Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“___” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти

на тему

“Дослідження та програмна реалізація експертної системи для управління персонажами у комп’ютерній грі”

Виконав здобувач вищої освіти
ІІ курсу, групи _____
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Дощенко В.О.
«____» _____ 20 р.

Керівник проекту
доктор технічних наук, професор
_____ Єлизавета МЕЛЕШКО
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Галузь знань *12 "Інформаційні технології"*
Спеціальність *123 "Комп'ютерна інженерія"*
Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
« __ » _____ 20__ року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Дощенку Владиславу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація експертної системи для управління персонажами у комп'ютерній грі*

2. Керівник роботи *Мелешко Єлизавета Владиславівна, доктор техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу №19-13 від 17.08.2022 року

3. Строк подання роботи до захисту *10.12.2022 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Дослідження та програмна реалізація експертної системи для управління персонажами у комп'ютерній грі*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схема алгоритму роботи додатку	*4 аркуші*
Діаграма процесів	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В., к.т.н., доцент	25.10.2022	11.11.2022
Охорона праці	Оришака О.В., к.т.н., доцент	04.11.2022	21.11.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2022 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2022 р.	
3.	Розробка моделі компонента	20.10.2022 р.	
4.	Розробка структур даних	25.10.2022 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2022 р.	
6.	Програмування алгоритмів	10.11.2022 р.	
7.	Розрахунок економічної ефективності	13.11.2022 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2022 р.	
9.	Оформлення ПЗ	17.11.2022 р.	
10.	Попередній захист роботи	10.12.2022 р.	

Дата видачі завдання
«__»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Дощенко В.О. Дослідження та програмна реалізація експертної системи для управління персонажами у комп'ютерній грі. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2022.

В даній магістерській роботі розроблено програмне забезпечення, яке призначено для реалізації експертної системи для управління персонажами у комп'ютерній грі.

Метою розробки є дослідження та програмна реалізація експертної системи для управління персонажами у комп'ютерній грі.

Об'єктом дослідження є процес інтелектуального управління персонажами у комп'ютерній грі.

Предметом дослідження є методи побудови експертних та інтелектуальних систем, методи інтелектуального управління персонажами у комп'ютерних іграх.

Методи дослідження базуються на методах штучного інтелекту, методах математичної статистики, методах розробки програмного забезпечення, методах розробки комп'ютерних ігор.

Результат роботи – програмна реалізація експертної системи для управління персонажами у комп'ютерній грі.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено на мові програмування C# у ігровому рушії Unity.

Ключові слова: комп'ютерна інженерія, експертна система, інтелектуальна система, штучний інтелект, комп'ютерна гра.

ABSTRACT

Doshchenko V.O. Research and software implementation of an expert system for managing characters in a computer game. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2022.

In this master's thesis, software designed for an expert system for the management of personages in a computer game.

The purpose of the development is the research and program implementation of an expert system for the management of personages in a computer game.

The object of the research is methods of building expert and intelligent systems, methods of intelligent control of personages in computer games.

The subject of the research is the methods of collection and analysis of data to identify user preferences on the website.

Research methods are based on artificial intelligence methods, mathematical statistics methods, software development methods, and computer game development methods.

The result of the work is the software implementation of an expert system for the management of personages in a computer game.

In the process of working on a software model an analysis of existing hardware and software was performed. All components of the software developed are fully described.

User-friendly user interface is developed. These are instructions for working with software.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the C# programming language in the Unity game engine.

Keywords: computer engineering, expert system, intelligent system, artificial intelligence, computer game

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

NavMesh – система для побудові маршрутів юнітів;

Nemesis – система по актоматичному створенні реалістичного світу через взаємодії з неігровими персонажами;

NPC – Non-Player Character, у ігровій індустрії, юніт, що не підпадає під контроль гравця, та має власну історію та прописаний характер;

OpanAI – система на базі нейронної мережі, що є базою для зародження розробок у сфері штучного інтелекту;

Unity – Unity Engine, сучасний ігровий рушій, що дозволяє створювати від невеликих ігрових проектів до об’ємних розробок великих компаній;

БЗ – база знань;

ГПУ – графічний процесор;

ЕС – експертна система;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер;

ЦПУ – центральний процесор.

ВСТУП

Актуальність теми. У наш час гостро стоїть проблема покращення поведінки штучно створених, так званих Non-Player Character (NPC) персонажів, у комп'ютерних іграх. У багатьох відеоіграх до сих пір застосовують застарілі методики написання сценарію поведінки NPC-персонажів з жорстко прописаними правилами дій через складність нових методів, пов'язаних з застосуванням теорії штучного інтелекту та методами машинного навчання. Штучний інтелект потрібен для імітації розумності NPC, при цьому його завдання не в тому, щоб обіграти користувача, а тому, щоб розважити його. У сучасних іграх використовуються різні підходи до створення штучного інтелекту. У основі лежить загальний принцип: отримання інформації → аналіз → дія. В основі штучного інтелекту для NPC часто лежить ситуаційне дерево поведінки, яким можуть керуватися одночасно декілька персонажів гри. Залежно від того, що вони знають і що поруч із ними відбувається, вони приймають рішення – і часто такі, навколо яких гравець може збудувати свій геймплей. Щоб ситуаційне дерево працювало, кожен NPC повинен мати певну базу знань. База знань може наповнюватися з публічної бази з ігрової локації: наприклад, про стан або положення інтерактивних предметів або інших персонажів, а також з особистої бази, завдяки якій штучний інтелект запам'ятовує останнє розташування тих самих предметів або персонажів.

Відеоігри вже давно перетворилися на окремий вид мистецтва, який за кількістю прихильників змагається з кінематографом. Це експірієнс майбутнього, де кожен може приміряти роль головного героя та вплинути на історію. Останнім часом розвиток відеоігор сповільнився – дивувати гравців стає все важче. Цей вид мистецтва сильно залежить від появи нових технологій, до яких гравці дуже швидко звикають та починають сприймати їх як даність. Але «живі» та розумні NPC ще можуть по-справжньому дивувати і вражати. Як би глибоко розробники

не продумали сюжет: прописували репліки, взаємодії, гілки діалогу – світ навколо стає мертвою декорацією, коли місія виконана або гравець вирішив взаємодіяти з перехожим. Цей прорив може дати технологія штучного інтелекту, яка дозволить наділити NPC особистістю, характером, мріями, історією. Застосування штучного інтелекту в комп'ютерних іграх часто обмежується вузькоспеціалізованими системами, які відповідали переважно за поведінку противників. Однак ці системи інтелектуальними важко назвати, це набори правил і алгоритмів, які створюють ілюзію інтелектуальності. В той же час надання штучного інтелекту NPC-персонажам є новим та актуальним напрямком.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація експертної системи для управління персонажами у комп'ютерній грі.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Дослідження існуючих систем для управління персонажами у комп'ютерних іграх, а також існуючих методів побудови експертних систем.

– Розробка методів та алгоритмів експертної системи для управління персонажами у комп'ютерній грі.

– Програмна реалізація експертної системи для управління персонажами у комп'ютерній грі.

Об'єктом дослідження є процес інтелектуального управління персонажами у комп'ютерній грі.

Предметом дослідження є методи побудови експертних та інтелектуальних систем, методи інтелектуального управління персонажами у комп'ютерних іграх.

Методи дослідження базуються на методах штучного інтелекту, методах математичної статистики, методах розробки програмного забезпечення, методах розробки комп'ютерних ігор.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Запропоновано метод управління персонажами у комп'ютерній грі на основі продукційної експертної системи.

2. Розроблено вітчизняний продукт інтелектуального управління персонажами у комп'ютерній грі, який має більш широкі можливості, на відміну від існуючих аналогів та може бути використаний при розробці різних відеоігор.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі інтелектуального управління персонажами у комп'ютерній грі.

Достовірність наукових результатів підтверджена теоретичними викладками, результатами комп'ютерного імітаційного моделювання, а також результатами тестування розробленого програмного забезпечення, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація експертної системи для управління персонажами у комп'ютерній грі, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній магістерській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Розроблюване програмне забезпечення призначене для реалізації експертної системи, що повинна здійснювати управління персонажами у комп'ютерній грі. А саме розроблювана експертна система повинна керувати NPC-персонажами.

NPC-персонажі – неігрові персонажі, що відрізняються від звичайних персонажів, які управляються комп'ютером, тим, що запрограмовані імітувати поведінку та характер людини. NPC можуть бути або протагоністами, або другорядними персонажами, що населяють відкритий світ для надання йому глибини. NPC найчастіше зустрічаються в розрахованих на багато користувачів іграх, наприклад, в онлайн-стратегіях, рольових іграх тощо.

Віртуальна особистість NPC-персонажу повинна мати певні властивості, основні з них наступні:

– набір індивідуальних характеристик, що представляють віртуальну особистість NPC;

– гнучкість – можливість налаштовувати риси характеру окремої особистості, змінювати манеру спілкування, спосіб мислення, стать, ім'я, голос, швидко формувати історію життя, коригувати цілі;

– здатність «розуміти» світ, у якому вона «живе» і адекватно діяти при різних зовнішніх обставинах та змінах в оточуючому середовищі;

– здатність вербально комікувати з гравцем та іншими ігровими персонажами.

NPC правдоподібний, якщо його дії та контакт з гравцем відповідають психологічним очікуванням щодо поведінки. Правдоподібні NPC відчуваються як живі навіть, якщо гравець знає, що це не так.

1.2 Область застосування

Розроблену експертну систему можна використовувати у декількох сферах діяльності.

По-перше, найбільш доцільним є її використання за призначенням у розробці різних відеоігор. У наші дні ситуація на ринку відеоігор така, що потребує інтегрування різноманітних покращень для неігрових NPC-персонажів, а саме їх поведінки та відтворення звичної життєдіяльності задля задоволення потреб користувачів. Необхідна проста та надійна система, яку б можна було переносити з проекта на проект без витрати великої кількості ресурсів.

По-друге, таку систему можна було б використовувати для моделювання соціальних процесів та ситуацій, які мають місце у реальному світі, і досліджені поведінки окремо взятих моделей при заданні чітких правил та параметрів.

Крім цього вона буде корисна також не як кінцевий продукт, а як інструмент по розробці ігрового програмного забезпечення, що зможе полегшити роботу розробникам та геймдизайнерам у створені більш натурального світу, шляхом впровадження такого інструменту як асет і зостосовуючи різноманітні налаштування.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми магістерської роботи

Проблема візуального реалізму ігрових персонажів одержала назву uncanny valley – ефект зловісної долини. Соціальна нейронаука вчить, що мозок не знає, як зрозуміти зміст вчинків істот, які виглядають як живі, але при цьому поводяться неможливо чи ненормально для живих організмів. Це стосується не лише зображень, але також рухів та поведінки. Реалізм персонажів порушується елементарними помилками – відсутністю зорового контакту або мінімальної чуйності персонажів під керівництвом комп'ютера.

АВ-рушії реалістичної поведінки NPC-персонажів

Artificial Behavior (AB) – штучна поведінка ігрових персонажів.

АВ-рушій – компонент середовища розробки комп'ютерних ігор, що надає інструменти для генерації реалістичної поведінки NPC-персонажів.

При розробці АВ-рушіїв враховується ряд принципів, що забезпечують відтворення ними реалістичної поведінки ігрових персонажів.

Розробники відеоігор зіставили поведінку людини з поведінкою NPC та отримали наступні принципи, що характеризують їх правдоподібність в іграх:

1. Поведінка піддається спостереженню. Спостережувана поведінка – дії, які можна побачити. До неї не належать психічні процеси, які не видно людському оку. Для АВ-рушіїв перший принцип спрощений: розробник не намагатися відтворити саме життя – лише його зовнішній вигляд.

2. Поведінка безперервна. Живі істоти діють постійно – з народження і до смерті. Це не суперечить першому принципу – навіть коли, начебто, ми нічого не робимо, можна спостерігати поведінку: людина спить, сидить нерухомо, затамувала подих. Люди живуть у безперервних потоках поведінки. Проблема

АВ-рушіїв у тому, щоб генерувати такі потоки з окремих пов’язаних між собою видів поведінки.

3. Поведінка інтерактивна. У реальному житті будь-яка поведінка інтерактивна. Наприклад, частота дихання залежить від густини кисню в атмосфері. Якщо усунути контекст (кисень), то поведінка (дихання) втратить сенс. Поведінка завжди в контексті, вона зумовлює взаємодію людей зі світом. Для АВ-рушіїв цей принцип означає таке: вся поведінка має генеруватися процедурно.

4. Поведінка обмежена. Контекст створює умови та накладає обмеження, які можуть бути пасивними чи активними. Обмеження динамічно та непередбачено спрямовують поведінку. Реалістичний персонаж повинен вийти за рамки простого процедурного вибору поведінки та запропонувати повноцінну процедурну анімацію, що адаптується до обмежень на льоту.

5. Поведінка впорядкована. У підручниках із штучного інтелекту часто розрізняють сценарну та несценарну поведінку. Образ дій людини завжди комбінація того й іншого. Наш мозок зберігає гігантські основи послідовностей рухів, які полегшують стандартні форми поведінки. У той же час послідовності легко адаптуються до середовища та контексту. Замість кожного разу наново вирішувати, які м’язи потрібні для дії, мозок використовує шаблони – вони зменшують складність кожної окремої дії. Також цей підхід спрощує синхронізацію між людьми, наприклад, у танцях. Розробники сучасних АВ-рушіїв черпають натхнення в нейронауці і включають в їхню архітектуру параметричну, адаптивну послідовність дій. Наприклад, вороги не лише реагують на дії гравця, а й планують свою мету перемогти супротивника.

6. Поведінку можна перервати. У дотриманні принципу безперервності є складнощі. Вимога безперервності і можливість швидко перервати поведінку суперечать одна одній.

7. Поведінка демонструє закономірну варіативність. Природна поведінка завжди варіативна. Якісь відмінності, навіть невеликі, є завжди.

Вимовляючи слова, людина може по-різному тримати голову або відкривати губи, але не може закрити рота. Для створення правдоподібних варіацій АВ-рушій не повинен моделювати весь голосовий апарат. На практиці можливі варіації часто обмежені та спрощені.

8. Поведінка ієрархічна. Людина одночасно робить кілька дій – і вони вишиковуються в певному порядку. Поведінка майже завжди організована ієрархічно, що добре ілюструє потиск рук: вона передбачає координацію багатьох частин тіла, які знаходяться в ієрархічних відносинах. Під час рукостискання все починається з тулуба, який визначає положення руки (підпорядкована тулубу частина тіла) і нахиляється вперед. Голова також залежить від тулуба: особа звернена у бік другої людини. Очі залежить від голови: спочатку дивляться вниз для координації руки, потім – вгору, щоб погляди людей перетнулися.

9. Поведінка паралельна. Це нормально робити кілька речей паралельно. Люди можуть поєднувати окремі процеси. Наприклад, якщо йти і курити, хода не заважатиме курінню. Також є дії, що перетинаються. Скажімо, йти і тремтіти від страху – два види поведінки, які впливатимуть на ті самі частини тіла різними способами. Наслідки восьмого та дев'ятого принципу ідентичні – потік поведінки має складатися з багатьох ієрархічно організованих «підповедінок». В якості додаткового ускладнення підповедінку можуть контролювати різні області тіла.

10. Поведінка обумовлена, контрольована і читана. Останній принцип, що поєднує кілька близьких понять, стосується відносин між поведінкою та пізнанням. Еволюція навчила нас зчитувати причини поведінки, що не спостерігаються, надавати їй навмисність і сенс. Таке присвоєння сенсу діям відбувається автоматично. Для ігрових персонажів десятий принцип означає, що неможливо відокремити поведінку від сенсу. Вона завжди щось висловлює.

Рівні достовірності поведінки NPC-персонажів

Розробка АВ-рушія – насамперед психологічна задача, і лише потім інженерна. Розробник повинен переконати гравця в свідомості генерованої

поведінки. Спираючись на психологічні дослідження, очікування працюють на трьох рівнях: фізичному, біологічному та психологічному.

Фізичний рівень. Належить до нашого повсякденного і частково вродженого розуміння того, як влаштований фізичний світ. Ми очікуємо, що одухотворені та неживі об'єкти підпорядковуватимуться законам того, що психологи називають «наївною фізикою». Об'єкти мають вагу і мають інерцію, рухаються, а не з'являються з повітря, стикаються, а не проникають один в одного. Однією з головних тем мемів сучасних ігор є порушення фізичних очікувань від гри.

Біологічний рівень. Постійне спостереження за поведінкою людей, свійських тварин, птахів та комах дає нам дані для взаємодії з іншими істотами. Вони неоднакові для різних груп людей: створити віртуального вихованця, правдоподібного в очах малюків, простіше, ніж зробити такого для ветеринарів.

Психологічний рівень. Стосується очікувань інших істот. Створення штучних персонажів, які будуть відповідати всім нашим очікуванням щодо інших розумних істот, все ще щось із розряду наукової фантастики. Такі персонажі мають пройти тест Тюрінга. Ідея емпіричного тесту була запропонована Аланом Тюрінгом у статті «Обчислювальні машини та розум», опублікованій у 1950 році у філософському журналі Mind. Тюрінг ставив метою визначити, чи може машина мислити. Стандартна інтерпретація тесту: «Людина взаємодіє з комп'ютером та людиною. На підставі відповідей на запитання вона має визначити, з ким розмовляє: з людиною чи з комп'ютерною програмою. Завдання комп'ютерної програми – ввести людину в оману, змусивши зробити неправильний вибір».

Найвідоміші інтелектуальні NPC-персонажі у комп'ютерних іграх

Хоча багато NPC – лише фоновий матеріал, деякі з них все ж таки зуміли привернути загальну увагу і стати іконами в ігровій спільноті.

1. Кейв Джонсон (Portal 2). В топ онлайн-ігр для ПК, зазвичай, включають Portal 2. Кейв Джонсон - найцікавіший з неігрових персонажів у франшизі Portal. Діалог Джонсона, який майстерно веде Дж. К. Сіммонс,

дотепний і захоплюючий, а заздалегідь записані повідомлення наголошують на його індивідуальності.

2. HK-47 (Зоряні війни: Лицарі Старої Республіки). Star Wars: Knights of the Old Republic також входить до топу MMORPG онлайн-ігор, а HK-47 – серйозний кандидат на включення до списку легендарних NPC. Головна причина в тому, що гравцям доведеться керувати ним під час бойової місії. Проте HK-47 – це NPC, з яким ви швидко вступаєте у взаємодію у багатьох аспектах KOTOR. У HK-47 до комічності простий моральний кодекс. Фактично, він вважає кожну людину «шматком м'яса» і вважає, що фізична конфронтація – найпростіший спосіб вирішити будь-яку суперечку.

3. Санс (Undertale). Серед неперсічних онлайн-ігор безкоштовно можна відкрити для себе Undertale – гру, що рясніє NPC, що запам'ятовуються. Зокрема, персонаж Санс. Цей моторошний, але водночас усміхнений скелет втілює в собі все найкраще, що є у цій грі. Часом він може здатися марним доповненням, але він також нагадує про те, що не варто сприймати гру надто серйозно.

4. Догмат (Fallout). Серія Fallout багата на відмінні NPC, але Догмат виділяється з їх ряду. Ви постійно бачите і чуєте цього пса-компаньйона у ваших пригодах по всесвіту Fallout. Догмат слідуватиме за вами всюди, куди б ви не вирушили, прикрашаючи вам самотність безплідної пустки.

5. Кортана (Halo). Halo була успішно адаптована під мобільні пристрої для тих, хто любить браузерні онлайн-ігри на телефон. Кортана – сама по собі дуже сильна особистість. Без цього NPC Halo не була б такою, якою ми її знаємо зараз. Вважайте, що Кортана – це голос розуму та знань, що спрямовує вас на вашому шляху. Багато таких NPC – це джерело розчарування для геймерів минулих років. Проте Кортана – безцінний додаток до франшизи.

6. Елізабет (BioShock Infinite). У BioShock Infinite, можливо, багато не так, як хотілося б, але в одному більшість геймерів одностайні – це успіх NPC Елізабет. Елізабет бере участь у грі зовсім не для того, щоб вас обтяжувати. Вона гнучка та здатна. Персонаж має безліч унікальних якостей, які допоможуть вам

виплутатися зі складних ситуацій.

