровадження у додатку;

– дослідити ігрові рушії, порівняти їх характеристики та аргументувати вибір рушія для розробки;

– скласти алгоритм роботи додатку, згенерувати список завдань;

– організувати взаємодію рушія з світлодіодами на відкритій апаратній платформі для створення електронних пристроїв та прототипів;

– забезпечити програмну реалізацію базової системи руху, генерації задач і організувати взаємодію компонентів додатку;

– виконати компіляцію і запуск додатку, перевірити його роботу на різних операційних системах родини Windows.

Практичне значення розробки полягає у створенні програмного додатку для адміністрування комп'ютерних систем, що дозволить позиціонувати його для більшої аудиторії в навчальному процесі. Крім того, досліджені в кваліфікаційній роботі методи формування об'єктів взаємодії можуть слугувати теоретичним та практичним матеріалом при вивченні учнями, чи студентами відповідних дисциплін.

Наукова новизна роботи полягає в унікальності і власному комплексному підході до роботи з проектування мультимедійної розробки.

Апробація результатів роботи:

Бірук Б. В., Христинець Н. А. Симулятор системного адміністратора в контексті інтеграції з IoT. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво*. 2024. №54. С. 49-52 (Додаток И).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Жанр симулятору

Симулятори – це [1] жанр відеоігор, які намагаються відтворити реальні або уявні процеси, ситуації чи дії з максимальною точністю. Гравець отримує можливість взаємодіяти з віртуальним середовищем, наслідуючи реальні аспекти життя або певної діяльності.

В переважній більшості розробники відео ігор даного жанру намагаються реалізувати якомога детальніше всі аспекти обраної тематики гри, за рахунок цього гравці мають змогу повністю зануритися в гру, і відчути на собі усі тонкощі певної професії.

Розглянутий в роботі [2] симулятор системного адміністратора – це одна з небагатьох ігор (рис. 1.1), яка була створена виключно для спеціалістів в області системного адміністрування. Якщо гравець не належить до числа системних адміністраторів, то проходження даної гри є цілком неможливим. Основні елементи геймплею – схема мережі та стандартна консоль, які є головними помічниками будь-якого системного адміністратора.

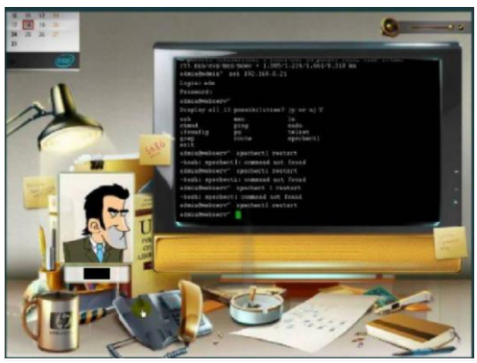

Рисунок 1.1 – Скріншот з гри «Симулятор системного адміністратора» [2]

Наведений симулятор системного адміністратора володіє обмеженим функціоналом і висвітлює певні аспекти даної професії. Розглянутий приклад не позиціонує себе, як засіб для навчання системних адміністраторів, але завдяки

атмосфері гри можна відчути себе справжнім системним адміністратором виконуючи певні задачі з цього напрямку.

Перевагою розробленого в даному дипломному проектуванні симулятора є те, що він забезпечує комплексну імітацію різноманітних аспектів професійної діяльності системного адміністратора, для візуального стилю використовується вдосконалена графіка а також реалізована взаємодія з платами Arduino. Основна мета даного симулятора полягає у навчанні користувачів різноманітним аспектам роботи системного адміністратора, включаючи управління мережею, конфігурацію серверів, забезпечення безпеки та розв'язання проблем.

1.2 Механіки жанру 2D симулятора

Жанр 2D симуляторів включає в себе різноманітні механіки, що відтворюють реальні та уявні аспекти дійсності [3].

Механіка управління персонажем є невід'ємною складовою в різних жанрах комп'ютерних ігор. Вона надає гравцеві можливість керування віртуальним персонажем в 2D або 3D середовищі, контролюючи його рухи, надає можливість взаємодії з об'єктами та виконання різних дій.

Системи взаємодії з оточенням надають гравцеві можливість взаємодіяти з різними об'єктами та персонажами в грі, виконуючи дії, такі як збирання ресурсів, взаємодія з іншими персонажами або об'єктами, а також використання різних інструментів.

Механіка прогресії та розвитку персонажа забезпечує тісну взаємодію гравця і персонажа, гравець має можливість удосконалювати свого персонажа чи вдосконалювати свої навички, що дозволяє йому виконувати більше завдань або отримувати нові можливості.

Механіка динамічної змінни погоди та часу забезпечує максимальне занурення в вілртуальний світ, в окремих симуляторах враховується погода та цикли дня та ночі, що може впливати на геймплей та взаємодію з оточенням,

також для більш реалістичної симуляції оточення використовується моделювання фізики, що імітує реалістичну або вигадану фізику для відтворення руху об’єктів в просторі, включаючи рух персонажа, предметів, транспорту тощо.

Також для загального наповнення гри використовується механіка завдання та виклики, завдяки даній механіці гравець може отримувати завдання або виклики, пов’язані з конкретними аспектами симуляції, які ставлять перед ним різні виклики та цілі.

Дані механіки можуть комбінуватися для створення унікального геймплею в різних симуляторах, що буде використано в ході кваліфікаційної роботи.

1.3 Основи піксель-арту

Піксель-арт – це стиль графічного дизайну, який використовує ретро-підхід до створення малюнків та спеціальних ефектів, використовуючи пікселі як основний елемент графіки [4]. У розробці комп’ютерних ігор піксель-арт використовується зазвичай для створення стилізованих, ретро або ностальгійних графічних образів (рис. 1.2).

Рисунок 1.2 – Приклад піксельної графіки, гра «Starbound» [5]

Графіка в піксель-арті зазвичай створюється на полотні низького розширення, наприклад, 8x8, 16x16, або 32x32 пікселів. Під час розробки гри,

дані зображення низького розширення можуть бути масштабовані для використання в віртуальному світі.

Кольорова палітра в піксель-арті зазвичай обмежена, іноді використовуються обмежені палітри або кольорові обмеження для реалізації певного стилю.

Завдяки обмеженню розширення, деталізація в піксель-арті теж здебільшого обмежена. Анімації можуть включати пересування пікселів або змінну кадрів для створення плавних рухів, анімація створюється здебільшого одним png-файлом з усіма рухами (рис. 1.3).

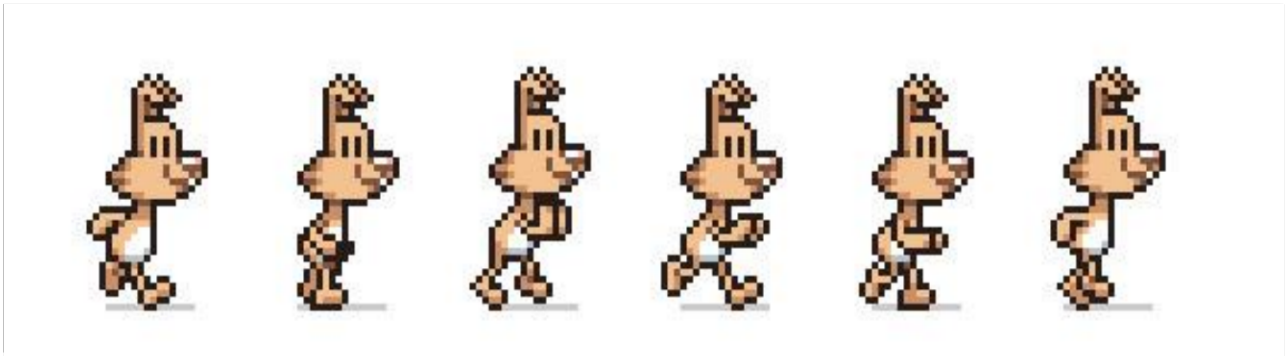

Рисунок 1.3 – Приклад розкадровки для піксельної анімації [6]

Більшість піксель-артів створюється вручну, користуючись графічними програмами, такими як Aseprite, GraphicsGale чи Photoshop.

Піксель-арт часто використовується для відтворення ретро-стилю, ностальгічного для старших гравців, або для надання особливого арт-стилю для гри.

В піксель-арті часто використовуються тайли – невеликі блоки графіки, які можна повторно використовувати для швидкої побудови віртувального світу (рис. 1.4). Це допомагає в зручності розробки та оптимізації.

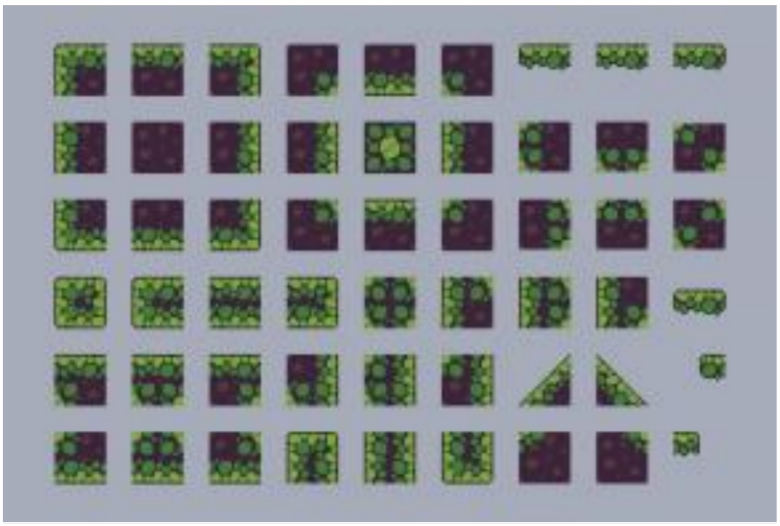

Рисунок 1.4 – Приклад тайл-сету [7]

Піксель-арт є ефективним стилем для створення графіки в комп’ютерних додатках, особливо в тих, які спрямовані на забезпечення ефекту ностальгії або для створення унікального стилю гри. Основна ідея піксель-арту – в обмеженні ресурсів для створення виразних та захоплюючих графічних образів.

