Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія підвищення рівня інтелектуальності персонажів комп'ютерної гри»

Виконав: студент 2-го курсу, групи 1КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
_____ Городецький Ю.Г.
(прізвище та ініціали)

Керівник: к. т. н., доц. кафедри КН
_____ Барабан С.В.
(прізвище та ініціали)
«15» 12 2022 р.

Опонент: д.т.н., професор, в.о. зав. каф. КСУ
_____ Ковтун В.В.
(прізвище та ініціали)
«15» 12 2022 р.

Допущено до захисту
Завідувач кафедри КН
_____ дтн., проф. Яровий А.А.
(прізвище та ініціали)
«16» 12 2022 р.

Вінниця ВНТУ – 2022 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Д.т.н., проф. Яровий А.А.
14. 09. **2022 року**

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Городецький Юрій Григорович
(прізвище та ініціали)

1. Тема роботи Інформаційна технологія підвищення рівня інтелектуальності персонажів комп'ютерної гри
керівник роботи к. т. н., доцент кафедри КН Барабан С.В.
затверджені наказом вищого навчального закладу від "14" 09 2022 року №203
2. Строк подання студентом роботи 18 листопада 2022 року
3. Вихідні дані до роботи:
вхідні дані – дані натиснутих клавіш користувачем (клавіші «стрілочок» або клавіші «W», «A», «S», «D», клавіша «E» та ліва кнопка миші), дані про положення комп'ютерної миші, дані про використовувану ігрову карту,
4. Зміст текстової частини: вступ, обґрунтування доцільності розробки інформаційної технології підвищення рівня інтелектуальності персонажів комп'ютерної гри, аналіз методів та засобів для реалізації інформаційної технології підвищення рівня інтелектуальності персонажів комп'ютерної гри, програмна реалізація інформаційної технології підвищення рівня інтелектуальності персонажів комп'ютерної гри, економічна частина, висновки, перелік використаних джерел, додатки.

5. Перелік ілюстративного матеріалу: дерево поведінки штучного інтелекту, UML діаграма класів комп'ютерної гри, програмна реалізація гри, результати порівняння розробленого ШІ з інтелектом аналогом.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-3	Барабан С.В. к.т.н., доц. каф. КН	14.09.2022	14.12.2022
4	Нікіфорова Л. О., к.е.н., доц. каф. ЕПВМ	13.09.2022	12.12.2022

7. Дата видачі завдання 14.09. 2022 року

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ МКР

№ етапу	Назва етапу	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного рівня розвитку інформаційних технологій штучного інтелекту в іграх. Постановка задач дослідження	14.09.2022р. - 1.10.2022	Розділ 1
2	Побудова математичних моделей функціонування штучного інтелекту	2.10.2022р. - 16.10.2022р	Розділ 2
3	Практичне застосування та оцінення ефективності розроблених моделей	17.10.2022р. - 07.11.2022р	Розділ 3
4	Підготовка економічної частини	08.11.2022 - 07.11.2022	розділ 4
5	Апробація та/або впровадження результатів дослідження	23.11.2022р. - 01.12.2022р	Тези доповіді
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	02.12.2022р. - 14.12.2022р.	Пояснювальна записка, графічний матеріал, презентація

Студент _____ (підпис) Городецький Ю. Г.

Керівник роботи _____ (підпис) Барабан С. В.

АНОТАЦІЯ

УДК 004.8

Городецький Ю.Г. Інформаційна технологія підвищення рівня інтелектуальності персонажів комп'ютерної гри. Магістерська кваліфікаційна робота зі спеціальності 122 – комп’ютерні науки, освітня програма – комп’ютерні науки. Вінниця: ВНТУ, 2022. 88 с.

На укр. мові. Бібліогр.: 33 назв; рис.: 4; табл. 8.

У даній магістерській кваліфікаційній роботі було проведено аналіз існуючих реалізацій штучного інтелекту в комп’ютерних іграх. Проведено дослідження актуальних технологій які дозволяють покращити штучний інтелект внутрішньо ігрових персонажів та здійснено порівняння з існуючим аналогом. Дана робота допомагає проаналізувати існуючі можливості технологій реалізації штучного інтелекту персонажів комп’ютерної гри, та допомагає покращити взаємодію гравців та внутрішньо ігрових персонажів, тим самим даючи більше задоволення від гри. До реалізації гри для демонстрації підвищення інтелектуальності персонажів комп’ютерної гри було обрано мову C#, середовище розробки Visual Studio, фреймворк MonoGame. Результати тестування відповідають встановленим досягненням мети даної магістерської кваліфікаційної роботи.

Графічна частина складається з 4 плакатів.

У економічному розділі розраховано суму витрат на розробку та виготовлення нового технічного рішення, яка складає 85731,32 гривень, спрогнозовано орієнтовану величину витрат по кожній з статей витрат, розраховано чистий прибуток, термін окупності витрат для виробника 2,6 роки та економічний ефект для споживача при використанні даної розробки.

Ключові слова: штучний інтелект, алгоритм пошуку A*, комп’ютерний персонаж, комп’ютерна гра.

ANNOTATION

Gorodetskyi Y.G. Information technology for increasing the level of intelligence of computer game characters. Master's qualification thesis on specialty 122 - computer science, educational program - computer science. Vinnytsia: VNTU, 2022. 88 p.

In Ukrainian language. Bibliographer: 33 titles; fig .: 4; table 8.

In this master's thesis, an analysis of existing implementations of artificial intelligence in computer games was carried out. A study of current technologies that allow improving the artificial intelligence of in-game characters was conducted and a comparison was made with the existing analogue. This work helps to analyze the existing possibilities of technologies for the implementation of artificial intelligence of computer game characters, and helps to improve the interaction of players and in-game characters, thereby giving more pleasure to the game. The C# language, the Visual Studio development environment, and the MonoGame framework were chosen before the game was implemented to demonstrate the increase in the intelligence of the characters of the computer game. The test results correspond to the established goals of this master's qualification work.

The graphic part consists of 4 posters.

In the economic section, the amount of costs for the development and production of a new technical solution is calculated, which is 85731.32 hryvnias, the estimated amount of costs for each of the cost items is predicted, the net profit is calculated, the payback period for the manufacturer is 2.6 years, and the economic effect for the consumer when using this developments.

Keywords: artificial intelligence, A* search algorithm, computer character, computer game.

ПЕРЕЛІК СКОРОЧЕНЬ

ШІ – штучний інтелект

NPC (Non-Player Character) – персонаж, що не керується гравцем

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі тема штучного інтелекту стає актуальною як ніколи. Кількість галузей, де використовується штучний інтелект постійно зростає як і підвищується складність самого інтелекту. Однією з галузей, де штучний інтелект використовувався майже з самої появи є відеоігри.

Ще в перших комп'ютерних іграх використовувався штучний інтелект. Наприклад в одній з перших комп'ютерних ігор «Pong» був варіант гри проти опонента, що керується комп'ютером, тим самим позбутись необхідності в другому гравцеві [1]. На той момент сам інтелект в грі був надзвичайно примітивним по причині технічних обмежень та правил самої гри (так як в «Pong» можна здійснювати тільки рухи вверх або вниз).

З роками галузь відеоігор розвивалася, а разом з нею і штучний інтелект, що використовується в іграх. З'являлись все більше ігор високої складності свого функціонування та штучного інтелекту. Ігрові персонажі набували все більшого покращення репертуару вмінь та реакцій на внутрішньо ігрові ситуації. Але це в свою чергу принесло простоту свого функціоналу в базисі і взаємно повторення між різними іграми. Таке взаємне повторення приводить до меншої унікальності та відкидає велику кількість внутрішньо ігрових ситуацій що могли привнести гравцеві унікальні і неповторні почуття від самої гри. Таким чином утворюється ситуація, де розвиток штучного інтелекту в комп'ютерних іграх є скоріше косметичним і показується лише в деталях.

З сказаного вище слідує, що задача підвищення рівня інтелектуальності персонажів комп'ютерної гри є достатньо важливою і для свого розв'язання потребує розробки нових способів взаємодії між гравцем та комп'ютерним персонажів, а також нових більш розвинутих реакцій від останнього. Щоб досягти цього необхідно надати штучному інтелекту вмінь та навчити його використовувати їх способами, що створять гравцеві цікаві внутрішньо ігрові ситуації та зроблять простий геймплей захопливим та складнішим.

Для спрощення роботи над підвищенням інтелектуальності внутрішньо ігрових персонажів є актуальним використання в розробці ігрових двигунів [2]. Цей засіб дозволить підвищити концентрацію саме на розробці та покращенню штучного інтелекту а також спростить розробку самої гри.

Також тема магістерської кваліфікаційної роботи є актуальною по причині того, що новий розроблений штучний інтелект може стати базисом для подальшого розвитку штучного інтелекту в нових іграх і тим самим розкрити геймплей з нових неочікуваних для гравця сторін.

Таким чином, підвищення рівня інтелектуальності персонажів комп'ютерної гри забезпечує подальше покращення ігрового досвіду для гравців, а також є актуальною темою дослідження нових методів використання штучного інтелекту.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська робота виконана відповідно до напрямку наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки, навчання та комунікацій» та плану наукової та навчально-методичної роботи кафедри

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є підвищення рівня майстерності комп’ютерних ігор за допомогою розробки інформаційної технології підвищення рівня інтелектуальності персонажів комп’ютерної гри. Для досягнення поставленої мети необхідно виконати такі завдання:

- провести аналіз проблеми рівня інтелектуальності персонажів комп'ютерної гри;
- розглянути існуючі методи вирішення задачі підвищення рівня інтелектуальності та обрати й обґрунтувати вибір методу, який задовольняє мету магістерської кваліфікаційної роботи;
- розробити модель інтелектуальної поведінки персонажів комп’ютерної гри, що дозволить виконати удосконалення згідно з метою роботи;

- сформулювати стадії інформаційної технології та їх основі розробити структуру та алгоритм роботи програмного модулю штучного інтелекту персонажів комп'ютерної гри;
- виконати програмну реалізацію запропонованої моделі інтелектуальної поведінки;
- провести тестування програмного продукту та виконати аналіз отриманих результатів;
- провести обрахунок вартості розробки технології підвищення інтелектуальності персонажів в комп'ютерних іграх та період окупності інвестицій.

Об'єкт дослідження – це процес взаємодії комп'ютерного персонажа, керованого штучним інтелектом з персонажем гравця.

Предмет дослідження – це програмні засоби взаємодії комп'ютерного персонажа, керованого штучним інтелектом з персонажем гравця.

Методи дослідження. Для досягнення мети дослідження застосовувалися методи побудови дерев поведінки та UML діаграм, ООП, структурного програмування.

Наукова новизна одержаних результатів полягає в такому:

Удосконалення інформаційної технології підвищення рівня інтелектуальності персонажів комп'ютерної гри яка на відміну від інших використовує модель поєднання методів дерев рішень та штучного інтелекту.

Практичне значення одержаних результатів полягає у тому, що на основі проведених досліджень розроблено гру, що може продемонструвати взаємодію модель інтелектуальної поведінки персонажа та гравця на практиці.

Запропонована інформаційна технологія підвищенню рівня інтелектуальності персонажів комп'ютерної гри, зокрема:

- розроблено алгоритм роботи персонажу комп'ютерної гри на основі алгоритму пошуку A*;
- розроблено гру для демонстрації покращеної моделі інтелектуальної поведінки персонажів комп'ютерної гри.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується коректністю постановки завдання, коректністю використання математичного апарату методів дослідження, експериментальними дослідженнями тестування програмної реалізації інформаційної технології підвищення рівня інтелектуальності персонажів комп'ютерної гри. Адекватність розроблених математичних моделей підтверджується результатами експериментальних досліджень.

Особистий внесок здобувача. Усі результати, що наведені у магістерській кваліфікаційній роботі, отримані самостійно. У працях, які написано у співавторстві, здобувачу належать: аналіз процесу роботи штучного інтелекту персонажів комп’ютерних ігор та методів підвищення рівня інтелектуальності персонажів комп'ютерної гри [3].

Апробація результатів роботи. Результати досліджень апробовані на конференції «Молодь в науці: дослідження, проблеми, перспективи», Вінниця, листопад 2022 – травень 2023 року [3].

Публікації. За результатами досліджень опубліковано одні тези доповіді на науково-технічній конференції.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПІДВИЩЕННЯ РІВНЯ ІНТЕЛЕКТУАЛЬНОСТІ ПЕРСОНАЖІВ КОМП'ЮТЕРНОЇ ГРИ

1.1Аналіз предметної області штучного інтелекту в комп'ютерних грах

Штучний інтелект — це широка галузь інформатики, яка займається створенням розумних машин, здатних виконувати завдання, які зазвичай вимагають людського інтелекту.

Різні підгалузі досліджень ШІ зосереджені навколо конкретних цілей і використання конкретних інструментів. Традиційні цілі досліджень ШІ включають міркування, представлення знань, планування, навчання, обробку природної мови, сприйняття та здатність переміщати об'єкти та маніпулювати ними. Загальний інтелект (здатність вирішувати довільну проблему) є однією з довгострокових цілей галузі. Щоб вирішити ці проблеми, дослідники ШІ адаптували та інтегрували широкий спектр методів вирішення проблем, включаючи пошук і математичну оптимізацію, формальну логіку, штучні нейронні мережі та методи, засновані на статистиці, ймовірності та економіці. ШІ також використовує комп'ютерні науки, психологію, лінгвістику, філософію та багато інших галузей [4].

Програми штучного інтелекту включають просунуті веб-пошукові системи (наприклад, Google), системи рекомендацій (використовуються YouTube, Amazon і Netflix), розуміння людської мови (такі як Siri та Alexa), безпілотні автомобілі, автоматизоване прийняття рішень і конкуренцію на найвищому рівні [4]. рівень у стратегічних ігрових системах (таких як шахи та го). Оскільки машини стають все більш спроможними, завдання, які вважаються такими, що вимагають «інтелекту», часто вилучаються з визначення штучного інтелекту, явище, відоме як ефект ШІ. Наприклад,

оптичне розпізнавання символів часто виключається з речей, які вважаються ШІ, ставши рутинною технологією.

Штучний інтелект був заснований як академічна дисципліна в 1956 році, і з тих пір пережив кілька хвиль оптимізму, за якими послідували розчарування та втрата фінансування (відома як «зима штучного інтелекту»), а потім нові підходи, успіх і відновлення фінансування [4]. Дослідження штучного інтелекту випробували та відкинули багато різних підходів з моменту свого заснування, включаючи симуляцію мозку, моделювання вирішення людських проблем, формальну логіку, великі бази даних знань та імітацію поведінки тварин. У перші десятиліття 21-го століття в галузі домінувало математико-статистичне машинне навчання, і ця техніка виявилася дуже успішною, допомагаючи вирішувати багато складних проблем у промисловості та академічних колах.

У відеоіграх ШІ використовується для генерування чуйної, адаптивної або інтелектуальної поведінки, насамперед у NPC, подібних до людського інтелекту. ШІ був невід’ємною частиною відеоігор з моменту їх заснування в 1950-х роках. ШІ у відеоіграх є окремою підсферою і відрізняється від академічного ШІ. Він служить для покращення досвіду гравця, а не для машинного навчання чи прийняття рішень. Протягом золотого віку аркадних відеоігор ідея супротивників з ШІ була значною мірою популяризована у вигляді ступеневих рівнів складності, чітких моделей рухів та ігрових подій, які залежали від внеску гравця. Сучасні ігри часто реалізують існуючі методи, такі як пошук шляху та дерева рішень, щоб керувати діями NPC. ШІ часто використовується в механізмах, які не відразу видно користувачеві, як-от інтелектуальний аналіз даних і створення процедурного вмісту [5].