7. Торговець (Resident Evil 4). Торговець постійно з'являється на екрані, щоб врятувати гравців від мешканців села, і завжди вітає питанням «що купуємо?». Якщо потрібно десь роздобути величезну кількість зброї, покращень чи лікувальних предметів, потрібно звертатися саме до нього.

8. Джокер (трилогія Mass Effect). Mass Effect – це легендарна трилогія для тих, хто любить ігри про космос. Джефф «Джокер» Моро – один із найкращих NPC усіх часів, і не лише в серії Mass Effect. Він на голову вищий за решту пілотів флоту Альянсу. Вищий пілотаж він показує не тільки у небі, а й у гуморі. Не дивно, якщо зважити, що озвучував його Сет Грін.

9. Отакон (Metal Gear Solid). Хел «Отакон» Еммеріх – зануда-науковець, який, на подив, став однією з найвідоміших постатей у франшизі Metal Gear Solid. Незважаючи на те, що Отакон демонструє свою вразливість, у міру розвитку франшизи він поступово росте як персонаж, нерідко виходячи за межі обов'язку для того, щоб допомогти гравцю.

10. Том Нук (Animal Crossing). Незважаючи на те, що Том Нук – NPC, йому належить ключова роль у всіх аспектах Animal Crossing. Він керує сільським магазином, і саме до нього варто звертатись, якщо вам потрібні кошти для продовження ваших пригод. Нук – відомий у Японії як Танукічі – практично виступає в ролі воротаря у грі, стежачи за тим, щоб гравці просувалися вперед у потрібний час.

Експертні системи як засіб вирішення неформалізованих задач в ігровій індустрії

Експертна система – це програмний засіб, що використовує експертні знання для забезпечення високоефективного вирішення неформалізованих завдань у вузькій предметній галузі.

Основним призначенням експертних систем є надання програмних засобів, які при вирішенні завдань, важких для людини, одержують результати, що не поступаються за якістю та ефективністю рішенням, що робить людина-експерт.

ЕС використовуються для вирішення так званих неформалізованих завдань, загальним для яких є те, що:

- завдання не можуть бути задані у числовій формі;
- цілі не можна виразити в термінах точно визначеної цільової функції;
- не існує алгоритмічного розв'язання задачі;
- а якщо алгоритмічне рішення є, його не можна використовувати через обмеженість ресурсів (час, пам'ять).

Крім того неформалізовані завдання мають неповноту, неоднозначність і суперечливість як вихідних даних, так і знань про задачу, що розв'язується.

До таких неформалізованих завдань цілком належить задача управління NPC-персонажами у комп'ютерній грі.

Основу ЕС складає база знань (БЗ) про предметну область, яка накопичується в процесі побудови та експлуатації ЕС. Накопичення та організація знань – найважливіша властивість усіх ЕС.

База знань – це сукупність моделей, правил і факторів, що породжують аналіз та висновки для знаходження рішень складних завдань у певній предметній галузі. База знань, що обумовлює компетентність експертної системи, втілює у собі знання експертів і є інституційними знаннями (зведення кваліфікованих, оновлюваних стратегій, методів, рішень).

Існують такі типи **моделей представлення знань**:

- логічні бази знань;
- семантичні мережі;
- фреймові бази знань;
- продукційні бази знань.

Логічні моделі – це моделі представлення знань, засновані на обчисленні висловлювань та предикатів, запозичених із логіки. **Предикат** – частина судження, що відображає предмет мислення. Кожен факт у БЗ представляється у вигляді деякого набору предикатів, а складна структура фактів задається формулами, що пов'язують предикати за допомогою логічних сполучень

(заперечення, кон'юнкції, імплікації, диз'юнкції, логічного слідування). Перевага моделі: можливість використання для пошуку в БЗ фактів і закономірностей, досить добре розроблених у логіці процедур логічного висновку. Недоліки: погана наочність представлення знань, громіздкість записів, при формуванні записів можна припуститися помилок.

Семантичні мережі – це орієнтований граф, вершини якого – поняття, а дуги – відносини між ними. Як поняття зазвичай виступають абстрактні чи конкретні об'єкти, а відносини – це зв'язки типу «це», «є частиною», «належить» та ін. Семантичні мережі надають розробникам багато можливостей для представлення у базі знань необхідних фактів і закономірностей предметної області. Недоліками моделі є складність організації процедури пошуку та виведення на семантичній мережі, а також неоднозначність представлення знань та неоднорідність зв'язків.

Фреймові бази знань одні з найпопулярніших у галузі розробки експертних систем. **Фрейм** – це мінімально можливий опис сутності будь-якого явища, події, процесу чи об'єкта. Мінімально можливий означає, що при подальшому спрощенні опису втрачається його повнота, він перестає визначати ту одиницю знань, для якої він призначений. Розрізняють фрейми – зразки або прототипи (протофрейми), що зберігаються в базі знань, і фрейми – екземпляри (екзофрейми), які створюються для відображення реальних фактичних ситуацій на основі даних, що надходять. Фрейм має майже однорідну структуру і складається із стандартних одиниць, які називаються **слотами**. Кожна така одиниця має назву та своє значення. Зображається кадр у вигляді ланцюжка типу:

Фрейм = <слот 1> <слот 2> … <слот N>.

Як слот може бути вказано ім'я іншого кадру. Значеннями слота може бути конкретні дані, процедури і навіть продукції.

Фрейм визначається як структура такого виду:

(ім'я кадру,

ім'я слоту 1 (значення слоту 1)

ім’я слоту 2 (значення слоту 2)

…

ім’я слоту N (значення слоту N))

Якщо значення слотів не визначені, то кадр називають кадром – прототипом, інакше – конкретним кадром або екземпляром кадру.

З усіх розглянутих раніше моделей представлення знань лише фреймам властива висока структурованість, внутрішня інтерпретованість за допомогою імен та значень, зв’язність слотів та їх значень. Фрейми також мають високу наочність і модульність. Однак кадри найбільш ефективні при обробці семантичної складової знань. У кадрів, як і в семантичних мереж, відсутні універсальні процедури їх обробки, що призводить до неефективного використання ресурсів обчислювальної техніки (пам’яті та швидкодії).

Продукційні моделі представлення знань, що грунтуються на правилах, дозволяють представити знання у вигляді виразів типу:

Якщо < умова > **то** < дія >

Якщо < причина > **то** < слідство >

Якщо < ситуація > **то** < рішення >

Продукційні моделі можуть бути реалізовані як процедурно, так і декларативно. У процедурних системах присутні три компоненти:

– база даних;

– деяка кількість продукційних правил, що складаються з умов та дій;

– інтерпретатор, який послідовно визначає, які продукції можуть бути активовані в залежності від умов, що містяться в них.

У базі даних зберігаються відомі факти обраної предметної області. Продукційні правила (продукції) містять специфічні знання предметної області у тому, які додаткові факти можуть бути враховані, чи є специфічні дані у БД.

Завдяки властивості модульності, властивій продукційним моделям, можна додавати та змінювати знання (правила, факти). Тому продукційні моделі

застосовуються в предметній галузі, де немає чіткої логіки та завдання вирішуються на основі незалежних правил (евристик).

Правила продукції несуть інформацію про послідовність цілеспрямованих процесів. Продукція виражається мовною конструкцією виду *«якщо виникає певна ситуація, то можна зробити певний набір дій»*.

Позитивні якості моделі: наочність, висока модульність, легкість внесення доповнень і змін, простота механізму логічного висновку. Недоліком є те, що зі збільшенням обсягу знань ефективність інформаційних одиниць моделі падає.

У даній роботі було вирішено використати експертну систему з продукційною моделлю представлення знань для управління поведінкою персонажів у комп'ютерній грі.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для розробки експертної системи для управління персонажами у комп'ютерній грі був обраний ігровий рушій Unity. Рішення було прийнято озираючись на те, що редатор даного ігрового рушія досить простий у використанні. Будь який геймдизайнер може досить швидко налаштувати параметри для функціонування програми навіть не спілкуючись з розробниками так, як Unity має зрозумілий Drag & Drop інтерфейс через який можна з легкістю встановлювати об'єкти на сцені (абстрактний простір у якому відбіваються основні процеси програми, іншими словами це основний файл додатку де крім усього сказаного ще й зберігаються геометричні закони) або розташовувати скрипти для подальшої їх роботи.

Така простота та множинна направленість вищезгаданного рушія являється набором спроб та помилок розробників Unity, що розвивали свій проект впродовж багатьох років. На сьогодняшній день ми маємо змогу працювати вже з п'ятою версією програми. Дана версія влючає в себе розробку для трьо- і двовимірного

простору, доповненої реальності, віртуальної реальності тощо. Дуже важливим є те, що Unity є кросплатформовим рушієм. Тобто на його основі можна розроблювати додатки для великої кількості існуючих платформ, таких як IOS, Widows, WebGL, Linux, Android, тощо.

Основу роботи будь-якого додатку на даному ігровому рушії (крім сцен) складають ігрові об’єкти. У ігровому просторі дані об’єкти є просто абстракціями, вони не мають жодних форм, розмірів, позиції і тому подібного. Додати їм якісь функції або властивості можна за допомогою компонентів, і це є основною властивістю Unity. Ця особливість і робить цю програму дуже простою для вивчення та зручною для розробки.

Даний рушій підримує велику кількість різних форматів аудіофайлів, графічних файлів, а також 3Д моделей. Крім цього розробник або геймдизайнер може легко створювати або редагувати анімації майже для будь-яких об’єктів. Це є можливим завдяки досить добре пропрацьованому аніматору вбудованому у сам рушій.

Неможливо не згадати про Unity Asset Store у якому розробник може знайти для себе будь-який набір ассетів (ассетом називають усі файли які використовуються під час створення додатків). Дане розширеня дуже пришвидшує розробку програмного зебезпечення.

І останнє, що можна продемонструвати про Unity, але не останнє по значимості, це взаємодія з законами фізики. Цей рушій включає в себе фізику твердих тіл, пружин, а також основні сили, що діють у реальному світі. Крім цього можна легко прописати свої власні закони по яким буде відбуватися взаємодія між усіма об’єктами у створеному додатку.

Варто зазначити деякі недоліки рушія Unity насамперед це деякий брак графічної складової. Наприклад, щоб зробити дуже гарну картинку необхідно буде пожертвувати частиною швидкості додатку. Також у даному ігровомі рушії є проблеми з налаштуванням світла на сцені.

На даний момент Unity підтримує дві основні мови програмування. Це за замовченням C#, а також JavaScript. Для розробки була обрана саме мова C#.

C# це об'єктно-орієнтована мова програмування зі строгою системою типизації, а також без прямого доступа до пам'яті. Синтаксис даної мови програмування є близьким до C++ та Java. Вона підтримує поліморфізм, наслідування, перевантаження операторів, тощо. Хоча ця мова не має прямого доступу до пам'яті але потрібно її звільняти і цим займається автоматичний збірник сміття. Він функціонує таким чином, що якщо деякий об'єкт перестає використовуватися то посилання да деяку ділянку пам'яті очищається, однак це не приводить до очистки самої ділянки у стеку, і в цей момент у гру вступає очисник, що відсліджує, що на дану ділянку посилань немає і очищає її.

Недоліком у використанні C# є те, що у ньому відсутня реалізація множинного наслідування класів, хоча є множинне наслудівання інтерфейсів.

Що ж до використання цієї мови програмування у ігровому рушії Unity, то кожен класс, він же у просторі рушія являється скриптом, який буде поміщений на сцену має наслідувати основний клас Unity – MonoBehaviour, або наслідуватися від класа або інтерфеса, що наслідує даний основний клас. Основний клас рушія включає в себе доступ до взаємодіє з ігровими обьєктами та їх компонентами такими як Transform, MeshRenderer, RigidBody, тощо, а також включає в себе основні методи Unity, що викликаються при деяких діях. Наприклад, старт програми, її кінець, упродовж роботи програми, абож на початку та в кінці роботи самого скрипта.

Для написання самого коду було обрано Microsoft Visual Studio, що включає в себе інтегроване середовище для розробки програмного забезпечення, а також низку розширень саме для розробки ігрових додаткові, що є дуже зручним, адже не потребує встановлення додаткових програм. Важливим пунктом у виборі саме цього програмного забезпечення є те, що Microsoft Visual Studio дозволяє дуже зручно і швидко провести аналіз коду і може допомогти знайти найбільш вразливі місця, що можуть уповільнювати роботу додатку. Крім цього дане

програмне забезпечення включає багато засобів для саме написання коду. Інтерактивні підказки, що пришвидшують роботу, можливість переглянути схему залежностей та посилань між класами та багато іншого.

Також дуже важливим є те, що дане програмне забезпечення і ігровий рушій є безкоштовними і дозволяють працювати навіть над комерційними проектами, що в перспективі дозволить розвивати свій проект у щось більше, що зможе допомоги більшій кількості розробників.

2.3 Розгорнута постановка завдання

Наступним пунктом є аналіз поставленої задачі в усіх напрямках та виокремлення основних положень, що потребуватимуть розгляду.

Згідно з навчальними матеріалами, а також з технічним завданням на квалікаційну магістерську роботу, підлягає реалізації програмне забезпечення експертної системи для управління персонажами у комп'ютерній грі.

В процесі розробки даного додатку необхідно провести групування виокремлених пунктів у деякий план та паставити правильні задачі.

Отже, при постановці завдання були виділені такі пункти плану:

− проведення аналізу існуючих систем, а також виявити їх позитивні та негативні сторони;

− визначити методику створення експертної системи для управління персонажами у комп'ютерній грі для використання її у різних сферах. Побудувати структурну та функціональну схему;

− розробити програмне забезпечення для виконання поставленої задачі. Побудувати блок-схеми алгоритмів і підалгоритмів;

− розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

− сформувати висновки про разроблену систему та методики, що були використані при розробці.

Як можна побачити аналіз був проведений, пункти виділені, основна робота, яку необхідно виконати була проаналізована та записана. Наступним прококом для створення системи «штучного інтелекту», буде розписання запланованого функціонали та розробка архітектури програмного забезпечення.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Для управління ігровими персонажами у розробленому програмному забезпеченні було використано продукційну експертну систему. У якості ігрових персонажів виступають жителі деякого віртуального міста.

Продукційні експертними системами є системи заснованими на правилах. Правила організовані як IF-THEN структури і називаються продукціями.

У продукційних системах знання представлені у формі множинних правил, на основі яких формуються висновки, які мають бути зроблені (або не зроблені) у різних ситуаціях.

Висновки робляться з урахуванням методів прямого чи зворотного логічного висновку. Залежно від методу логічного висновку розрізняють два види продукційних систем: системи із прямим логічним висновком та системи із зворотним логічним висновком.

Загальна стратегія вирішення завдань полягає у розбитті їх на фрагменти, які можна легше довести. При цьому *системи з прямим логічним висновком* знаходяться під керуванням фактів. Вони починають свою роботу з відомих початкових фактів і продовжують, використовуючи правила створення висновків чи виконання певних дій.

Системи із зворотним логічним висновком керуються гіпотезами. Вони розпочинають свою роботу з гіпотези, або мети, яку користувач намагається довести і продовжують, шукаючи правила, які дозволять довести правдивість гіпотези.

Широке застосування систем, що базуються на продукційних правилах, обумовлене наявністю в них наступних особливостей:

– *Модульна організація.* Завдяки модульній організації спрощується представлення знань та розширення експертної системи, нарощуючи її можливості крок за кроком.

– *Наявність засобів пояснення.* Продукційні експертні системи за допомогою правил дають змогу легко створювати засоби пояснення. Засіб пояснення відстежує послідовність активованих правил і, на цій основі, дає можливість відновити перебіг міркувань, що привели до певного висновку.

– *Наявність аналогії з пізнавальним процесом людини.* Згідно з результатами, отриманими Ньюеллом та Саймоном, правила є природним способом моделювання процесу вирішення завдань людиною. Тому, в процесі виявлення експертних знань, не виникають зайві складності у поясненні експертам структури подання знань, оскільки застосовується представлення у вигляді правил IF-THEN.

Загальну структуру експертної системи, заснованої на правилах, можна подати у вигляді схеми, зображеної на рис. 3.1.

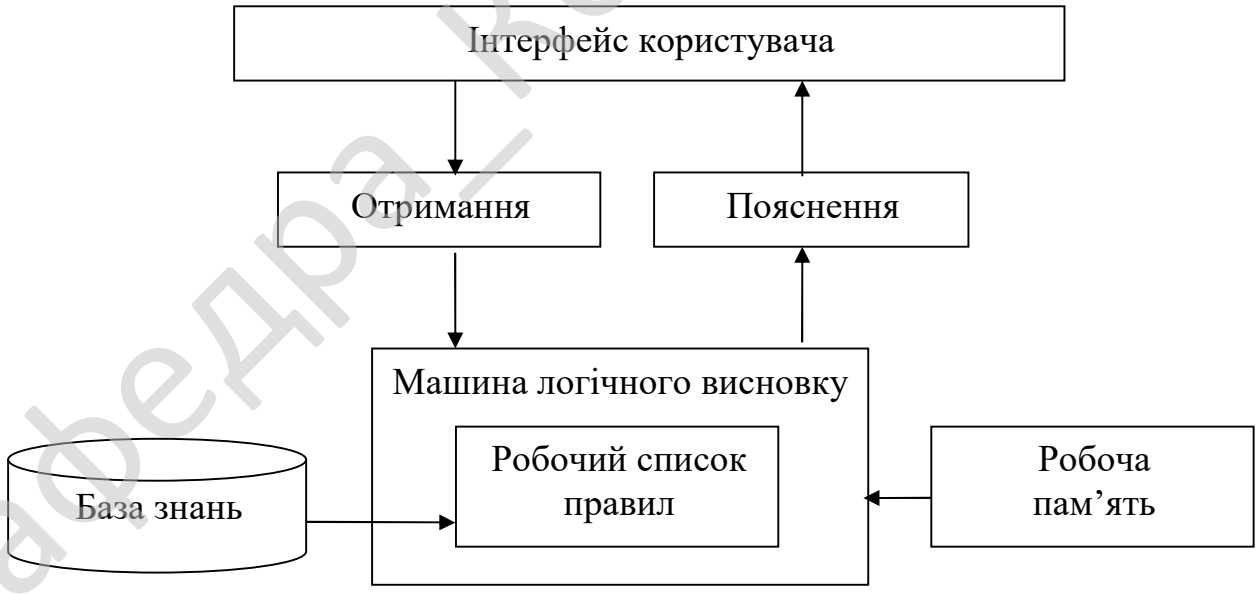

Рисунок 3.1 – Загальна структура продукційної експертної системи

Наведена структура наочно ілюструє основні аспекти продукційної моделі експертних систем, тому докладно розглянемо сутність та призначення її компонентів.

Інтерфейс. Інтерфейс – це механізм, за допомогою якого відбувається спілкування користувача з експертною системою. У нашому випадку, результат роботи експертної системи відображається у діях ігрових персонажів, їх поведінці і способі реагування на різні ситуації та події.

Засіб отримання знань. Засіб отримання знань є автоматизованим способом, який дозволяє користувачеві вводити знання в систему, не застосовуючи явного кодування знань за допомогою інженера знань. Цей інструментальний засіб, в деяких експертних системах, здатний навчатися, здійснюючи автоматичне формування правил на підставі прикладів. Для формування правил у машинному навчанні застосовуються такі методи та алгоритми, як ID3, C4.5, C5.1, штучні генетичні алгоритми, нейронні мережі. У розроблюваній програмі деякі правила одразу прописані у базі знань експертної системи, а деякі формуються автоматично в результаті взаємодії ігрових персонажів з віртуальним середовищем та гравцем.

База знань. Продукційні експертні системи зберігають знання, необхідні для вирішення завдань, у форматі продукційного псевдокоду IF-THEN. Кожне правило позначається ім’ям. Після нього починається IF-частина правила. Ця частина продукційного правила розташована між ключовими словами IF і THEN та має назву – антецедент або ліва частина (LHS – left-hand-side) правила. Насправді застосовуються також назви: умовний елемент і шаблон. Після частини IF починається частина THEN правила. Вона містить висновки чи список дій, які мають бути виконані згідно з правилом. Ця частина правила називається консеквентною або правою частиною (RHS – Right-Hand Side). До складу дій консеквентних правил зазвичай входить додавання і видалення фактів з робочої пам’яті, або формування результатів. Формат опису цих дій залежить від синтаксису мови експертної системи.