Дані основні методики являються базовими методами створення піксельних зображень, в результаті дані методики будуть використані в ході розробки візуальної частини кваліфікаційної роботи.

1.4 Штучний інтелект в комп’ютерних застосунках

У комп’ютерних іграх штучний інтелект відіграє ключову роль у створенні непередбачуваних, адаптивних і захоплюючих геймплейних досвідів для гравців [8]. Дана технологія базується на використанні різних алгоритмів і підходів, зокрема машинного навчання, генетичних алгоритмів та еволюційних стратегій.

Машинне навчання використовується для створення моделей, які можуть навчатися з зростанням досвіду гравців. Наприклад, алгоритми нейронних мереж можуть адаптуватися до геймплейних ситуацій, навчаючись з реакцій гравців та результатів їх дій. Це дозволяє створювати ботів, які реагують на стратегії гравців і розвивають нові тактики.

Евристичні алгоритми ігрового штучного інтелекту використовуються в широкій розмаїтості в багатьох галузях усередині гри. Найочевидніше застосування ігрового штучного інтелекту проявляється в контролюванні неігрових персонажів, хоча скриптинг теж є дуже розповсюдженим способом контролю. Пошук шляху є іншим широко розповсюдженим застосуванням ігрового штучного інтелекту, він особливо проявляється в стратегіях реального часу. Пошук шляху є методом для визначення того, як неігровому персонажеві перейти з однієї точки на мапі до іншої: потрібно враховувати ландшафт, перешкоди тощо. Ігровий штучний інтелект також пов'язаний із динамічним ігровим балансуванням.

Концепція непередбачуваного штучного інтелекту була досліджена в таких іграх як Creatures, Black & White і Nintendogs і в таких предметних іграшках, як тамагочі. Персонажі у цих іграх мають здатність «навчатися» на діях, вчинених гравцем, і їхня поведінка змінюється відповідно. У той час, як ці рішення взяті з обмеженої множини можливих рішень, це дійсно часто дає бажану ілюзію інтелекту по іншу сторону екрана.

Узагальнюючи, використання наукових методів у створенні штучного інтелекту в комп'ютерних іграх дозволяє створювати більш інтелектуальних і адаптивних ботів, які можуть надати гравцям захоплюючі виклики і непередбачувані ігрові ситуації.

Отже, для зменшення навантаження на ігровий рушій, вирішено використовувати методику скриптингу для неігрових персонажів, оскільки використання штучного інтелекту досить ресурсозатратний процес, як для рушія так і для розробки гри з його використанням.

1.5 Вибір рушія

На сьогоднішній день існує велика кількість рушіїв, деякі з них орієнтовані для певної платформи або для певного ігрового стилю, для певних ігор компанії розробляють власні ігрові рушії які призначені лише для користування компанією, власне ті що наведені в таблиці 1.1 є умовно безкоштовними.

Таблиця 1.1 – Порівняння рушіїв

Критерії	Unity	Unreal Engine	Godot
Мова програмування	C#, UnityScript, Boo	C++, Blueprint	GDScript, C#
Графічний інтерфейс	Unity Editor	Unreal Editor	Godot Editor
Розширення	Asset Store, Unity Hub	Marketplace, Unreal Editor	Asset Library, Godot Asset Store
Анімація	Mecanim, Timeline	Animation Blueprint, Sequencer	Animation Player, AnimationTree
Фізика	Built-in фізика, PhysX	Built-in фізика, PhysX	2D та 3D фізика
Швидкість розробки	Висока, багато готових рішень	Висока, багато готових рішень	Висока, простий API, гнучкість
Спільнота	Велика спільнота користувачів	Велика спільнота користувачів	Швидко розвивається, зростає
Вартість	Безкоштовний для особистого використання	Безкоштовний для особистого використання	Безкоштовний та відкритий код

Джерело: [9]

Самими популярними рушіями на сьогоднішній день є Unity, Unreal Engine, Godot, їхня популярність пояснюється тим що вони являються повністю або частково безкоштовними, наприклад Unity безкоштовний до того моменту доки розробник отримав за власний проект менше 100 000$, якщо певний проект надасть розобникові суму більше ніж 100 000$ то розробник повинен віддавати певний відсоток прибутку.

Рушій Unity був вибраний для проекту в першу чергу через те, що він зручніший, ніж інші схожі рушії, даний рушій досить простий в освоєні а також він має широкий функціонал та високу швидкість розробки. Навколо даного рушія сформована велика спільнота користувачів, що дозволяє швидко знаходити рішення на форумах та в документації. Unity також має велику кількість готових рішень та розширень, таких як Asset Store, які значно полегшують процес розробки.

1.6 Взаємодія Unity з платою Arduino

Платформа Arduino є однією з найпопулярніших та найбільш доступних платформ для розробки пристроїв Інтернету речей (IoT) та інтерактивних електронних проектів. Її високий рівень доступності, широкий спектр сенсорів та модулів розширення, а також велика спільнота розробників роблять Arduino ідеальним вибором для створення 2D симулятора системного адміністратора в середовищі розробки Unity.

Створення симулятора системного адміністратора у середовищі Unity з використанням інструментів платформи Arduino відкриває широкі перспективи для розробки інтерактивних електронних систем та їх імітації віртуально. Платформа Arduino надає можливість підключення різноманітних сенсорів, актуаторів та інших електронних компонентів, що дозволяє створити відтворення реальних сценаріїв управління та моніторингу комп’ютерних систем.

Завдяки можливостям платформи Arduino і інтеграції з середовищем розробки Unity, розробники мають можливість моделювати різноманітні аспекти роботи системного адміністратора, від моніторингу та управління мережею до виявлення та виправлення несправностей у апаратному забезпеченні. Використання Arduino у поєднанні з Unity дозволяє створити віртуальне середовище, де користувач може ефективно взаємодіяти з електронними пристроями та відтворювати реальні сценарії роботи системного адміністратора.

У контексті розробки 2D симулятора системного адміністратора в Unity з використанням інструментів Arduino, важливо враховувати можливості і обмеження обох платформ, а також забезпечити оптимальну взаємодію між ними для досягнення бажаних результатів. Такий підхід дозволяє не лише створити цікавий та корисний продукт, але й розширити можливості навчання та дослідження в галузі системного адміністрування та IoT.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ДОДАТКУ

2.1 Алгоритм роботи додатку

Завданнями програмних розробок мультимедійних проектів є симуляція та моделювання систем, що надає можливість вивчення поведінки складних систем та відображення їх динаміки. У цьому контексті, створення симулятора системного адміністратора у середовищі розробки Unity відкриває можливості для дослідження та аналізу різноманітних аспектів адміністрування.

Даний проект призначений для розуміння та моделювання роботи системного адміністратора, який відповідає за підтримку та оптимізацію комп'ютерних систем. Описаний симулятор створений з метою надання віртуальної платформи для вивчення та вдосконалення навичок управління комп'ютерною інфраструктурою та реагування на непередбачені ситуації.

Моделювання завдань адміністратора полягає у тому, що симулятор включає в себе набір завдань, що характеризують різноманітні аспекти роботи системного адміністратора, такі як резервне копіювання даних, моніторинг мережі, апаратний ремонт тощо. Управління ресурсами передбачає можливість управляти різними ресурсами комп'ютерної системи, такими як потужність процесора, обсяг пам'яті, мережеві з'єднання тощо. Реалістична модель реагування заключається в тому, що програма передбачає реалістичні моделі реагування на виникнення непередбачених ситуацій, таких як відмова обладнання, атаки на мережу, витоки даних тощо. Аналіз та вдосконалення програми відбувається через гравця, який може аналізувати результати своєї діяльності та вдосконалювати свої навички управління системою на основі здобутого досвіду.

В додатку реалізовано систему завдань для гравця, алгоритм роботи даної системи виглядає, як на схемі рисунку 2.1.