Ігри були сферою досліджень ШІ з самого початку. Одним із перших прикладів штучного інтелекту є комп’ютеризована гра Nim, створена в 1951 році та опублікована в 1952 році. Незважаючи на передову технологію в рік її створення, за 20 років до Pong, гра мала форму відносно невеликої коробки і була здатна регулярно вигравати ігри навіть проти висококваліфікованих

гравців гри [5]. У 1951 році, використовуючи машину Ferranti Mark 1 Манчестерського університету, Крістофер Стрейчі написав програму для шашок, а Дітріх Прінц — для шахів. Це були одні з перших комп'ютерних програм, коли-небудь написаних. Програма шашок Артура Семюела, розроблена в середині 50-х і початку 60-х років, зрештою досягла достатньої майстерності, щоб кинути виклик респектабельному любителю. Робота над шашками та шахами завершилася поразкою Гаррі Каспарова від комп'ютера IBM Deep Blue в 1997 році [6].

Саме в золотий вік відео-аркадних ігор ідея NPC супротивників була значною мірою популяризована завдяки успіху Space Invaders, який мав зростаючий рівень складності, чіткі моделі пересування та події в грі, залежні від хешування функції на основі введення гравця. Galaxian додав більш складні та різноманітні рухи противника, включаючи маневри окремих ворогів, які вириваються із строю. Pac-Man представив шаблони штучного інтелекту в іграх з лабіринтами, з додаванням різних особистостей для кожного ворога. Karate Champ пізніше представив шаблони штучного інтелекту в іграх-файтингах [7]. First Queen була тактичною екшн-рольовою ігрою, яка представляла персонажів, якими можна керувати за допомогою штучного інтелекту комп'ютера, ідучи за лідером. Рольова відеогра Dragon Quest IV представила систему «Тактики», де користувач може налаштувати підпрограми штучного інтелекту негравових персонажів під час бою, концепцію, пізніше введену в жанр екшн-рольової гри Secret of Mana [7].

Процес розробки комп'ютерної гри потребує не тільки великих затрат в часі, а й розбиття розробки на кілька різних етапів з метою простоти і зрозумілості процесу розробки. Самі етапи розробки гри зазвичай включають себе 6 кроків [8], а саме програмування, кодування, рендеринг, розробку та тестування гри (і всіх її елементів: звуку, рівнів, персонажів та інших ресурсів тощо). Далі розглянемо кожен з 6 кроків.

Першим кроком є висока концепція. Цей крок заключається в розробці ідеї гри. Це дуже короткий опис, у якому ви намагаєтеся скласти свої ідеї до

кількох речень. Це дозволяє напрям, в якому буде розвиватись гра та на яку аудиторію вона буде орієнтована.

Наступним другим кроком є пітч. Під час пітчу ви викладаєте свою ідею в узагальненій формі, щоб найкраще передати команді розробки свої ідеї та їхні цілі. Це також час пояснити, чому було б доцільно проводити розробку гри та ідеї як сам це робити.

Третім кроком є концепція. Це крок який створюється на основі перших двох кроків. Його ціль лежить в конкретизації всіх аспектів гри. Саме тут окреслюється історія світу гри, її основний сюжет, персонажі, додаткові механіки, аналіз ризиків та купа іншої додаткової ніормації.

Далі слідує крок чотири – написання документу дизайну гри. Цей документ містить аспекти, пов’язані з реальним процесом гри, і може містити прототипні аспекти самого проекту. Його задача остаточно сформувати бачення того, якою має бути гра і дати основний напрям ідеї програмістам, щоб розробити тестову версію гри.

Кроком п’ять є створення прототипу. Саме тут команда можете проявити себе на ранніх стадіях розробки гри. Створення прототипу дозволить розробити методи, за допомогою яких можна повністю реалізувати основну ідею гри. Це хороший спосіб побудувати доказ концепції, адаптуючи свої цілі до більш відчутної форми. Не дивлячись на те, що прототип є надзвичайно далеким від фінальної версії гри, він вже дозволяє побачити і спробувати на практиці чим може стати гра.

Фінальним кроком є виробництво гри. В цей етап починається розробка дизайну, створення рівнів, розширення функціоналу прототипу. Починають впроваджувати такі речі, як звук і ранні графічні ресурси. Гра нарешті починає набувати форму. В кінці цього кроку йде тестування гри з подальшим виправленням помилок – багів.

Ігрові алгоритми ШІ використовуються в широкому спектрі досить різнорідних сфер всередині гри. Найбільш очевидним є контроль над будь-якими NPC в грі, хоча «скриптинг» на даний момент є найпоширенішим

засобом контролю. Ці рукописні дерева рішень часто призводять до «штучної дурості», як-от повторюваність поведінки, втрата занурення або ненормальна поведінка в ситуаціях, які розробники не планували.

Пошук шляху, ще одне поширене використання ШІ, широко зустрічається в стратегічних іграх у реальному часі. Пошук шляху — це метод визначення того, як дістатися NPC з однієї точки на карті до іншої, беручи до уваги місцевість, перешкоди та, можливо, «туман війни». У комерційних відеоіграх часто використовується швидкий і простий «пошук шляху на основі сітки», коли місцевість відображається на жорсткій сітці однорідних квадратів, а алгоритм пошуку шляху, такий як A* або IDA*, застосовується до сітки [9]. Замість просто жорсткої сітки деякі ігри використовують неправильні багатокутники і збирають навігаційну сітку з областей карти, до яких NPC можуть пройти. Як третій метод, іноді розробникам зручно вручну вибирати «маршрути», які NPC повинні використовувати для навігації; ціна полягає в тому, що такі шляхові точки можуть створювати неприродні рухи. Крім того, шляхові точки, як правило, працюють гірше, ніж навігаційні сітки в складних середовищах. Крім статичного пошуку шляху, навігація є підполем ігрового штучного інтелекту, зосередженим на тому, щоб надати NPC можливість орієнтуватися в динамічному середовищі, знаходити шлях до цілі, уникаючи зіткнень з іншими сутностями (іншими NPC, гравцями...) або співпрацювати з їх (групова навігація). Навігація в динамічних стратегічних іграх з великою кількістю одиниць, наприклад Age of Empires або Civilization V, часто працює погано; одиниці часто заважають іншим підрозділам.

Однією з найбільш позитивних і ефективних функцій, які є в сучасних відеоіграх ШІ, є здатність полювати. ШІ спочатку відреагував дуже чорно-білим чином. Якби гравець перебував у певній зоні, то ШІ відреагував би або повністю наступаючим способом, або був би повністю оборонним. В останні роки впроваджується ідея «полювання»; у цьому стані «полювання» ШІ шукатиме реалістичні маркери, такі як звуки, які видає персонаж, або сліди, які вони могли залишити. Ці розробки в кінцевому підсумку дозволяють створити

більш складну форму гри. За допомогою цієї функції гравець може реально обміркувати, як наблизитися або уникнути ворога. Це особливість, яка особливо поширена в жанрі стелсу.

Іншим розвитком останніх ігор ШІ є розвиток «інстинкту виживання». Ігрові комп'ютери можуть розпізнавати різні об'єкти в навколишньому середовищі та визначати, чи це вигідно чи шкідливо для його виживання. Як і користувач, ШІ може шукати прикриття під час перестрілки, перш ніж вживати заходів, які в іншому випадку залишають його вразливим, наприклад, перезарядити зброю або кинути гранату. Можна встановити маркери, які вказують, коли потрібно реагувати певним чином. Наприклад, якщо NPC дається команда перевірити його здоров'я під час гри, тоді можна встановити додаткові команди, щоб він реагував певним чином при певному відсотку здоров'я. Якщо здоров'я нижче певного порогу, ШІ можна налаштувати так, щоб він втік від гравця та уникав його, поки не буде запущена інша функція. Іншим прикладом може бути, якщо ШІ помітить, що в нього закінчилися кулі, він знайде об'єкт укриття і сховається за ним, поки не перезарядиться. Такі дії роблять ШІ більш людським. Однак у цій сфері все ще є потреба в удосконаленні.

1.2Аналіз сучасних технологій створення штучного інтелекту

Методи створення ШІ в комп'ютерних іграх можна поділити на 2 типи, а саме детерміновані та не детерміновані методи.

Детерміновані прийоми штучного інтелекту – це прийоми, які використовують рівняння або алгоритми, які були раніше розроблені для подібних ситуацій. Ці методи не передбачають стохастичних чи статистичних підходів. Детерміновані методи, як правило, легше та швидше застосовуються і легко піддаються комп'ютерним додаткам. Однак вони можуть не забезпечувати найбільш детальні чи найточніші моделі пласта. Детерміновані методи є найбільш широко використовуваними ігровими методами ШІ.

Детермінована поведінка або продуктивність вказується і є дуже передбачуваною. У цих методах просто немає елемента невизначеності. Вони досить швидкі та прості в реалізації, розумінні, тестуванні та налагодженні. Проблема в тому, що детерміновані методи змушують розробників передбачати всі можливі сценарії і самостійно кодувати всю поведінку. Ці методи навіть не дозволяють навчатися або розвиватися, що робить поведінку гри передбачуваною після невеликого ігрового процесу і навіть має обмежуючий вплив на ігрове життя [7].

Недетермінований алгоритм — це алгоритм, який навіть для одного і того ж вхідного сигналу може проявляти різну поведінку під час різних запусків, на відміну від детермінованого алгоритму. Існує кілька способів, яким алгоритм може вести себе по-різному від виконання до виконання. Поведінка ймовірнісного алгоритму залежить від генератора випадкових чисел. Алгоритм, який вирішує проблему за недетермінований поліноміальний час, може виконуватися за поліноміальний або експоненційний час залежно від вибору, який він робить під час виконання. Недетерміновані алгоритми часто використовуються для пошуку наближення до рішення, коли точне рішення було б надто дорогим для отримання з використанням детермінованого. Недетермінована поведінка має певний рівень невизначеності (який залежить від методу ШІ, який використовується, і від того, наскільки добре цей метод ШІ зрозумілий). Якщо ви хочете краще зрозуміти, про що йдеться, просто подивіться на NPC, який вивчає ходи та тактику гравця та пристосовується проти них. Для такого навчання може бути використана нейронна мережа, байєсівська техніка або генетичний алгоритм.

Розробникам ігор навіть не потрібно буде передбачати всі можливі сценарії та поведінку коду відповідно до них. Ці методи можуть навіть вчитися та екстраполювати самостійно і сприяти виникненню поведінки – поведінці, яка виникає без чітких інструкцій [10].

1.3Аналіз об’єкту проектування штучного інтелекту персонажів комп’ютерної гри

1.3.1Постановка задачі підвищення рівня штучного інтелекту комп’ютерної гри

Сучасні рішення штучного інтелекту, що схожі по своєму призначенню містять в собі занадто детерміновані, а це в свою чергу зменшує варіативність дій противника і цікавість самої гри.

Вхідними даними для інформаційної технології є рання альфа версія гри з режимом гри зверху жанру шутер з елементами Rouge-like та покрашений штучний інтелект, що може бути протестований в самій грі.

Вихідними даними є штучний інтелект, що здатен взаємодіяти з використовуваними внутрішньо ігровими предметами, наносити атаки по гравцеві та ухилятися від атак гравця.

Для коректної роботи інформаційної технології створення штучного інтелекту комп’ютерної гри система повинна відповідати наступним вимогам:

– операційна система сімейства Windows 7\8\10.

1.3.2 Особливості створення штучного інтелекту в комп’ютерній грі

При створенні ШІ доводиться працювати зі стандартними комп’ютерними платформами, програмним забезпеченням для 3D або 2D анімації, ігровими двигунами, інструментами профілювання та програмним забезпеченням штучного інтелекту. Також зазвичай програмісту необхідно написати спеціальний код для інструментів ШІ, необхідних для гри.

ШІ надає грі свій розум, за допомогою алгоритмів, які встановлюють поведінку персонажів і елементів на основі ігрового процесу окремого гравця. Це робиться шляхом налаштування реакції ігрового процесу на дії гравця. Такі елементи, як пошук шляху, груповий рух і керування камерою, усі вбудовані в стратегію ігрового процесу.

Програмування штучного інтелекту — це вузькоспеціалізована сфера розробки ігор, вона має бути бездоганною та непомітною для гравця, покращуючи досвід на підсвідомому рівні. Ця сфера розробки полягає в забезпеченні гри «мозком», який працює інстинктивно та незалежно на основі ігрового процесу окремого гравця.

У ігровій сфері програмісти створюють дерева рішень і проектують нейронні мережі в грі, створюючи штучні нервові системи. Програмування штучного інтелекту є передовим у розробці ігор, оскільки воно має глибокий вплив на ігровий процес, про який гравці можуть не усвідомлювати, що забезпечує динамічний та інтуїтивно зрозумілий досвід.

ШІ використовується для створення історій і ситуацій. ШІ найчастіше використовується в інтерактивних наративах. Користувачі можуть створювати або впливати на драматичну історію своїми діями або тим, що вони говорять у такій грі. Аналіз тексту використовується алгоритмами ШІ, які потім створюють сценарії на основі минулого досвіду оповідання. Одним із найвідоміших прикладів такої програми є Dungeon 2 [11]. У грі використовується технологія генерації тексту з відкритим вихідним кодом, розроблена OpenAI, навчена на романах Choose Your Own Adventure.

Здатність алгоритмів ШІ моделювати складні системи є головною привабливістю. Геймери постійно прагнуть зробити свої ігри більш захоплюючими та реалістичними. Однак моделювати реальність важко. Алгоритми штучного інтелекту гри можуть прогнозувати наслідки рішень гравців і такі речі, як погода та емоції, щоб врахувати складність гри.

ШІ більшості сучасних ігор — це попередньо запрограмовані NPC; однак це на порозі зміни. Це зробить їх більш непередбачуваними та з ними буде цікавіше взаємодіяти. ШІ має кілька переваг для гри. Найпомітнішим є те, що з розвитком гри NPC стають розумнішими та реагують на ігрове середовище інноваційними та оригінальними способами.

1.3.3 Характеристика та аналіз інтелектів аналогів персонажів комп’ютерної гри

До порівняння було обрано гру схожого жанру і напрямку геймплею під назвою «Enter the Gungeon».

Enter the Gungeon — це відеогра, жанру булетхелл рогалик, розроблена Dodge Roll та опублікована Devolver Digital. Вона був випущена у всьому світі для Microsoft Windows, OS X, Linux та PlayStation 4 5 квітня 2016 року, на Xbox One 5 квітня 2017 року, на Nintendo Switch 14 грудня 2017 року та на Stadia 22 грудня 2020 року [12].

Ігровий процес працює наступним чином: гравець вибирає одного з чотирьох головних героїв на ім’я «Marine», «Convict», «Pilot», і «Hunter», кожен з яких має різні спеціальні здібності, такі як виклик підтримки або злом скринь. Поки гравець спускається в Gungeon, він має пройти кілька поверхів, кожен з яких має випадкову кількість кімнат. Поки кімнати попередньо визначені, сузір’я кімнат на поверсі, вороги, які з’являються в кімнатах, і скарби генеруються процедурно. Кожна кімната містить набір ворогів, які різняться за силою, витривалістю та поведінкою в атаці, де атака може варіюватися від простих, прямих пострілів до складної суміші пострілів, здійснених одночасно. Гравець може ухилятися від атак, виконуючи кидок ухилення, який був натхненний серією відеоігор Souls, і є невразливим під час дії, або, як альтернатива, перевертати столи та використовувати їх як укриття, хоча столи можуть бути знищені, якщо в них стріляють. Гравець має обмежену кількість «заготовок» для кожного поверху, які можна використовувати, щоб знищити всі поточні снаряди та завдати 10 шкоди всім ворогам у цій кімнаті, певні предмети додають бафи до «заготовок». Щоб перемогти ворогів, гравець повинен використовувати зброю, яку можна знайти в скринях, виграти, перемігши босів, або купити в магазинах, розкиданих по поверхах в Gungeon. В кінці кожного поверху гравця чекає бос; Побиття боса дає гравцеві зброю або предмет і валюту, які можна витратити в магазинах, і відкриває наступний поверх.