Машина логічного висновку. Машина логічного висновку є програмним компонентом, який визначає антецеденти правил (якщо є), які виконуються згідно з фактами. Для цього машина логічного виводу виконує такі дії:

- вибирає правила, яким відповідають факти;
- розподіляє обрані правила з пріоритетів;
- виконує правило із найвищим пріоритетом.

Як класичні стратегії вирішення завдань в експертних системах використовуються два загальні методи логічного висновку: прямий логічний висновок і зворотний логічний висновок. До інших методів, застовуються для виконання конкретизованих завдань, можуть входити: аналіз цілей та засобів, спрощення завдання, перебір із поверненнями, метод «запланувати-зробити-перевірити», ієрархічне планування.

Робоча пам'ять. Робоча пам'ять застосовується для розміщення фактів, що стосуються поточного стану об'єкта досліджень. Факти, що у робочої пам'яті, не взаємодіють один з одним, на відміну правил, які у знаходяться базі знань. Якщо в робочій пам'яті є факт, що відповідає умовній частині правила, машина логічного виводу розміщує це правило у списку правил.

Якщо правило має кілька шаблонів, то для того, щоб правило можна було розмістити в робочому списку правил, всі ці шаблони повинні бути розпізнані, як відповідні. Як умову відповідності деяких шаблонів можна назвати відсутність певних фактів у робочій пам'яті. Машина логічного висновку працює у режимі здійснення циклів «розпізнавання-дія».

Робочий список правил. Робочий список правил являє собою створений машиною логічного висновку і розташований за пріоритетами список правил, шаблони яких відповідають фактам, що знаходяться в робочій пам'яті.

Правило, всі шаблони якого розпізнані як відповідні, називають активізованим або реалізованим. У списку правил можуть бути одночасно присутні кілька активізованих правил. У цьому випадку машина логічного виводу повинна вибрати, залежно від пріоритету, одне з правил запуску дії.

Після завершення всіх правил керування повертається до інтерпретатора команд верхнього рівня, щоб користувач міг надати командному інтерпретатору експертної системи додаткові інструкції. Роль верхнього рівня системи виконує інтерфейс, який є механізмом інтерпретації команд користувача.

Засіб пояснення. Головною особливістю експертної системи є передбачений у ній засіб пояснення, який відображає інформацію про те, як система дійшла певного висновку. У системах, заснованих на правилах, нескладно організувати пояснення, як було отримано певний висновок, оскільки хронологія активізації правил і зміст робочої пам'яті можна зберігати в стеку.

Перше, на що варто звернути увагу при моделювання жителів міста, це те яким саме чином будуть з'являтися нові юніти на можливій території міста. Вони повинні мати якісь первинні параметри для налаштування, які будуть керувати їх подальшим станом, а також будуть давати сенс усім діям, що вони будуть виконувати. Для цього необхідно створити обробник, що відповідав би за створення юнітів, заповнення їх параметрів, а також обробляв їх подальшу взаємодію з середовищем у якому вони з'являться.

Отже, після створення чергового юніта йому передаються основні параметри. Для кожного юніта є свій ідентичний набір параметрів, що заповняється випадковим чином, а саме: голод, спрага, втомленість. Ці параметри є основними рушіями, що штовхають агентів виконувати поставлені перед ними задачами. Найголовніша задача – зберігати рівномірний процес життєдіяльності і підтримувати свої головні показники у балансі. Крім цього на початку функціонування нового «містянина» у скрипті, що його контролює запускається життєвій цикл, що віднімає або додає випадкове значення до його показників. Крім цього невеликою особливістю являється те, що у кожного юніта є своє ім'я завантажене з файлу і може бути використане при подальшому розвитку квестової системи для потенційної гри.

Першою задачею, що стоїть перед новим агентом являється пошук, якщо можна так назвати, своєї власної «домівки», а саме області у яку б він міг

повертатися вночі, ця особливість буду описана нижче, а також зміг би відновити параметр, що відповідає за втому, який не можна опускати нижче певного рівня інакше юніт не зможе нормально функціонувати та забезпечувати себе необхідними ресурсами. Цей процес починається з того, що скрипт UnitController.cs, що відповідає за керування одним конкретним юнітом, тобто представляє його абстрактну сутність передає запит до менеджера юнітів, що був описаний вище, іншими словами обробник, що у свою чергу оброблює його і в свою чергу передає запит до менеджера будівель. Цей менеджер будівель повертає обробнику данні по усім житловим приміщенням, які існують на мапі в даний момент. Після цього ці дані потрапляють послідовно до контроллера юнітом і він починає свій шлях до першої поставленої мети.

Окремо слід виразити декілька слів про те, яким чином усі юніти рухаються та визначають свій рух по мапі, обходячі перешкоди і знаючи куди вони можуть дістатися, а куди ні. Тут у гру вступає NavMesh система ігрового рцшія Unity, що є дуже зручною і дозволяє створювати просто величесні ігрові зони для переміщень як штучного інтелекту так і самих гравців (рис. 3.2).

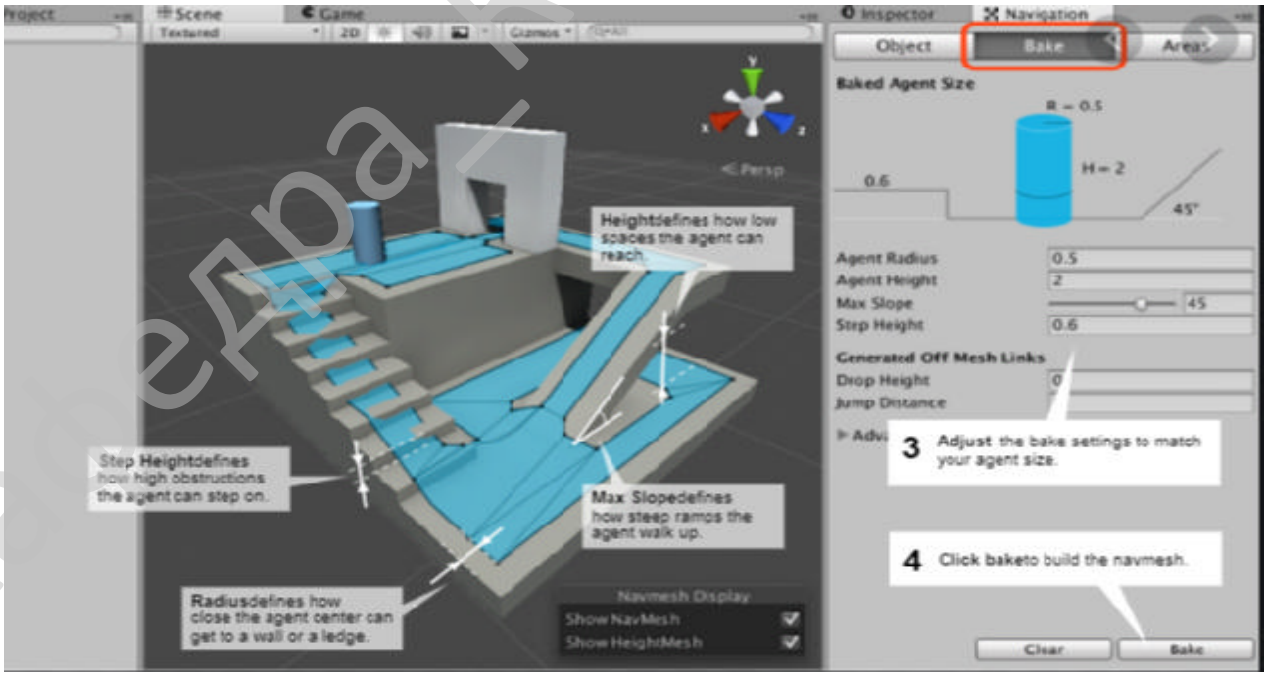

Рисунок 3.2 – Зображення NavMesh системи

Пошук маршруту відбувається за допомогою АПІ сценаріїв агента NavMesh. Для того щоб правильно налаштувати дану утиліту необхідно відкрити вікно управління, вибрати необхідний тип агента або створити новий, у нашому випадку гуманоїд, обрати необхідні параметри такі як радіус агента, його висота або кут на який він може піднятися і натиснути на кнопку запікання, після деякого часу ми отримуємо повністю створену мапу яка прораховує усі статичні та нестатичні об’єкти.

Наступним кроком для юніта є пошук точки у просторі, що являється так би мовити житловою зоною, для того щоб зайняти її і закріпити за собою. По першому з правил на одній житловій точці не може бути більше ніж два юніти. Якщо кількість персонажів у даний проміжок часу перевищує два, система повідомляє про це агента і він повинен продовжити свій пошук, до того часу доки не знайде вільний простір. Слід також зазначити, що на даній момент кількість житлових камер обмежена і ще не була додана можливість юнітам самим створювати собі житло, але така функція буде розглядатися у майбутньому.

Далі відбувається налаштування середовища, і у цей час можна сказати декілька слів про механіку зміни дня і ночі. Даний функціонал є дуже важливим для підтримання більш реального ходу речей і напряму впливає на поведінку і поставлені задачі для юнітів. Ефект створено за допомогою спрямованого джерела світла, а також налаштувань для освітлення сцени. Керується все єдиним скриптом, що відсилає івенти усім підписаним на нього скриптам про зміну часу, крім цього у майбутньому планується додавання погодних умов (локальних та глобальних).

Після переналаштувань система виходить на стандартну роботу. При цьому починаючи життєвий цикл юнітів, відбувається процес пошуку їжі та води. По правилам у кожного ресурсу є своя вартість, але є такі, що не коштують нічого. Вся суть у тому, що безкоштовній ресурс віднімає показники енергії, і тут була прописана залежність чим вище вартість ресурсу тим менше він віднімає енергії і тим довше юніт може функціонувати. Також такий хід варто було

зробити аби персонажі витрачали б кошти, що вони заробили на роботі, що буде описано трохи пізніше.

Тобто вже вимальовується чітка послідовність: пошук житла, потім пошук ресурсів і втрата енергії. Це і запускає наступний етап по пошуку робочого місця для кожного юніта.

Коли показник витривалості або, як було названо вище, втомленості падає нижче двадцяти відсотків юніту необхідно пересунутися на точку, що записана у його контролер, і являється для нього житловою зоною. Починаючи з цього моменту відбувається процес, що буде відновлювати цей параметр по закінченню якого, юніт зможе визначити, що такий порядок занадто сильно заважає йому виконувати свій життєвий цикл. Тобто поповнення життєво важливих ресурсів вимагає занадто великих затрат у показнику витривалості через те, що юніт використовує лише доступні йому джерела. Наприклад, воду ти їжу номінально він може отримати лише на безкоштовніх точках на яких затрати енергії є дуже чуттєвими.

Для того щоб це змінити необхідно створити систему, що має бути доволі збалансованою, для циркулювання грошових одиниць по усій моделі, щоб забезпечити усі юніти необхідною кількістю життєво потрібних ресурсів. Для створення такої системи перш за все необхідно створити початкову роздачу робочих місць саме для отримання деякої кількості грошових одиниць.

Якщо розглядати цей процес більш наглядно, то можна виділити головні пункти, або ж можна ще назвати критеріями за якими буде відбуватися пошук робочого місця для кожного окремо взятого персонажа:

- кількість робочих місць;
- кількість працівників, що дане робоче місце може забезпечити;
- пріорітетність виконання завдань;
- кваліфікація юніта (плани на майбутнє).

Як можна побачити така система містить мінімум три обов’язкових параметра, що мають бути інтегровані для коректної роботи процедури по знаходженню точки отримання грошових одиниць.

Отже, була виконана дослідницька, а також робота по плануванню по розробці такої системи і працює вона наступним чином. Починається все з контролера юнітом, що відсилає до обробника повідомлення про те, що йому потрібно знайти робоче місце. Обробник отримує повідомлення і перенаправляє його до менеджера будівель, який має доступ до усіх робочим місць на сцені та їх доступності. Кожне робоче місце це окрема сутність, що виконує своє внутрішні процеси незалежно одна від одної, і має цілу низку керуючих параметрів. Це наприклад, кількість вільних місць, що дана точка може запропонувати, кількість грошових одиниць, що буде отримувати агент, а також час, що буде затрачено на процес видачі. Ці три основні параметри керують також низкою менш суттєвих параметрів. Вони усі утворюють окрему незалежну екосистему, що дозволяє дуже легко створювати нові місця для роботи юнітів, а також дуже легко їх кастомізувати.

Менеджер будівель подає запит на отримання необхідного робочого місця по тим параметрам, що йому вислав обробник, менеджер юнітів, по конкретному агенту. Після оброблення даних він повертає той пункт для роботи, що найбільш добре підходить до запиту і контролер юніта записує найявність робочого місця для агента, а також його місце розташування.

Далі відбувається процес по самій роботі, тобто поставленя задачі персонажу дійти до робочого місця та почати викоувати свої обов’язки. Він розпочинається з того, що контролер який відповідає за погодні умови та зміну дня і ночі пропорційно до реального часу, рівно о восьмій годині ранку по ігровом часу, передає команду усім агентам, що вже знайшли для себе роботу, почати робочий день та відправитися на точку видачі грошових одиниць, де вони проводять більшість часу. А також з настанням вечора, а це сьома годи на ігровим часом, усі юніти покидають робочі місця та розпочинають процес відпочинку. Він

утворений простою прогулянкою по місту, щоб відтворити вечірнє життя його звичайного аналогу, що існує у реальному світі, або юніти, що мають занизький параметр енергії повертаються до житловох зони.

Крім цього, треба зазначити пункт, що для відтворення реальни процесів було додано вихідні дні у які юніти можуть лише витрачати свої кошти, а також ті агенти, що не мають роботи можуть просто рухатись по локаціїї, що є також важливим для подальшого розвитку кввестової системи для повноцінної ігрової локації.

На даному етапі для системи циркуляції грошових одиниць вже прописано план по видачі їх юнітам, а також модель поведінки юніта при настанні моменту, що визначає початок робочого дня. Але для балансу слід також визначити як саме будуть витрачатися ці одиниці.

Як було зазначено вище кожен персонаж має свої базові параметри від яких буде залежати правильність функціонування життєвого циклу кожного окремого агента. Для задоволення їх потреб були створені точки у яких можна було б отримати той чи інший ресурс, надалі вони будуть називатися джерелами ресурсів. До цтого часу усі джерала були виключно безкоштовніми з великими затратами показників втоменості. Але тепер коли почала відбуватися консентрація грошових одиниць у кожного персонажа можна внести деякі корективи у систему поповнення ресурсів.

А саме:

- створення більшої кількості джерел;
- налаштування кожного джерела у балансі коштів та затрат енергії;
- написання відповідних контролерів;
- визначення доступності того чи іншого джерела.

Перші три пункти плану були успішно імпортовані в систему, через те, що ще на початку розробки системи було передбачено такий хід подій і закладено базову логіку для налаштування кожного окремого джерела.

Про доступність джерела повідомляє менеджер будівель, а саме для того

щоб кошти юнітів не застоювалися надовго і не відбувалося великого накопичення грошових одиниць у одного персонажа, менеджеру передається кількість грошей у агента і він повертає джерело ресурсів, що має набільшу вартість. Це дозволяє зменшити загальні накопичення у юнітів, а також збільшити їх продуктивність упродовж їх життєвого циклу. Після створення системи грошообігу вона була протестована і виявлено, що вона являєтеся досить стабільною та збалансованою для подальшої роботи з нею.

Дуже важливим пунктом виникла потреба у збалансуванні станів персонажів, щоб одні завдання не заважали іншим, а треті не створювали яких небудь ситуацій при яких юніт не може вирішити яке завдання виконати наступним. Ця проблема є досить складною у вирішенні через великий набір станів та параметрів, але була створена базова модель, або можна сказати набір правил для таких ситуацій, зо дозволили дещо стабілізувати систему. Такі правила представлені у таблиці 3.1:

Таблиця 3.1 – Таблиця пояснення основних правил розробленої експертної системи

Правило	**Основна суть правила**
Правило почерговості ресурсів	Кожен юніт має задовольняти свої потреби у ресурсах чітко дотримуючись пріоритетності даного ресурсу і не порушувати послідовність виконання команд, наданих системою.
Правило визначення втоми	Кожен юніт може втратити скільки завгодно значення параметра енергії, але це не повинно впливати на виконання ним своїх базових функцій для підтримання життєдіяльності. Параметр втоми напряму керує процесом знаходження і виконання ресурсом роботи.

Продовження таблиці 3.1

Правило	**Основна суть правила**
Правило втрати ресурсів	Упродовж життєвого циклу кожен юніт має втрачати ресурси поступово, залежно від умов середовища, в якому він знаходиться.
Правило вибору джерела ресурсів	Кожен юніт повинен вибирати найкраще джерело ресурсів, яке йому доступне.
Правило розподілу часу	Кожен юніт має чітко дотримуватися розпорядку послідовних дій, що залежать від ігрового часу.
Правило реагування на подразник	Кожен юніт повинен реагувати на подразник позачергово для того, щоб вирішити проблему у системі найшвидшим способом.

У цей набір правил входить взаємодія між усіма станами, а також їх пріоритетність по виконанню і почерговість.

Насамперед потрібно зазначити два основних параметра це голод та спрага, що мають найвищий пріоритет серед інших станів, і працюють вони наступним чином. При потребі поповнити один із цих параметрів система перевіряє зайнятість юніта, якщо він не виконує інший головний параметр, то система направляє персонажа у потрібну точку, якщо ж виконується поповнення іншого головного параметра, то поточний запис стає у чергу і очікує закінчення процесів.

Крім цього похід на роботу буде виконаний лише за умови, що він не перерве виконання процесів по поновленню основних параметрів. Також, знаходячись на робочому місці, юніт може покинути його задля поповнення показників спраги чи голоду, але, наприклад, якщо занадто сильно опуститься параметр втоми, то агент не може покидати своєї роботи, тому що цей параметр стоїть за пріоритетністю на останньому місці. Така ж ситуація і з прогулянкою. Але за одним винятком, юніт може перервати процес прогулянки якщо параметр

його витривалості опуститься нижче двадцяти відсотків.

Цей простий набір правил успішно керує системою і дозволяє зробити її більш природною.

Найважливішою особливістю створеної системи є те, що юніти могли б адаптуватися до різких змін обстановки, тобто реагувати на якісь подразники самостійно без участі користувача, а також навчатися самим, для того щоб вирішувати такі завдання швидше та ефективніше.

З цією метою було створено чітку модель події з її параметрами, а також відповідними подіями, такими як вирішення задачі або вплив на якийсь юніт. Дана модель може використовуватись у багатьох випадках, і кожному окремому показнику можна буде прописати ще цілу низку особливостей, що будуть так чи інакше впливати на персонажів та їх оточення.

При створенні нової раптової ситуації на сцені запускаєтеся процес, що відповідає за те, якому персонажу, за який проміжок часу прийде сповіщення про те, що на нього впливає подразник, а також юніту приходить ймовірність успішної та неуспішної дії. На стороні агента прораховується поточна ймовірність і відповідно посилається в зворотню стороно дія, що повинна бути виконана відповідно до відсотків. Тут слід зазаначити, що виконуєтьс якась абстракна дія на стороні агента, але сама вона реалізована у подразнику, тобто ці дії залежать лише від виду подразника і не прописуються на боці юніта, такий підхід дозволяє створити дуже гнучку систему взаємодії.

Якщо персонажу випала позитивна ймовірність, то автоматично наступна подія за участю даного виду подразника, що буде впливати на юніт буде вирішена з ймовірністю сто відсотків. Якщо ж навпаки випала негативна ймовірність, то ймовірність позитивного рішення задачі на стороні юніта збільшується і зберігається. Таким способом вдалося отримати систему, що буде самонавчатися і само розвиватися відповідно до ситуацій, що відбуваються у даному середовищі. А також з’явилася змога вивантажувати файли з даними агентів, для подальшого їх імпорту на іншу сцену зі збереженням показників та вже навченими до появи

деяких подразників. Таку систему можна легко реалізувати у майбутніх ітераціях розробки моделі.