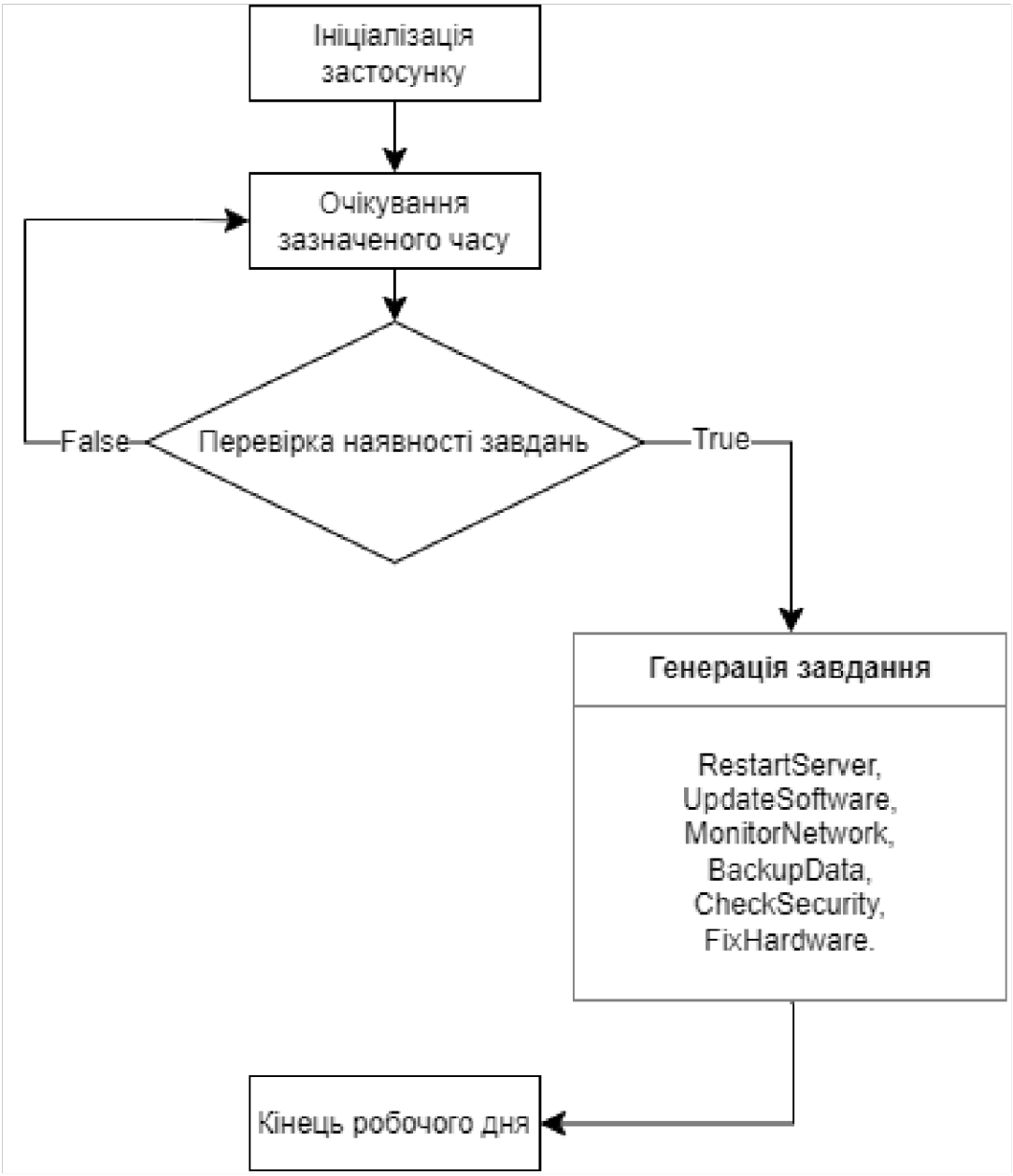

Рисунок 2.1 – Схема роботи ігрових завдань

Після ініціалізації ігрової симуляції розпочинається відлік ігрового часу, який імітує робочий день системного адміністратора, після початку даного відліку виконується постійна перевірка на заздалегідь зазначений час, після настання певного ігрового часу гравцеві приходить на віртуальну пошту завдання, яке випадковим чином вибирається з створеного списку завдань. Гравцеві потрібно виконати поставлене завдання після чого відбувається змінна ігрового дня.

2.2 Способи інтеграції IoT та взаємодія платформи Arduino з симулятором

Інтернет речей (IoT) та платформа Arduino представляють собою важливі компоненти в екосистемі сучасних технологій, які знаходять широке застосування в різних сферах, включаючи системи автоматизації, моніторингу та управління. У контексті розробки 2D симулятора системного адміністратора в середовищі розробки Unity, інтеграція IoT та платформи Arduino може відкривати нові можливості для створення більш реалістичного та інтерактивного досвіду.

Одним з основних методів інтеграції IoT у симулятор може бути використання спеціалізованих плагінів або SDK, які забезпечують зв’язок між програмною частиною додатку Unity та реальними пристроями IoT. Це дозволяє взаємодіяти з датчиками, засобами управління та іншими пристроями, що підтримують протоколи зв’язку IoT, такі як MQTT, CoAP тощо.

Стосовно взаємодії платформи Arduino з симулятором в середовищі Unity, можливість інтеграції полягає в створенні віртуальних моделей пристроїв Arduino, які симулюють реальне поведінку таких пристроїв. Це може бути досягнуто за допомогою розробки спеціалізованих скриптів в Unity, що моделюють функціонал пристроїв Arduino, а також застосування плагінів або бібліотек, які забезпечують взаємодію з Arduino через USB-підключення або мережеві інтерфейси, такі як Ethernet або Wi-Fi.

Однак, варто враховувати те, що взаємодія з реальними пристроями IoT та Arduino може потребувати додаткового обладнання та програмного забезпечення, а також враховувати особливості платформи та обмеження в зв’язку з характером роботи симулятора та можливостями Unity.

В ході розробки даного проекту використовується реальна плата Arduino, для її інтеграції в розробку, використано стандартні бібліотеки Unity, а саме

System.IO.Ports, дана бібліотека відповідає за надсилання даних на зазначені серійні порти комп’ютера (рис. 2.2).

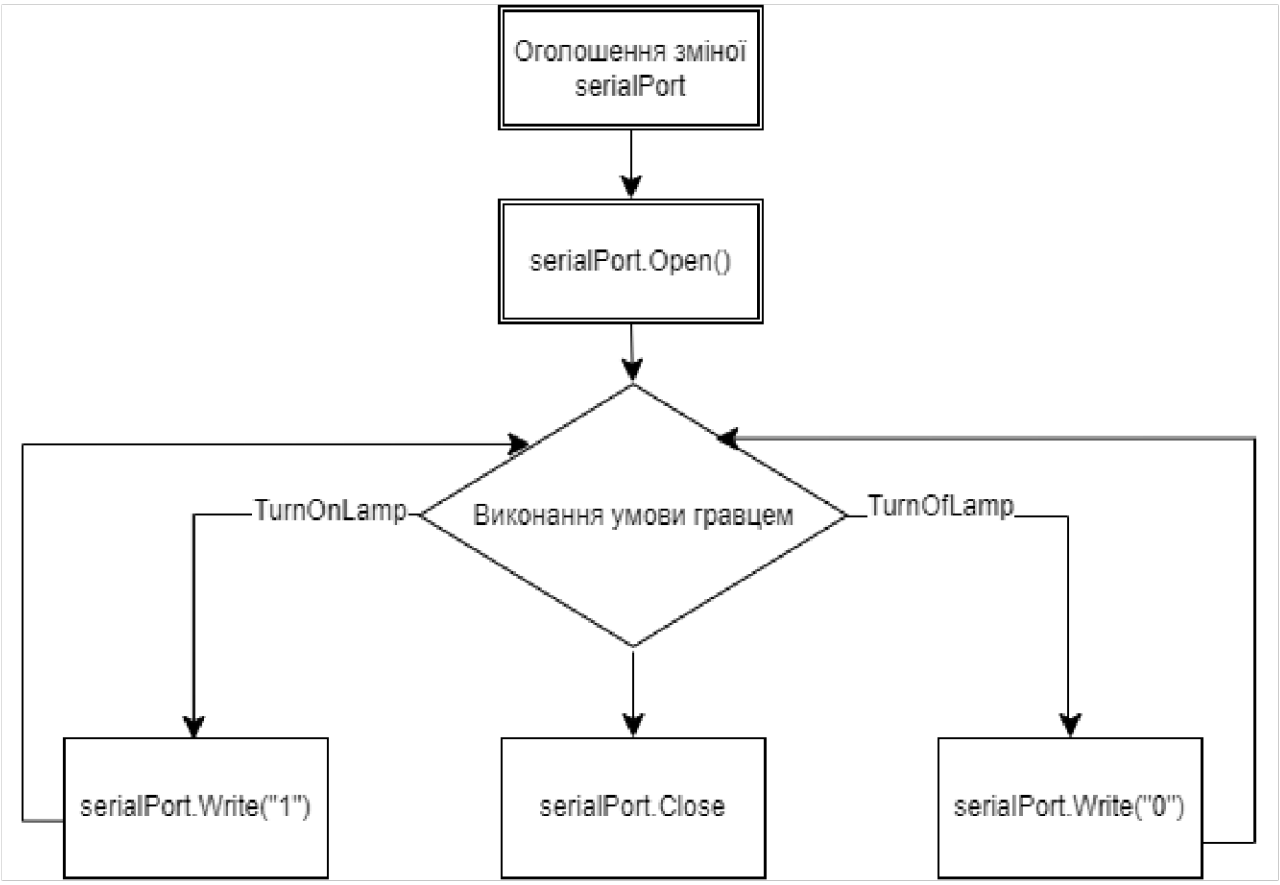

Рисунок 2.2 – Алгоритм взаємодії з платою Arduino

Алгоритм взаємодії з платформою Arduino базується на діях гравця, оскільки для того щоб тригери в грі спрацювали потрібно щоб гравець зайшов в них персонажем, після запуску ігрової симуляції відбувається оголошення змінної serialPort яка приймає значення порту, наприклад «COM3» і значення швидкості передачі даних, після цього відбувається відкриття серійного порту для середовища розробки, тоді після виконання гравцем умови відбувається надсилання даних на порт, після чого виконується перевірка даних платою Arduino.

2.3 Інструменти платформи Arduino

Arduino є апаратною обчислювальною платформою для аматорського конструювання, основними компонентами якої є плата мікроконтролера з елементами вводу/виводу та середовищем розробки Processing/Wiring на мові програмування, що є спрощеною підмножиною C/C++ [10]. Arduino може використовуватися як для створення автономних інтерактивних об'єктів, так і підключатися до програмного забезпечення, яке виконується на комп'ютері (наприклад: Processing, Adobe Flash, Max/MSP, Pure Data, SuperCollider). Інформація про плату (малюнок друкованої плати, специфікації елементів, програмне забезпечення) знаходяться у відкритому доступі і можуть бути використані тими, хто воліє створювати плати власноруч.

Взаємодія Arduino з середовищем розробки Unity відбувається через вбудовану бібліотеку System.IO.Ports. У скрипті Unity прописується відкриття серійного порта, до якого підключено плату Arduino і прописується швидкість передачі даних, після чого через скрипт відправляються дані, які активують умову, запрограмовану в плату. Наприклад, при надсиланні з скрипта значення 1 на визначений порт буде активуватись перевірка умови if на платі, після чого буде виконано запрограмовану дію.

Для даного проекту була обрана плата Arduino UNO R3 (рис. 2.3), оскільки дана плата має всі необхідні порти підключення і є досить дешевою.