Рисунок 1.1 – Геймплей «Enter the Gungeon»

Далі перейдемо до порівняння штучного інтелекту звичайного ворожого NPC з гри «Enter the Gungeon» з ім’ям «Bullet Kin» з розвинутим штучним інтелектом.

Таблиця 1.1 – Порівняльний аналіз основних характеристик інтелекту NPC

	«Bullet Kin»	Розвинутий ШІ
Можливість атакувати	Рідкісні не точні атаки	Помірні точні атаки
Швидкість противника	Низька	Висока
Можливість ухилятись	-	+
Можливість робити укриття	+	-
Можливість використовувати укриття	+	+/-
Можливість використовувати предмети	-	+

1.4 Висновок до розділу 1

У даному розділі було проведено аналіз предметної області штучного інтелекту в комп’ютерних іграх було визначено засоби для реалізації поставленої задачі, технології створення штучного інтелекту в комп’ютерних іграх та аналоги реалізації штучного інтелекту в грі з поглядом зверху. Серед інформаційних даних було проаналізовано методи створення штучного інтелекту, що містять необхідний потенціал для покращення та подальшої реалізації власного штучного інтелекту.

2 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПІДВИЩЕННЯ РІВНЯ ІНТЕЛЕКТУАЛЬНОСТІ ПЕРСОНАЖІВ КОМП'ЮТЕРНОЇ ГРИ

2.1 Аналіз алгоритмів пошуку шляху штучним інтелектом персонажів комп'ютерної гри

Враховуючи рухливий геймплей комп'ютерної гри що розроблюється з'являється необхідність в рухливості NPC. Забезпечити це можна за допомогою алгоритмів пошуку шляху. Але є багато алгоритмів пошуку шляху. Тому потрібно визначити який саме алгоритм необхідно обрати з великої кількості їх різновидів, відомих на сьогоднішній день. Розглянемо деякі з алгоритмів, а саме алгоритм алгоритм Дейкстри, Беллмана — Форда, алгоритм пошуку A* та оберемо той, що найбільше підходить для розв'язання поставленої задачі.

Алгоритм Дейкстри — це алгоритм найкоротшого шляху з одним джерелом. Тут єдине джерело означає, що задано лише одне джерело, і ми повинні знайти найкоротший шлях від джерела до всіх вузлів. Він був задуманий комп'ютерним науковцем Едсгером В. Дейкстрою в 1956 році. Дейкстра думав про проблему найкоротшого шляху, коли працював програмістом у Математичному центрі в Амстердамі, щоб продемонструвати можливості нового комп'ютера під назвою ARMAC.[13] Його мета полягала в тому, щоб вибрати як проблему, так і рішення (яке було б створено за допомогою комп'ютера), які могли б зрозуміти люди, які не володіють комп'ютерами. Він розробив алгоритм найкоротшого шляху, а пізніше реалізував його для ARMAC для дещо спрощеної транспортної карти 64 міст у Нідерландах (64, так що 6 біт було б достатньо для кодування номера міста) [14]. Через рік він зіткнувся з іншою проблемою інженерів апаратного забезпечення, які працювали над наступним комп'ютером інституту:

мінімізувати кількість дроту, необхідного для з’єднання контактів на задній панелі машини. Як рішення він заново відкрив алгоритм, відомий як алгоритм мінімального остовного дерева Пріма. Дейкстра опублікував алгоритм у 1959 році, через два роки після Прима та через 29 років після Ярніка [15].

Виконується алгоритм наступним чином: нехай вузол, з якого ми починаємо, буде називатися початковим вузлом. Відстань вузла Y буде відстанню від початкового вузла до Y. Алгоритм Дейкстри спочатку почне з нескінченних відстаней і намагатиметься покращити їх крок за кроком [16]. Сам алгоритм виконується наступним чином:

1. Всі вузли позначаються як не відвідані. Створюється набір усіх не відвіданих вузлів, який називається не відвіданим набором.

2. Кожному вузлу призначається умовне значення відстані: для нашого початкового вузла значення становить рівним нулю та нескінченністю для всіх інших вузлів. Під час роботи алгоритму орієнтовна відстань вузла V — це довжина найкоротшого шляху, знайденого на даний момент, між вузлом V і початковим вузлом. Оскільки спочатку не відомий жоден шлях до будь-якої іншої вершини, крім самого джерела (який є шляхом нульової довжини), усі інші попередні відстані спочатку встановлені на нескінченність. Встановіть початковий вузол як поточний.

3. Для поточного вузла розглядаються всі його не відвідані сусіди і обчислюється їхні орієнтовні відстані через поточний вузол. Після цього виконується Порівняння щойно розрахованої орієнтованої відстань із тією, яка зараз призначена сусідові, і призначається їй менша. Наприклад, якщо поточний вузол A позначено відстанню 6, а ребро, що з’єднує його з сусіднім B, має довжину 2, то відстань до B через A буде 6 + 2 = 8. Якщо B раніше було позначено відстань більше 8, потім змініть його на 8. В іншому випадку поточне значення буде збережено.

4. Коли розгляд усіх не відвіданих сусідів поточного вузла закінчується виконується позначення поточного вузла як відвіданий і видалення його з не відвіданого набору. Відвіданий вузол більше ніколи не

перевірятиметься (це є дійсним і оптимальним у зв'язку з поведінкою на кроці 6: наступні відвідувані вузли завжди будуть у порядку «спочатку найменша відстань від початкового вузла», тому будь-які відвідування після цього будуть мають більшу відстань).

5. Якщо вузол призначення було позначено як відвіданий (під час планування маршруту між двома певними вузлами) або якщо найменша орієнтовна відстань між вузлами в не відвіданому наборі дорівнює нескінченності (під час планування повного обходу; відбувається, коли немає зв'язку між початковим вузлом і залишилися не відвіданими вузлами), алгоритм завершено.

6. В іншому випадку необхідно вибрати не відвіданий вузол, позначений найменшою орієнтовною відстанню, та установити його як новий поточний вузол і повернутися до кроку 3.

Алгоритм Дейкстри використовує структуру даних для зберігання та запиту часткових рішень, відсортованих за відстанню від початку. Тоді як оригінальний алгоритм використовує чергу з мінімальним пріоритетом і виконується за час $O((|V| + |E|)\log|V|)$, де $|V|$ кількість вузлів і $|E|$ це кількість ребер), його також можна реалізувати у $O(V^2)$ за допомогою масиву [16].

Алгоритм Беллмана–Форда — це алгоритм, який обчислює найкоротші шляхи від однієї вихідної вершини до всіх інших вершин у зваженому графі. Він повільніший, ніж алгоритм Дейкстри для тієї ж проблеми, але більш універсальний, оскільки він здатний обробляти графіки, в яких деякі ваги ребер є від'ємними числами. Алгоритм вперше був запропонований Альфонсо Шимбелом (1955), але замість цього названий на честь Річарда Беллмана та Лестера Форда молодшого, які опублікували його в 1958 та 1956 роках відповідно [17]. Едвард Ф. Мур також опублікував варіацію алгоритму в 1959 році, і з цієї причини його також іноді називають алгоритмом Беллмана–Форда–Мура.

Подібно до алгоритму Дейкстри, алгоритм Беллмана–Форда діє шляхом релаксації, у якій наближення до правильної відстані замінюються кращими,

доки вони врешті-решт не досягнуть рішення. В обох алгоритмах приблизна відстань до кожної вершини завжди є переоцінкою справжньої відстані та замінюється мінімальним значенням її старого значення та довжини щойно знайденого шляху. Однак алгоритм Дейкстри використовує чергу пріоритетів для жадібного вибору найближчої вершини, яка ще не була оброблена, і виконує цей процес релаксації на всіх її вихідних ребрах; навпаки, алгоритм Беллмана–Форда просто розслаблює всі ребра, і робить це $|V|-1$ разів, де $|V|$ кількість вершин у графі. У кожному з цих повторів кількість вершин з правильно розрахованими відстанями зростає, з чого випливає, що в кінцевому підсумку всі вершини матимуть свої правильні відстані [18]. Цей метод дозволяє застосовувати алгоритм Беллмана–Форда до ширшого класу вхідних даних, ніж Дейкстри. Проміжні відповіді залежать від порядку розслаблення ребер, але остаточна відповідь залишається незмінною.

Алгоритм Беллмана–Форда працює за час $O(|V| \cdot |E|)$, де $|V|$ і $|E|$ є числом вершин і ребер відповідно [18].

A* («A-star») — це алгоритм обходу графа та пошуку шляхів, який використовується в багатьох галузях інформатики завдяки повноті, оптимальності та оптимальній ефективності. Порівняно з алгоритмом Дейкстри, алгоритм A* знаходить лише найкоротший шлях від зазначеного джерела до вказаної цілі, а не дерево найкоротших шляхів від зазначеного джерела до всіх можливих цілей. Це необхідний компроміс для використання евристики, спрямованої на конкретну мету. Для алгоритму Дейкстри, оскільки генерується все дерево найкоротшого шляху, кожен вузол є ціллю, і не може бути евристики, спрямованої на конкретну мету.

A* був створений в рамках проекту Shakey, який мав на меті побудувати мобільного робота, здатного планувати власні дії [19]. Нільс Нільссон спочатку запропонував використовувати алгоритм пошуку по графу для планування шляху Shakey. Пошук по графу керується евристичною функцією *h(n)*, оціненою відстанню від вузла n до цільового вузла: він повністю ігнорує *g(n)*, відстань від початкового вузла до n. Бертрам Рафаель запропонував

використовувати суму *g(n) + h(n)* [20]. Пітер Харт винайшов концепції, які ми зараз називаємо допустимістю та узгодженістю евристичних функцій. A* спочатку був розроблений для пошуку шляхів з найменшою вартістю, коли вартість шляху є сумою його витрат, але було показано, що A* можна використовувати для пошуку оптимальних шляхів для будь-якої задачі, яка задовольняє умови алгебри вартості.

Щоб пояснити принцип роботи алгоритму розглянемо квадратну сітку з багатьма перешкодами, і нам дано початкову клітинку та цільову клітинку. Ми хочемо досягти цільової клітини (якщо це можливо) з початкової клітини якомога швидше.

Алгоритм пошуку A* на кожному кроці вибирає вузол відповідно до значення *«f»*, яке є параметром, що дорівнює сумі двох інших параметрів — *«g»* і *«h»* [21]. На кожному кроці він вибирає вузол/комірку з найменшим *«f»* і обробляє цей вузол/комірку.

Нижче ми визначаємо *«g»* і *«h»* якомога простіше

g це вартість руху для переміщення від початкової точки до заданого квадрата на сітці, дотримуючись шляху, згенерованого для того, щоб дістатися туди.

h це розрахункова вартість переміщення від даного квадрата на сітці до кінцевого пункту призначення. Це часто називають евристикою, яка є не чим іншим, як розумним припущенням. Ми дійсно не знаємо фактичну відстань, поки не знайдемо шлях, тому що всілякі речі можуть бути на шляху (стіни, вода тощо). Існує багато способів обчислення цього *«h»*, які обговорюються в наступних розділах.

Далі алгоритм виконується за наступними кроками:

1. Початковий вузол знаходиться в списку відкритих вузлів;
2. Виконується перевірка чи є пустим список відкритих вузлів. Якщо список пустий то алгоритм закінчується помилкою;

3. З відкритих вузлів обирається той, який має найменше значення функції оцінки (g+h), якщо вузол n є цільовим вузлом то цільва задача виконана, інакше наступний пункт;

4. Вузол n розгортається і здійснюється розгляд всіх його наступників. Вузол n стає закритим. Для кожного наступника n' здійснюється перевірка на наявність в списках і якщо вони відсутні в списках, тоді обчислюється функція оцінки для n' і переміщення його у список відкритих;

5. Інакше, якщо значення вузла n' уже знаходиться у відкритих та закритих, його слід прикріпити до заднього вказівника, який відображає найнижче значення g(n');

6. Повернення до кроку 2.

Часова складність алгоритму A* становить $O(|V|^2)$, де $|V|$ є множиною вершин в графі. Але алгоритм може бути прискореним, а саме якщо кожна вершина зберігатиме вказівник на відповідний об'єкт в купі, то час роботи функції зменшиться до $O(|V| \cdot \log |V|)$ [21].

Серед розглянутих алгоритмів найбільш підходящим для даної роботи є алгоритм A*. Даний алгоритм пропонує вищу швидкодію за алгоритми аналоги, а також при цьому залишаючись простим в реалізації. Ще одним плюсом в користь A* є те, що він призначений для пошуку шляху в місцях складної будови. В свою чергу алгоритм містить в собі недолік, у вигляді великої кількості пам’яті необхідної на обрахунок, який не є значним для даної задачі так як одночасно в грі знаходиться низька кількість NPC.

2.2 Аналіз способів реалізацій штучного інтелекту персонажів в комп’ютерних іграх

Традиційно NPC програмувалися за допомогою автоматів на основі правил і кінцевих автоматів. Для побудови цих систем було потрібно багато умовних умов, які надають NPC детерміновані дії. Розробники використовували нечітку логіку, щоб скоротити час розробки та додати до ігор

ступінь непередбачуваності. Алгоритми пошуку шляху був одним із перших застосувань штучного інтелекту в програмуванні ігор. Іншими методами використовуються сценарії, експертні системи та методи штучного життя.

У популярних іграх, таких як Black & White, Battlecruiser 3000AD, Creatures, Dirt Track Racing, Fields of Battle і Heavy Gear, дизайнери використовували не детерміновані методи, такі як дерева рішень, дерево поведінки, (глибинні) нейронні мережі, генетичні алгоритми та методи навчання з підкріпленням [22]. Розглянемо ці підходи докладніше.

Почнемо з дерев рішень, які є методами навчання під наглядом, які можна навчити виконувати класифікацію та регресію. Це один із найпростіших алгоритмів машинного навчання для розробки ігор. Вони можуть допомогти вам оцінити значення змінної, яка вас цікавить, виводячи прості правила прийняття рішень з характеристик даних.

Дерева рішень легко зрозуміти та інтерпретувати, а оцінка результатів не потребує багато часу. Також є багато складних методів візуалізації дерева. Моделі білого ящика розробили моделі, які можна перевірити за допомогою різноманітних статистичних тестів.

Дерево рішень — це форма техніки, яка використовується для створення ігор зі штучним інтелектом. У ігровому дизайні використовуються таблиці рішень (прогнозування дій) [23]. У більшості сучасних відеоігор використовуються дерева рішень, особливо сюжетно-базовані. Дерева рішень можуть допомогти гравцям зрозуміти, як їхні рішення вплинуть на майбутнє, якщо вони пройдуть через них.

Дерево поведінки — це дерево ієрархічних вузлів, які контролюють процес прийняття рішень об’єктом штучного інтелекту. У межах дерева, листках, є фактичні команди, які керують об’єктом штучного інтелекту, а формування гілок – це різні типи службових вузлів, які керують проходженням штучного інтелекту деревами, щоб отримати послідовності команд, які найкраще відповідають ситуації.

Дерева можуть бути надзвичайно глибокими, з вузлами, що викликають піддерева, які виконують певні функції, дозволяючи розробнику створювати бібліотеки поведінки, які можна об’єднати разом, щоб забезпечити дуже переконливу поведінку ШІ. Розробка є дуже ітеративною, де є можливість почати з формування базової поведінки, а потім створити нові гілки, щоб мати справу з альтернативними методами досягнення цілей, з гілками, упорядкованими за їх бажаністю, дозволяючи штучному інтелекту мати запасну тактику, якщо певна поведінка не вдасться. Дерева поведінки мають певну схожість з ієрархічними кінцевими автоматами з тією ключовою різницею, що основним будівельним блоком поведінки є завдання, а не стан. Його легкість для розуміння людиною робить дерева поведінки менш схильними до помилок і дуже популярними в спільноті розробників ігор.