І останнім пунктом у роботі над задачею являється додавання анімацій до моделей, які візуально представляють агентів. Зазвичай для анімації при створенні моделі художник пророблює кістки для того щоб анімації виглядали більш природно та анімує їх прямо у редакторі для створення моделей. Але при розробці програмного забезпечення я зіштовхнувся з відсутністю анімацій, тому було вирішено їх створити. У цьому мені допомогла утиліта UMotionEditor, що дозволяє використовуючи пензлі, записувати різні анімації прямо у редакторі Unity. Такий підхід допоміг зекономити час та сили у створені більш живих персонажів.

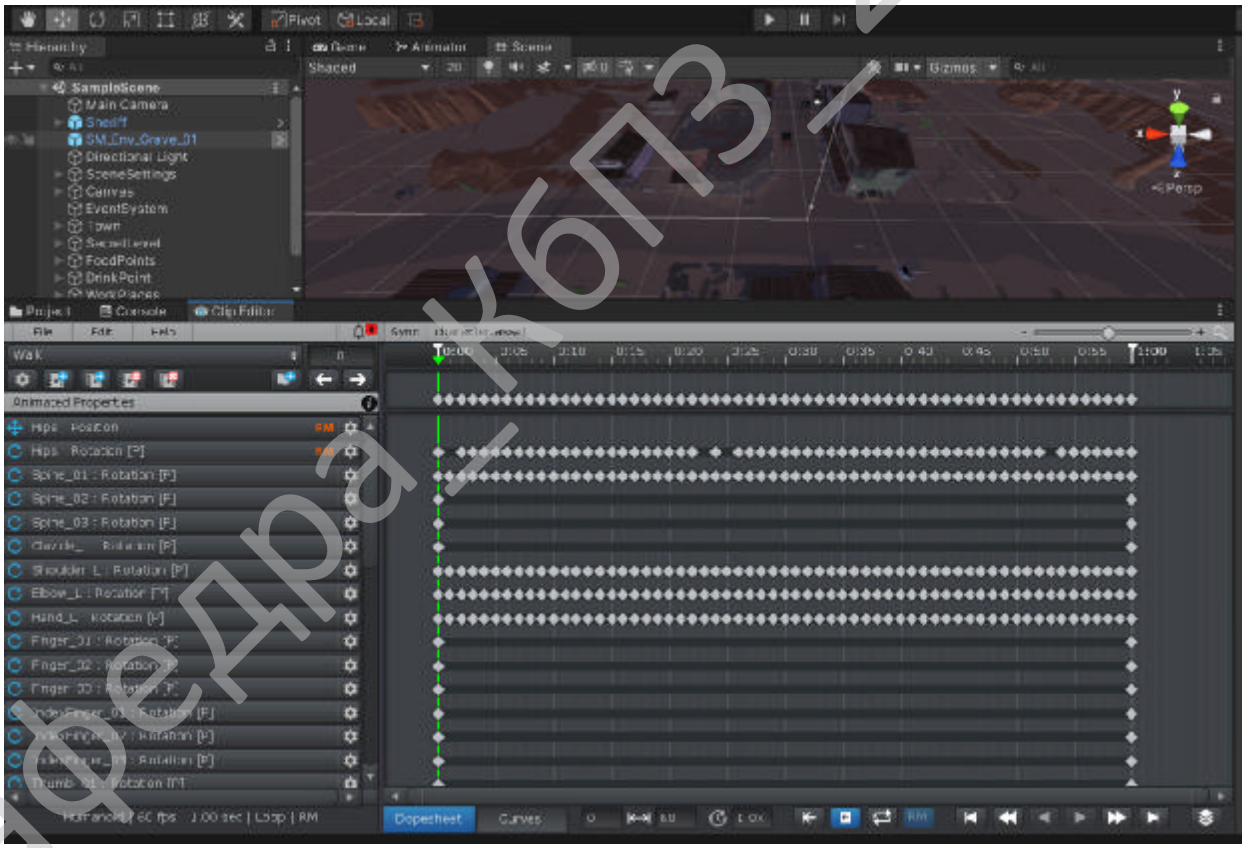

Рисунок 3.2 – UmotionEditor, вікно по керуванню записом анімацій

3.2 Розробка структурної схеми

Структурна схема розробленого програмного забезпечення зображена на рисунку 3.3.

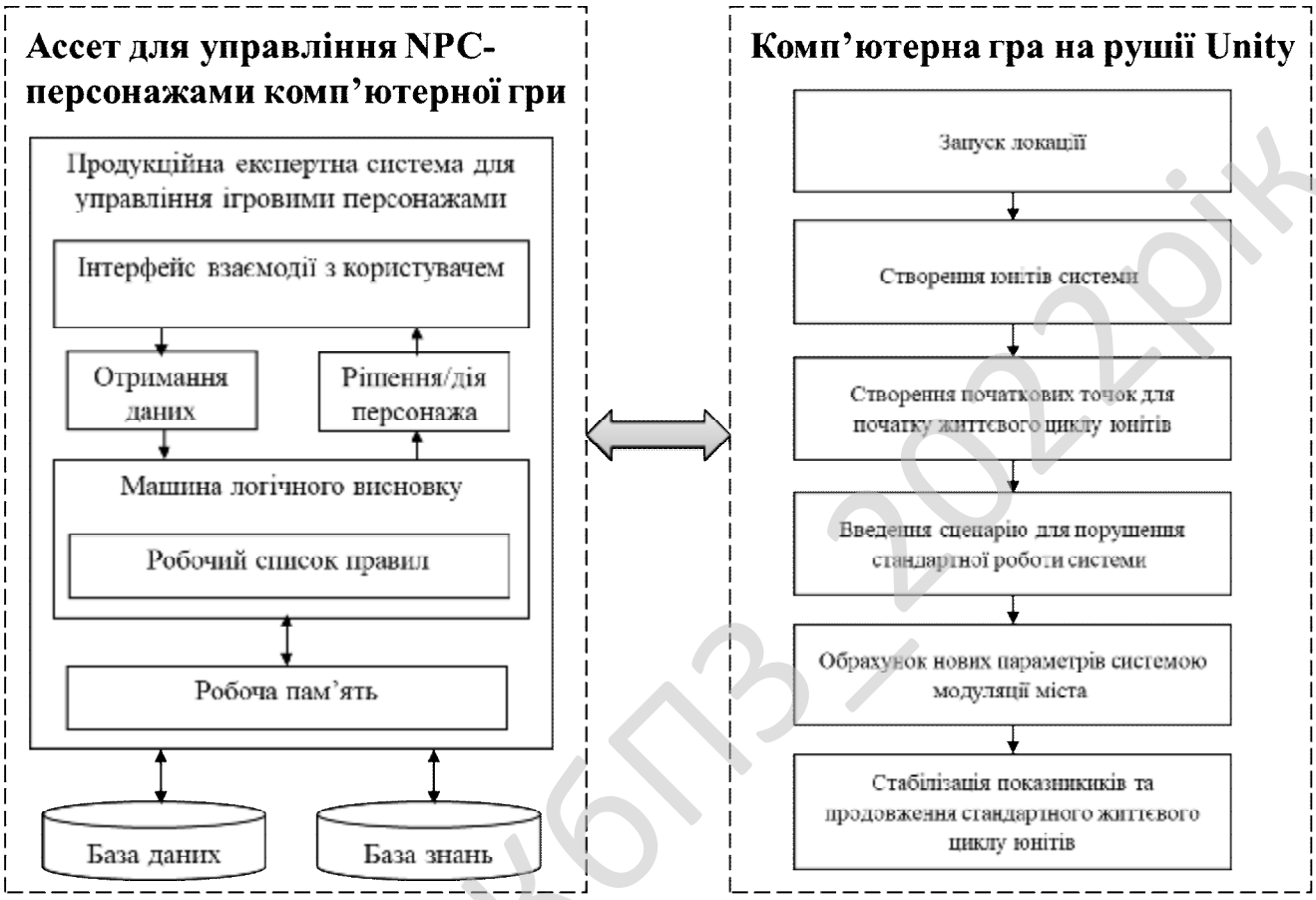

Рисунок 3.3 – Структурна схема системи

На рисунку видно, що система складається з наступних частин:

- запуск основної локації;
- створення юнітів системи;
- створення початкових точок інтересу;
- введення сценарію для подальшого життєвого циклу агентів;
- обрахунок нових параметрів, що можуть з'явитися при запуску системного подразника;
- стабілізація показників та системи в цілому.

Необхідно розглянути кожен блок окремо.

Запуск локації та створення агентів системи, два дуже важливих процеси, що розпочинають роботу усієї системи в цілому. На цьому етапі створюється середовище у якому агенти будуть налагоджувати свій життєвий цикл, а також самі юніти. Також на цьому рівні виконується налаштування показників юнітів та відбувається видача перших цілей.

Далі система налаштовує основні точки маршрутів для агентів, запам’ятовує їх місцезнаходження задане гейм-дизайнером, а також їх тип, призначення, тощо. Надалі це спростить навігацію для юнітів, а також передачу інформації того, що знаходиться на сцені, від менеджерів до потрібного агента.

Після цих трьох кроків відбувається запуск основного сценарію по якому буде функціонувати система. Таким чином буде чітко визначеною стандартна поведінка система у своєму звичному циклі.

Обрахунок нових параметрів відбувається коли у систему запускається подразник, що порушує стандартний життєвий цикл системи чи агента, у такому випадку система намагається якомога швидше адаптуватися до нових умов.

І нарешті блок зі стабілізацією параметрів завершує процес адаптації системи до нового подразника.

3.3 Розробка функціональної схеми

При розробці функціональної схеми експертної системи для управління персонажами у комп’ютерній грі необхідно поставити перед собою задачу доволі чітко та просто описати загальну структуру створюваного програмного забезпечення. Дана схема повинна включати у себе відображення зв’язків між блоками, що представляють собою схематичне зображення різних модулів системи.

Функціональна схема системи наведена на рис. 3.4.

Рисунок 3.4 – Функціональна схема системи

Були виділені основні підсистеми, що будуть забезпечувати функціювання моделі у тому виді, як вона була запланована. По-перше це підсистема для базової взаємодії з користувачем, для того, щоб гравець міг мінімально впливати на ситуацію у середовищі. Надалі виділено, ще низку підсистем: файлова підсистема, підсистема управління агентами, а також підсистема, що керуватиме системними подразниками.

Як видно на схемі представленій вище усі заплановані підсистеми зв’язані між собою та представлені так, як вони повинні працювати у створеній моделі міста.

Підсистема взаємодії з користувачем пов’язана з основною підсистемою і може мати деякий вплив на неї.

Підсистема управління агентами та їх параметрами включає у себе дві менші підсистеми, що забезпечують коректну роботу юнітів, налаштування їх параметрів, а також допомагають налаштувати середовище для існування агентів. Крім цього основна підсистем включає у себе експертну систему, базу даних та базу знань, у якій будуть зберігатися уся інформація про функціонуючі юніти, а також інформація про середовище.

Також варто зазначити, що підсистема подразників буде керувати усіма процесами, що не являються звичними для системи, а також матиме у собі підсистему по візуалізації даних процесів, для того щоб симулювати справжні різкі ситуації, що можуть трапитися у місті, а також відображати адаптування системи до них.

І остання підсистема, що взаємодіє з усіма іншими це – файлова підсистема. У майбутньому при подальшому розвитку програмного забезпечення планується зберігання вже адаптованої системи у файл і подальше завантаження цієї інформації у нову систему, для швидкого пристосування.

3.4 Розробка діаграми процесів

На рисунку 3.5 зображена діаграма процесів, що описує наявні у системі процеси та їх взаємодію.

На початку роботи основного процесу користувач має розпочати сесію, що запустить цілу низку підпроцесів, які будуть утворювати єдину системи. Таким чином можна буде виявити чи достатньо система самодостатня та можна побачити чи має кожен процес логічне завершення і перевірити наявність тупиків, що можуть повністю зруйнувати побудовану архітектуру.

Також слід враховувати послідовність дій, що можуть призвести до різних результатів, позитивних чи негативних.

Наприклад, система не повинна входити у процес стандартного життєвого циклу при подразнені, без проходу процесу адаптації та стабілізування експертною системою. Такі малі деталі створюють цілісну картину програмного забезпечення і дозволяють регулювати його.

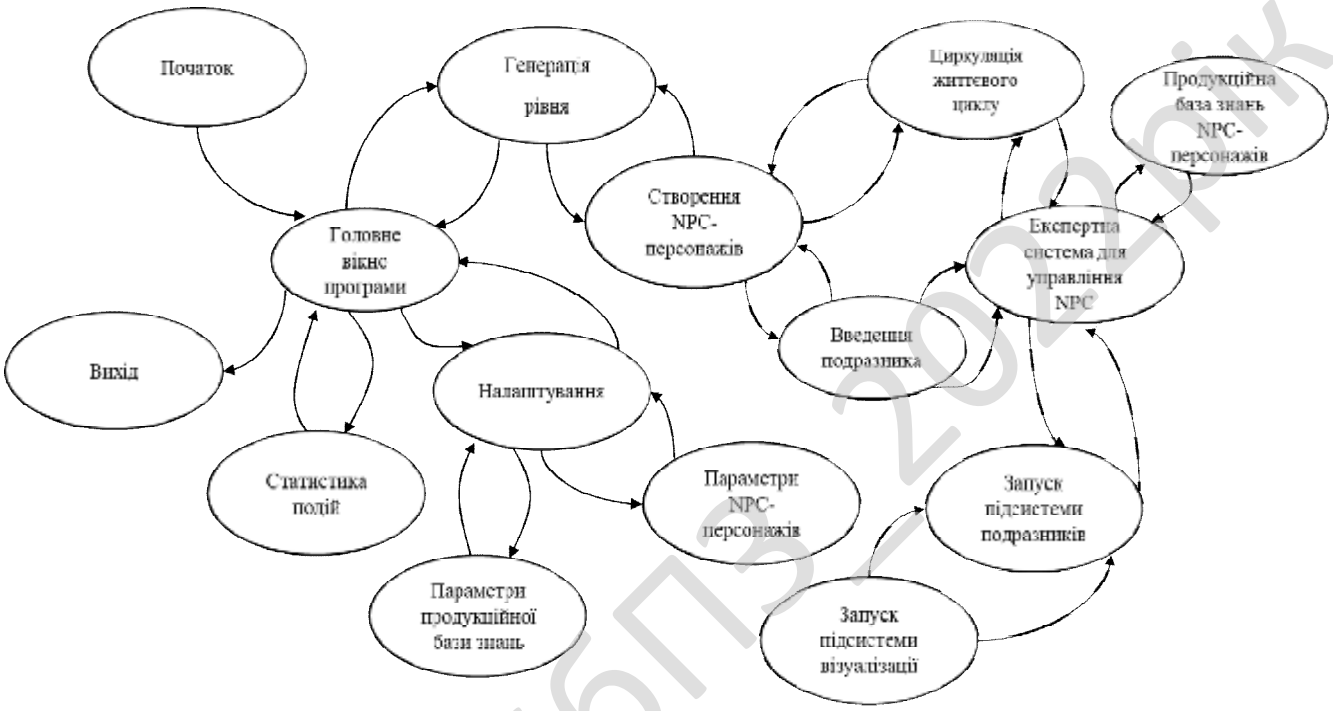

Рисунок 3.5 – Діаграма процесів системи

У розробленій системі присутні наступні процеси:

- Головне вікно програми;
- Генерація рівня;
- Створення NPC-персонажів;
- Циркуляція життєвого циклу;
- Введення подразника;
- Експертна система для управління NPC;
- Продукційна база знань NPC- персонажів;
- Запуск підсистеми подразників;
- Запуск підсистеми візуалізації;

- Налаштування;
- Параметри NPC-персонажів;
- Параметри продукційної бази знань;
- Статистика подій.

Розроблена діаграма відображає послідовність роботи основних процесів, що являються рушіями усієї моделі, що повністю підтверджує її повноту як графічного зображення протікання роботи системи та її підсистем.

Тож, опис структури, функціоналу і процесів системи повністю виконані та готові до використання. Це дозволяє перейти від проектування до розробки системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Було визначено основний життєвий цикл роботи програми, її основні залежності, а також підсистеми, що керуватимуть різними сферами діяльності.

Робота програми починається з моменту створення першого юніта. Це запускає низку підпроцесів, які повинні будуть допомогти усім агентам розпочати свої життєві цикли і виконувати поставлені перед ними задачі. Для цього їм необхідно буде передати початкові параметри, а також деякі класи, з якими вони зможуть взаємодіяти.

Для цього було створено клас UnitManager, який займається створенням агентів, а також передає їм усі необхідні дані. Нижче представлено функцію по створенню деякого юніта:

```
private void CreateUnitEntity(GameObject unit, bool isEnemy = false)
    {
        UnitConctoller person = unit.GetComponent<UnitConctoller>();
        if(isEnemy)
        {
            person.SetUpEnemy();
        }
        int tmp = Random.Range(m_minRandValue, m_maxRandValue);
        person.Hunger = tmp;
        tmp = Random.Range(m_minRandValue, m_maxRandValue);
        person.Thirst = tmp;
        person.Health = m_maxRandValue;
        person.GameName = m_names[Random.Range(0, m_names.Count - 1)];
        m_unitsList.Add(person);
        person.GenerateHouseList(m_houseManager.GetAvailableHouses());
        person.OnNeedDrink += GetDrinkBuilding;
        person.OnNeedEat += GetEatBuilding;
```

```
        person.OnWorkFinding += GetWorkPlace;
        person.OnWalk += GetWalkPoint;
            }
```

Як можна побачити на кожному юніті є компонент UnitController, що представляє собою клас агента і, що найголовніше, він може використовуватися для різних юнітів, що виконують різні функції. Це дозволить гейм дизайнерам швидше розібратися з тим як влаштована базова система керування життєвим циклом моделі.

Наприклад, цей клас забезпечує пошук житлових точок для проживання юнітів, отримання даних та пошук маршрутів до точок поповнення ресурсів та робочого місця, і найважливіший функціонал цього класу це урегулювання конфліктних ситуацій між параметрами агента, а також керування його циклом життєдіяльності. Зверху усього цього цей клас відповідний за передачу юніту інформації про подразник, що потрапив до системи.

Далі запускається налаштування усіх інших параметрів системи, налаштування дня та ночі, дня тижня, плин часу, тощо. І крім цього відбувається установка точок інтересів для юнітів.

Так як дана система виконується у циклі то кожна ітерація перевіряє кількість життєздатних юнітів, показники їх параметрів, а також показники середовища, в якому вони знаходяться. Коли кількість юнітів падає до нуля система перестає функціонувати і відбувається її вимкнення.

Крім цього у циклі роботи системи відбуваються періодичні перевірки на присутність подразників. Якщо такі присутні, система починає адаптуватися до тих умов, що почали змінюватися раптово, для того щоб зберегти свою функціональну здатність. На виході система має повністю пристосуватися до нової ситуації і продовжувати працювати у звичному ритмі, при чому система не має знати заздалегідь про те як адаптуватися. Для того щоб більш чітко визначити принцип роботи основної програми була створена блок-схема продемонстрована на рисунку 4.1.