Рисунок 2.3 – Плата Arduino UNO R3

Також для підключення всіх компонентів до плати придбано:

– Макетну безпайну плату МВ-102 400 точок (рис. 2.4);

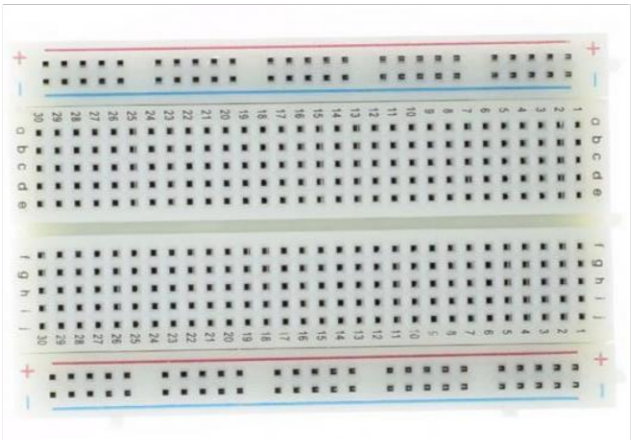

Рисунок 2.4 – Макетна плата МВ-102

– Набір перемичок для Arduino (рис. 2.5);

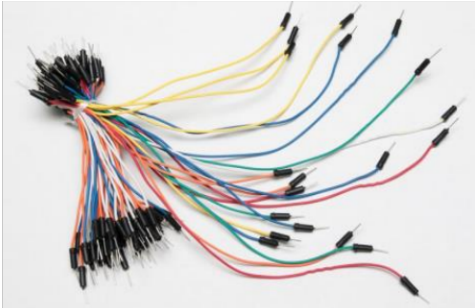

Рисунок 2.5 – Набір перемичок

– Світлодіод 3мм червоний (рис. 2.6) – номінал даного діода 20 мА;

Рисунок 2.6 – Світлодіод 3мм

– Резистор 220 Ом – даний резистор був обраний згідно обраного світлодіода, оскільки його номінал 20 мА.

Такі компоненти дозволяють реалізувати інтеграцію IoT з програмним середовищем розробки Unity, а саме індикацію ігрових подій а також дозволять ознайомитись з базовими можливостями плати Arduino UNO R3.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА РОЗРОБКА ДОДАТКУ

3.1 Створення головного меню

Елемент додатку «Головне меню» ініціалізується запуску додатку. З головного меню відбувається налаштування гри, запуск ігрових рівнів і відбувається правильне закриття додатку без помилок, які можуть виникнути при аварійному завершені гри або завершені роботи через комбінацію «Alt+F4». Завдяки головному меню гравець може ознайомитись з загальною тематикою гри, стилем оформлення і дізнатися детальну інформацію ігрових предметів якщо це передбачено розробником.

Зазвичай, головне меню складається з заднього фону, клавіш «Start», «Option», «Quit», в масштабних проектах добавляють більше функціоналу для головного меню, але в даному проекті буде реалізовано лише три клавіші для взаємодії з головним меню, а саме: Start, Load, Quit.

Розпочато проект з розробки графіки головного меню, для заднього фону (рис. 3.1), на зображені заднього фону зображено сервер, на якому розміщені клавіші для виконання основної функції головного меню, лого університету і комп'ютерний стіл з комп'ютером і стільцем для відображення основної тематики і стилю гри.

Рисунок 3.1 – Зображення для заднього фону головного меню

Також, розроблено тайл-сет (рис. 3.2) з загальними клавішами які використовуються при розробці головного меню і меню паузи.

Рисунок 3.2 – Тайл-сет для головного меню і меню паузи

Для задання логіки клавішам реалізовано скрипт з основним функціоналом клавіш, його розміщено на головний UI елемент, «Canvas».

Для реалізації функціоналу клавіші Start, створено метод PlayGame() (лістинг 3.1) в скрипті MainMenu, в даному методі зазначено запуск конкретної сцени через SceneManager, тут прописано назву сцени та індекс рівня з BuildSeting.

Лістинг 3.1 – Метод для запуску головної сцени PlayGame()

```
27    public void PlayGame(){
28        //checkStart = 1;
29        SceneManager.LoadScene("Work");
30    }
```

Кінець лістингу 3.1

Для реалізації клавіші завантаження гри створено метод LoadGame() (лістинг 3.2) в скрипті MainMenu, а для того щоб пов’язати клавішу Load з клавішею Save, створено скрипт SaveLoadManager, який відповідає за підготовку ігрових даних до збереження їх в файл з розширенням «.dat».

Лістинг 3.2 – Метод для завантаження гри LoadGame()

```
public void LoadGame(){
    // Збереження гри
    GameData gameData = new GameData();
    // Заповнення даними gameData
    SaveLoadManager.SaveGame(gameData);

    // Завантаження гри
    GameData loadedData = SaveLoadManager.LoadGame();
    if (loadedData != null)
    {
        // Обробка завантажених даних
        Debug.Log("Loaded game data: " + loadedData.ToString());
        // Наприклад, встановлення позиції гравця збереженим значенням
        player.transform.position = loadedData.playerPosition;
    }
    else
    {
        // Обробка випадку, коли немає збереженого файлу
        Debug.LogWarning("No saved game data found. Starting a new game.");
        // Наприклад, початкова ініціалізація гри
        SceneManager.LoadScene("Work");
    }

}
```

Кінець лістингу 3.2

Для реалізації клавіші виходу з гри створено метод ExitGame() (лістинг 3.3). В даному методі використовується загальна функція для керування додатком Application і метод Quit створений рушієм. Для того, щоб відображати роботу клавіші в редакторі, додано вивід інформації в консоль. Це реалізовано. оскільки працездатність клавіші виходу з додатку можна побачити лише в повністю скомпільованому застосунку.

Лістинг 3.3 – Метод для завершення роботи застосунку ExitGame()

```
57      public void ExitGame(){
58
59          Debug.Log("Quit");
60
61          Application.Quit();
62      }
63  }
```

Кінець лістингу 3.3

Для того щоб клавіші виконували свою функцію, на UI елементі Canvas створено дочірній UI елемент Button. Це зроблено для того, щоб зручніше використовувати редактор клавіші меню. Такий елемент є дочірнім елементом пустого об’єкта, формуючи групу елементів. В роботі задано йому в полі Source Image потрібний спрайт, і в полі OnClick() додати елемент, на якому розміщений скрипт з прописаними окремими методами для клавіш (рис. 3.3).

Рисунок 3.3 – Створення функціоналу клавіш головного меню

В даному застосунку не передбачено клавіші налаштувань, оскільки основний функціонал і налаштування запрограмовані заздалегідь в скриптах, а також, опираючись на загальне споживання системних ресурсів системи, відпадає необхідність реалізації елемента налаштувань даного додатку.

3.2 Розробка графіки для додатку

Графіка відіграє значну роль на загальний ігровий досвід. Комп'ютерна графіка займає третє місце після ігрових механік і сюжету гри, але іноді розробники роблять максимальний опір на графіку ігноруючи інші два аспекти, від яких на нашу думку відбувається більш ефективне занурення в гру. Якщо ці два аспекти перебувають на середньому рівні ігрова графіка надасть позитивні враження від ігрового процесу.

Отже, для графічного оформлення використано стороні тайл-сети, які в результаті були перетворено в програмі Photoshop для того, щоб вони відповідали потребам проекту, тобто, були взяті спрайти з інтернету і перемальовані під необхідні розміри з додаванням певних елементів.

Тайл-сет ігрового персонажа взято з gif-файла, в якому показано покадрово зображення переміщення персонажа. Після конвертації gif-файла в велику кількість JPEG зображень, видалено зайві фрагменти анімації, і перетворено в загальний тайл-сет з необхідними кадрами анімації (рис. 3.4).

Рисунок 3.4 – Загальний тайл-сет анімацій персонажа

Після створення анімацій для персонажа, створено загальний тайл-сет з графікою рівня (рис. 3.5), а саме: стіни в декількох варіаціях, підлога, дві варіації персональних комп’ютерів, сервер, принтер, пустий стіл, настільна лампа в двох станах, корзина і стілець.

Всі ці елементи були перемальовані з різних наборів і адаптовані під потреби проекту.

Рисунок 3.5 – Загальний тайл-сет з графікою рівня

Фон для меню паузи (рис. 3.6) адаптовано для загального використання, оскільки це дозволить створити відповідну атмосферу.

Рисунок 3.6 – Універсальний фон для меню паузи

Даний фон використовується також як імітація віртуального дисплея для персонального комп’ютера.

Для того, щоб реалізувати максимальну симуляцію, створено піктограми для програм на персональному комп’ютері (рис. 3.7), всі піктограми були розміщені на загальному тайл-сеті з інтерфейсом віртуального комп’ютера.

Рисунок 3.7 – Тайл-сет з піктограмами програм

Також для створення користувацького інтерфейсу створено тайл-сет з зображеннями інтерфейсу програм (рис. 3.8). Даний тайл-сет використовується в якості шаблону та заднього фону для віртуальних програм.

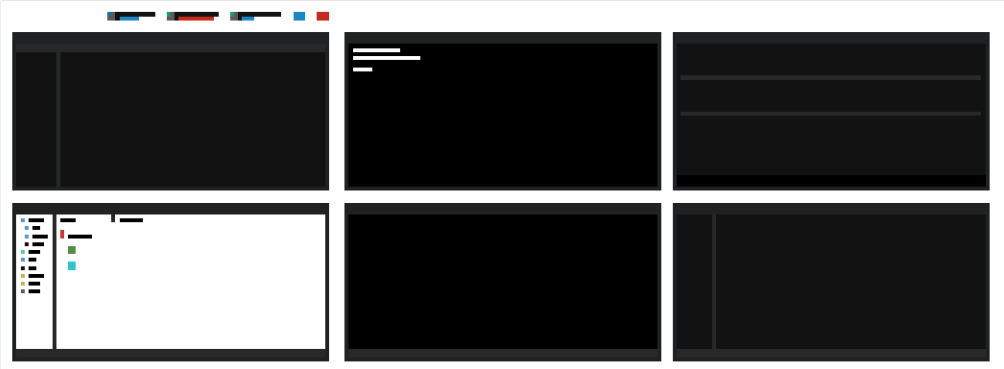

Рисунок 3.8 – Тайл-сет з зображеннями інтерфейсу програм

Отже, дані фрагменти графіки формують загальний вигляд застосунку, і дозволяють створити загальний користувацький інтерфейс, і індикацію ігрових подій.