Ключовим аспектом дерев поведінки є те, що на відміну від методу у кодовій базі, для завершення певного вузла чи гілки в дереві може знадобитися багато кроків гри. У базовій реалізації дерев поведінки система проходитиме вниз від кореня дерева кожен окремий кадр, перевіряючи кожен вузол вниз по дереву, щоб побачити, який з них активний, повторно перевіряючи будь-які вузли на цьому шляху, доки не досягне поточного активного вузла до поставте галочку ще раз.

Штучні нейронні мережі — це штучні мізки, побудовані на основі алгоритмів навчання, структура яких нагадує структуру людського мозку. Глибоке навчання може вивчати різні характеристики з навчальних даних і, як наслідок, може моделювати надзвичайно складні реальні та ігрові ситуації [24]. На відміну від класичних підходів штучного інтелекту, мережеві мережі долають певні недоліки в дизайні ігрового агента. Крім того, глибоке навчання є само адаптивними та легко пристосовуються до змін налаштувань гри в режимі реального часу.

Глибоке навчання наразі набуває популярності як інструмент розробки ігрових агентів. Глибоке навчання в іграх використовує кілька рівнів нейронних мереж, щоб «витягувати ознаки з вхідних даних» шляхом їх поступового

розкладання [24]. При керуванні одним або декількома ігровими агентами багаторівневий підхід глибокого навчання і підвищена архітектурна складність дозволяють отримувати кращі результати порівняно з попередніми підходами. Залежно від сценарію, це можуть бути NPC або саме ігрове середовище.

Генетичний алгоритм — це більш складний підхід, відомий як евристичний, заснований на ідеї природної еволюції. Генетичний алгоритм імітує природний відбір, вибираючи найсильніших особин для створення нащадків наступного покоління [25].

Генетичні алгоритми широко використовуються для оптимізації. Порівняно з іншими методами оптимізації, генетичні алгоритми дають видатні результати для багатокритеріальної оптимізації. Генетичні алгоритми використовувалися в настільних іграх, які використовували різні стратегії пошуку для пошуку найкращих ходів у минулому. Адаптація поведінки NPC за допомогою сучасних застосувань генетичний алгоритм допомагає їм захищатися від сильних, але передбачуваних тактик, які можуть використовувати гравці-люди. Ігрові агенти штучного інтелекту створені, щоб бути більш реалістичними, але вони також мають недоліки. Генетичні алгоритми робить ігровий процес більш реальним, обмежуючи людей-гравців або інших агентів штучного інтелекту від виявлення лазівок і перемоги в грі за допомогою нескінченних кроків, які завжди ведуть до успіху. Розширена ігрова здатність є кінцевим наслідком генетичних алгоритмів.

Навчання з підкріпленням — це форма машинного навчання, яка передбачає навчання методом проб і помилок. Під час навчання модель може відтворювати події та дізнаватися про те, вдалися вони чи провалилися [25].

У динамічних і невизначених умовах навчання з підкріпленням є перевагою. Навчання з підкріпленням вже деякий час використовується у відеоіграх. Як наслідок, доменів ігрової індустрії для тестування алгоритмів навчання з підкріпленням достатньо. У той же час деякі з найкращих комп’ютерних геймерів світу використовують навчання з підкріпленням

(AlphaGo). З іншого боку, алгоритми навчання з підкріпленням недостатньо потужні для ігор високого рівня.

Враховуючи всі особливості штучного інтелекту, що розроблюється найбільш підходящим способом розробки штучного інтелекту персонажа комп’ютерної гри є дерево поведінки. Даний спосіб дозволить NPC паралельно виконувати декілька дій паралельно, тим самим бути достатньо непередбачуваним для гравця і в той же час мати логічну поведінку. Ще ця паралельність дозволить персонажу здаватись розумним і небезпечним супротивником в очах гравця за рахунок мультизадачності.

2.3 Розробка структури інформаційної технології підвищення рівня інтелектуальності персонажів комп'ютерної гри

При розробці розвинутого штучного інтелекту в NPC буде правильним розділити його на декілька станів, між якими будуть здійснюватись переходи в залежності від ігрової ситуації.

Першим станом є пасивний стан. Цей стан є найменш агресивним у відношенні до гравця. Під час цього стану персонаж бродить по карті шукаючи корисні предмети. Якщо такий предмет буде знайдено персонаж перейде в наступний стан. Також в цьому стані персонаж намагається уникати гравця триматись на безпечній відстані. У випадку якщо NPC оточує стіна, за якою можна сховатись, він скористається цим. При спробі гравця зблизитись з комп’ютерним персонажем, штучний інтелект буде змушений втікати від гравця збільшуючии дистанцію та ставлячи в пріоритет пошук укриттів, в яких можна заховатись і продовжувати пошук внутрішньо ігрової зброї.

У випадку якщо комп’ютерний персонаж зміг помітити разом з собою внутрішньо ігровий предмет він переходить у наступний стан, а саме активний стан. Основною задачею цього стану є отримання підсилення до того як його помітить і отримає гравець. В залежності від кількості пунктів здоров’я дії персонажа, що керується комп’ютером варіруються. Якщо здоров’я персонажу

є більшим половини, то він не буде рухатись в сторону цього предмету на встановленим найкоротшим шляхом на максимально доступній йому швидкості. В інакшому випадку, а саме якщо шкала здоров’я є меншою або рівною половині то персонаж буде рухатись до предмету вибираючи найбільш безпечний шлях, враховуючи позицію де був помічений гравець раніше. Також в цьому підстані якщо комп’ютерний персонаж буде знаходитись не далеко від гравця, він буде намагатись ухилятись від атак гравця. У випадку якщо гравець підбере предмет раніше NPC буде змушений втікати повернувшись до пасивного стану.

Після цього якщо внутрішньо ігровому персонажу вдалось отримати зброю він переходить в стан бою. Це найбільш агресивний зі станів. В ньому комп’ютерний персонаж ставить собі в пріоритет знищення гравця. Першим ділом якщо навколо персонажу відсутній гравець то персонаж рухається в місце, де він останній раз зустрічався з гравцем. Якщо таке місце відсутнє, то він починає бродити по карті шукаючи гравця. Побачивши гравця штучний інтелект намагається атакувати його знайденою зброєю, при цьому тримаючись від нього на середній дистанції та ухиляючись від його атак. У випадку якщо персонаж в цьому стані помітить предмет, він почне рухатись до нього паралельно відстрілюючись від гравця. Якщо гравця біля знайденого предмету немає, то штучний інтелект починає охороняти цей предмет.

На основі описаного вище потрібно створити дерево поведінки. Для того, щоб створити своє дерево поведінки необхідно розглянути основні складові дерева. Цими складовими є основа, гілки та листя.

Основний вузол — це вузол, який може мати одного чи кількох дочірніх вузлів. Вони оброблятимуть один або декілька з цих дочірніх елементів у послідовності від першої до останньої або у випадковому порядку залежно від конкретного складеного вузла, про який йде мова, і на певному етапі вважатимуть їхню обробку завершеною та передадуть успіх або невдачу своєму батьківському вузлу, що часто визначається за успіх або невдача дочірніх

вузлів. Протягом часу, коли вони обробляють нащадків, вони продовжуватимуть повертати значення до батьків.

Найбільш часто основний вузол використовується для створення послідовностей, що просто запускають кожну дочірню частину послідовно, повертаючи помилку в момент невдачі будь-якого з дочірніх елементів і повертаючи успіх, якщо кожна дочірня частина повернула успішний статус.

Вузол гілка, як і основний вузол, може мати дочірній вузол. На відміну від складеного вузла, у них може бути лише один дочірній вузол. Їхня функція полягає в тому, щоб або трансформувати результат, отриманий від статусу дочірнього вузла, завершити дочірній елемент, або повторити обробку дочірнього вузла, залежно від типу вузла-гілки.

Зазвичай використовуваним прикладом гілки є інвертор, який просто інвертує результат нащадка. Якщо нащадок зазнає невдачі, вона поверне успіх своїм батькам, або якщо нащадок досягне успіху, вона повернеться невдачею батькам.

Листя є вузлом найнижчого рівня, і вони не можуть мати нащадків. Проте листя є найпотужнішими типами вузлів, оскільки вони будуть визначені та реалізовані вашою грою для виконання специфічних для гри чи персонажів тестів або дій, необхідних для того, щоб дерево справді робило корисні речі.

Розглянувши основні елементи можна перейти до створення свого дерева поведінки.

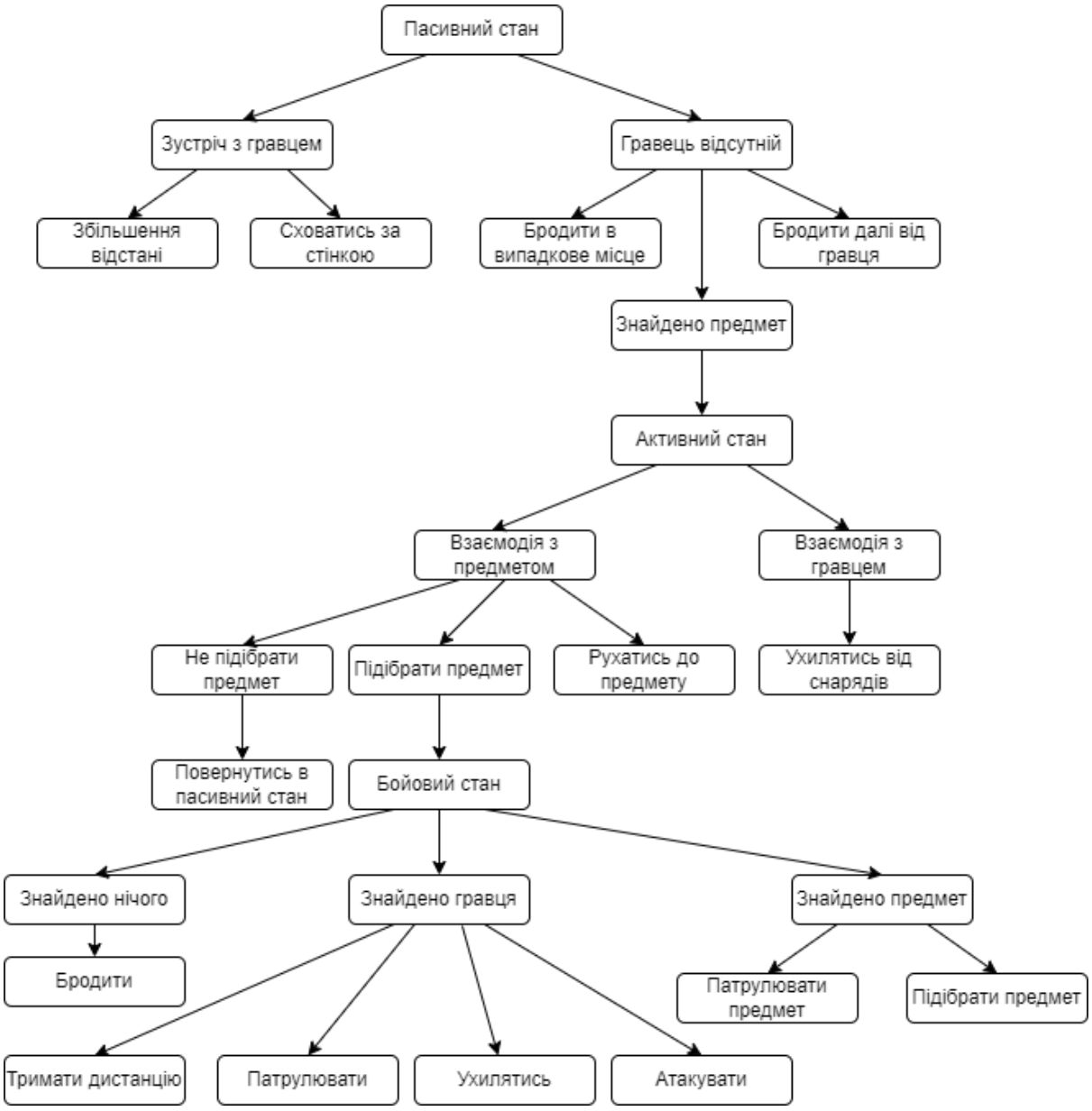


Рисунок 2.1 – Дерево поведінки штучного інтелекту

UML (уніфікована мова моделювання) - це мова моделювання, що використовується розробниками програмного забезпечення. UML можна використовувати для розробки діаграм та надання користувачам (програмістам) готових до використання виразних прикладів моделювання. Створення UML спочатку мотивувалося бажанням стандартизувати різні нотаційні системи та підходи до проектування програмного забезпечення. Він був розроблений в Rational Software у 1994–1995 рр., Подальший розвиток під керівництвом них до 1996 р [26].

Важливо розрізняти модель UML від набору діаграм системи. Діаграма - це часткове графічне зображення моделі системи. Набір діаграм не повинен повністю охоплювати модель, а видалення діаграми не змінює модель. Модель може також містити документацію, яка керує елементами моделі та діаграмами (наприклад, письмові випадки використання). UML діаграма класів гри зображено на рисунку 2.2.

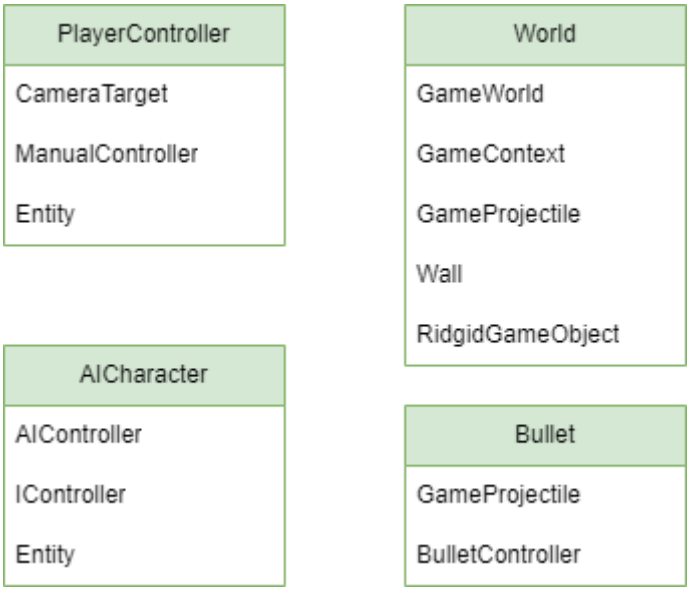


Рисунок 2.2 – UML діаграма класів комп’ютерної гри

Окрім цього потрібно не забувати, що створений штучний інтелект повинен підсилювати головну ціль гри, а саме бути цікавою для гравця. Зробити це можна врахувавши наступні рекомендації в розробці ШІ для персонажів гри [27]:

1.Додати час реакції. ШІ може просто знати, що все відбувається в грі відразу, але гравець не може. Отже, в іграх у реальному часі рекомендовано додати 100–200 мс затримки між діями ШІ.

2.Недосконалі дії. ШІ може ввести ідеальний кут із точністю 32 біта, де гравцеві пощастило прицілитися з точністю 12 бітів. Занадто точний ШІ може не давати шансів на перемогу гравцеві.

3.ШІ не повинен знати, чого не знає гравець. Приклад: якщо гравець бачить панель здоров’я монстра, ШІ також може не знати, скільки у гравця здоров’я.