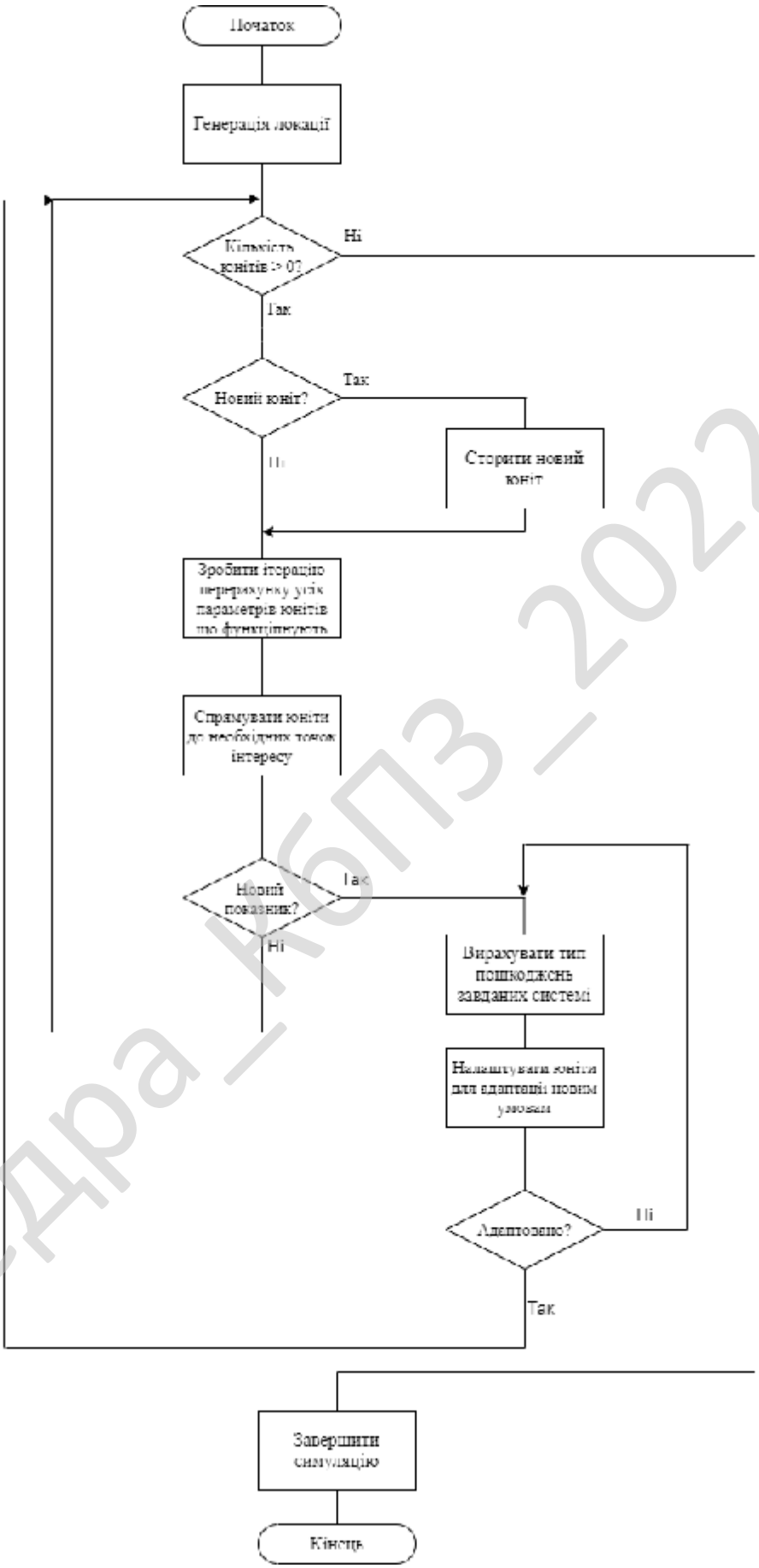

Рисунок 4.1 – Блок-схема основної програми

Як можна побачити, на даній блок схемі дуже добре видно як працює система в цілому. Усе, що було описано вище подано у графічному вигляді. Це дозволяє дослідити правильність проектування, а також методу розробки даного програмного забезпечення.

Надалі необхідно більш детально описати алгоритм роботи підсистеми, що керує агентом окремо, але насамперед потрібно розібрати алгоритм роботи підсистеми, що дозволяє працювати подразникам.

Перш за все є потреба сказати, що запуск подразників планувався як задовільно так і за втручання користувача, але на даний момент реалізовано тульки самозапуск.

Архітектурно дана система розроблялася для максимальної гнучкості налаштування. Тобто, для того, щоб створити новий подразник з унікальними властивостями розробнику, або гейм дизайнеру необхідно створити свій довільний клас подразника і унаслідувати його від класу Invasion, який є базовим класом для кожного подразника. Реалізація базового класу представлена нижче:

```
public abstract class Invasion : MonoBehaviour
{
    public InvationType Type { get; set; } = InvationType.robber;

    [SerializeField]
    private DayTime m_activationTime;
    [SerializeField]
    private int m_successPercent;

    protected List<UnitConctoller> m_unitList;
    protected LightController m_dayController;
    protected UnitConctoller m_target;

    private IEnumerator m_lifeRoutine;
    #region MonoBehaviour
    private void Start()
    {
        m_lifeRoutine = ActionRoutine();
        StartCoroutine(m_lifeRoutine);
```

```
    }

    private void OnDestroy()
    {
        if(m_lifeRoutine != null)
        {
            StopCoroutine(m_lifeRoutine);
        }
    }
    #endregion

    public void SetUpInvasion(List<UnitConctoller> list, LightController day)
    {
        m_unitList = list;
        m_dayController = day;
    }

    protected IEnumerator ActionRoutine()
    {
        while(true)
        {
            yield return new WaitForSecondsRealtime(60f);

            if(m_dayController.LightState == m_activationTime)
            {
                SelectAction();
            }
        }
    }

    protected void SelectAction()
    {
        int index = Random.Range(0, m_unitList.Count - 1);
        m_target = m_unitList[index];

        m_target.ReactOnInvasion(this, m_successPercent, DoSuccess, DoFault);
    }

    public abstract void DoFault();
    public abstract void DoSuccess();

}
```

Як можна побачити дії що буде виконувати агент при необхідності відреагувати на подразник не визначені заздалегідь у базовому класі. Вони визначаються при створенні унікального подразника. Крім цього у кожного агента є метод ReactOnInvasion(Invasion invasion, int successPercent, Action sucess, Action failure), у який передається саме вторгнення, мінімальний поріг для визначення успішності реакції, а також для методу, що будуть викликані у разі успіху чи невдачі. Така побудова архітектури дозволяє з легкістю налаштовувати і створювати подразники для системи.

Після запуску відбувається ініціалізація подразника. Через UnitManager подразник має доступ до кожного окремого агента, що дозволяє йому вільно взаємодіяти з ними.

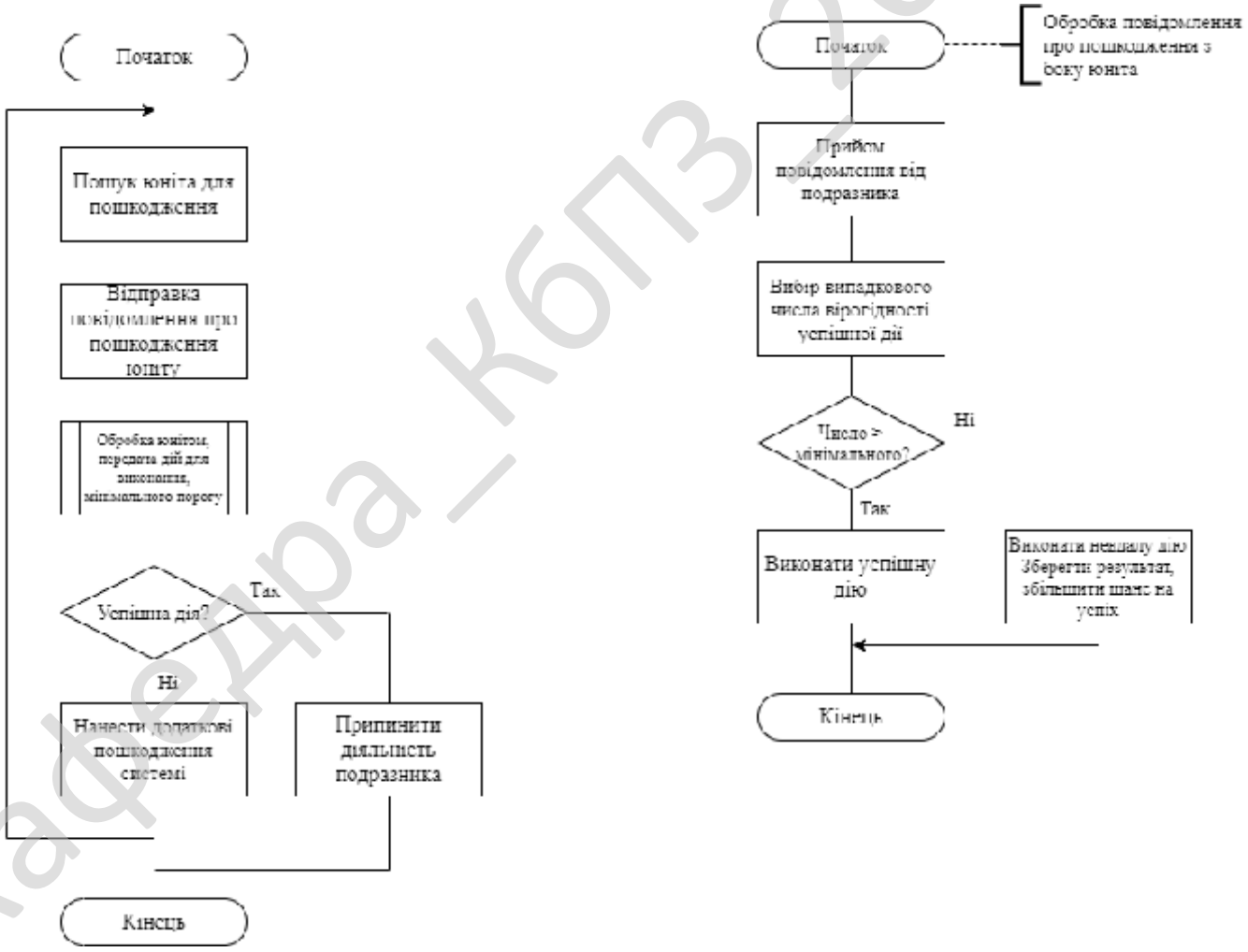

Рисунок 4.2 – Блок-схема алгоритму роботи подразника юнітів

Як можна побачити, на блок-схемах представлених у тексті магістерської роботи вище, алгоритм роботи вторгнення у систему є визначеним, чітким та скінченим. Крім цього як раніше було сказано дуже легкий для розуміння і відтворення.

Після одержання інформації про усіх агантів, що функціонують на сцені програми виконується передача інформації про подразнення через метод ReactOnInvasion у юніта і таким чином юніт на своїй стороні спочатку розуміє, що йому прийшло таке повідомлення, реєструє його і запам'ятовує тип пошкодження, що заважає йому функціонувати стабільно. Далі виконується прорахунок можливості виконати необхідні дії успішно і у разі позитивного результату агент зберігає такий варіант вирішення проблеми, що дозволяє її вирішити з шансами на успіх рівними ста відсоткам.

У разі невдачі виконується дія, яка ніяк не впливає на подразник але може призвести до більших пошкоджень системі. Але такий результат дозволяє агенту навчатися і, якщо він знову зіткнеться з проблемою конкретного типу, на яку він відреагував некоректно, шанси на успіх будуть значно вищими. Функція обробки даних по подразнику:

```
public void ReactOnInvasion(Invasion invasion, int successPercent, Action sucess,
Action failure)
    {
        int percent = (UnityEngine.Random.Range(0, 1000) * 100) / 1000;

        ReactionEnity reaction = m_reactions.Find(x => x.invasion == invasion);

        if(reaction == null)
        {
            reaction = new ReactionEnity();
            reaction.invasion = invasion;
            reaction.successPercent = successPercent;
            m_reactions.Add(reaction);
        }

        if(percent > 100 - reaction.successPercent)
```

```
        {
            sucess.Invoke();
            reaction.successPercent = 100;
        }
        else
        {
            failure.Invoke();
            reaction.successPercent += 10;
        }

        m_isReacted = true;
           }
```

Функція є досить простою для розуміння, але досить ефективною для вирішення поставлених задач.

Отже, підбивши підсумки по роботі системи загалом та підсистеми подразників можна визначити, що аналіз роботи алгоритмів був проведений успішно, кожен алгоритм чітко вирішує поставлені перед ними проблеми і виконує свою роботу дуже ефективно. Крім цього можна сказати, що написання коду та будування архітектури, що буде базуватися на даних принципах досить легко, а що найголовніше гнучко. Така гнучкість дозволяє виходити за рамки стандартної роботи і створювати різноманітні системи по моделюванню міст у ігрових, і не тільки, проектах. Це відкриває гейм дизайнерам величезний простір для створення широкого спектру серидовищ.

Але крім роботи самої системи в ній повинні бути активні агенти для коректної роботи, адже і робота підсистеми пошкоджень і процеси, що протікають у життєвому циклі моделі потребують функціонуючих юнітів у серидовищі.

При побудуванні основних принципів взаємодії агенті з усіма процесами необхідно пам’ятати про правила, що були зазначені у таблиці 3.1. Керуючись ними можна створити базу для моделювання NPC, що зможуть поводитись максимально наближено до поведінки людей у реальному світі.

Перш за все базовим класом для роботи агента, функціонування його життєвого циклу, являється згаданий вище UnitController, який можна виставити у стандартний тип агента або ж якийсь специфічний. При цьому юніти поведінка яких буде відрізнятися від стандартної повинні будуть наслідуватися від цього класу, адже саме у ньому прописані основні правила життєдіяльності агентів, а також пошук маршрутів, встановлення правильних анімацій, тощо.

Коли об'єкт цього класу створюється на сцені у ньому запускається життєвий цикл для агента:

```
private IEnumerator LifeCircle()
    {
        while(Health != 0)
        {
            if(m_isDead)
            {
                yield break;
            }

            yield return new WaitForSecondsRealtime(5f);

            if(Hunger == 0 || Thirst == 0)
            {
                Health -= 15;
            }
            else
            {
                if(UnitManager.State == EnvironmentState.normal)
                {
                    Hunger -= UnityEngine.Random.Range(1, 2);
                    Thirst -= UnityEngine.Random.Range(1, 3);
                    Fatigue -= UnityEngine.Random.Range(1, 5);
                    if (Fatigue <= 0)
                    {
                        Fatigue = 0;
                    }
                }
            }
        }
```

```
            m_isDead = true;
}
```

Цей невеликий метод визначає у якому стані зараз юніт, віднімає його показники і визначає коли він перестає функціонувати. Тобто можна сказати, що це серце жителя віртуального міста, що виконує дуже важливу функцію.

Також велику роль у функціонуванні юніта грає процес відстеження показників та їх поповнення. Кожен фрейм ігрового часу процес повинен досліджувати стан життєвоважливих показників і при необхідності направляє агента на їх поповнення згідно з правилами. Наприклад, якщо є дефіцит параметру спраги дана підсистеми перевіряє чи не зайнятий юніт поповненням іншого ресурсу, якщо він вільний, по підсистема має направити його на поповнення, якщо ж зайнятий, то необхідно запустити чергу очікування для поповнення.

Також цей клас отримує інформацію про стан навколишнього середовища через UnitManager, що у свою чергу пов’язаний з класом LightController. Ці дані необхідні для того, щоб кожен агент знав стан дня, коли необхідно йти на робоче місце, а коли на житлову точку. Крім цього така взаємодія задає правильний графік для більш збалансованого налаштування усіх агентів.

Інформацію про робочі місця, а також про точки поповнення ресурсів та житлові точки включає у себе HouseManager, у якому зберігаються усі місцезнаходження точок, а також тип будівлі та відносно типу визначаються додаткові параметри.

Вищеописаний алгоритм слід представити графічно – блок-схемою, що має бути досить простою для розуміння, відображати усю повноту функціоналу розробленого алгоритму поведінки, а також буде відповідати стандартам.

Для цього необхідно провести аналіз алгоритму на усі точки входу та виходу, розробити правильну математичну і логічну моделі і перевірити усі вразливості, що можуть проявитися у подальшій роботі з ним. Це допоможе скоротити час та зменшити витрачання ресурсів на побудову блок-схеми.

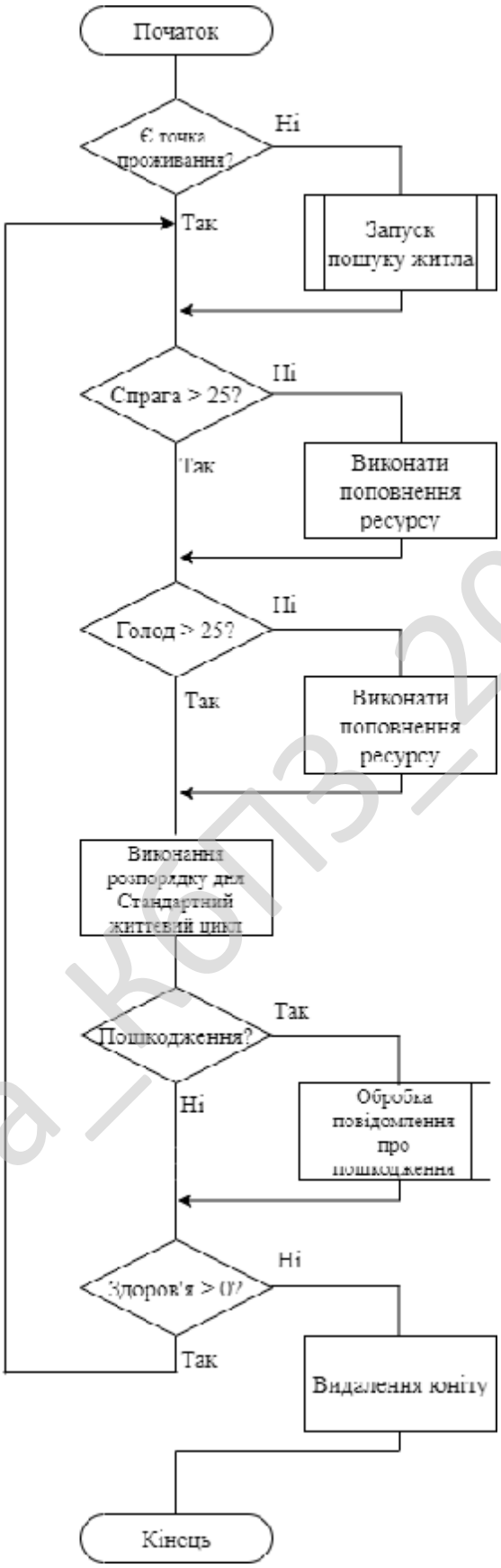

Рисунок 4.3 – Блок-схема роботи алгоритму по керуванню юніта

Рисунок 4.4 – Блок-схема алгоритму знаходження житлової точки

Взявши до уваги усе вищесказане, а також блок-схеми алгоритмів по керуванню агентами, можна з упевненістю сказати, що спланована архітектура як системи так і окремих юнітів повністю оправдовує очікування і дуже добре виконує свою роботи.

Як висновок можна заявити, що така модель може спростити задачі розробникам ігрового програмного забезпечення і пришвидшити їх розробку, що є важливим показником і демонструє необхідність продовження розвитку експертної системи для управління персонажами у комп’ютерній грі.

4.2 Захист розробленого програмного забезпечення

Для захисту програмного забезпечення від декомпілювання бібліотеки в яку воно буде скомпільоване і збереження основних функцій ніким не зміненими був обраний метод обфускації.

Обфускація або заплутування коду – приведення початкового коду або виконуваного програмного коду до вигляду, що зберігає його функціонал та робочі процеси, але ускаладнює його аналіз, розуміння роботи алгоритмів та процесів, заважає вносити модифікації у вже створену бібліотеку.

Заплутування коду може буди застосовано на рівні алгоритму, початкового коду або асемблерного коду. Для цього можуть використовуватися спеціальні компілятори, які використовують неочевидні або незадокоментовані можливості середовища виконання процесів. Існують також спеціальні програми, що виконують обфускацію, так звані обфускатори.

Такий підхід хоч і досить простий але може допомогти зберігати функціонал системи без змін з боку розробників, а також завадить легко скопіювати розроблену модель.

Основні цілі обфускації:

− ускладнення декомпіляції пропрієтарних програм з метою копіювання функціоналу;

− ускладнення декомпіляції пропрієтарних програм з метою запобігання зворотної розробки або обходу ліцензій;

− приховання авторства коду;

− оптимізація програми з метою зменшення розміру працюючого коду і прискорення роботи.

Таким чином створене програмне забезпечення буде мати мінімальний захист необхідний для подальшої інтеграції у виробничий процес, що забезпечить безпеку для створеного коду, а також завадить копіюванню функціоналу системи.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Отже, у фінальній частині розробки програмного забезпечення, що реалізує експертну систему для управління персонажами у комп'ютерній грі необхідно визначити як інтегрувати її у звичний робочий процес.