Завдяки даним компонентам формується атмосфера і загальна стилістика в комп’ютерній грі, яка забезпечує максимальне занурення в ігровий процес.

3.3 Розробка основних механік

Розробка основних механік є ключовим етапом створення 2D симулятора системного адміністратора. Цей процес включає створення функціональності, яка дозволяє гравцям взаємодіяти з віртуальним світом, виконувати завдання та отримувати зворотний зв’язок від гри.

Ключовою механікою в комп’ютерних іграх є механіка руху персонажа, вона дозволяє користовачу взаємодіяти з віртуальним світом. Здебільшого реалізують стандартне пересування ігрового об’єкту Player стрілками або клавішами «W, A, S, D» на клавіатурі. Оскільки Unity є ігровим рушієм, для зручності розробки використовуються стандартні команди, які дозволяють прописувати керування завдяки загальним направленням, так як зображено на лістингу 3.4, в даному коді з скрипта Player реалізовано пересування персонажа по віртувальному полі.

Лістинг 3.4 – Код реалізації руху з скрипта Player

```
0 references
private void Update(){
    movement.x = Input.GetAxisRaw("Horizontal");
    movement.y = Input.GetAxisRaw("Vertical");

    animator.SetFloat("Horizontal", movement.x);
    animator.SetFloat("Vertical", movement.y);
    animator.SetFloat("Speed", movement.sqrMagnitude);
}

0 references
private void FixedUpdate() {
    rb.MovePosition(rb.position + movement * movingSpeed * Time.fixedDeltaTime);
}
```

Кінець лістингу 3.4

Після реалізації механіки руху персонажа створено скрипт «CameraFollow» для того щоб ігрова камера пересувалась за персонажем, створено обмеження пересування ігрової камери для того щоб камера не виходила за межі ігрового рівня, код реалізації зображено на лістингу 3.5.

Лістинг 3.5 – Кодова реалізація пересування і обмеження ігрової камери

```
void Update()
{
    if (target != null)
    {
        // Визначаємо цільову позицію для камери
        Vector3 targetPosition = new Vector3(target.position.x, target.position.y, transform.position.z);

        // Плавне переміщення камери до цільової позиції
        transform.position = Vector3.SmoothDamp(transform.position, targetPosition, ref velocity, smoothTime);
    }

    transform.position = new Vector3
        (
        Math.Clamp(transform.position.x, leftLimit,rightLimit),
        Math.Clamp(transform.position.y, bottomLimit,upperLimit),
        transform.position.z
        );
}

0 references
private void OnDrawGizmos() {
    Gizmos.color = Color.red;
    Gizmos.DrawLine(new Vector2(leftLimit, upperLimit), new Vector2(rightLimit, upperLimit));
    Gizmos.DrawLine(new Vector2(leftLimit, bottomLimit), new Vector2(rightLimit, bottomLimit));
    Gizmos.DrawLine(new Vector2(leftLimit, upperLimit), new Vector2(leftLimit, bottomLimit));
    Gizmos.DrawLine(new Vector2(rightLimit, upperLimit), new Vector2(rightLimit, bottomLimit));
}
```

Кінець лістингу 3.5

Метод OnDrawGizmos призначений для відображення меж пересування камери, дані для даних обмежень задаються через інспектор Unity. Для того щоб змінні відображалися в інспекторі (рис. 3.9), потрібно перед типом даних в скрипті прописати [SerializeField].

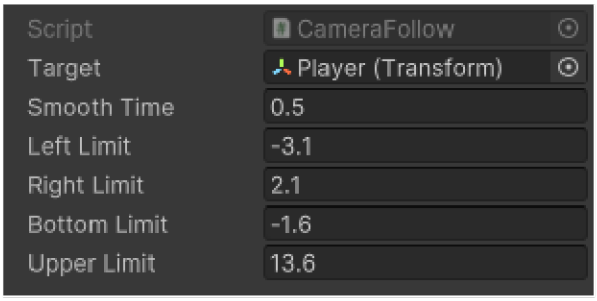

Рисунок 3.9 – Відображення змінних в інспекторі

Також створено скрипт «GameManager» в якому описується взаємодія різних компонентів, і основна логіка гри. Для взаємодії з платформою Arduino, в скрипті «GameManager» виконується ініціалізація підключення платформи, для цього використовується код зображений на лістингу 3.6.

Лістинг 3.6 – Код для ініціалізації платформи Arduino

```
public static SerialPort serialPort = new SerialPort("COM3", 9600); // Порт з'єднання та швидкість передачі даних

0 references
public void Start()
{
    Time.timeScale = 1f;

    try
    {
        serialPort.Open(); // Відкрити порт

        if (serialPort.IsOpen)
        {
            Debug.Log("Порт COM3 успішно відкритий.");
        }
    }
    catch (IOException e)
    {
        Debug.LogError("Помилка підключення до порту COM3: " + e.Message);
    }
    catch (UnauthorizedAccessException e)
    {
        Debug.LogError("Доступ до порту COM3 заборонений: " + e.Message);
    }
    catch (System.Exception e)
    {
        Debug.LogError("Невідома помилка: " + e.Message);
    }
```

Кінець лістингу 3.6

Насамперед, виконується ініціалізація змінної serialPort і запис в неї даних про визначений порт підключення, і швидкість передачі даних. Після ініціалізації змінної виконується метод Start, в якому використовується структура try/catch для запобігання аварійного завершення роботи зостосунку. Тут виконується перевірка на можливі помилки, пов’язані з підключенням, або з доступом до цільового порту, після чого в консоль виводиться інформація про помилку.

Після вдалого відкритя порту, в скриплі «Lamp» виконується перевірка з умовою «OnTriggerEnter2D» і «OnTriggerExit2D», що зображено на лістингу 3.7.

Лістинг 3.7 – Умова «OnTriggerEnter2D» і «OnTriggerExit2D» скрипта Lamp

```
private void OnTriggerEnter2D(Collider2D col) {
    if(col.gameObject.tag == "Player"){
        if (GameManager.serialPort.IsOpen)
        {
            TurnOnLamp();
        }
    }
}

0 references
private void OnTriggerExit2D(Collider2D col) {
    if(col.gameObject.tag == "Player"){
        if (GameManager.serialPort.IsOpen)
        {
            TurnOffLamp();
        }
    }
}
```

Кінець лістингу 3.7

Дані умови виконують перевірку на взаємодію колайдера ігрового персонажа з колайдером настільної лампи (рис. 3.10). Якщо колайдер персонажа перетинає межу колайдера настільної лампи, то виконується ввімкнення світлодіода на платі Arduino, і змінна спрайта в додатку. Для імітації ввімкнення лампи, при виході за межі колайдера настільної лампи, виконується вимкнення світлодіоду на платі і змінна спрайта в додатку.

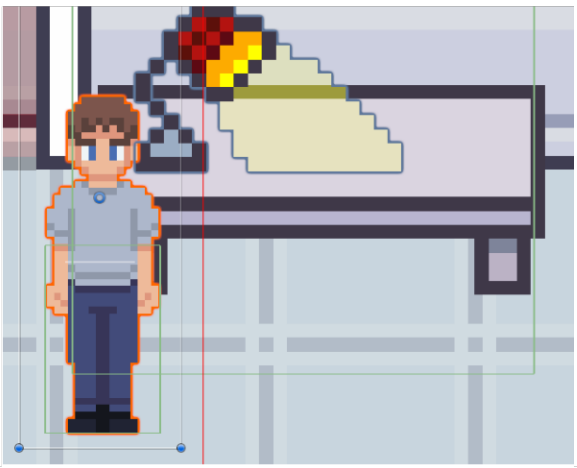

Рисунок 3.10 – Взаємодія колайдерів персонажа і настільної лампи

Під час виконання умови в скрипті «Lamp», виконується виклик методів з скрипта «GameManager», для надсилання даних на плату, код подано на лістингу 3.8.

Лістинг 3.8 – Код для надсилання і отримання даних з плати Arduino

```
public static void TurnOnLampGameManager()
{
    // Відправляємо дані до Arduino
    serialPort.Write("1"); // Наприклад, відправляємо "1" для увімкнення

    // Отримуємо дані з Arduino
    string data = serialPort.ReadLine();
    Debug.Log("Data from Arduino: " + data);
}

// Метод для вимкнення лампочки
1 reference
public static void TurnOffLampGameManager()
{
    // Відправляємо дані до Arduino
    serialPort.Write("0"); // Відправляємо "0" для вимкнення лампочки

    // Отримуємо дані з Arduino
    string data = serialPort.ReadLine();
    Debug.Log("Data from Arduino: " + data);
}
```

Кінець лістингу 3.8

Після того, як плата отримує дані з гри, виконується код, поданий на лістингу 3.9, запрограмований в плату.

На платформі Arduino постійно виконується перевірка швидкості вхідних даних, і зчитування даних з порту. Після отримання даних виконується перевірка умови згідно вхідних даних. Якщо в змінну lampChar записано 1, виконується цифровий запис на lampPin, який дорівнює 13. Оновлюється значення HIGH, також виконується надсилання підтвердження увімкнення «Lamp On»: якщо записано 0 тоді записується LOW, і надсилається підтвердження «Lamp Off». Ці

дані відображаються в консолі Unity. Таким чином, розробник може контролювати стан лампи в режимі реального часу. Це забезпечує зручність і оперативність у керуванні пристроєм.

Лістинг 3.9 – Код для опрацювання даних з гри

```
int lampPin = 13;

void setup() {

  pinMode(lampPin, OUTPUT); // Встановлюємо пін як вихід
  Serial.begin(9600); // Ініціалізуємо зв'язок через SerialPort з швидкістю 9600

}

void loop() {

  if (Serial.available() > 0) { // Якщо доступні дані через SerialPort
    //noticePinChar();
    //lampPinChar();
    char lampChar = Serial.read(); // Зчитуємо отримані дані

    if (lampChar == '1') { // Якщо отримано "1"
      digitalWrite(lampPin, HIGH); // Увімкнути лампочку
      Serial.println("Lamp On"); // Надіслати підтвердження увімкнення
    }
    else if (lampChar == '0') { // Якщо отримано "0"
      digitalWrite(lampPin, LOW); // Вимкнути лампочку
      Serial.println("Lamp Off"); // Надіслати підтвердження вимкнення
    }
  }
}
```

Кінець лістингу 3.9

В скрипті «GameManager» реалізовано відображення ігрового часу. Для розрахунку поточного часу створено метод «UpdateGameTime» (лістинг 3.10), також в даному методі виконується виклик ігрових завдань при досягнені певного часу. Крім того, цей метод забезпечує синхронізацію ігрових подій з реальним часом. Таким чином, гравці отримують плавний та безперервний ігровий досвід.