4.Переробити код ШІ з параметрами. Сердитий штучний інтелект максимізує шкоду, роблячи необдумані дії, Обережний діятиме з затримкою, але прийматиме зважені рішення, Злий натхненник накопичуватиме ресурси, перш ніж розв’язати тотальну атаку. Це робить гру набагато цікавішою, оскільки гравець вивчає різні підходи.

5.Група ШІ. Якщо гравець може керувати групою одиниць, виконуйте прості команди високого рівня, як-от «слідуй за мною», «нападай на те, що я атакую», «залишайся тут, до біса» та «виживай».

6.Група противника ШІ. Шахрайський спосіб зробити хорошу поведінку у зграї полягає в тому, щоб вибрати лідера, запропонувати іншим слідувати за ним і залишатися на випадковій відстані від нього залежно від кількості одиниць. Коли лідер помирає, лідером стає найближчий.

7.Базовий штучний інтелект «ножиці для паперу». Якщо гравець продовжує кидати камінь, виберіть більше паперу. Цього достатньо, щоб створити ілюзію набагато більшого інтелекту.

Слідуючи даним рекомендаціям можна створити ШІ, який буде цікавим для гравця, становити загрозу, але в той же час все ще здатним до програшу.

2.4 Висновок до розділу 2

У цьому розділі було проаналізовано різні алгоритми пошуку шляху та обрано найбільш підходящий з них а саме алгоритм A*. Розроблено структуру та основні особливості ШІ. Також було визначено, а також реалізовано основний спосіб створення штучного інтелекту, а саме дерево прийняття рішень.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПІДВИЩЕННЯ РІВНЯ ІНТЕЛЕКТУАЛЬНОСТІ ПЕРСОНАЖІВ КОМП'ЮТЕРНОЇ ГРИ

3.1 Обґрунтування вибору програмно-апаратної платформи

Microsoft Visual Studio - лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення і ряд інших інструментів. Дані продукти дозволяють розробляти як консольні додатки, так і ігри та програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, підтримуваних Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework і Silverlight [28].

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторінга коду. Вбудований відладчик може працювати як відладчик рівня вихідного коду, так і відладчик машинного рівня. Решта вбудовуються інструменти включають в себе редактор форм для спрощення створення графічного інтерфейсу додатку, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати і підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду (як, наприклад, Subversion і Visual SourceSafe), додавання нових наборів інструментів (наприклад, для редагування і візуального проектування коду на предметно-орієнтованих мовах програмування) або інструментів для інших аспектів процесу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server).

Ще одною важливою перевагою Visual Studio є підтримка набору програмних інструментів MonoGame.

MonoGame — це безкоштовний фреймворк C# із відкритим кодом, який використовується розробниками ігор для створення ігор для кількох платформ та інших систем. Даний фреймворк реалізує інтерфейс прикладного програмування Microsoft XNA 4 Framework.

Графічні можливості MonoGame походять від OpenGL, OpenGL ES або DirectX. Починаючи з MonoGame версії 3, OpenGL 2 був у центрі уваги щодо можливостей. Попередні версії MonoGame (2.5) використовували OpenGL 1.x для візуалізації графіки. Використання OpenGL 2 дозволило MonoGame підтримувати шейдери, щоб розширити можливості візуалізації на платформі. MonoGame дозволяє розробляти ігри на різні платформи, а саме Windows, Linux, Xbox, PlayStation, iOS, Android, Mac OS [29].

3.2 Обґрунтування вибору мови програмування

До порівняння представлені мови програмування C#, Python, JavaScript.

C# — об'єктно-орієнтована мова програмування, що підтримує також і компонентно-орієнтоване програмування. Важлива особливість таких компонентів - це модель програмування на основі властивостей, методів і подій. Кожен компонент має атрибути, які надають декларативні відомості про компоненті, а також вбудовані елементи документації. C # надає мовні конструкції, безпосередньо підтримують таку концепцію роботи. Завдяки цьому C # відмінно підходить для створення і застосування програмних компонентів [30].

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня з строгою динамічною типізацією. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій так і у вихідній формі на

всіх основних платформах. В мові програмування Python підтримується декілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована [31].

JavaScript, часто скорочений як JS, — це мова програмування, яка відповідає специфікації ECMAScript. JavaScript є високорівневим, часто вчасно скомпільованим і багатопарадигмальним. Він має синтаксис фігурних дужок, динамічне введення тексту, орієнтацію на об’єкти на основі прототипу та функції першого класу. Як мова багатопарадигми, JavaScript підтримує керовані подіями, функціональні та імперативні стилі програмування. Він має інтерфейси прикладного програмування (API) для роботи з текстом, датами, регулярними виразами, стандартними структурами даних та об’єктною моделлю документа (DOM). Стандарт ECMAScript не включає введення / виведення (введення / виведення), наприклад, мережеві, сховища чи графічні засоби [32].

Порівняння мов програмування показано в таблиці 3.1.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Мова	Використання	Парадигми	Типізація	ООП	Стандар-тизація
C#	Застосування, RAD, бізнес, клієнтська та серверна сторони, загальний, веб, робототехніка	Імперативна, функціональна, рефлексивна, узагальнене програмування, декларативна, розподільча	Строга	+	ISO ECMA

Продовження Таблиці 3.1

Python	Створення застосувань, загальне, Web, скрипт, ШІ	Імперативна, функціональна, рефлексивна, узагальнене програмування, декларативна , розподільча	Строга	+	-
JavaScript	Серверна сторона, Web-застосування, Web	Імперативна, елементи функціональної, об'єктно-орієнтована	Не строга	+	ECMAScript

Мною було вибрано цю мову програмування, тому що C#:

1. Використовує проміжний код і тому може виконуватись на будь-яких комп'ютерах, незалежно від апаратного і системного забезпечення;
2. Є повністю об'єктно-орієнтованою − код можна писати тільки у класах;
3. Має багато доповнень, які полегшують програмування;
4. Має засоби автоматичного вивільнення непотрібної пам'яті;
5. Має зручний набір типів;
6. Підтримка стратегії .NET;
7. Наявність бібліотеки MonoGame.

Тому вона є найбільш актуальною і практичною, а також має більше переваг ніж недоліків порівняно із іншими мовами.

3.3 Програмна реалізація гри та штучного інтелекту персонажа комп'ютерної гри

При створенні комп'ютерної гри необхідно визначитись з особливостями її базового геймплею та того як саме він буде відображатись для гравця.

В розробленому прототипі гри гравець повинен змагатись в поєдинку з супротивниками, що керуються ШІ. Як гравець так і ШІ здатні взаємодіяти з внутрішньо ігровими предметами і використовувати їх один проти одного. Перемагаючи гравець буде розвиватись і отримувати більше варіантів способів боротьби проти NPC.

Враховуючи описаний геймплей буде доречним визначити жанр гри як rogue-like. Rogue-like розуміється як жанр ігор, що є подібними своєму "батькові" "Rogue". "Rogue" (також відомий як Rogue: Exploring the Dungeons of Doom) — це відеогра, що повзає підземеллями від Майкла Тоя та Глена Віхмана з пізнішим внеском Кена Арнольда. Rogue спочатку був розроблений приблизно в 1980 році для систем мейнфреймів на базі Unix як вільно розповсюджуваний виконуваний файл. [33]

У "Rogue" гравець бере на себе типову роль авантюриста ранніх фентезійних рольових ігор. Гра починається на самому верхньому рівні непозначеного підземелля з безліччю монстрів і скарбів. Мета полягає в тому, щоб пробитися на нижній рівень, отримати амулет Йендора ("Родні" пишеться задом наперед), а потім піднятися на поверхню. Перемагати монстрів на рівнях стає все важче. Поки амулет не буде отримано, гравець не може повернутися на попередні рівні.

По причині того, що жанр базується на Rogue він володіє тими ж особливостями, а саме:

1.Процедурна генерація ігрового процесу , починаючи з одержуваних нагород, закінчуючи генерацією цілих просторів, за якими ви будете переміщатися протягом цілого забігу.

2.Відсутність будь-яких обмежень перед гравцем : він вільний зробити будь-яку дурість або набір продуманих дій, які можуть вплинути на успіх в межах даного забігу в певній мірі.

3.Permadeath або «вічна смерть» , виконана в тій чи іншій формі, але виконує своє головне призначення - повернути гравця в самий початок після смерті його ігрового персонажа та необхідність почати з самого початку, що веде до втрати абсолютно всього досягнутого прогресу. Таким чином гравець постійно знаходиться в ситуації ризику і концентрації.

Окрім правил, які характерні для жанру Rouge-like також необхідно додати свої внутрішньо ігрові правила, щоб створити унікальність геймплею порівняно з іншими представниками аналогами. Тому головними правилами гри можна виділити наступне:

1. Основні правила жанру Rouge-like;
2. Ігрові предмети з’являються при переході на наступний рівень або з малим шансом під час знаходженні на рівні;
3. На кожному рівні знаходиться від одного до семи супротивників;
4. Персонаж гравця та NPC мають обмежену кількість здоров’я в розмірі десяти очок (при цьому гравець при переході на наступний рівень відновлює свої повністю);
5. Гра закінчується у випадку смерті персонажу гравця або досягненні двадцять першого рівня.

Тепер перейдемо до реалізації самого штучного інтелекту. Як було сказано в розділі два для реалізації пересування NPC було використано метод A*. Коли персонаж, що керується ШІ знаходиться в процесі «бродіння» по карті він вибирає випадкову точку, що є не заповненою перешкодами та здійснює рух до неї будуючи шлях за допомогою вищезгаданого алгоритму. У випадку знаходження предмету під час «бродіння» фінальною точкою пересування стає сам предмет. Під час зустрічі з гравцем в пасивному стані персонаж буде вибирати найближчу точку, що знаходиться за межами видимості гравця, беручи в пріоритет внутрішні стіни та інші перешкоди. В

агресивному стані навпаки пріоритетними точками стають ті, що є в видимості гравця, але є все ще далекими від нього, що створює чесну ситуацію для обох сторін.

Фрагмент коду алгоритму пошуку шляху має наступний вигляд:

```
class NPCwalk
{
      public float Xposition { get; set; }
      public float Yposition { get; set; }
      public float Cost { get; set; }
      public float Distance { get; set; }
      private float CostDistance => Cost + Distance;
      public float Parent { get; set; }

      public void SetDistance(int targetX, int targetY)
      {
            this.Distance = Math.Abs(targetX - Xposition) + 
Math.Abs(targetY - Yposition);
      }
}
```

Далі перейдемо до логіки стрільби штучного інтелекту. Після того як NPC підбирає внутрішньо ігрову зброю він може здійснювати стрільбу по гравцеві. Алгоритм обрахунку є наступним персонаж бере позицію персонажа гравця, його напрямок та прискорення руху гравця. Після цього він здійснює обрахунок приблизної траєкторії гравця роблячи не значну похибку в обрахунках, щоб не здійснювати стрільбу, з точністю якої гравець не здатен змагатись. Далі можна побачити фрагмент реалізації цього методу.

```
public ControlResult Hangle(Entity entity, GameContext context)
        {
            var nearestTarget = context
```

```
                    .GameObjects
                    .Where(obj => obj is Entity && (obj as Entity).Team
!= entity.Team)
                    .MinBy(obj   =>   (entity.Position   -   (obj   as
Entity).Position).Length()) as Entity;

                _state += 0.1f;

                return new ControlResult()
                {
                    Move      =      new      Vector2(MathF.Sin(_state),
MathF.Cos(_state)),
                    Angle     =     nearestTarget     !=     null     ?
MathUtil.GetAngle(nearestTarget.Position - entity.Position) : 0,
                    Action = false,
                    Shoot = true
                };
```

Ще одним важливим елементом є можливість до уникання пострілів гравця. Відбувається це за рахунок зчитування інформації про снаряди, що знаходяться в близькій відстані до NPC. Після цього персонаж здійснює рух в лівий або правий бік від напряму вектора прискорення снаряду. Так як фізика снарядів заповільнює рухомі снаряди з часом це дозволяє встановити залежність між відстанню, на якій гравець здійснив постріл та якістю уникання. Фрагмент коду реалізації можна побачити далі.

```
        Vector2 IGameObject.Position { get { return Position; } }

            public  GameProjectile(Texture2D  texture,  float  radius,
Material material) : base(radius, material)
            {
                Texture = texture;
                TextureDiameter = radius * 2;
```

```
        }

        public void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(Texture, Position, null, Color.White,
MathUtil.GetAngle(Velocity), new Vector2(Texture.Width / 2,
Texture.Height / 2), TextureDiameter / MathF.Max(Texture.Width,
Texture.Height), SpriteEffects.None, 1);
        }

        public void Update(float dt, GameContext context)
        {
            LifeTime -= dt;
            if (LifeTime < 0 || Velocity.LengthSquared() < 40)
            {
                context.RemoveObject(this);
            }
        }
```

3.4 Тестування інтелектуального штучного інтелекту та аналіз результатів

Після розробки гри запустимо та протестуємо розроблену гру. Початкове вікно гри показано на рисунку 3.1.

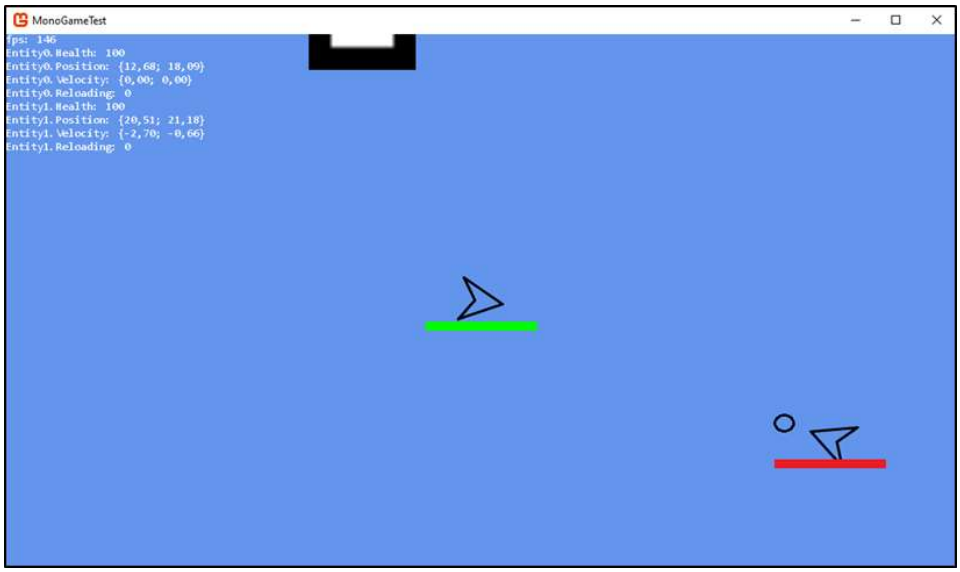


Рисунок 3.1 – Програмна реалізація гри

Після перевірки працездатності гри можна перейти до порівняння штучного інтелекту персонажа з персонажем аналогом. Для цього частину характеристик розробленого ШІ було зведено до значень близьких з інтелектом аналогом. Дані тестування внесено в таблицю 3.2.

Таблиця 3.2 Результати порівняння розробленого ШІ з інтелектом аналогом

Властивість	Розвинений ШІ	«Bullet Kin»	Відсоток покращення
Точність попадання	20 з 20	18 з 20	+10%
Точність попадання по рухомій цілі	12 з 20	5 з 20	+35%
Уникання попадання	17 з 20	3 з 20	+70%
Рух до кінцевої точки на одній швидкості	8 с	10 с	+20%
Середнє значення покращення			+33,75%

Як показує тестування розроблений ШІ є кращим за інтелект аналог на 33,75%. Окрім цього в переваги над аналогом можна внести можливість взаємодіяти з внутрішньо ігровими предметами до того як їх отримає гравець. Ця особливість створює цікавіший і важчий геймплей, змушуючи гравця діяти активно, а іноді більш агресивно, щоб досягти перемоги.