Перший пункт інтеграції це визначення способу розповсюдження програмного забезпечення для його подальшого використання. Так як модель являє собою набір асетів для швидкої розробки комп'ютерних ігор з NPC-персонажами і цей набір розроблений за допомогою рушія Unity для його зручного використання логічною є потреба опублікувати його в Unity Asset Store. У цьому магазині кожен розробник, що потребує якийсь додатковий функціонал до свого розроблюваного програмного забезпечення, може з легкістю знайти його. Також це дає змогу оновлювати модель і постачати кінцевому користувачу найсвіжішу версію асетів. Тобто такий спосіб розповсюдження є найбільш пріорітетним. Так як дозволяє і актуалізувати версії так і доносити свої розробки до користувачів з надзвичайною легкістю.

Інший спосіб розповсюдження, який є також досить популярним серед розробників ігрового програмного забезпечення це використання репозиторіїв і викладення системи у вільний доступ, як Open Source проект. Це дозволяє розробникам самим приймати участь у оновленні та покращені розробленої моделі. Це може бути досить вигідно, але є велика загроза безпеці системи, а також ймовірність копіювання функціоналу. Для такого способу розповсюдження необхідно змінювати спосіб захисту розробленого програмного забезпечення, що був би достатній для магазину асетів Unity.

Крім цього необхідно написати детальну документацію по користуванню даним набором асетів, насамперед її треба розділити на частини для розробників та геймдизайнерів. Для розробників описується функціонал моделі, основні класи

та зв’язки між ними. Крім цього слід зазначити базові класи за допомогою яких можна розширити уже існуючий функціонал.

Для геймдизайнерів є потреба в описанні як саме налаштувати середовище, а також як розташувати необхідні точки з компонентами на самій сцені.

Крім цього у документації мають бути описані основні цілі програмного забезпечення, його призначення та можливості.

І останнім пунктом у інтеграцію системи, що була розроблена, являється створення тестової сцени, що вже є налаштованою і служить для демонстрації того, що функціонал моделі у нормі і може виконувати свої задачі. Крім цього створення сцени-прикладу, допоможе провести рекламну діяльність стосовно програмного забезпечення по моделюванню жителів міста, що є важливим пунктом у її просуненні та інтеграції у робочий процес великих компаній.

Рисунок 5.1 – Демонстраційна сцена з активним агентом

Таким чином можна провести інтеграцію розробленої системи у робочій процес досить швидко та ефективно, не витрачаючи занадто великої кількості ресурсів та сил.

6 НАУКОВА НОВИЗНА

У магістерській роботі розроблено програмне забезпечення, яке призначено для експертної системи для управління персонажами у комп'ютерній грі.

Метою роботи є дослідження та програмна реалізація експертної системи для управління персонажами у комп'ютерній грі.

Об'єктом дослідження є процес інтелектуального управління персонажами у комп'ютерній грі.

Предметом дослідження є методи побудови експертних та інтелектуальних систем, методи інтелектуального управління персонажами у комп'ютерних іграх.

Методи дослідження базуються на методах штучного інтелекту, методах математичної статистики, методах розробки програмного забезпечення, методах розробки комп'ютерних ігор.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Запропоновано метод управління персонажами у комп'ютерній грі на основі продукційної експертної системи.

2. Розроблено вітчизняний продукт інтелектуального управління персонажами у комп'ютерній грі, який має більш широкі можливості, на відміну від існуючих аналогів та може бути використаний при розробці різних відеоігор.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми магістерської роботи

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці).

В магістерській роботі було проведене дослідження та виконана програмна реалізація експертної системи для управління персонажами у комп'ютерній грі.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність

Таблиця 7.1 - Початкові данні

Показники	Позна-чення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт	N	1
2. Кількість екземплярів програм, шт	Ne	50
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2
7. Кількість макетів вхідної інформації	–	3

Продовження табл. 7.1

1	2	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	1
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	_	2
13. Рівень спрацьованості колективу (1-6)	_	2
14. Ступінь вимірності процесів (1-6)	_	3
15. Необхідна надійність програмного забезпечення (1-6)	_	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	_	2
17. Складність кінцевого програмного продукту (1-6)	_	2
18. Необхідний рівень забезпечення повторного використання (1-6)	_	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	_	2
20. Вимоги до швидкодії ПП (1-6)	_	2
21. Обмеження на розміри основного сховища даних (1-6)	_	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	_	2
23. Професійний рівень аналітиків (1-6)	_	2
24. Професійний рівень програмістів (1-6)	_	2
25. Постійність складу команди розробників (1-6)	_	2
26. Досвід розробки додатків (1-6)	_	2
27. Досвід роботи з обчислювальною платформою (1-6)	_	2

Продовження табл. 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПО для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн	–	50000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	55
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \, \text{Size}^B , \qquad (7.1)$$

де A - коефіцієнт Боема, A=2,45; Size - загальний об'єм відлагодженого програмного коду, тис. рядків; B - показник ступеня, що визначається співвідношенням

$$B = 1{,}01 + 0{,}001\sum W_i \qquad (7.2)$$

де Wi - сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B=1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}026$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн}=T_{ном}\, ПВ_j, \qquad (7.3)$$

де $ПВj$ - добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн}= 6{,}78\cdot(0{,}88\cdot0{,}93\cdot0{,}88\cdot0{,}91\cdot0{,}95\cdot1\cdot1\cdot0{,}87\cdot1{,}22\cdot1{,}16\cdot1{,}1\cdot1{,}1\cdot$$
$$\cdot1{,}12\cdot1{,}1\cdot1\cdot1{,}1\cdot1{,}1)=9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3CT_{уточн}^{0{,}33+0{,}2(B-1{,}01)}S, \qquad (7.4)$$

де С - визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); S - коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПО згідно встановленим вимогам. Вибираємо в межах (25…350)%

$$T_{РП}=0{,}3\cdot3{,}23\cdot9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)}\cdot75=153 \text{ люд/день}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 - Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	153	Ф 7.1-7.4
Впровадження	13	Д13
Всього	194	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де Fpq - плановий фонд робочого часу одного спеціаліста, днів, Тпз – трудомісткість розробки програмного забезпечення люд-дні,

$$Ч = \frac{194 \cdot 1}{60\text{-}5} = 3,5 \text{ ставки}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3

Таблиця 7.3 - Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	**Профілактичне обслуговування**			
	Кількість хв. на один. обл.	**Кількість облад-нання**	**Затрати часу в хв.**	**Затрати часу в год.**
Системний блок ПК	90	10	900	15
Монітор	60	10	600	10
Клавіатура	30	10	300	5
Маніпулятор «мишка»	30	10	300	5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор–маршрутизатор	30	3	90	1,5
Кабельні господарства ЛВС на 1 м. п.	2,5	250	625	10,42
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	52,58

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{міс}}{1,2} \quad (7.6)$$

$$Ф_{др}^{с} = \frac{53 \cdot 3}{1,2} = 132,5 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}} \quad (7.7)$$

$Ч_{ел}$=132,5/(60·8)=0,28 ставки

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів – електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 - Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	**Вид роботи**	**Час**	**Кількість штатних одиниць**
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, серверу доступу АДСЛ (ОС Linux), Wi-Fi налаштування ADSL, VPN, PPPoE, Frame Relay	0,5	0,25
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	0,5	
Всього		2	

Продовження таблиці 7.4

Посада	Вид роботи	Час	Кількість штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,25	
	Контроль взаєморозрахунків з постачальниками	0,25	
Всього		2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,25
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Складемо штатний розклад виконавців:

Таблиця 7.5 - Штатний розклад виконавців

Посада	Кількість ставок	Середньо-місячний оклад, грн.	Всього за період роз-робки, грн.
Керівник (IT-менеджер)	1	16000	48000
Продакт-менеджер	0,25	14000	10500
Інженер-програміст	3,5	15000	157500
Інженер-електронщик	0,28	12000	10080
Інженер-системотехнік	0,25	12000	9000
Адміністратор мережі	0,25	13000	9750
Системний програміст	0,25	13000	9750
Дизайнер WEB	0,25	15000	11250
Інженер-верстальник	0,25	14000	10500
Бухгалтер-економіст	0,5	15000	22500
Всього за період розробки	$R_{сп}$=6,78	-	$Ф_{роб}$=298830

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де Фроб – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{298830}{6,78 \cdot 60} = 734 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць. Sу – питома площа на одне робоче місце, $м^2$; $Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно $8 м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн на одне робоче місце. Тобто

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де Цм – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу Інтернет магазину Компбест за 16.11.22 – джерело https://compbest.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	**Тип**	**Оптова ціна**
Персональний комп’ютер		10947
Системний блок		7347
Процесор	Socket 1151 Intel Core i3-6100 (2 (4) ядра по 3.7 GHz) BOX	-
Системна плата	FUJITSU Mainboard D3402-B µATX. 5 x Serial ATA III 600 Interface (up to 6GBit/s, NCQ, AHCI, RAID 0/1/5/10) 1 x M.2 Socket (2260/2280) for PCIe (4 lanes) or SATA based M.2 modules, AHCI Boot support	-
Відеокарта	Вбудована в процесор Intel HD Graphics 530	-
Жорсткий диск	1000 GB HDD + SSD M.2 2280 250GB ADATA	-
Оперативна пам’ять	8 GB DDR4 (2 модулі по 4 GB)	-
DVD-привод	DVD -RW/+RW , LG SATA SuperMulti Bulk 22x, SecurDisc, black	-
Корпус	Terra Tower, 400W (120mm big fan), full-ATX, БЖ 4xSATA, Air Duct, 2xUSB 3.0, Mic+Audio, silver/black	-
Кардрідер внутрішній	USB 3.0 Card reader STORM CR, int. 3.5", 2*USB3.0+AUDIO+1394, multi: All Type Cards, black	-
інше	Клавіатура, мишка, DISPLAY PORT TO HDMI 1.0M CABLEXPERT	-

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	**Тип**	**Оптова ціна**
Монітор	LG W2363V-WF Wide LCD 2ms, 70 000:1, 300кд/м2, 170/160, D-Sub / Glossy White	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струменевий	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 - Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	**Кількість, шт.**	**Ціна за одиницю, грн.**	**Витрати на транспортування, монтаж та випробовування.**	**Загальна вартість, грн.**
Персональні комп’ютери	8	10947	8757,6	96333,6
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	114885,1

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	**Балансова вартість, грн.**	**Амортизація**	
		Норма, %	**Відрахування, грн.**
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400
Група 4			
3. Обчислювальна техніка	114885	-	-
Всього по групі	114885	50	57442,5
Група 5			
4. Вимірювальні пристрої	5190	-	-
5. Господарський інвентар	28000	-	-
Всього по групі	33190	25	8297,5
Нематеріальні активи			
6. Нематеріальні активи	50000	10	5000
Разом	$К_р$= 1606075		$А_р$= 141140

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_e}, \quad (7.11)$$

де Ne – Кількість екземплярів програм, шт.

$$З_о = 734 \cdot 194/50 = 2848 \text{ грн}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де $Н_q$ – норматив додаткової зарплати, %

$$З_д = 2848 \cdot 10 \cdot 0{,}01 = 285 \text{ грн}$$

Відрахування на соціальні потреби за нормативом Нс=22% від суми основної та додаткової зарплати

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де $Н_с$ – відрахування на соціальні потреби, %

$$С_{оц} = 0{,}01 \cdot 22(2848+285) = 689 \text{ грн}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом Нг=15% від основної зарплати

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де $Н_г$ – загальногосподарські витрати, %

$$Г_{осп} = 2848 \cdot 15 \cdot 0{,}01 = 427 \text{ грн}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де $З_{М1}$ – вартість паперу, грн., $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн., $З_{М3}$ – вартість фарби, картріджей, тонеру, грн., $N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{міс}$ приймаємо 2 пачки паперу на місяць розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=220 грн., визначаємо вартість паперу за період розробки $N_м$ =3 міс:

$$З_{М1} = Ц_п \cdot N_м \, n_{міс}. \quad (7.16)$$

$$З_{М1} = 220 \cdot 3 \cdot 2 = 1320 \text{ грн}.$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості

коробочних версій запропонованого продукту (приймаємо 50):

$$З_{М2} = \sum Ц_{д}., \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 25 грн./шт., DVD-R box – 35 грн./шт.

$$З_{М2} = 50 \cdot 25 + 35 = 1285 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_{з}., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574+558+570=1702 \text{ грн.}$$

$$З_{М} = (1320 + 1285 + 1702)/50=86 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом (Нп = 15%) від основної зарплати виконавців

$$О_{п} = З_{о} \cdot Н_{п} \cdot 0{,}01, \quad (7.19)$$

де Нп - норматив витрат на освоєння нових мов програмування, %

$$О_{п} = 2848 \cdot 15 \cdot 0{,}01=427 \text{ грн}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 50 прим.)

$$А_{м} = \frac{А_{р} \cdot N_{міс}}{N_{е} \cdot 12}, \quad (7.20)$$

де Ар – загальна річна сума амортизаційних відрахувань, грн.

$$А_{м} = 141140 \cdot 3/(50 \cdot 12)=706 \text{ грн}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції

$$С_{п} = З_{о} + З_{д} + С_{оц} + Г_{осп} + З_{м} + О_{п} + А_{м}. \quad (7.21)$$

$$С_{п} = 2848+285+689+427+86+427+706 = 5468 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності (P_n) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 55%

$$П_р = 0{,}01 \cdot P_n \cdot C_n, \quad (7.22)$$

де Рс – рівень рентабельності, %

$$П_р = 0{,}01 \cdot 55 \cdot 5468 = 3007 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	**Позначення**	**Величина, грн.**
1. Основна зарплата виконавців	$З_о$	2848
2. Додаткова зарплата виконавців	$З_д$	285
3. Відрахування на соціальні потреби	$С_{оц}$	689
4. Загальногосподарські витрати	$Г_{осп}$	427
5. Витрати на матеріали	$З_М$	86
6. Освоєння нових операційних систем, мов програмування	$О_п$	427
7. Амортизація основних фондів	$А_м$	706
8. Повна собівартість програмного забезпечення	$С_п$	5468
9. Плановий прибуток	$П_р$	3007
10.Ціна підприємства $Ц_п=С_п+П_р$	$Ц_п$	8475
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	1695
12.Відпускна ціна програмної продукції Ц = Цп + ПДВ	$Ц$	11482

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.9.

Таблиця 7.10 - Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	**Сума за варіантами, грн**	
	Базовий	**Новий**
Вартість програмної продукції	–	11482
Всього капітальних витрат	–	11482

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 - Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	**Позначення**	**Сума витрат за варіантами, грн.**	
		Базовий	**Новий**
1. Витрати на обслуговування	$З_р$	69784	17446
2. Витрати на електроенергію	$З_{ел}$	1045	261
Витрати на амортизацію	$З_{ам}$	0	2871
Всього витрат за рік	$І$	70829	20578

Витрати на оплату праці:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \qquad (7.23)$$

де Тр – кількість годин обслуговування системи за рік, *год.*; Зг - заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення кількість годин на реалізацію поставленої задачі зменшилась з 1000 годин до 250 годин на рік, тому витрати на обслуговування склали:

$$З_{р\ баз} = 1000 \cdot 52 \cdot 1{,}1 \cdot 1{,}22 = 69784\ грн.$$

до

$$З_{р\ нов} = 250 \cdot 52 \cdot 1{,}1 \cdot 1{,}22 = 17446\ грн.$$

Витрати на електроенергію визначаються з урахуванням спожитої потужності (Пел) в кіловатах, часу експлуатації технічних засобів (Тр) в годинах та ціни однієї кіловат-години (Цел).

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \qquad (7.24)$$

Зел баз=·0,475·1000·2,2 = 1045 грн

Зел нов=·0,475·250·2,2 = 261 грн

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 - Розрахунок амортизаційних відрахувань

Групи основних фондів	**Норма амортизації %**	**Балансова вартість, грн., за варіантами**		**Сума відрахувань, грн., за варіантами**	
		Базовий	**Новий**	**Базовий**	**Новий**
Програмна продукція	25	–	11482	–	2870,5
Всього відрахувань	-	–	11482	–	2870,5

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (8475 - 5468) \cdot 50 - (0,05 \cdot 1408000 + 0,5 \cdot 114885 +$$
$$+0,25 \cdot 33190 + 0,1 \cdot 50000) \cdot 3/12 = 115065 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р^*$ – балансова вартість основних фондів розробника.

$$T_в = \frac{1606075}{(8475 - 5468) \cdot 50 \cdot 12 / 3} = 2,67 \text{ років}.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно, $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються

$$E_{сп} = (70829 - 20578) - 0,25 \cdot 11482 = 47380,5 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н} \quad (7.28)$$

$$T_{сп} = \frac{11482}{70829 \ - \ 20578} = 0,23 \ року$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 - Показники економічної ефективності програмної продукції

Найменування показників	**Одиниця виміру**	**Величина**
1. Кількість екземплярів програми	Прим.	50
2. Повна собівартість розробленої програми	Грн.	5468
3. Ціна розробленої програми	Грн.	8475
4. Плановий прибуток від реалізації розробленої програми	Грн.	3007
Рентабельність програмної продукції	%	55
Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1606075
Загальний прибуток від реалізації програмної продукції	Грн.	150350
Величина економічного ефекту при виготовлені програмної продукції	Грн.	115065
Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	2,67
0.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	11482
1.Величина економічного ефекту у користувача програмної продукції	Грн.	47381
2.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,23

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Законом України “Про охорону праці” регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби польців та м’язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

Керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98, та «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18 розглянемо шкідливі чинники роботи персоналу.

Щоб запропонувати заходи щодо зменшення негативного впливу комп'ютера на організм людини визначемо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Електронно-обчислювальна машин (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. У приміщенні, в якому працюють люди (у т.ч. програмісти) необхідно створити належний мікроклімат, праметри якого регламентуються, Державними санітарними правилами і нормами, зокрема ДСанПіН 3.3.2.007-98.

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

- ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.
- ризик виникнення пожежі;
- ризики ураження електричним струмом;
- негативний вплив на органи зору людини;
- недостатня, або надмірна освітленість робочого місця;
- несприятливі мікрокліматичні умови;
- нервово-емоційна напруженість праці;
- інтелектуальні навантаження;
- монотонність праці;
- невідповідність ергономічних показників робочого місця діючим вимогам;
- шум;
- електромагнітні (у т.ч. високочастотні) електромагнітні випромінювання (коливання);
- статичні навантаження на кістково-м'язовий апарат;

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	4
Довжина	5
Висота	3

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого*

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	6,6
Об'єм, V	$м^3$	не менше 20.0	20

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

У зазначеному приміщенні працюють троє людей. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об'єм приміщення у розрахунку на одно робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин», але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»). Таким чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці

програміста відповідають вимогам.

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	**Оптимальні для Іа**			**Фактичні**		
	Температура, ºС	**Воло-гість,%**	**Швидкість повітря, м/с**	**Температура, ºС**	**Воло-гість%**	**Швид-кість повітря, м/с**
Холодна	22-24	40-60	0,1	21-22	46-54	0,1
Тепла	23-25	50-70	0,1	24-25	50-65	0,11

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер *XEROX PHASER 3020BI*, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

З 2019 року діють Державні будівельні норми україни "Природне і штучне освітлення" – ДБН В.2.5-28:2018, у яких прописані вимоги до використання всіх освітлювальних приладів, у т.ч. світлодіодних.

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5-28:2018, можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк., Крім того все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.4 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

- розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

- мікроклімат відповідає нормативному значенню;

- акустичні умови роботи не перевищують нормативних значенньь;

Таким чином можна припустити, що основною причиною можнивого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга).

Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв, які працюють при напрузі вище 36 В.

8.5 Розрахункова частина

Проводемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 4 м,

довжина – 5 м, висота – 3 м.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, необхідно враховувати, що, з одного боку «середня освітленість робочих місць з постійним перебуванням людей повинна бути не менш як 200 люкс», а з іншого «штучне освітлення при системі комбінованого освітлення для зорової роботи найвищої точності повинна складати з00 люкс» та вимогу ДБН В.2.5-28:2018, що “створювати освітленість більше ніж 300 лк при світлодіодних світильникх дозволяється тільки за наявності обґрунтування».