Лістинг 3.10 – Метод «UpdateGameTime» для розрахунку ігрового часу

```
public void UpdateGameTime()
{
    // Розрахунок поточного часу гри з урахуванням швидкості
    currentGameTime += Time.deltaTime * (60.0f / dayDurationInMinutes) * clockSpeedMultiplier;

    if(currentGameTime > 490.0f && currentGameTime < 490.3f){
        enableTask = true;
    }else if(currentGameTime > 490.9f){
        enableTask = false;
    }

    // Перевірка для визначення нового дня
    if (currentGameTime >= 1440.0f) // 24 години у хвилинах
    {
        currentGameTime = 480.0f;
    }

    if(currentGameTime >= 960.0f){
        Debug.Log("End work day");

        currentGameTime = 480.0f;
    }
}
```

Кінець лістингу 3.10

Даний метод містить розрахунок поточного ігрового часу, він вираховується і записується в змінну currentGameTime, після чого, для відображення на графічному інтерфейсі ця змінна використовується в методі «UpdateClockDisplay» (лістинг 3.11).

Лістинг 3.11 – Метод «UpdateClockDisplay» для відображення годинника

```
public void UpdateClockDisplay()
{
    // Оновлення відображення годинника
    int hours = Mathf.FloorToInt(currentGameTime / 60);
    int minutes = Mathf.FloorToInt(currentGameTime % 60);

    string clockText = string.Format("{0:00}:{1:00}", hours, minutes);

    if (clockDisplay != null)
    {
        clockDisplay.text = clockText;
    }
}
```

Кінець лістингу 3.11

Дані методи викликаються в методі «Update» (лістинг 3.12) для того, щоб відображати ігровий час постійно. Завдяки методу «Update» виконується велика кількість методів, які створені для перевірки ігрових подій.

Лістинг 3.12 – Метод «Update» в скрипті «GameManager»

```
public void Update()
{
    UpdateGameTime();
    UpdateClockDisplay();

    if(enableTask == true){
        TaskManager();
    }
}
```

Кінець лістингу 3.12

Також в методі «Update» який запропоновано у лістингу 3.12, виконується виклик методу «TaskManager» (лістинг 3.13). Даний метод випадково генерує завдання з списку завдань.

Лістинг 3.13 – Метод «TaskManager»

```
public static void TaskManager(){

   if(availableTask == false){
      // Генеруємо випадкове завдання
      UnityTasks.TaskType randomTask = (UnityTasks.TaskType)UnityEngine.Random.Range(0, (int)UnityTasks.TaskType.FixHardware + 1);
      //UnityTasks.TaskType randomTask = (UnityTasks.TaskType)(int)UnityTasks.TaskType.FixHardware;

      unityTasks.ExecuteTask(randomTask);
   }
}
```

Кінець лістингу 3.13

Для списку завдань, створено скрипт з назвою «UnityTask». У даному скрипті розміщено перелік можливих завдань (лістинг 3.14).

Лістинг 3.14 – Скрипт «UnityTask»

```
public enum TaskType
{
    1 reference
    RestartServer,
    1 reference
    UpdateSoftware,
    1 reference
    MonitorNetwork,
    1 reference
    BackupData,
    0 references
    CheckSecurity,
    1 reference
    FixHardware
}

// Метод для виконання завдання
0 references
public void ExecuteTask(TaskType task)
{
    switch (task)
    {
        case TaskType.RestartServer:
            GameManager.availableTask = true;
            RestartServer();
            break;
        case TaskType.UpdateSoftware:
            GameManager.availableTask = true;
            UpdateSoftware();
            break;
        case TaskType.MonitorNetwork:
            GameManager.availableTask = true;
            MonitorNetwork();
            break;
        case TaskType.BackupData:
            GameManager.availableTask = true;
            BackupData();
            break;
        case TaskType.FixHardware:
            GameManager.availableTask = true;
            FixHardware();
            break;
        default:
            GameManager.availableTask = false;
            Debug.LogError("Unknown task type");
            break;
    }
}
```

Кінець лістингу 3.14

У скрипті «UnityTask» створено метод для виконання завдань «ExecuteTask», код продемонстровано на лістингу 3.14.

Метод «ExecuteTask» призначений для виклику завдань, використовуючи умову switch. Після виклику певного завдяня з списку, виконується присвоєння змінній availableTask значення true для того, щоб відобразити наявність завдань в імітованій поштовій скринці.

Ці основні ігрові механіки надають базову функціональність для 2D симулятора системного адміністратора в Unity. Кожна з них може бути розширена та вдосконалена відповідно до потреб гравців, забезпечуючи реалістичну ігрову симуляцію та навчальну платформу для користувачів.

3.4 Тестування та запуск додатку

Процедура тестування та запуску додатку Unity на Windows 10 та 11 включає кілька етапів: підготовку проекту, збірку додатку, перенесення на тестові машини, виконання тестів та оцінку результатів.

Розпочинали з підготовки проекту. Для цього виконано налаштування платформи збірки. У Build Setting у вікні (рис. 3.11) вибрано цільову платформу, архітектуру і додано ігрові сцени.

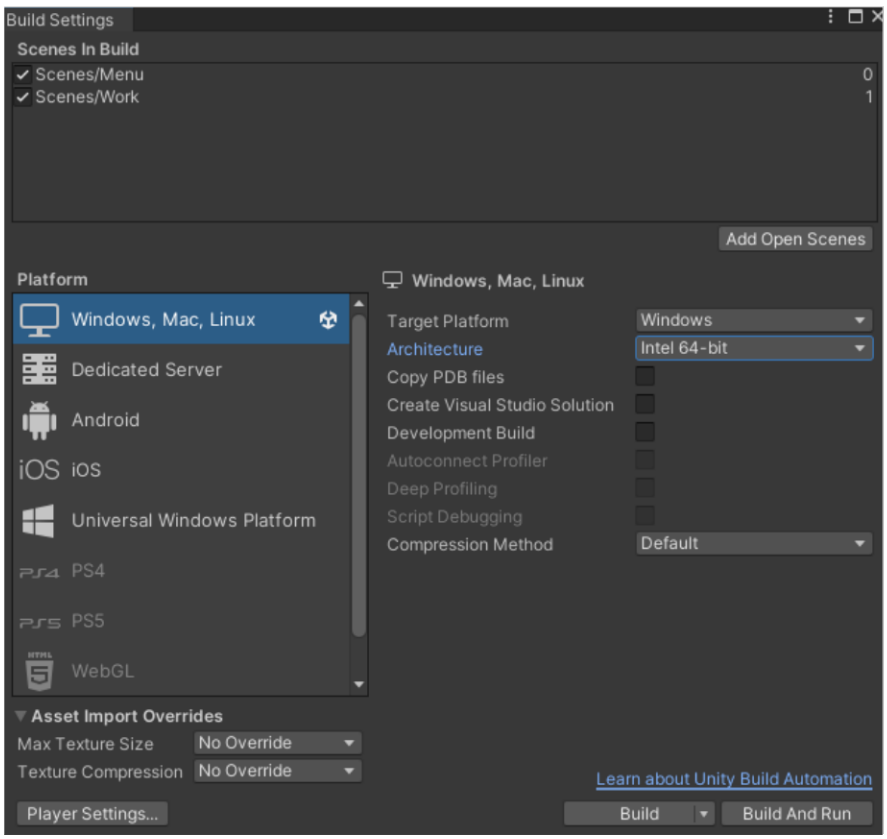

Рисунок 3.11 – Меню Build Setting

Після налаштування платформи збірки, перейшли до налаштувань опцій збірки, для цього з меню Build Setting перейшли в Player Settings (рис. 3.12), в даному меню налаштовано параметри екрану та роздільної здатності, встановлено налаштування графіки.