3.5 Висновок до розділу 3

У розділі було здійснено обґрунтування використання середовища Microsoft Visual Studio та програмних інструментів MonoGame. Також було здійснено порівняння трьох мов програмування, обрано C# та обґрунтовано її використання в розробці. Також було здійснено програмну реалізацію комп’ютерної гри, що демонструє покращений штучний інтелект, здійснено порівняння з інтелектом аналогом.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку нового програмного продукту штучного інтелекту персонажа комп'ютерної гри. Особливістю програми є те, що дана технологія є комп'ютерним додатком, що покликаний підвищити рівень інтелектуальності персонажів комп'ютерної гри і тим самим покращити геймплей та ігровий досвід гравців.

Аналогами можуть бути Enter the Gungeon, ціна 325 грн; The Binding of Isaac, ціна 229 грн. Ціна мого продукту буде становити 190 грн.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п'ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 4.1.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах

Продовження табл. 4.1

Ринкові переваги					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження табл. 4.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено до таблиці 4.2

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	3	4
Наявність аналогів на ринку	3	3	4
Цінова політика	5	4	4
Технічні та споживчі властивості виробу	3	4	4
Експлуатаційні витрати	3	3	2
Ринок збуту	4	4	4
Конкурентоспроможність	3	3	2
Фахівці з технічної і комерційної реалізації	4	5	4
Фінансування	3	3	3
Матеріально-технічна база	4	4	4
Термін реалізації ідеї	4	5	5
Супровідна документація	3	4	3
Сума	42	45	43
Середньоарифметична сума балів	(42+45+43) / 3 = 43		

За даними таблиці 4.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 4.3.

Таблиця 4.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок того, що програмний продукт відрізняється від існуючих тим, що дана технологія є більш інтелектуальним штучним інтелектом для персонажів комп'ютерної гри, що досягається за рахунок покращеної взаємодії з персонажем гравця, а також більш складної поведінки.

4.2 Прогнозування витрат на виконання науково-дослідної роботи

Основна заробітна плата дослідників Витрати на основну заробітну плату дослідників ($З_о$) розраховують відповідно до посадових окладів працівників, за формулою 4.1.

$$З_о = \sum_{i=1}^{k_i} \frac{M_{ni} \cdot t_i}{T_p} \quad , \qquad (4.1)$$

де k – кількість посад дослідників, залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – кількість днів роботи конкретного дослідника, дн.;

t_i – середня кількість робочих днів в місяці, T_p= 28 дні.

Результати розрахунків зведемо до таблиці 4.4.

Таблиця 4.4 – Витрати на заробітну плату дослідників

Посада	Місячний посадовий оклад (грн)	Оплата за робочий день (грн)	Кількість днів роботи	Витрати на заробітну плату (грн)
Керівник проекту	20000,00	43,47	28	20000,00
Інженер програміст	18000,00	41,06	28	19285,71
Всього				39285,71

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

Додаткова заробітна плата дослідників та робітників Додаткова заробітна плата розраховується як 11% від суми основної заробітної плати дослідників та робітників за формулою 4.2.

$$З_{дод} = З_о \cdot \frac{Н_{дод}}{100\%}, \quad (4.2)$$

де $Н_{дод}$– норма нарахування додаткової заробітної плати.

В даному випадку додаткова заробітна плата буде становити:

$$З_{дод} = 39285{,}71 \cdot \frac{11}{100} = 4321{,}43 \text{ грн.}$$

До статті «Відрахування на соціальні заходи» належать відрахування внеску на загальнообов’язкове державне соціальне страхування та для

здійснення заходів щодо соціального захисту населення (ЄСВ – єдиний соціальний внесок).

Нарахування на заробітну плату дослідників та робітників розраховується як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою 4.3.

$$З_н = (З_о + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (4.3)$$

де $Н_{зп}$ – норма нарахування на заробітну плату.

В даному випадку відрахування на соціальні заходи будуть обраховуватись наступним чином:

$$З_н = (39\,285{,}71 + 4321{,}43) \cdot \frac{22}{100} = 9593{,}57 \text{ грн.}$$

Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

До статті «Амортизація обладнання, програмних засобів та приміщень» відносять амортизаційні відрахування по кожному виду обладнання, устаткування та інших приладів і пристроїв, а також програмного забезпечення для проведення науково-дослідної роботи, за його наявності в дослідній організації або на підприємстві.

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо можуть бути розраховані з використанням прямолінійного методу амортизації за формулою 4.4.

$$А_{обл} = \frac{Ц_б}{Т_в} \cdot \frac{t_{вик}}{12}, \quad (4.4)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$Т_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Проведені розрахунки необхідно звести до таблиці.

Таблиця 4.5 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	*Балансова вартість (грн)*	*Строк корисного використання (роки)*	*Термін використання обладнання (місяці)*	*Амортизаційні відрахування (грн)*
Персональний комп'ютер	25 000,00	2	1,27	2 083,33
Офісне приладдя	23 000,00	4	1,27	958,33
Приміщення	600 000,00	20	1,27	5000,00
Всього				8041,67

До статті «Паливо та енергія для науково-виробничих цілей» належать витрати на придбання у сторонніх підприємств, установ і організацій будь-якого палива, що витрачається з технологічною метою на проведення досліджень.

Витрати на силову електроенергію ($В_е$) розраховують за формулою (4.5).

$$В_е = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i} \quad , \qquad (4.5)$$

де W_{yi} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії. На момент виконання роботи вартість становить 6,28 грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі}$ становить 0,6;

η_i – коефіцієнт корисної дії обладнання, η_i становить 0,5.

Так як з електронних приборів в розробці приймає участь тільки комп'ютер значення n буде становити 1. В даному випадку витрати на електроенергію будуть обраховуватись наступним чином:

$$В_е = \sum_{i=1}^{1} \frac{0{,}6 \cdot 240 \cdot 22{,}8 \cdot 0{,}6}{0{,}7} = 775{,}13 \text{ грн.}$$

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50% від суми основної заробітної плати дослідників та робітників за формулою 4.6.

$$І_в = З_о \cdot \frac{Н_{ів}}{100\%}, \qquad (4.6)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

Інші витрати будуть становити:

$$І_в = 39\,285{,}71 \cdot \frac{50}{100} = 19642{,}86 \text{ грн.}$$

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 120% від суми основної заробітної плати дослідників та робітників за формулою 4.7.

$$В_{нзв} = З_о \cdot \frac{Н_{нзв}}{100\%}, \quad (4.7)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

Витрати за статтею Накладні (загальновиробничі) витрати будуть становити:

$$В_{нзв} = 39\,285{,}71 \cdot \frac{120}{100} = 47142{,}86 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою 4.9.

$$В_{заг} = З_о + З_{дод} + З_н + А_{обл} + В_е + І_в + В_{нзв}. \quad (4.9)$$

Витрати на проведення науково-дослідної роботи становлять:

$$В_{заг} = 39285{,}71 + 4321{,}43 + 9593{,}57 + 8041{,}67 + 775{,}13 + 19642{,}86 + \\ + 22909{,}09 = 114888{,}23 \text{ грн.}$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (4.10)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи. Так, якщо науково-технічна розробка знаходиться на стадії розробки промислового зразка, то η =0,5.

В даній роботі загальні витрати на завершення науково-дослідної будуть становити:

$$ЗВ = \frac{114888,23}{0,5} = 229776,45 \text{ грн.}$$

4.3 Розрахунок економічної ефективності науково-технічної розробки та її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

а) вказати, з якого часу можуть бути впроваджені результати науковотехнічної розробки;

б) зазначити, протягом скількох років після впровадження цієї науковотехнічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 4-х років після її впровадження);

в) кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

г) визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу.

При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

1. абсолютного економічного ефекту (чистого дисконтованого доходу);
2. внутрішньої економічної дохідності (внутрішньої норми дохідності);
3. терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об’єднати, враховуючи визначені ситуації з відповідними умовами.

Дану роботу можна віднести до розробки чи суттєвого вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем. В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних: ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик; N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки; $Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки; $\pm\Delta Ц_о$ – зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу.

Для обрахунку можливого збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, необхідно використати формулу:

$$\Delta П_і = (\pm \Delta Ц_о \cdot N + Ц_о \cdot \Delta N) \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right), \quad (4.11)$$

де $\pm \Delta Ц_о$ – зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай, таким показником може бути зміна ціни реалізації одиниці нової розробки в аналізованому році (відносно року до впровадження цієї розробки); ±ΔЦо може мати як додатне, так і від'ємне значення (від'ємне – при зниженні ціни відносно року до впровадження цієї розробки, додатне – при зростанні ціни);

N – основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році, $Ц_о = Ц_б \pm \Delta Ц_о$;

$Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів;

ΔN– зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай таким показником може бути зростання попиту на науково-технічну розробку в аналізованому році (відносно року до впровадження цієї розробки);

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2022 році ставка податку на додану вартість становить 20%, а коефіцієнт λ =0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). ρ =0,5;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій ціні 190 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на 50 грн. Кількість одиниць

реалізованої продукції також збільшиться: протягом першого року – на 12000 шт., протягом другого року – на 7000 шт., протягом третього року на 7000 шт. До моменту впровадження результатів наукової розробки реалізації продукту набуло:

$$\Delta П_1 = (0 + (190 + 50) \cdot 12000) \cdot 0{,}8333 \cdot 0{,}5 \cdot \left(1 - \frac{\vartheta}{100}\right) =$$

$$= 1082395{,}67 \text{ грн};$$

$$\Delta П_2 = (0 + (190 + 50) \cdot (12000 + 7000)) \cdot 0{,}8333 \cdot 0{,}5 \cdot \left(1 - \frac{\vartheta}{100}\right) =$$

$$= 1713793{,}145 \text{ грн};$$

$$\Delta П_3 = (0 + (190 + 50) \cdot (12000 + 7000 + 7000)) \cdot 0{,}8333 \cdot 0{,}5 \cdot \left(1 - \frac{\vartheta}{100}\right) =$$

$$= 2345190{,}619 \text{ грн}.$$

Далі розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (4.12)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,1;

t – період часу (в роках) від моменту початку впровадження науковотехнічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році

$$\text{ПП} = \frac{1082395{,}67}{(1+0{,}1)^1} + \frac{1713793{,}145}{(1+0{,}1)^2} + \frac{2345190{,}619}{(1+0{,}1)^3} = 4673981{,}30 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 4673981,30 грн.

Далі розраховують величину початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу 4.13.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \qquad (4.13)$$

де $k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; $k_{\text{інв}}$=3;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 3 \cdot 229\,776{,}45 = 689\,329{,}36 \text{ грн}$$

Тоді абсолютний економічний ефект $\text{Е}_{\text{абс}}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$\text{Е}_{\text{абс}} = \text{ПП} - PV, \qquad (4.14)$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн.

Якщо величина $E_{абс}$ буде мати велике додатне значення, то це може свідчити про потенційну зацікавленість інвесторів у впровадженні та комерціалізації цієї науково-технічної розробки. Але для остаточного прийняття рішення про впровадження науково-технічної розробки та виведення її на ринок (тобто її комерціалізації) цього недостатньо.

$$E_{абс} = 4673981{,}30 - 689\,329{,}36 = 12057892{,}38 \text{ грн.}$$

Як показує обрахунок значення $E_{абс}$ є додатнім, а значить науково-технічна розробка може викликати у інвесторів зацікавленість у впровадженні та комерціалізації.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність $E_в$ або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науковотехнічної розробки, розраховується за формулою:

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1 \quad , \qquad (4.15)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

Тж – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$E_в = \sqrt[1,5]{1 + \frac{12057892,38}{689\ 329,36}} - 1 = 0,55866.$$

Далі визначають бар'єрну ставку дисконтування $\tau_{мін}$, тобто мінімальну внутрішню економічну дохідність інвестицій, нижче якої кошти у впровадження науково-технічної розробки та її комерціалізацію вкладатися не будуть. Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$ визначається за формулою:

$$\tau_{мін} = d + f, \qquad (4.16)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d =0,9;

f – показник, що характеризує ризикованість вкладення інвестицій; зазвичай величина f =0,1.

$$\tau_{мін} = 0,9 + 0,1 = 1.$$

Якщо величина $E_в > \tau_{мін}$, то потенційний інвестор може бути зацікавлений у фінансуванні впровадження науково-технічної розробки та виведенні її на ринок, тобто в її комерціалізації. Далі розраховуємо період окупності інвестицій $T_{ок}$ (DPP, Discounted Payback Period), які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \quad (4.17)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$$T_{ок} = \frac{1}{0{,}55866} = 1{,}8 \text{ роки.}$$

В моєму випадку $T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

Висновок до розділу 4

Під час виконання четвертого розділу було спрогнозовано витрати на виконання науково-дослідної роботи. Ці витрати становлять 229776,45 грн. Ще було розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. Період окупності інвестицій становить 1,8 роки, а це говорить про те, що розробка є комерційно привабливою.

ВИСНОВКИ

Під час виконання магістерської кваліфікаційної роботи було розв’язано задачу розробки інформаційної технології підвищення рівня інтелектуальності персонажів комп'ютерної гри за допомогою використання алгоритму A* та дерева прийняття рішень.

У першому розділі магістерської роботи було проведено аналіз предметної області штучного інтелекту в комп’ютерних іграх, а саме – сформульовано постановку задачі, проведено огляд сучасних технологій створення штучного інтелекту. Обгрунтовано вибір гри аналогу Enter the Gungeon. Недоліком аналогу є низька інтелектуальність та простота персонажів, а звідси і випливає мета даної роботи – підвищення рівня інтелектуальності персонажів комп'ютерної гри для покращення ігрового досвіду, що отримують гравці.

У другому розділі магістерської кваліфікаційної роботи було проаналізовано алгоритмів пошуку шляху штучним інтелектом персонажів комп’ютерної гри та обґрунтовано вибір для даної задачі алгоритму A* який є простим в реалізації і в той же час швидким і ефективним для поставленої задачі. Обрано та розроблено структуру та основний принцип роботи штучного інтелекту.

У третьому розділі розглянуто плюси та мінуси об’єктно-орієнтованої мови програмування C# та обґрунтовано її вибір. У результаті було розроблено гру з реалізацією розвинутого штучного інтелекту персонажів, створену мовою програмування C# із застосуванням безкоштовної бібліотеки MonoGame та середовища Microsoft Visual Studio. Було проведено тестування гри. Аналіз результатів тестування показує, що розроблений новий штучний інтелект має кращу точність за інтелект аналог на 35%. Крім цього, розроблена програма має також можливість ухилятись від атак противника на 70% краще, а також можливість підбирати предмети гравця, що робить геймплей важчим і цікавішим для гравця. Тобто мета магістерської кваліфікаційної роботи

досягнута – розроблена модель інтелектуальної поведінки краще за модель аналог на 33,75% при схожих налаштуваннях.

У четвертому розділі було виконано розрахунок витрат на розробку та виготовлення нового технічного рішення, сума яких складає 229776,45 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який потенційно може отримати виробник від реалізації розробленого технічного рішення, знайдено термін окупності витрат виробника та економічний ефект для споживача при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розробка у виробництві та використанні є дешевша за аналог, а також висококонкурентоспроможною. Період окупності складе близько 1,8 роки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Steven L. Kent. The Ultimate History of Video Games: From Pong to Pokemon--The Story Behind the Craze That Touched Our Lives and Changed the World September 6, 2001

2 Technologies and Innovation Second International Conference, Guayaquil, Ecuador, November 23-25, 2016

3 Городецький Ю.Г. «Інформаційна технологія підвищення рівня інтелектуальності персонажів комп'ютерної гри» Матеріали конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)», Вінниця, 2022. [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2023/paper/view/16852

4 Pennachin, C.; Goertzel, B. (2007). "Contemporary Approaches to Artificial General Intelligence". Artificial General Intelligence. Cognitive Technologies. Berlin, Heidelberg: Springer.