Таким чином для розрахунку приймаэмо середню освітленість робочих місць 300 Люкс.

Визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F = E \cdot S \cdot K \cdot Z / n,$$

де: F - світловий потік, що розраховується, Лм;

E - нормована мінімальна освітленість, Лк; E = 300 Лк.;

S - площа поверхні, на яку падає світловий потік, м^2;

Z - відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку Z = 1,1);

K - коефіцієнт запасу, що враховує зменшення світлового потоку світильники в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку K = 1,5);

n - коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.

Обчислимо індекс приміщення за формулою:

$$i = S / (h \cdot (A + B)),$$

де: S - площа поверхні, на яку падає світловий потік, м^2;

h - розрахункова висота підвісу, h = 3 м;

A - ширина приміщення, A = 4 м;

B - довжина приміщення, B = 5 м.

Підставимо всі значення у формулу та визначемо індекса приміщення: і=0,74. Знаючи індекс приміщення, за знаходимо n = 0.29 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників відповідного типу). Підставимо всі значення у формулу, визначемо світловий потік: *F=43043 Лм.* Будемо використовувати світильники (світлодіодні панелі) Matt White Армстронг 36W 6000К, світловий потік яких $F_л$ = 3000 Лм.

Число ламп визначається по формулі:

$$N = F / F_л$$

де: F - світловий потік,

$F_л$ - світловий потік одного світильника.

Підставимо всі значення у формулу та визначемо індекса приміщення: N= *43043/* 3000=14,34 шт.

Приймаємо необхідну кількість світильників 15 шт.

8.6 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання магістерської роботи, призначено для експертної системи для управління персонажами у комп'ютерній грі.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У магістерській роботі наведені теоретичне узагальнення й рішення наукового завдання дослідження методів інтелектуального управління персонажами у комп'ютерних іграх.

Рішення даного завдання полягало у вирішенні наступних задач:

– Було проведене дослідження існуючих систем інтелектуального управління персонажами у комп'ютерних іграх, а також існуючих методів побудови експертних систем.

– На основі проведеного дослідження розроблено методи та алгоритми для експертної системи для управління персонажами у комп'ютерній грі.

– Використовуючи розроблені методи та алгоритми, створена програмна реалізація експертної системи для управління персонажами у комп'ютерній грі.

Розроблені під час виконання магістерської роботи алгоритми дозволяють успішно вирішувати завдання інтелектуального управління персонажами у комп'ютерній грі.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня C# у середовищі розробки Unity. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для захисту програмного забезпечення від несанкціонованого використання та поширення програмного коду був обраний метод обфускації.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 47381 грн. З урахуванням вартості розробки програми та обладнання, строк окупності становить 0,23 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Довідник по C # | Microsoft Docs, Ключові слова C #, Директиви препроцесора, Параметри компілятора C #

2. Архітектура персонального комп'ютера, Тема 3 - Загальні принципи архітектури комп'ютерів, 3.1 Принципи побудови комп'ютера. Архитектура Фон Неймана, 3.3 Архітектура і структура ПК

3. Седерхольм, Д. Пуленепробиваемый дизайн. Библиотека специалиста / Д. Седерхольм. - СПб.: Питер, 2012. - 304 с.

4. Джозеф Хокинг, Unity в дії. Глава 10 Звукові ефекти та музика 242с.

5. Джозеф Хокинг, Unity в дії. Глава 11 Об'єднання фрагментів в готову гру 267с.

6. Джозеф Хокинг, Unity в дії. Глава 12 Розгортання ігор на пристроях гравців 298с.

7. Е. Таненбаум, Архітектура комп'ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, глава 5. Рівень архітектури команд 334с.

8. Е. Таненбаум, Архітектура комп'ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Глава 8. Архітектури комп'ютерів паралельної дії 556с.

9. Е. Таненбаум, Архітектура комп'ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Додаток А. Двійкові числа 663с.

10. Е. Таненбаум, Архітектура комп'ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Додаток Б. Числа з плаваючою точкою 674с.

11. Гудфеллоу Я.Бенджіо І.Курвілль А., Глибоке навчання 2017р. Глава 2. Лінійна алгебра 44с.

12. Гудфеллоу Я.Бенджіо І.Курвілль А., Глибоке навчання 2017р. Глава 3. Теорія ймовірності і теорія інформації 61с.

13. Гудфеллоу Я.Бенджіо І.Курвілль А., Глибоке навчання 2017р. Глава 5.

Основи машинного навчання 96с.

14. Бхаргава А. Грокаем алгоритми. (Пітер)

15. Кормен Томас Х., Лейзерсон Чарльз І., Ривест Рональд Л., Штайн Кліффорд, Алгоритми. Побудова і аналіз

16. «Плоский дизайн»: с чего начать? Пять основных принципов Flat дизайна [Електронний ресурс] – Режим доступу до ресурсу: http://powerbranding.ru/design/flat-design-june13/.

17. «Альтернативы для замены ArcGIS» [Електронний ресурс] – Режим доступу до ресурсу: https://ruprogi.ru/software/arcgisdesktop.

18. «.NET Framework Guide» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/framework/.

19. «Common Language Runtime (CLR) overview» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/standard/clr.

20. Рихтер Д. CLR via C# Программирование на платформе Microsoft .NET Framework 2.0 на языке C# / Джеффри Рихтер. – Киев: Питер, 2008. – 656 с. – (2).

21. Троелсен Э. Язык программирования C# и платформа .NET 4.5 / Эндрю Троелсен. – Киев: вильямс, 2014. – 1312 с. – (6).

22. «Visual Studio» [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/ru/.

23. «Windows Forms overview» [Електронний ресурс] – 10. Режим доступу до ресурсу: https://docs.microsoft.com/enus/dotnet/framework/winforms/windows-forms-overview.

24. «Entity Framework Documentation» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/ef/.

25. «SQL Server 2017» [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/ru-ru/sql-server/sql-server-2017.

26. «About ArcGIS» [Електронний ресурс] – Режим доступу до ресурсу: https://www.esri.com/ru-ru/arcgis/about-arcgis/overview.

27. «Understanding Onion Architecture» [Електронний ресурс] – Режим

доступу до ресурсу: https://www.codeguru.com/csharp/csharp/cs_misc/designtechniques/understandingonion-architecture.html.

28. «Концептуальная модель системы» [Електронний ресурс] – Режим доступу до ресурсу: http://studepedia.org/index.php?vol=1&post=2123.

29. «MVC, MVP and MVVM Design Pattern» [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvmdesign-pattern-6e169567bbad.

30. «UnitOfWork And Repository Pattern » [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@utterbbq/c-unitofwork-and-repository-pattern305cd8ecfa7a.

31. «What is ArcGIS, and where is it available at IU?» [Електронний ресурс] – Режим доступу до ресурсу: https://kb.iu.edu/d/avzt.

32. Кузін А.В «Базы данных, 5-е издание» / Кузін А.В., Левонисова С.В. – К. : «Академия», 2012. – 317 с.

33. Гольцман В.І. «MySQL 5.0. Библиотека программиста» / Гольцман В.І. – К. : «Питер», 2010. – 253 с.

34. Бен Сміт «Beginning JSON» / Бен Сміт – К. : «Apress», 2015. – 324 с.

35. Sedgewick, Robert. Algorithms in C++. Parts 5: Graph Algorithms. 3rd Ed. Addison-Wesley, 2002.

36. Роберт Седжвік. Фундаментальные алгоритмы на C++. Части 1-4: Анализ/Структуры данных/Сортировка/Поиск. - К.: Издательство ДиаСофт?, 2001.

37. Роберт Седжвік. Фундаментальные алгоритмы на C++. Часть 5: Алгоритмы на графах. - К.: Издательство ДиаСофт?, 2002.

38. Роберт Седжвік. Фундаментальные алгоритмы на C. Части 1-4: Анализ/Структуры данных/Сортировка/Поиск. - К.: Издательство ДиаСофт?, 2003.

39. Роберт Седжвік. Фундаментальные алгоритмы на C Часть 5:

Алгоритмы на графах. - К.: Издательство ДиаСофт?, 2003.

40. Джон Макгрегор, Девід Сайке. Тестирование объектно-ориентированного программного обеспечения. Практическое пособие. - К.: Издательство ДиаСофт?, 2002.

41. Knuth, Donald E. The Art of Computer Programming: Fundamental Algorithms. 3rd Ed. Addison-Wesley, 1997.

42. Knuth, Donald E. The Art of Computer Programming: Seminumerical Algorithms. 3rd Ed. Addison-Wesley, 1998.

43. Knuth, Donald E. The Art of Computer Programming: Sorting and Searching. 2nd Ed. Addison-Wesley, 1998.

44. L"Ecuyer, Pierre. "Efficient and Portable Combined Random Number Generators." Communications of the ACM, Vol. 31 (1988), pp. 742-749, 774.

45. Nelson, Mark. The Data Compression Book. M& T Publishing, 1991.

46. Park, S.K., and K.W. Miller. "Random Number Generators: Good Ones are Hard to Find." Communications of the ACM, vol. 31 (1988), pp. 1192-1201.

47. Pham, Thuan Q. and Pankaj K. Garg. Multithreaded Programming with Win32. Prentice Hall, 1999.

48. Pugh, William. "Skip Lists: A Probabilistic AItemative to Balanced Trees." Communications of the ACM, Vol. 33 (1990), pp. 668-676.

49. Robbins, John. Debugging Applications. Microsoft Press, 2000.

50. Wood, Derick. Data Structures, Algorithms, and Performance. Addison-Wesley, 1993.

51. Державні будівельні норми України: ДБН В.2.5-28:2018. - Режим доступу до ресурсу: https://goo.su/9AkQ

52. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

53. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

54. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

55. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508

56. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. - Кіровоград: КІСМ, 1997. - 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358

57. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99

58. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : "Колос", 1973. – 238 с.

59. Центр післядипломної освіти та підвищення кваліфікації. - Режим доступу до ресурсу: https://cpo.stu.cn.ua

60. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : ЦНТУ, 2022. - 175 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12161 (дата звернення 19.09.22).

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію експертної системи для управління персонажами у комп’ютерній грі.

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. №19-13 від 17.08.2022 року).

3 Мета та призначення розробки

Метою магістерської роботи є дослідження та програмна реалізація експертної системи для управління персонажами у комп’ютерній грі.

4 Джерела розробки

Джерелом цієї магістерської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки

програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– управління персонажами у комп'ютерній грі на основі експертної системи;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Мова програмування C#, середовище розробки Unity.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2022 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи повинна бути розглянуті умови праці програмістів під час розробки програмного забезпечення.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 4 аркуші.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 94 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі магістерської роботи. Постановка задачі на виконання магістерської роботи (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень магістерської роботи.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання магістерської роботи на попередній захист 10.12.2022 р.

11.2 Подання магістерської роботи на захист .12.2022 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ
Керівник випускної кваліфікаційної роботи
за другим (магістерським) рівнем вищої освіти
_____ Є.В. Мелешко

Дослідження та програмна реалізація експертної системи для управління персонажами у комп’ютерній грі

Лістинг програми

Код документу 12

Носій: DVD-диск

Загальна кількість аркушів: 36

Літера: РП

Кропивницький – 2022 року

Основна програма

Файл UnitManager.cs основної програми

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

public enum EnvironmentState
{
    normal
}

public class UnitManager : MonoBehaviour
{
    public static EnvironmentState State = EnvironmentState.normal;

    [SerializeField]
    private HouseManager m_houseManager;
    [SerializeField]
    private ScenarioManager m_scenarioManager;
    [SerializeField]
    private int m_unitNumber;
    [SerializeField]
    private TextAsset m_namesData;
    [SerializeField]
    private List<GameObject> m_menPrefabs;
    [SerializeField]
    private List<GameObject> m_womenPrefabs;

    private List<UnitConctoller> m_unitsList = new List<UnitConctoller>();

    private List<string> m_names;

    private bool isEnemySpawned;

    private int m_minRandValue = 60;
    private int m_maxRandValue = 100;

    #region MonoBehavior
    private void Start()
    {
        m_names = new List<string>(m_namesData.text.Split('\n'));
        StartCoroutine(SpawnUnits());
```

```
}
#endregion

public List<UnitConctoller> GetUnitList()
{
    return m_unitsList;
}

private IEnumerator SpawnUnits()
{
    int tmp = 0;
    int enemyIndex = Random.Range(0, m_unitNumber-1);
    while(tmp != m_unitNumber)
    {
        yield return new WaitForSecondsRealtime(5f);

        if(tmp == enemyIndex)
        {
            SpawnUnit(true);
        }
        else
        {
            SpawnUnit();
        }
        tmp++;
    }
}

private void SpawnUnit(bool isEnemy = false)
{
    GameObject newUnit;

    newUnit = Instantiate(GetPrefab());

    if(isEnemy)
    {
        CreateUnitEntity(newUnit, true);
    }
    else
    {
        CreateUnitEntity(newUnit);
    }
}

private GameObject GetPrefab()
{
```

```
            int gender = Random.Range(0,100);

            if(gender > 50)
            {
                return m_womenPrefabs[Random.Range(0, m_womenPrefabs.Count - 1)];
            }
            else
            {
                return m_menPrefabs[Random.Range(0, m_menPrefabs.Count - 1)];
            }
        }

        private void CreateUnitEntity(GameObject unit, bool isEnemy = false)
        {
            UnitConctoller person = unit.GetComponent<UnitConctoller>();

            if(isEnemy)
            {
                person.SetUpEnemy();
            }

            int tmp = Random.Range(m_minRandValue, m_maxRandValue);
            person.Hunger = tmp;

            tmp = Random.Range(m_minRandValue, m_maxRandValue);
            person.Thirst = tmp;

            person.Health = m_maxRandValue;

            person.GameName = m_names[Random.Range(0, m_names.Count - 1)];
            m_unitsList.Add(person);

            person.GenerateHouseList(m_houseManager.GetAvailableHouses());

            person.OnNeedDrink += GetDrinkBuilding;
            person.OnNeedEat += GetEatBuilding;
            person.OnWorkFinding += GetWorkPlace;
            person.OnWalk += GetWalkPoint;
        }

        private void GetEatBuilding(UnitConctoller unit)
        {
           MealBuilding building =  m_houseManager.GetMealBuilding(HouseType.eat,
unit.Money);
            unit.FindMealBuilding(building);
        }
```

```
    private void GetDrinkBuilding(UnitConctoller unit)
    {
        MealBuilding building = m_houseManager.GetMealBuilding(HouseType.drink,
unit.Money);
        unit.FindMealBuilding(building);
    }

    private void GetWorkPlace(UnitConctoller unit)
    {
        WorkPlace work = m_houseManager.GetWorkPlace();
        unit.FindWorkPlace(work);
    }