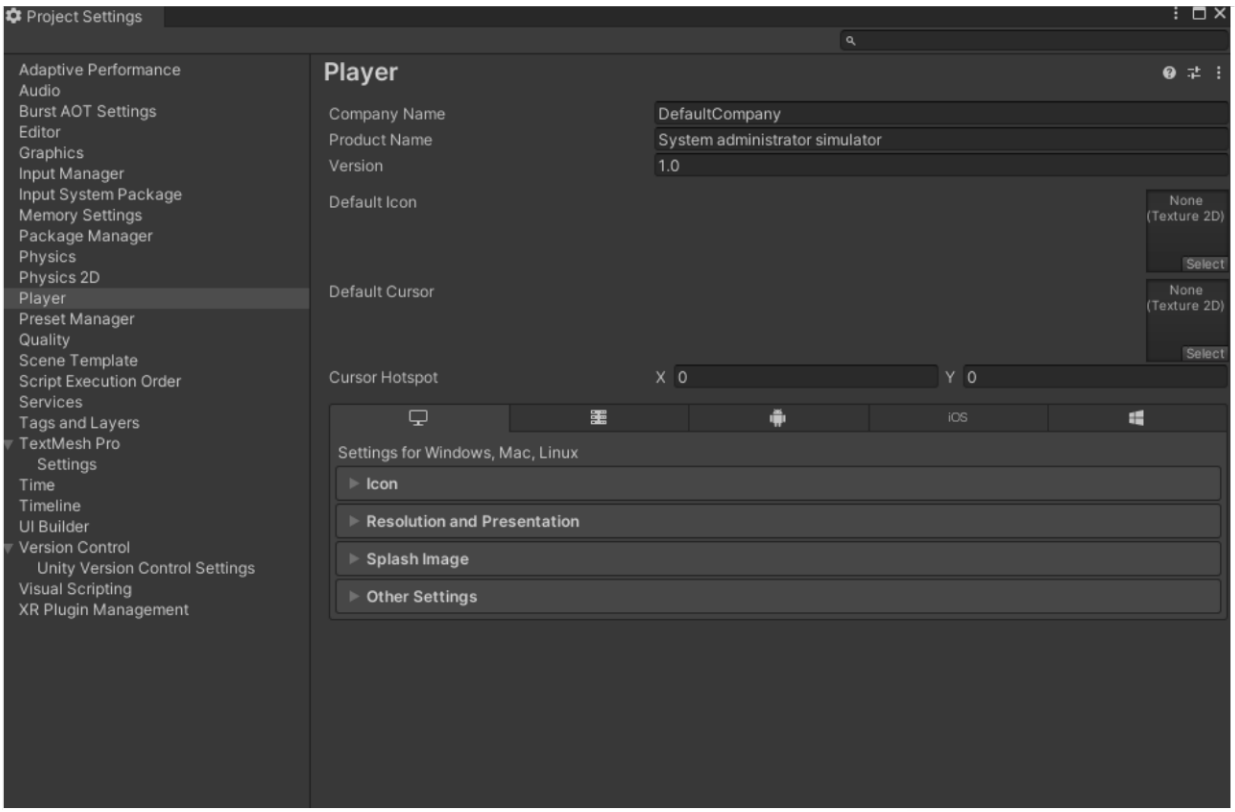

Рисунок 3.12 – Меню Player Settings

Завершивши налаштування перейшли до збірки додатку, для цього в меню Build Setting через Build і вибрати шлях збереження виконавчого файлу.

Збірка проекту пройшла коректно, помилок виявлено не було. Наступним етапом виконано тестування додатку на двох популярних операційних системах, Windows 10 і Windows 11. Ця процедура забезпечує комплексний підхід до тестування додатку на різних платформах, що дозволяє виявити потенційні проблеми і забезпечити високу якість продукту.

Для проведення тестування сформовано так званий «Check list», в нього увійшли наступні критерії тестування:

- перевірка коректності запуску додатку;

- перевірка роботи основних механік;
- перевірка коректності відображення користувацького інтерфейсу;
- перевірка взаємодії з платформою Arduino.

Згідно результату тестування сформовано в таблицю 3.1.

Таблиця 3.1 – Оцінка результатів тестування

Параметри	Windows 10	Windows 11
Запуск додатку	Додаток запускається коректно	Додаток запускається коректно
Робота основних механік	Усі механіки працюють коректно	Усі механіки працюють коректно
Користувацький інтерфейс	Користувацький інтерфейс відображається коректно	Користувацький інтерфейс відображається коректно
Взаємодія з платформою Arduino	Взаємодіє коректно	Взаємодіє коректно

Процедура тестування показала, що додаток «Симулятор системного адміністратора», зібраний в Unity, коректно працює як на Windows 10, так і на Windows 11. Всі основні функції додатку виконуються без помилок, інтерфейс користувача відображається належним чином, і взаємодія з обладнанням через COM-порт працює належним чином. Це свідчить про високий рівень сумісності додатку з обома версіями операційної системи.

ВИСНОВКИ

Додаток «Симулятор системного адміністратора» надає можливість гравцеві поглибитись в технічний світ адміністрування, також він є важливим інструментом для навчання та тренування нових поколінь системних адміністраторів. Завдяки інтеграції Arduino в гру, відкривається можливість навчання гравців в галузі електроніки, схемотехніки, конструювання тощо.

В результаті виконання кваліфікаційної роботи:

– проведено аналітичний огляд питань формування мультимедійних додатків і визначено елементи професійної діяльності системного адміністратора які впроваджено в розробку: обслуговування сервера, оновлення програмного забезпечення, адміністрування мереж, виконання резервного копіювання, перевірка систем безпеки, заміна комплектуючих комп’ютера;

– проведено огляд популярних рішень з питань застосування елементів штучного інтелекту та визначено доцільність їх впровадження у додаток, зокрема вирішено використовувати для зменшення навантаження на рушій методику скриптингу, оскільки використання штучного інтелекту досить ресурсозатратний процес;

– досліджено рушії, порівняно їх характеристики, та виявлено, що рушій Unity вибраний для проекту в першу чергу через те, що він зручніший, ніж інші схожі рушії, даний рушій досить простий в освоєнні та має широкий функціонал і високу швидкість розробки;

– складено алгоритм роботи додатку, згенеровано список завдань які є базовими для загального розуміння професії системного адміністратора: RestartServer, UpdateSoftware, MonitorNetwork, BackupData, CheckSecurity, FixHardware;

– організовано взаємодію рушія з світлодіодами на відкритій апаратній платформі для створення електронних пристроїв та прототипів, а саме

на платформі Arduino UNO, а взаємодія з рушієм відбувається завдяки стандартній бібліотеці System.IO.Ports;

– забезпечено програмну реалізацію базової системи руху, генерації задач і організовано взаємодію компонентів додатку: базову систему руху реалізовано стандартними методами бібліотеки UnityEngine; генерація задач описана в окремому скрипті UnityTacks.cs, де описано перелік завдань, і метод для їх виклику; взаємодія компонентів додатку організовано публічним методом public static при оголошені змінних; подальше їх використання виконується через зазначення назви скрипта перед назвою змінної;

– виконано компіляцію і запуск додатку, перевірено роботу додатку на різних операційних системах родини Windows: після компіляції, і при подальшому запуску на різних операційних системах Windows 10 та Windows 11, проблем не виявлено, усі функції працюють коректно.

Отже, усі поставлені завдання виконано. Тому, можна стверджувати що мета роботи досягнута.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Симуляційна відеогра. URL: http://surl.li/uafcb (дата звернення: 13.03.2024).

2. Симулятор системного адміністратора. URL: http://surl.li/uaffx (дата звернення 14.03.2024).

3. Що таке механіка гри сьогодні?. URL: http://surl.li/tznvh (дата звернення 20.03.2024).

4. Вікіпедія Піксельна графіка. URL: http://surl.li/tznvp (дата звернення 15.03.2024).

5. Looking for a few good people to play multiplayer Starbound with? Come check us out!. URL: http://surl.li/tznvw (дата звернення: 16.03.2024).

6. Сайт викладача математичних та інформаційних дисциплін. URL: http://surl.li/tznwb (дата звернення 19.03.2024).

7. Free swamp game tileset pixel art. URL: http://surl.li/tznwp (дата звернення 28.03.2024).

8. Ігровий штучний інтелект. URL: http://surl.li/tznwu (дата звернення 23.03.2024).

9. 10 Кращих ігрових рушіїв. URL: http://surl.li/uafty (дата звернення 21.03.2024).

10. Arduino UNO. URL: http://surl.li/tznxl (дата звернення 25.03.2024).

11. Бірук Б. В., Христинець Н. А. Симулятор системного адміністратора в контексті інтеграції з IoT. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво*. 2024. №54. С. 49-52.

ДОДАТКИ

Додаток А
Скрипт MainMenu.cs

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

public class MainMenu : MonoBehaviour
{
   //public Button loadButton;

   //public int checkStart = 0;

   [SerializeField]public GameObject player;

   public void PlayGame(){
      //checkStart = 1;
      SceneManager.LoadScene("Work");
   }

   public void LoadGame(){
      // Збереження гри
      GameData gameData = new GameData();
      // Заповнення даними gameData
      SaveLoadManager.SaveGame(gameData);

      // Завантаження гри
      GameData loadedData = SaveLoadManager.LoadGame();
      if (loadedData != null)
      {
         // Обробка завантажених даних
         Debug.Log("Loaded game data: " + loadedData.ToString());
         // Наприклад, встановлення позиції гравця збереженим значенням
         player.transform.position = loadedData.playerPosition;
      }
      else
      {
         // Обробка випадку, коли немає збереженого файлу
         Debug.LogWarning("No saved game data found. Starting a new game.");
         // Наприклад, початкова ініціалізація гри
         SceneManager.LoadScene("Work");
      }

   }

   public void ExitGame(){
    Debug.Log("Quit");
    Application.Quit();
   }
}
```

Додаток Б
Скрипт PauseMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel.Design;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    public bool GameIsPaused = false;
    public GameObject pauseMenu;
    public GameObject playerObject;


    // Update is called once per frame
    void Update()
    {
        if (PC.compuhterWindow == false){
            if(Input.GetKeyDown(KeyCode.Escape)){
                if (GameIsPaused)
                {
                    ResumeGame();
                }
                else
                {
                    Pause();
                }
            }
        }
    }

    public void ResumeGame(){
        pauseMenu.SetActive(false);
        playerObject.SetActive(true);
        Time.timeScale = 1f;
        GameIsPaused = false;
    }
    public void SaveGame(){
        // Збереження гри
        GameData gameData = new GameData();
        // Заповнення даними gameData
        SaveLoadManager.SaveGame(gameData);
    }
    public void BackToMenu(){
        //SaveGame();
        SceneManager.LoadScene("Menu");
    }
    public void Pause(){
        playerObject.SetActive(false);
        pauseMenu.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
    }
}
```

Додаток В
Скрипт Player.cs

```
    using System.Collections;
    using System.Collections.Generic;
    using System.Numerics;
    using Unity.VisualScripting;
    using UnityEngine;

    public class Player : MonoBehaviour
    {
        [SerializeField] private float movingSpeed = 4f;
        [SerializeField] private Rigidbody2D rb;
        [SerializeField] private Animator animator;
        UnityEngine.Vector2 movement;

        private void Awake() {
            rb = GetComponent<Rigidbody2D>();
        }
        private void Update(){
            movement.x = Input.GetAxisRaw("Horizontal");
            movement.y = Input.GetAxisRaw("Vertical");
            animator.SetFloat("Horizontal", movement.x);
            animator.SetFloat("Vertical", movement.y);
            animator.SetFloat("Speed", movement.sqrMagnitude);
        }
        private void FixedUpdate() {
            rb.MovePosition(rb.position + movement * movingSpeed *
Time.fixedDeltaTime);
        }
    }
```