5 It | The New Yorker [Електронний ресурс]. Режим доступу: https://www.newyorker.com/magazine/1952/08/02/it

6 McCorduck, Pamela (2004), Machines Who Think (2nd ed.), Natick, MA: A. K. Peters, Ltd.

7 History of AI Use in Video Game Design - Big Data Analytics News [Електронний ресурс]. Режим доступу: https://bigdataanalyticsnews.com/history-of-artificial-intelligence-in-video-games/

8 Beginners Guide to Video Game Development in 2023 [Електронний ресурс]. Режим доступу: https://www.gamedesigning.org/video-game-development/

9 A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games [Електронний ресурс]. Режим доступу: https://doi.org/10.1155%2F2015%2F736138

10 AI in Gaming | 5 Innovations Changing The Future of Gaming | Engati [Електронний ресурс]. Режим доступу: https://www.engati.com/blog/ai-in-gaming#toc-what-are-the-kinds-of-ai-in-games-

11 Dungeons 2 on Steam [Електронний ресурс]. Режим доступу: https://store.steampowered.com/app/262280/Dungeons_2

12 Enter the Gungeon [Електронний ресурс]. Режим доступу: https://enterthegungeon.com

13 "ARMAC". Unsung Heroes in Dutch Computing History. 2007. Archived from the original on 13 November 2013

14 Frana, Phil (August 2010). "An Interview with Edsger W. Dijkstra". Communications of the ACM.

15 V. Jarník: O jistém problému minimálním [About a certain minimal problem], Práce Moravské Přírodovědecké Společnosti, 6, 1930, pp. 57–63.

16 Fredman, Michael Lawrence; Tarjan, Robert E. (1987). "Fibonacci heaps and their uses in improved network optimization algorithms". Journal of the Association for Computing Machinery.

17 Schrijver, Alexander (2005). "On the history of combinatorial optimization (till 1960)" Handbook of Discrete Optimization. Elsevier

18 Bellman–Ford Algorithm | DP-23 – GeeksforGeeks [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/

19 SRI educational film demonstrating Shakey [Електронний ресурс]. Режим доступу: https://www.youtube.com/watch?v=qXdn6ynwpiI

20 Nilsson, Nils J. (2009-10-30). The Quest for Artificial Intelligence. Cambridge: Cambridge University Press.

21 A* Search Algorithm – GeeksforGeeks [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/a-search-algorithm/

22 16 best artificial intelligence games & AI gaming – Dataconomy [Електронний ресурс]. Режим доступу: https://dataconomy.com/2022/04/artificial-intelligence-games/

23 architecture - Difference between Decision Trees & Behavior Trees for Game AI - Game Development Stack Exchange [Електронний ресурс]. Режим доступу: https://gamedev.stackexchange.com/questions/51693/difference-between-decision-trees-behavior-trees-for-game-ai

24 Behavior trees for AI: How they work [Електронний ресурс]. Режим доступу: https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work

25 Applications of Neural-Based Agents in Computer Game Design | IntechOpen [Електронний ресурс]. Режим доступу: https://www.intechopen.com/chapters/10916

26 16 best artificial intelligence games & AI gaming – Dataconomy [Електронний ресурс]. Режим доступу: https://dataconomy.com/2022/04/artificial-intelligence-games/

27 ECMA-334 - Ecma International [Електронний ресурс] – Режим доступу до ресурсу: https://www.ecma-international.org/publications-and-standards/standards/ecma-334/

28 Visual Studio 2019 Preview Release Notes | Microsoft Docs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes-preview

29 Introduction | MonoGame Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.monogame.net/

30 C#: The Complete Reference - Herbert Schildt

31Paul Barry Head First Python: A Brain-Friendly Guide; O'Reilly Media; 1st edition: 2010 – 494 с.

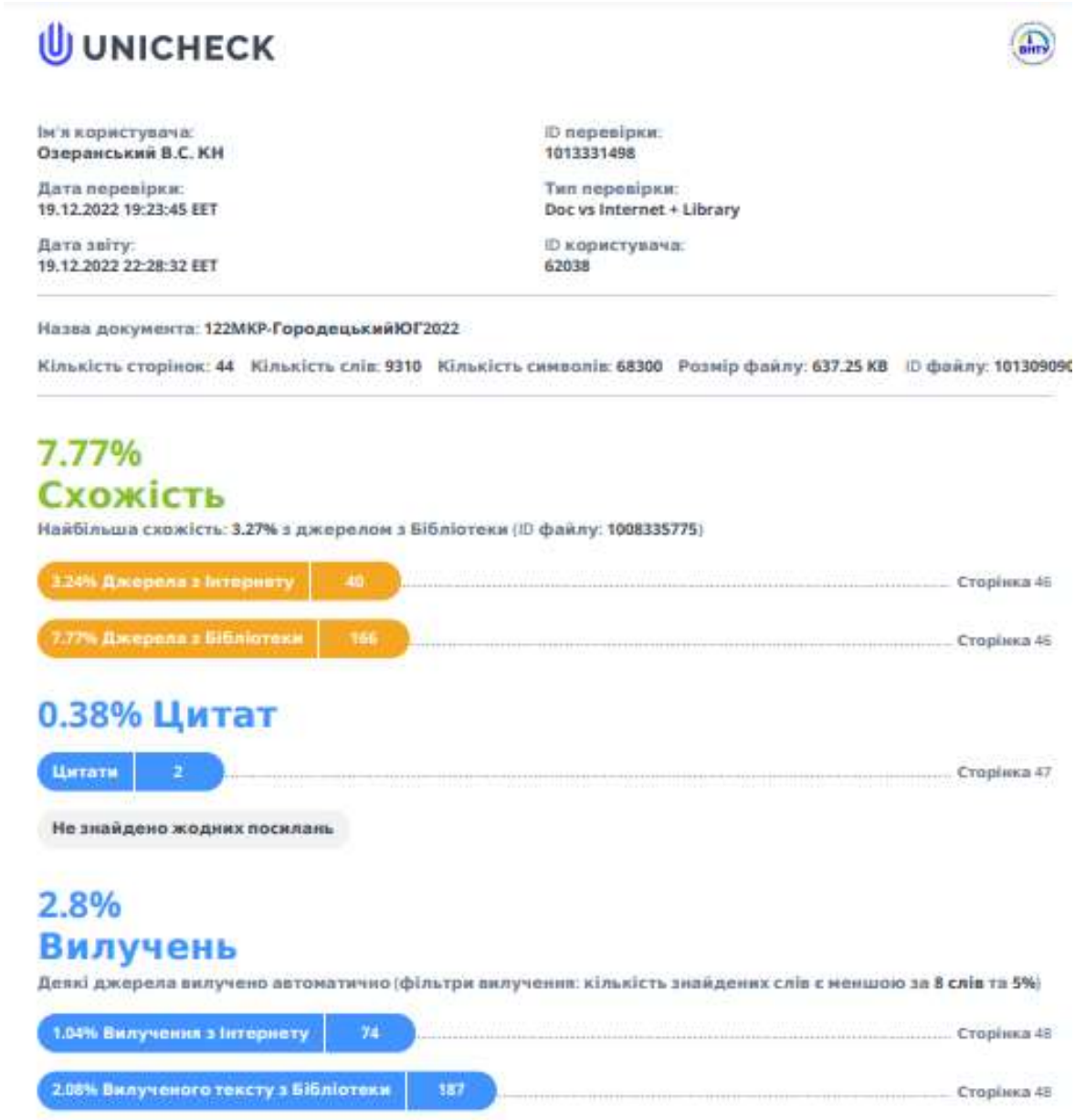
32 Mark Myers A Smarter Way to Learn JavaScript: The new approach that uses technology to cut your effort in half; 2013 – 256 с.

33 The Making Of: Rogue - Edge Magazine [Електронний ресурс] – Режим доступу до ресурсу: https://web.archive.org/web/20120815191124/http://www.edge-online.com/features/making-rogue

ДОДАТКИ

Додаток А

Результат перевірки на плагіат в онлайн-системі UNICHECK



UNICHECK

Ім'я користувача:
Озеранський В.С. КН

ID перевірки:
1013331498

Дата перевірки:
19.12.2022 19:23:45 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
19.12.2022 22:28:32 EET

ID користувача:
62038

Назва документа: 122МКР-ГородецькийЮГ2022

Кількість сторінок: 44 Кількість слів: 9310 Кількість символів: 68300 Розмір файлу: 637.25 KB ID файлу: 10130909

7.77%
Схожість

Найбільша схожість: 3.27% з джерелом з Бібліотеки (ID файлу: 1008335775)

3.24% Джерела з Інтернету 40 Сторінка 46

7.77% Джерела з Бібліотеки 166 Сторінка 46

0.38% Цитат

Цитати 2 Сторінка 47

Не знайдено жодних посилань

2.8%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за 8 слів та 5%)

1.04% Вилучення з Інтернету 74 Сторінка 48

2.08% Вилученого тексту з Бібліотеки 187 Сторінка 48

Додаток Б

Лістинг програми

CameraTarget.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MonoGameTest.Core.Game
{
    public class CameraTarget : IGameObject
    {
        public bool SmoothFollowing = true;
        public float Speed = 10f;
        public IGameObject FollowedGameObject;
        public Vector2 SmoothedPosition;

        public Vector2 Position
        {
            get
            {
                if (FollowedGameObject != null && !SmoothFollowing) {
                    return FollowedGameObject.Position;
                }
                return SmoothedPosition;
            }
        }

        public void Draw(SpriteBatch spriteBatch)
        {
        }

        public void Update(float dt, GameContext context)
        {
            if (FollowedGameObject != null) {
```

```
                    SmoothedPosition += (FollowedGameObject.Position -
SmoothedPosition) * dt * Speed;
                }
            }
        }
    }
```

GameProjectile.cs

```
    using Microsoft.Xna.Framework;
    using Microsoft.Xna.Framework.Graphics;
    using MonoGameTest.Core.Physics;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Threading.Tasks;

    namespace MonoGameTest.Core.Game
    {
        public class GameProjectile : Projectile, IGameObject
        {
            public Texture2D Texture;
            public float TextureDiameter = 1;
            public float Angle = 0;
            public float LifeTime = 10;

            Vector2 IGameObject.Position { get { return Position; } }

            public GameProjectile(Texture2D texture, float radius, Material material)
: base(radius, material)
            {
                Texture = texture;
                TextureDiameter = radius * 2;
            }

            public void Draw(SpriteBatch spriteBatch)
            {
                spriteBatch.Draw(Texture, Position, null, Color.White,
MathUtil.GetAngle(Velocity), new Vector2(Texture.Width / 2, Texture.Height / 2),
TextureDiameter / MathF.Max(Texture.Width, Texture.Height), SpriteEffects.None, 1);
```

```
        }

        public void Update(float dt, GameContext context)
        {
            LifeTime -= dt;
            if (LifeTime < 0 || Velocity.LengthSquared() < 40)
            {
                context.RemoveObject(this);
            }
        }
    }
}
```

GameWorld.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameTest.Core.Physics;
using System;
using System.Collections.Generic;

namespace MonoGameTest.Core.Game
{
    public class GameWorld
    {
        private PhysicWorld _physicWorld = new PhysicWorld();
        private GameContext _context;

        private float _cameraZoom = 50;


        private Matrix _transformMatrix;

        private GraphicsDevice _graphicsDevice;

        public IGameObject CameraTarget;

        public GameWorld(GraphicsDevice graphicsDevice)
        {
            _graphicsDevice = graphicsDevice;
```

```
            _physicWorld.HalfSize = new Vector2(100, 100);
            _physicWorld.Center = new Vector2(100, 100);

            var player = new Entity(new Vector2(0, 0), new ManualController()) {
Team = 1};
            AddGameObject(player);

            AddGameObject(new Entity(new Vector2(20, 20), new AIController()));

            CameraTarget = new CameraTarget() { FollowedGameObject = player };
            AddGameObject(CameraTarget);

            AddGameObject(new Wall(Textures.VerticalWall, new Vector2(10, 10),
0.1f));

            _context = new GameContext(this);


        }


        float t = 0;
        public List<IGameObject> GameObjects = new List<IGameObject>();
        public void Update(float dt)
        {
            var mouseState = Mouse.GetState();

            _context.MousePosition =
Vector2.Transform(mouseState.Position.ToVector2(), Matrix.Invert(_transformMatrix));

            //Debug.SetValue("mouseX", _context.MousePosition.X.ToString());
            //Debug.SetValue("mouseY", _context.MousePosition.Y.ToString());

            t += dt;
            foreach(var obj in GameObjects)
            {
                obj.Update(dt, _context);
            }
            _context.CommitAction();
            _physicWorld.Update(dt);
        }
```

```
        public void Draw(SpriteBatch spriteBatch)
        {

            var cameraPoint = CameraTarget?.Position ?? Vector2.Zero;

            _transformMatrix =
                Matrix.CreateTranslation(-cameraPoint.X, -cameraPoint.Y, 0)
                * Matrix.CreateRotationZ(0)
                * Matrix.CreateScale(_cameraZoom, _cameraZoom, 1f)
            * Matrix.CreateTranslation(_graphicsDevice.Viewport.Width / 2,
_graphicsDevice.Viewport.Height / 2, 0);

            spriteBatch.Begin(transformMatrix: _transformMatrix);
            foreach (var obj in GameObjects)
            {
                obj.Draw(spriteBatch);
            }
            spriteBatch.End();
        }

        public void AddGameObject(IGameObject obj)
        {
            if (obj is Projectile)
            {
                _physicWorld.AddProjectile(obj as Projectile);
            }
            else if (obj is RigidBody)
            {
                _physicWorld.AddBody(obj as RigidBody);
            }
            GameObjects.Add(obj);
        }
        public bool DeleteGameObject(IGameObject obj)
        {
            if (obj is Projectile)
            {
                _physicWorld.RemoveProjectile(obj as Projectile);
            }
```

```
                else if (obj is RigidBody)
                {
                    _physicWorld.RemoveBody(obj as RigidBody);
                }

                return GameObjects.Remove(obj);

            }
        }
    }
```

ManualController.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Input;
using System;

namespace MonoGameTest.Core.Game
{
    public class ManualController : IController
    {
        public ControlResult Hangle(Entity entity, GameContext context)
        {
            var state = Keyboard.GetState();

            Vector2 moveDir = Vector2.Zero;

            moveDir.X -= state.IsKeyDown(Keys.A) ? 1 : 0;
            moveDir.X += state.IsKeyDown(Keys.D) ? 1 : 0;
            moveDir.Y -= state.IsKeyDown(Keys.W) ? 1 : 0;
            moveDir.Y += state.IsKeyDown(Keys.S) ? 1 : 0;

            var mouseState = Mouse.GetState();


            return new ControlResult()
            {
                Move = MathUtil.ClampVector(moveDir),
                Angle = MathUtil.GetAngle(context.MousePosition -
entity.Position),
                Action = false,
```

```
                    Shoot = state.IsKeyDown(Keys.Space)
                };
            }
        }
    }
```

RidgidGameObject.cs

```
    using Microsoft.Xna.Framework;
    using Microsoft.Xna.Framework.Graphics;
    using MonoGameTest.Core.Physics;
    using System;

    namespace MonoGameTest.Core.Game
    {
        public abstract class RigidGameObject : RigidBody, IGameObject
        {
            public Texture2D Texture;
            public float TextureDiameter = 1;
            public float Angle = 0;