    private void GetWalkPoint(UnitConctoller unit)
    {
        Transform point = m_houseManager.GetWalkPoint();
        unit.Walk(point);
    }
}
```

Файл UnitController.cs основної програми

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

public class ReactionEnity
{
    public Invasion invasion;
    public int successPercent;
}

public class UnitConctoller : MonoBehaviour
{
    public Action<UnitConctoller> OnNeedDrink { get; set; }

    public Action<UnitConctoller> OnNeedEat { get; set; }

    public Action<UnitConctoller> OnWorkFinding { get; set; }

    public Action<UnitConctoller> OnWalk { get; set; }

    public string GameName { get; set; }

    public int Health { get; set; }

    public int Hunger { get; set; }

    public int Thirst { get; set; }

    public int Fatigue { get; set; } = 50;

    public float Money { get; set; }

    public bool m_isEnemy { get; set; }
    public bool m_isOpen { get; set; }

    [SerializeField]
    private NavMeshAgent m_avatar;

    private float m_standingTime = 3f;
    private bool m_isDead = false;

    private int m_houseChecked;
```

```
private Animator m_animator;
private List<House> m_houseList;
private IEnumerator m_houseEnumerator;
private HouseEntity m_home;

private MealBuilding m_mealBuilding;
private WorkPlace m_work;
private Transform m_walkPoint;

//Home finding
private List<HouseEntity> m_bedroomList;
private int m_currentRoomIndex;

private bool m_needDrink;
private bool m_needEat;
private bool m_needRest;
private bool m_workFinding;

private bool m_working;
private bool m_walkWorking;

private bool m_walking;

private bool m_inHouse;

private bool m_isReacted;

private List<ReactionEnity> m_reactions = new List<ReactionEnity>();

private IEnumerator m_restingRoutine;
private IEnumerator m_workRoutine;
private IEnumerator m_lifeRoutine;

#region MonoBehavior
private void Start()
{
    m_animator = GetComponent<Animator>();
    m_lifeRoutine = LifeCircle();
    StartCoroutine(m_lifeRoutine);
    StartHousesPatrul();
    LightController.OnWorkStart += StartWorkingDay;
    LightController.OnWorkEnd += EndWorkingDay;
    LightController.OnWeekendStart += StartWeekend;
    LightController.OnDayEnd += EndDay;
}
```

```
private void OnTriggerStay(Collider other)
{
    if (other.tag.Equals("Bedroom"))
    {
        if (!m_inHouse)
        {
            m_inHouse = true;

        }
    }
}

private void OnTriggerExit(Collider other)
{
    if (other.tag.Equals("Bedroom"))
    {
        if (m_inHouse)
        {
            m_inHouse = false;

            if (m_restingRoutine != null)
            {
                StopCoroutine(m_restingRoutine);
                m_restingRoutine = null;
            }
        }
    }

    if (other.tag.Equals("Work"))
    {
        if (m_working)
        {
            if (m_workRoutine != null)
            {
                StopCoroutine(m_workRoutine);
                m_workRoutine = null;
            }
        }
    }
}

private void OnTriggerEnter(Collider other)
{
    if (other.tag.Equals("Reaction") || other.tag.Equals("Sheriff"))
    {
        if (m_isReacted)
```

```
        {
            Stop();
            StartTaskAfterMeal();
            m_isReacted = false;

            m_needDrink = false;
            m_needEat = false;
            Debug.LogError("Reacted " + m_isReacted);
        }
    }

    if (other.tag.Equals("Bedroom"))
    {
        Stop();
        if (m_home == null)
        {
            ComeToHomePoint(other.gameObject.GetComponent<Bedroom>());
        }

        m_restingRoutine = Resting();
        StartCoroutine(m_restingRoutine);
    }

    if (other.tag.Equals("Drink"))
    {
        if (!m_needDrink)
        {
            return;
        }
        Stop();
        StartCoroutine(Drink(m_mealBuilding));
    }

    if (other.tag.Equals("Eat"))
    {
        if (!m_needEat)
        {
            return;
        }
        Stop();
        StartCoroutine(Eat(m_mealBuilding));
    }

    if (other.tag.Equals("Work"))
    {
        if (!m_working)
```

```
            {
                return;
            }
            Stop();
            m_workRoutine = Work();
            StartCoroutine(m_workRoutine);
        }

        if (other.tag.Equals("Walk"))
        {
            if (m_walkPoint == null)
            {
                return;
            }

            if (!m_walking)
            {
                return;
            }

            Stop();

            while (other.transform.position.Equals(m_walkPoint.position))
            {
                OnWalk?.Invoke(this);
            }

            GoWalk();

            Debug.LogError("Come To Walk Point");
        }
    }

    private void Update()
    {
        if (Thirst <= 25)
        {
            if (!m_needDrink)
            {
                m_needDrink = true;

                if (m_needEat)
                {
                    StartCoroutine(DoingDelay(m_needEat, OnNeedDrink));
                }
                else
```

```
            {
                Stop();
                OnNeedDrink?.Invoke(this);
            }
        }
    }
    else if (Thirst >= 50)
    {
        if (m_needDrink)
        {
            m_needDrink = false;
            StartTaskAfterMeal();
        }
    }

    if (Hunger <= 25)
    {
        if (!m_needEat)
        {
            m_needEat = true;

            if (m_needDrink)
            {
                StartCoroutine(DoingDelay(m_needDrink, OnNeedEat));
            }
            else
            {
                Stop();
                OnNeedEat?.Invoke(this);
            }
        }
    }
    else if (Hunger >= 50)
    {
        if (m_needEat)
        {
            m_needEat = false;
            StartTaskAfterMeal();
        }
    }

    if (Fatigue <= 25 && !m_needDrink && !m_needEat && !m_working)
    {
        if (!m_needRest && m_home != null)
        {
            m_needRest = true;
```

```
                if (m_inHouse)
                {
                    m_restingRoutine = Resting();
                    StartCoroutine(m_restingRoutine);
                }
                else
                {
                    HomeFinding(m_home);
                }
            }
        }
        else if (Fatigue > 25)
        {
            if (m_needRest)
            {
                m_needRest = false;

                if (!m_workFinding)
                {
                    m_workFinding = true;
                    OnWorkFinding?.Invoke(this);
                }
            }
        }

        if (m_working && m_walkWorking && !m_needDrink && !m_needEat)
        {
            GoWork();
            m_walkWorking = false;
        }
    }
    #endregion

    public void ReactOnInvasion(Invasion invasion, int successPercent, Action
sucess, Action failure)
    {
        int percent = (UnityEngine.Random.Range(0, 1000) * 100) / 1000;

        ReactionEnity reaction = m_reactions.Find(x => x.invasion == invasion);

        if(reaction == null)
        {
            reaction = new ReactionEnity();
            reaction.invasion = invasion;
            reaction.successPercent = successPercent;
```

```
            m_reactions.Add(reaction);
        }

        if(percent > 100 - reaction.successPercent)
        {
            sucess.Invoke();
            reaction.successPercent = 100;
        }
        else
        {
            failure.Invoke();
            reaction.successPercent += 10;
        }

        m_isReacted = true;
    }

    public HouseEntity GetHome()
    {
        return m_home;
    }

    public void GoToPoint(Vector3 position)
    {
        m_avatar.ResetPath();
        m_avatar.SetDestination(position);
        m_animator.SetBool("Move", true);
    }

    public void FindMealBuilding(MealBuilding building)
    {
        m_mealBuilding = building;

        GoToPoint(building.housePoint.position);
    }

    public void FindWorkPlace(WorkPlace work)
    {
        //WorkPlacing
        m_work = work;
        Debug.LogError("Work Found!");
    }

    public void Walk(Transform transform)
    {
        m_walkPoint = transform;
```

```
    }

    private void  GoWalk()
    {
        if (m_needDrink || m_needEat || m_walkWorking)
        {
            return;
        }

        if (m_walkPoint != null)
        {
            GoToPoint(m_walkPoint.position);
        }
    }

    public void GenerateHouseList(List<House> houses)
    {
        m_houseList = houses;
    }

    public void SetUpEnemy()
    {
        m_isEnemy = true;
    }

    public void ResetAvatar(List<HouseEntity> interests)
    {
        m_avatar.isStopped = true;
        m_avatar.ResetPath();
        m_animator.SetBool("Move", false);

        m_bedroomList = null;

        if (interests != null)
        {
            List<HouseEntity> bedrooms = interests.FindAll(x => x.type ==
HouseInterests.Bedroom);
            m_bedroomList = bedrooms;
        }
        else
        {
            m_bedroomList = null;
        }

        m_houseEnumerator = HouseStanding();
        StartCoroutine(m_houseEnumerator);
```

```
}

public void InstallFinalPoint(Transform point)
{
    GoToPoint(point.position);
    m_isOpen = true;
    StartCoroutine(MoveToFinal(point));
}

public void ReceiveHit(int damage)
{
    Health -= damage;
}

public void Die()
{
    if(m_houseEnumerator != null)
    {
        StopCoroutine(m_houseEnumerator);
    }
    Destroy(gameObject);
}

private void StartTaskAfterMeal()
{
    if (m_working && !m_walkWorking)
    {
        m_walkWorking = true;
    }
    else if (m_walking)
    {
        OnWalk?.Invoke(this);
    }
    else
    {
        HomeFinding(m_home);
    }
}

private void GoWork()
{
    m_avatar.ResetPath();
    Debug.LogError("MY WORK " + m_work.housePoint.position);
    GoToPoint(m_work.housePoint.position);
}
```

```
private void HomeFinding(HouseEntity room)
{
    m_animator.SetBool("Move", true);
    m_avatar.SetDestination(room.point.position);
}

private void Stop()
{
    m_avatar.isStopped = true;
    m_avatar.ResetPath();
    m_animator.SetBool("Move", false);
}

private void StartWorkingDay()
{
    if(m_work != null)
    {
        Debug.LogError("Start Working Day");
        m_working = true;
        m_walkWorking = true;
    }
}

private void EndWorkingDay()
{
    if (m_work != null)
    {
        Debug.LogError("End Working Day");
        m_working = false;
        m_walkWorking = false;
        m_needDrink = false;
        m_needEat = false;
        m_walking = true;
        Stop();

        OnWalk?.Invoke(this);
        GoWalk();
    }
}

private void StartWeekend()
{
    m_needDrink = false;
    m_needEat = false;

    m_walking = true;
```

```
    OnWalk?.Invoke(this);
}

private void EndDay()
{
    Debug.LogError("End Day");

    m_walking = false;
    m_walkPoint = null;
    Stop();
    m_needDrink = false;
    m_needEat = false;

    if (!m_inHouse)
    {
        HomeFinding(m_home);
    }
}

private void ComeToHomePoint(Bedroom bedroom)
{
    if (bedroom.People < 2)
    {
        bedroom.People++;
        m_home = m_bedroomList[m_currentRoomIndex];
    }
    else
    {
        m_currentRoomIndex++;
        if (m_currentRoomIndex >= m_bedroomList.Count)
        {
            m_currentRoomIndex = 0;
            StartHousesPatrul();
        }
        else
        {
            HomeFinding(m_bedroomList[m_currentRoomIndex]);
        }
    }
}

private void StartHousesPatrul()
{
    if(m_houseChecked < m_houseList.Count)
    {
        GoToPoint(m_houseList[m_houseChecked].housePoint.position);
```

```
            m_houseChecked++;
        }
    }

    private IEnumerator Eat(MealBuilding building)
    {
        while(Hunger <= 50)
        {
            yield return new WaitForSecondsRealtime(1f);

            if (Money < building.serviceCost)
            {
                m_needEat = false;
                yield break;
            }

            Hunger += building.points;
            Money -= building.serviceCost;
            Fatigue -= (int)(5 - building.serviceCost);
        }
    }

    private IEnumerator Work()
    {
        while(m_working)
        {
            yield return new WaitForSecondsRealtime(3f);
            Money += m_work.payment;
        }
    }

    private IEnumerator Drink(MealBuilding building)
    {
        while (Thirst <= 50)
        {
            yield return new WaitForSecondsRealtime(1f);

            if (Money < building.serviceCost)
            {
                m_needDrink = false;
                yield break;
            }

            Thirst += building.points;
            Money -= building.serviceCost;
            Fatigue -= (int)(5 - building.serviceCost);
```

```
        }
    }

    private IEnumerator Resting()
    {
        while(m_inHouse)
        {
            yield return new WaitForSecondsRealtime(1f);

            if (Fatigue < 100)
            {
                Fatigue++;
            }
        }
    }

    private IEnumerator DoingDelay(bool condition, Action<UnitConctoller>
action)
    {
        while(condition)
        {
            yield return null;
        }

        action?.Invoke(this);
    }

    private IEnumerator MoveToFinal(Transform point)
    {
        while(Vector3.Distance(transform.position, point.position) > 2)
        {
            yield return null;
        }

        m_animator.SetBool("Move", false);
    }

    private IEnumerator HouseStanding()
    {
        yield return new WaitForSecondsRealtime(m_standingTime);
        m_houseEnumerator = null;

        if(m_home == null)
        {
            if (m_bedroomList != null && m_bedroomList.Count > 0)
            {
```

```
                HomeFinding(m_bedroomList[m_currentRoomIndex]);
            }
            else
            {
                StartHousesPatrul();
            }
            yield break;
        }
    }

    private IEnumerator LifeCircle()
    {
        while(Health != 0)
        {
            if(m_isDead)
            {
                yield break;
            }

            yield return new WaitForSecondsRealtime(5f);//Need balance

            if(Hunger == 0 || Thirst == 0)
            {
                Health -= 15;
            }
            else
            {
                if(UnitManager.State == EnvironmentState.normal)
                {
                    Hunger -= UnityEngine.Random.Range(1, 2);
                    Thirst -= UnityEngine.Random.Range(1, 3);
                    Fatigue -= UnityEngine.Random.Range(1, 5);
                    if (Fatigue <= 0)
                    {
                        Fatigue = 0;
                    }
                }
            }
        }

        m_isDead = true;
    }
}
```

Файл LightController.cs основної програми

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

public enum DayTime
{
    day,
    night
}

[ExecuteAlways]
public class LightController : MonoBehaviour
{
    public static Action OnWorkStart { get; set; }
    public static Action OnWorkEnd { get; set; }
    public static Action OnWeekendStart { get; set; }
    public static Action OnDayEnd { get; set; }

    public DayTime LightState { get; set; }

    [SerializeField]
    private Light m_directionalLight;
    [SerializeField]
    private LightningPreset m_preset;
    [SerializeField, Range (0, 156)]
    private float m_timeOfDay;

    private float m_timeMultiolier = 156f;

    private bool m_isWorkStart;
    private bool m_isNewDay;
    private bool m_isDayStart;

    private int m_dayCount = 1;

    private void OnValidate()
    {
        if(m_directionalLight != null)
        {
            return;
        }

        if(RenderSettings.sun != null)
```

```
        {
            m_directionalLight = RenderSettings.sun;
        }
        else
        {
            Light[] lights = FindObjectsOfType<Light>();

            foreach(Light light in lights)
            {
                if(light.type == LightType.Directional)
                {
                    m_directionalLight = light;
                    return;
                }
            }
        }
    }

    private void Update()
    {
        if(m_preset == null)
        {
            return;
        }

        if(Application.isPlaying)
        {
            m_timeOfDay += Time.deltaTime;
            m_timeOfDay %= m_timeMultiolier;
            UpdateLight(m_timeOfDay / m_timeMultiolier);

            if(m_timeOfDay <= 1 && !m_isNewDay)
            {
                m_isNewDay = true;
                if (m_dayCount != 7)
                {
                    m_dayCount++;
                }
                else
                {
                    m_dayCount = 1;
                }
            }

            if(m_timeOfDay > 51 && m_timeOfDay < 52 && !m_isWorkStart)
            {
```

```
                if(m_dayCount < 6)
                {
                    OnWorkStart?.Invoke();
                }
                else
                {
                    OnWeekendStart?.Invoke();
                }
                m_isDayStart = true;
                m_isNewDay = false;
                m_isWorkStart = true;
                Debug.LogError("Day");
                LightState = DayTime.day;
            }

            if (m_timeOfDay > 123 && m_timeOfDay < 123.5f && m_isWorkStart)
            {
                if (m_dayCount < 6)
                {
                    OnWorkEnd?.Invoke();
                }

                m_isWorkStart = false;
                Debug.LogError("Night");
                LightState = DayTime.night;
            }


            if (m_timeOfDay > 148 && m_timeOfDay < 148.5f && m_isDayStart)
            {
                m_isDayStart = false;
                OnDayEnd?.Invoke();
                Debug.LogError("DeepNight");
            }
        }
        else
        {
            UpdateLight(m_timeOfDay / m_timeMultiolier);
        }
    }

    private void UpdateLight(float timePreset)
    {
        RenderSettings.ambientLight =
m_preset.AmbientColor.Evaluate(timePreset);
        RenderSettings.fogColor = m_preset.FogColor.Evaluate(timePreset);
```

```
        if(m_directionalLight != null)
        {
            m_directionalLight.color =
m_preset.DirectionalColor.Evaluate(timePreset);
            m_directionalLight.transform.rotation = Quaternion.Euler(new
Vector3((timePreset * 360f) - 90f, -170, 0));
        }
    }
}
```

Файл InvasionManager.cs основної програми

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

public enum InvationType
{
    robber
}

public class InvasionManager : MonoBehaviour
{
    [SerializeField]
    private UnitManager m_unitManager;
    [SerializeField]
    private LightController m_dayController;
    [SerializeField]
    private List<Invasion> m_invasionList;

    private void Start()
    {
        StartCoroutine(Delay());
    }

    private IEnumerator Delay()
    {
        Debug.LogError("Delay");
        yield return new WaitForSecondsRealtime(60f);
        Invasion invasion = Instantiate(m_invasionList[0]);
        invasion.SetUpInvasion(m_unitManager.GetUnitList(), m_dayController);
    }
}
```

Файл Invasion.cs основної програми

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

public abstract class Invasion : MonoBehaviour
{
    public InvationType Type { get; set; } = InvationType.robber;

    [SerializeField]
    private DayTime m_activationTime;
    [SerializeField]
    private int m_successPercent;

    protected List<UnitConctoller> m_unitList;
    protected LightController m_dayController;
    protected UnitConctoller m_target;

    private IEnumerator m_lifeRoutine;
    #region MonoBehaviour
    private void Start()
    {
        m_lifeRoutine = ActionRoutine();
        StartCoroutine(m_lifeRoutine);
    }

    private void OnDestroy()
    {
        if(m_lifeRoutine != null)
        {
            StopCoroutine(m_lifeRoutine);
        }
    }
    #endregion

    public void SetUpInvasion(List<UnitConctoller> list, LightController day)
    {
        m_unitList = list;
        m_dayController = day;
    }

    protected IEnumerator ActionRoutine()
    {
        while(true)
        {
```

```
            yield return new WaitForSecondsRealtime(60f);

            if(m_dayController.LightState == m_activationTime)
            {
                SelectAction();
            }
        }
    }

    protected void SelectAction()
    {
        int index = Random.Range(0, m_unitList.Count - 1);
        m_target = m_unitList[index];

        m_target.ReactOnInvasion(this, m_successPercent, DoSuccess, DoFault);
    }

    public abstract void DoFault();
    public abstract void DoSuccess();
}
```

Файл RobberInvasion.cs основної програми

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

public class RobberInvasion : Invasion
{
    [SerializeField]
    private Transform m_reactionPointError;

    private Transform m_successPoint;

    private void OnEnable()
    {
        m_successPoint = GameObject.Find("SheriffPoint").transform;
    }

    public override void DoFault()
    {
        m_target.GoToPoint(m_reactionPointError.position);
    }

    public override void DoSuccess()
    {
        m_target.GoToPoint(m_successPoint.position);
    }
}
```

Файл SheriffController.cs основної програми

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

public class SheriffController : MonoBehaviour
{
    public bool IsGone { get; set; }

    [SerializeField]
    private NavMeshAgent m_agent;
    [SerializeField]
    private Animator m_animator;
    [SerializeField]
    private Transform m_homePoint;

    private Transform m_target;
    private bool m_isHome = true;

    private void OnTriggerEnter(Collider other)
    {
        if(other.tag.Equals("Unit") && !IsGone && m_isHome)
        {
            IsGone = true;
            m_target = GameObject.Find("KeyPoint").transform;
            m_agent.SetDestination(m_target.position);
            m_animator.SetBool("Move", true);
        }
    }

    private void Update()
    {
        if(IsGone)
        {
            if (Vector3.Distance(m_agent.transform.position, m_target.position)
< 1)
            {
                m_isHome = false;
                IsGone = false;
                m_agent.SetDestination(m_homePoint.position);
            }
        }

        if(!m_isHome)
```

```
        {
            if (Vector3.Distance(m_agent.transform.position,
m_homePoint.position) < 1.5f)
            {
                m_isHome = true;
                m_agent.isStopped = true;
                m_agent.ResetPath();
                m_animator.SetBool("Move", false);
            }
        }
    }
}
```

Файл HouseManager.cs основної програми

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

public enum HouseType
{
    standart,
    drink,
    eat
}

[System.Serializable]
public class House
{
    public HouseType type;
    public Transform housePoint;
}

[System.Serializable]
public class MealBuilding : House
{
    public float serviceCost;
    public int points;
}

[System.Serializable]
public class WorkPlace : House
{
    public float payment;
    public int vacancyLimit;
    public int lockVacany;
}

public class HouseManager : MonoBehaviour
{
    [SerializeField]
    private List<House> m_houseList;
    [SerializeField]
    private List<MealBuilding> m_foodHouses;
    [SerializeField]
    private List<WorkPlace> m_workPlaces;
    [SerializeField]
    private List<Transform> m_walkPoints;
```

```
    public List<House> GetAvailableHouses()
    {
        return m_houseList;
    }

    public MealBuilding GetMealBuilding(HouseType type, float money)
    {
        List<MealBuilding> neededBuildings = m_foodHouses.FindAll(x => x.type ==
type && x.serviceCost <= money);

        MealBuilding building = neededBuildings[0];

        foreach(MealBuilding meal in neededBuildings)
        {
            if(meal.serviceCost > building.serviceCost)
            {
                building = meal;
            }
        }

        return building;
    }

    public WorkPlace GetWorkPlace()
    {
        List<WorkPlace> availableWorks = m_workPlaces.FindAll(x => x.lockVacany
< x.vacancyLimit);

        if(availableWorks != null)
        {
            int index = Random.Range(0, availableWorks.Count);
            availableWorks[index].lockVacany++;
            return availableWorks[index];
        }
        else
        {
            return null;
        }
    }

    public Transform GetWalkPoint()
    {
        int index = Random.Range(0, m_walkPoints.Count - 1);
        return m_walkPoints[index];
    }
}
```

Файл HousePoint.cs основної програми

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

public enum HouseInterests
{
    Bedroom,
    Citchen,
}

[System.Serializable]
public class HouseEntity
{
    public HouseInterests type;
    public Transform point;
}

public class HousePointController : MonoBehaviour
{
    [SerializeField]
    private List<HouseEntity> m_houseInterests;

    private void OnTriggerEnter(Collider other)
    {
        if(other.gameObject.tag == "Unit")
        {
            UnitConctoller unit = other.GetComponent<UnitConctoller>();

            if(unit != null && unit.GetHome() == null)
            {
                unit.ResetAvatar(m_houseInterests);
            }
        }
    }
}
```

Файл RobberInteraction.cs основної програми

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

public class RobberInteraction : MonoBehaviour
{
    [SerializeField]
    private GameObject m_parent;

    private Transform m_sheriff;

    private void Start()
    {
        m_sheriff = GameObject.Find("Sheriff").transform;
    }

    private void Update()
    {
        if (Vector3.Distance(m_sheriff.transform.position, transform.position) <
1)
        {
            Destroy(m_parent);
        }
    }
}
```

Файл PlayerController.cs основної програми

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

[RequireComponent(typeof(CharacterController))]
public class PlayerController : MonoBehaviour
{
    public int Health { get; set; } = 100;

    [SerializeField]
    private GameObject m_UIObject;

    private Camera m_mainCamera;
    private NavMeshAgent m_agent;
    private Transform m_transform;
    private Vector3 m_currDest;
    private bool m_isMoving;
    private Animator m_animator;

    private int m_layer = 1 << 8;

    void Start()
    {
        m_mainCamera = Camera.main;
        m_transform = GetComponent<Transform>();
        m_agent = GetComponent<NavMeshAgent>();
        m_animator = GetComponent<Animator>();
    }

    void Update()
    {
        if(Input.GetMouseButtonDown(0))
        {
            RaycastHit hit;

if(Physics.Raycast(m_mainCamera.ScreenPointToRay(Input.mousePosition), out hit,
m_layer))
            {
                m_currDest = hit.point;
                m_isMoving = true;
                m_animator.SetBool("Move", true);
                m_agent.ResetPath();
                m_agent.SetDestination(hit.point);
```

```
            }
        }

        if(Input.GetKeyDown(KeyCode.J))
        {
            if(!m_UIObject.activeSelf)
            {
                m_UIObject.SetActive(true);
            }
        }


        if (Input.GetKeyUp(KeyCode.J))
        {
            if (m_UIObject.activeSelf)
            {
                m_UIObject.SetActive(false);
            }
        }

        if (m_isMoving && Vector3.Distance(m_currDest, m_transform.position) <=
2)
        {
            m_isMoving = false;
            m_animator.SetBool("Move", false);
            m_agent.ResetPath();
        }
    }

    public void ReceiveHit(int damage)
    {
        Health -= damage;
    }
}
```