Додаток Г
Скрипт GameManager.cs

```
    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;
    using TMPro;
    using UnityEngine.SceneManagement;
    using System;
    using System.IO;
    using System.Runtime.Serialization.Formatters.Binary;
    using System.IO.Ports;

    public class GameManager : MonoBehaviour
    {
        public float dayDurationInMinutes = 150.0f; // Тривалість одного ігрового дня в
хвилинах
        public float clockSpeedMultiplier = 1.0f; // Множник швидкості ігрового часу
        [SerializeField] public TextMeshProUGUI clockDisplay; // Поле для відображення
годинника
        [SerializeField] public TextMeshProUGUI aidaTempCP;
        [SerializeField] public TextMeshProUGUI aidaTempGPU;
        public static bool availableTask = false;
        public static int runningTask;
        public static UnityTasks unityTasks;
        public bool enableTask = false;
        public static SerialPort serialPort = new SerialPort("COM3", 9600); // Порт
з’єднання та швидкість передачі даних
```

```
        public void Start()
        {
            Time.timeScale = 1f;

            try
            {
                serialPort.Open(); // Відкрити порт
```

```
                if (serialPort.IsOpen)
                {
                    Debug.Log("Порт COM3 успішно відкритий.");
                }
            }
            catch (IOException e)
            {
                Debug.LogError("Помилка підключення до порту COM3: " + e.Message);
            }
            catch (UnauthorizedAccessException e)
            {
                Debug.LogError("Доступ до порту COM3 заборонений: " + e.Message);
            }
            catch (System.Exception e)
            {
                Debug.LogError("Невідома помилка: " + e.Message);
            }
```

```
            // Отримуємо доступ до компонента UnityTasks
            unityTasks = GetComponent<UnityTasks>();
        }
```

```
        public float currentGameTime; // Поточний час гри в хвилинах

        public void FixedUpdate(){
            //TemperatureAida();
        }

        public void Update()
        {
            UpdateGameTime();
            UpdateClockDisplay();

            if(enableTask == true){
                TaskManager();
            }
        }

        public static void TaskManager(){

            if(availableTask == false){
                // Генеруємо випадкове завдання
                UnityTasks.TaskType randomTask =
(UnityTasks.TaskType)UnityEngine.Random.Range(0, (int)UnityTasks.TaskType.FixHardware +
1);
                //UnityTasks.TaskType randomTask =
(UnityTasks.TaskType)(int)UnityTasks.TaskType.FixHardware;

                unityTasks.ExecuteTask(randomTask);
            }
        }

        public void TemperatureAida(){
            int randomNumberCP = UnityEngine.Random.Range(35, 110);
            aidaTempCP.text = randomNumberCP.ToString() + " °C";

            int randomNumberGPU = UnityEngine.Random.Range(35, 110);
            aidaTempGPU.text = randomNumberGPU.ToString() + " °C";
        }

        public void UpdateGameTime()
        {
            // Розрахунок поточного часу гри з урахуванням швидкості
            currentGameTime += Time.deltaTime * (60.0f / dayDurationInMinutes) *
clockSpeedMultiplier;

            if(currentGameTime > 490.0f && currentGameTime < 490.3f){
                enableTask = true;
            }else if(currentGameTime > 490.9f){
                enableTask = false;
            }

            // Перевірка для визначення нового дня
            if (currentGameTime >= 1440.0f) // 24 години у хвилинах
            {
                currentGameTime = 480.0f;
            }

            if(currentGameTime >= 960.0f){
                Debug.Log("End work day");
```

```
                currentGameTime = 480.0f;
            }
        }

        public void UpdateClockDisplay()
        {
            // Оновлення відображення годинника
            int hours = Mathf.FloorToInt(currentGameTime / 60);
            int minutes = Mathf.FloorToInt(currentGameTime % 60);

            string clockText = string.Format("{0:00}:{1:00}", hours, minutes);

            if (clockDisplay != null)
            {
                clockDisplay.text = clockText;
            }
        }

        public static void TurnOnLampGameManager()
        {
            // Відправляємо дані до Arduino
            serialPort.Write("1"); // Наприклад, відправляємо "1" для увімкнення
лампочки

            // Отримуємо дані з Arduino
            string data = serialPort.ReadLine();
            Debug.Log("Data from Arduino: " + data);
        }

        // Метод для вимкнення лампочки
        public static void TurnOffLampGameManager()
        {
            // Відправляємо дані до Arduino
            serialPort.Write("0"); // Відправляємо "0" для вимкнення лампочки

            // Отримуємо дані з Arduino
            string data = serialPort.ReadLine();
            Debug.Log("Data from Arduino: " + data);
        }

        void OnDestroy()
        {
            serialPort.Close(); // Закрити порт при завершенні
        }
    }
```

Додаток Д
Скрипт CameraFollow.cs

```
    using System;
    using UnityEngine;

    public class CameraFollow : MonoBehaviour
    {
        public Transform target; // Посилання на гравця, за яким слідує камера
        public float smoothTime = 0.3f; // Час для плавного переходу
        private Vector3 velocity = Vector3.zero;

        [SerializeField] float leftLimit;
        [SerializeField] float rightLimit;
        [SerializeField] float bottomLimit;
        [SerializeField] float upperLimit;

        void Update()
        {
            if (target != null)
            {
                // Визначаємо цільову позицію для камери
                Vector3 targetPosition = new Vector3(target.position.x,
target.position.y, transform.position.z);

                // Плавне переміщення камери до цільової позиції
                transform.position = Vector3.SmoothDamp(transform.position,
targetPosition, ref velocity, smoothTime);
            }

            transform.position = new Vector3
                (
                Math.Clamp(transform.position.x, leftLimit,rightLimit),
                Math.Clamp(transform.position.y, bottomLimit,upperLimit),
                transform.position.z
                );
        }

        private void OnDrawGizmos() {
            Gizmos.color = Color.red;
            Gizmos.DrawLine(new Vector2(leftLimit, upperLimit), new Vector2(rightLimit,
upperLimit));
            Gizmos.DrawLine(new Vector2(leftLimit, bottomLimit), new
Vector2(rightLimit, bottomLimit));
            Gizmos.DrawLine(new Vector2(leftLimit, upperLimit), new Vector2(leftLimit,
bottomLimit));
            Gizmos.DrawLine(new Vector2(rightLimit, upperLimit), new
Vector2(rightLimit, bottomLimit));
        }
    }
```

Додаток Е
Скрипт Lamp.cs

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

public class Lamp : MonoBehaviour
{
    [SerializeField] private GameObject lampOn;
    [SerializeField] private GameObject lampOff;

    private void OnTriggerEnter2D(Collider2D col) {
        if(col.gameObject.tag == "Player"){
            if (GameManager.serialPort.IsOpen)
            {
                TurnOnLamp();
            }
        }
    }

    private void OnTriggerExit2D(Collider2D col) {
        if(col.gameObject.tag == "Player"){
            if (GameManager.serialPort.IsOpen)
            {
                TurnOffLamp();
            }
        }
    }

    // Метод для включення лампочки
    public void TurnOnLamp()
    {
        GameManager.TurnOnLampGameManager();
        lampOn.SetActive(true);
        lampOff.SetActive(false);
    }

    // Метод для вимкнення лампочки
    public void TurnOffLamp()
    {
        GameManager.TurnOffLampGameManager();
        lampOn.SetActive(false);
        lampOff.SetActive(true);
    }
}
```

Додаток Ж
Скрипт PC.cs

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

public class PC : MonoBehaviour
{
    public static bool compuhterWindow;
    public GameObject pcWindow;
    public GameObject playerObject;
    public GameObject pc;
    public Animator anim;
    public Button myCompuhter;
    public GameObject myCompuhterWindow;
    public Button consoleCMD;
    public GameObject consoleCMDWindow;
    public Button printerSeting;
    public GameObject printerSetingWindow;
    public Button aida64;
    public GameObject aida64Window;
    public Button networkSeting;
    public GameObject networkSetingWindow;
    public Button message;
    public GameObject messageWindow;

    public GameObject noMessage;
    public GameObject avaibleMessage;

    // Start is called before the first frame update
    void Awake(){
        anim = pcWindow.GetComponent<Animator>();
        compuhterWindow = false;
    }

    // Update is called once per frame
    void Update(){
        if(GameManager.availableTask == true){
            noMessage.SetActive(false);
            avaibleMessage.SetActive(true);
        }
        else{
            noMessage.SetActive(true);
            avaibleMessage.SetActive(false);
        }
    }

    private void OnTriggerStay2D(Collider2D col) {
        if(col.gameObject.tag == "Player"){
            anim.SetBool("isOpen", true);
            compuhterWindow = true;
            pcWindow.SetActive(true);
        }
    }

    private void OnTriggerExit2D(Collider2D col) {
```

```
            if(col.gameObject.tag == "Player"){
                anim.SetBool("isOpen", false);
                compuhterWindow = false;
                //pcWindow.SetActive(false);
            }
        }

        public void MyCompuhter(){
            Debug.Log("Compuhter work");
            myCompuhterWindow.SetActive(true);


        }

        public void ConsoleCMD(){
            Debug.Log("CMD work");
            consoleCMDWindow.SetActive(true);

        }

        public void PrinterSeting(){
            Debug.Log("Printer work");
            printerSetingWindow.SetActive(true);

        }

        public void Aida64(){
            Debug.Log("Aida64 work");
            aida64Window.SetActive(true);

        }

        public void NetworkSeting(){
            Debug.Log("Network work");
            networkSetingWindow.SetActive(true);

        }

        public void Message(){
            Debug.Log("Message work");
            messageWindow.SetActive(true);

            if (GameManager.availableTask == true){
                GameManager.availableTask = false;
            }

        }


        public void CloseButton(){
            myCompuhterWindow.SetActive(false);
            consoleCMDWindow.SetActive(false);
            printerSetingWindow.SetActive(false);
            aida64Window.SetActive(false);
            networkSetingWindow.SetActive(false);
            messageWindow.SetActive(false);
        }
    }
```