            Vector2 IGameObject.Position { get { return Position; } }

            public RigidGameObject(Collider collider, Texture2D texture, float
textureDiameter, Material material = null) :
                base(collider, material)
            {
                Texture = texture;
                TextureDiameter = textureDiameter;
            }
            public RigidGameObject(CircleCollider circleCollider, Texture2D texture,
Material material = null) :
                base(circleCollider, material)
            {
                Texture = texture;
                TextureDiameter = circleCollider.Radius * 2;
            }

            public virtual void Draw(SpriteBatch spriteBatch)
            {
                spriteBatch.Draw(
```

```
                    Texture,
                    Position,
                    null,
                    Color.White,
                    Angle,
                    new Vector2(Texture.Width / 2, Texture.Height / 2),
                    TextureDiameter / MathF.Max(Texture.Width, Texture.Height),
                    SpriteEffects.None, 1
                );
            }

            public virtual void Update(float dt, GameContext context)
            {

            }
        }
    }
```

Wall.cs

```
    using Microsoft.Xna.Framework;
    using Microsoft.Xna.Framework.Graphics;
    using MonoGameTest.Core.Physics;
    using System;

    namespace MonoGameTest.Core.Game
    {
        public class Wall : RigidGameObject
        {
            public Wall(Texture2D texture, Vector2 position, float scale = 1): base(
                    new RectangleCollider(texture.Width * scale, texture.Height *
scale),
                    texture, Math.Max(texture.Width, texture.Height) * scale
                )
            {
                IsStatic = true;
                Position = position;
            }
        }
    }
```

Entity.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using MonoGameTest.Core.Physics;
using System;

namespace MonoGameTest.Core.Game
{
    public class Entity : RigidGameObject
    {
        public static int IdCounter = 0;

        public int Id = IdCounter++;

        public IController Controller;
        public float MaxSpeed = 5;

        public int Team = 0;
        public float Reloading = 0;

        public float Health = 100;
        public Entity(Vector2? position = null, IController controller = null) :
base(new CircleCollider(0.5f), Textures.PlayArrow, Material.Entity)
        {
            if (position != null)
            {
                Position = position.Value;
            }
            Controller = controller;
        }

        public void Fire(GameContext context)
        {
            var direction = new Vector2(MathF.Cos(Angle), MathF.Sin(Angle));
            float projectileVelocity = 60;
            var projectile = new Bullet()
            {
                Velocity = direction * projectileVelocity + Velocity
            };
```

```
                projectile.Position = Position + direction * ((Collider as
CircleCollider).Radius + projectile.Radius);

                Velocity -= 0.1f * direction * projectileVelocity * projectile.Mass /
Mass;

                context.AddGameObject(projectile);
            }

            public override void Update(float dt, GameContext context)
            {
                if (Controller != null)
                {
                    var controlState = Controller.Hangle(this, context);
                    var targetSpeed = controlState.Move * MaxSpeed;
                    Velocity += (targetSpeed - Velocity) * 10 * dt;
                    Angle = controlState.Angle;
                    if (controlState.Shoot && Reloading == 0)
                    {
                        Fire(context);
                        Reloading += 0.2f;
                    }
                }

                Reloading = Reloading < dt ? 0 : Reloading - dt;

                Debug.SetValue($"Entity{Id}.Health", Health.ToString());
                Debug.SetValue($"Entity{Id}.Position", Position.ToString("0.00"));
                Debug.SetValue($"Entity{Id}.Velocity", Velocity.ToString("0.00"));
                Debug.SetValue($"Entity{Id}.Reloading", Reloading.ToString());

                base.Update(dt, context);
            }

            public override void Draw(SpriteBatch spriteBatch)
            {
                base.Draw(spriteBatch);

                spriteBatch.Draw(
                    Textures.HealthBar,
```

```
                    Position + new Vector2(-Textures.HealthBar.Width / 2 * 0.05f,
0.55f),
                    null,
                    new Color(1 - Health / 100, Health / 100,0),
                    0,
                    new Vector2(0, Textures.HealthBar.Height/2),
                    new Vector2(0.05f * Health / 100, 0.02f),
                    SpriteEffects.None,
                    1
                );
            }

            public override void OnCollision(Collision collision)
            {
                if (collision.B is Bullet)
                {
                    Debug.SetValue("hitSpeed", $"{collision.Speed}");
                    ApplyDamage(75f * MathF.Pow(collision.Speed / 60, 2));

                }
                base.OnCollision(collision);
            }

            public void ApplyDamage(float damage)
            {
                Debug.SetValue("damage", $"{damage}");
                Health -= damage;
            }

        }
    }
```

AIController.cs

```
using Microsoft.Xna.Framework;
using System;
using System.Linq;

namespace MonoGameTest.Core.Game
{
    public class AIController : IController
```

```
        {
            private float _state = 0;
            public ControlResult Hangle(Entity entity, GameContext context)
            {

                var nearestTarget = context
                    .GameObjects
                    .Where(obj => obj is Entity && (obj as Entity).Team !=
entity.Team)
                    .MinBy(obj => (entity.Position - (obj as
Entity).Position).Length()) as Entity;

    {
            public float Xposition { get; set; }
            public float Yposition { get; set; }
            public float Cost { get; set; }
            public float Distance { get; set; }
            private float CostDistance => Cost + Distance;
            public float Parent { get; set; }

            public void SetDistance(int targetX, int targetY)
            {
                this.Distance = Math.Abs(targetX - Xposition) + Math.Abs(targetY -
Yposition);
            }
    }


                _state += 0.1f;

                return new ControlResult()
                {
                    Move = new Vector2(MathF.Sin(_state), MathF.Cos(_state)),
                    Angle = nearestTarget != null ?
MathUtil.GetAngle(nearestTarget.Position - entity.Position) : 0,
                    Action = false,
                    Shoot = false
                };
            }
        }
    }
```

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПІДВИЩЕННЯ РІВНЯ ІНТЕЛЕКТУАЛЬНОСТІ ПЕРСОНАЖІВ КОМП'ЮТЕРНОЇ ГРИ

Виконав: студент 2-го курсу, групи 1КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
_____ Городецький Ю.Г.
(прізвище та ініціали)

Керівник: к. т. н., доц. кафедри КН
_____ Барабан С.В.
(прізвище та ініціали)
«15» 12 2022 р.

Вінниця ВНТУ - 2022 рік

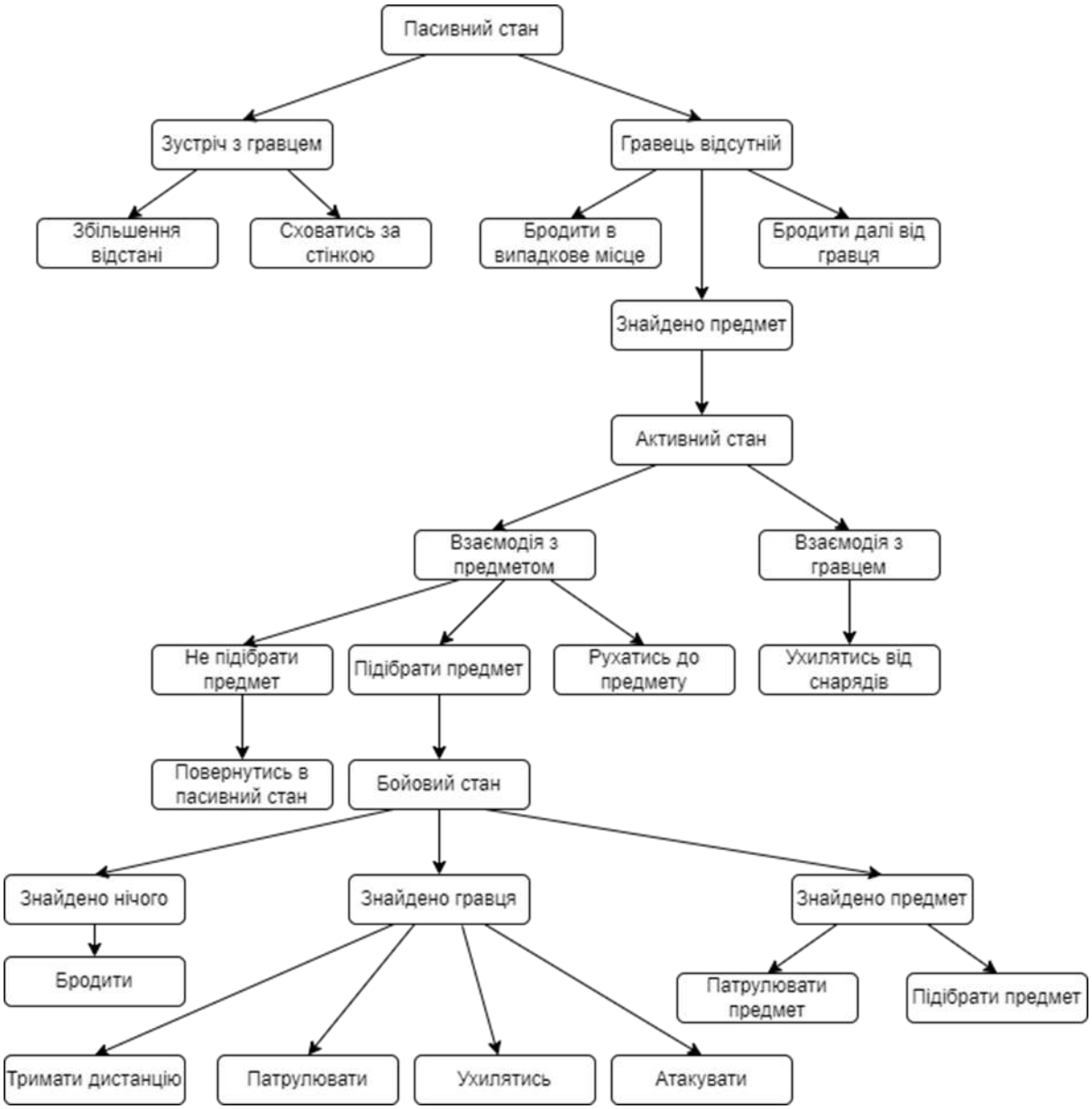


Рисунок В.1 – Дерево поведінки штучного інтелекту

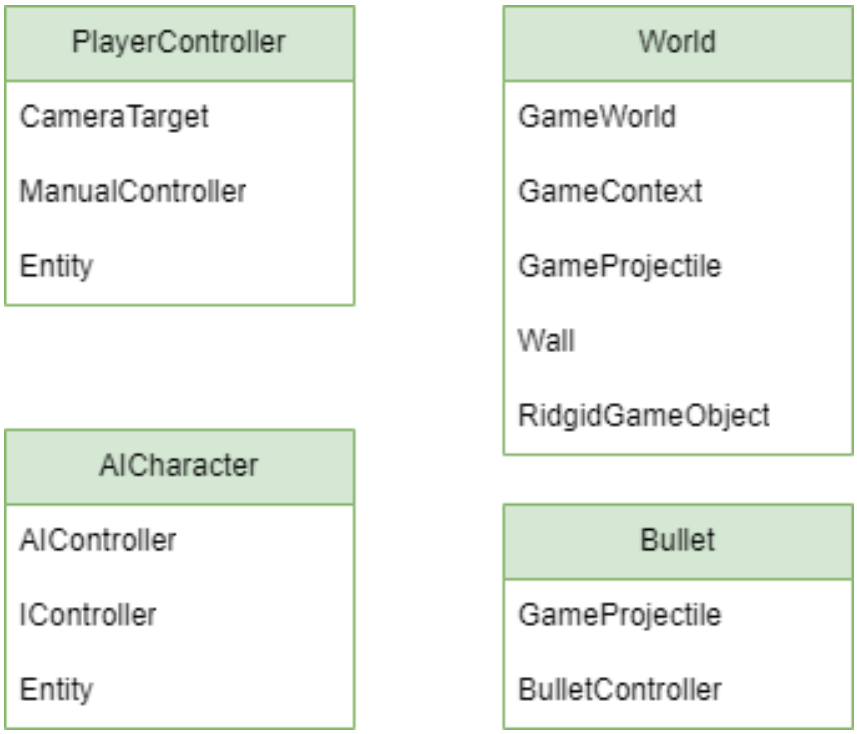


Рисунок 2.2 – UML діаграма класів комп’ютерної гри

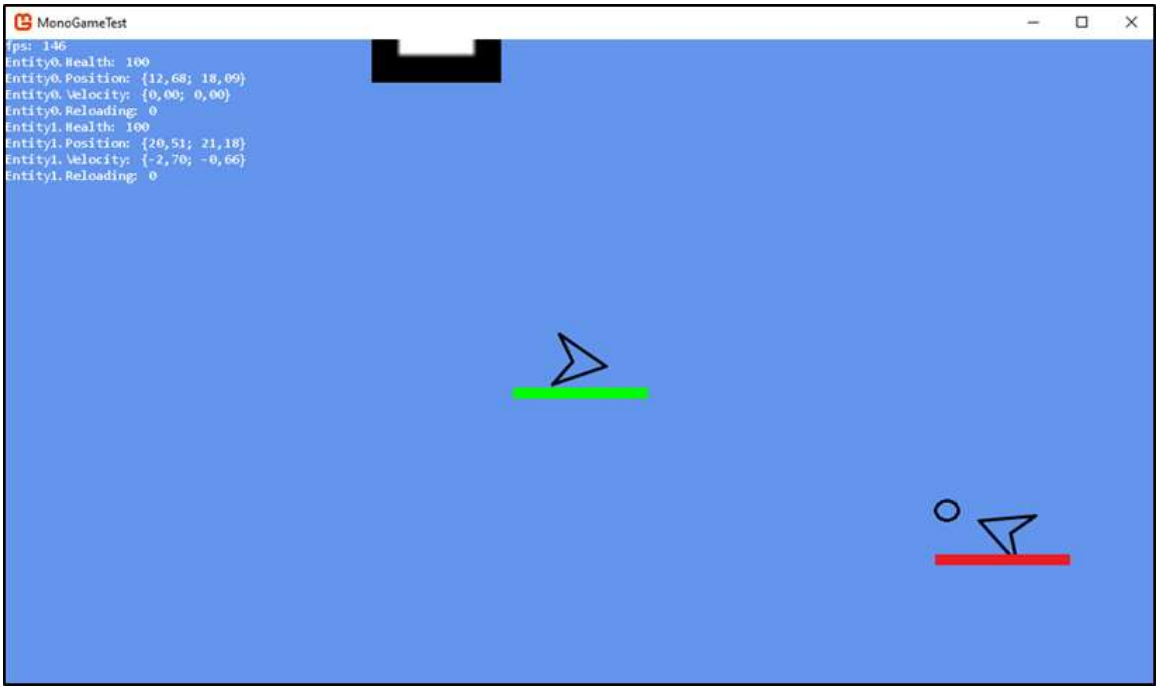


Рисунок В.3 – Програмна реалізація гри

Властивість	Розвинений ШІ	«Bullet Kin»	Відсоток покращення
Точність попадання	20 з 20	18 з 20	+10%
Точність попадання по рухомій цілі	12 з 20	5 з 20	+35%
Уникання попадання	17 з 20	3 з 20	+70%
Рух до кінцевої точки на одній швидкості	8 с	10 с	+20%
Середнє значення покращення			+33,75%

Рисунок В.4 Результати порівняння розробленого ШІ з інтелектом аналогом

Додаток Г

Інструкція користувача

Для запуску розробленої гри потрібно відкрити файл MonoTest.exe. Після запуску відкривається вікно гри, що показано на рис. Г.1. Керування персонажем здійснюється через клавіатуру та мишку. За допомогою клавіш «W» «A» «S» «D» здійснюється рух персонажем вверх, вліво, вниз, вправо відповідно. Клавіша «E» відповідає за підбор або використання предмету. Повороти персонажем здійснюються за допомогою комп’ютерної миші а стрільба за допомогою лівої кнопки миші.

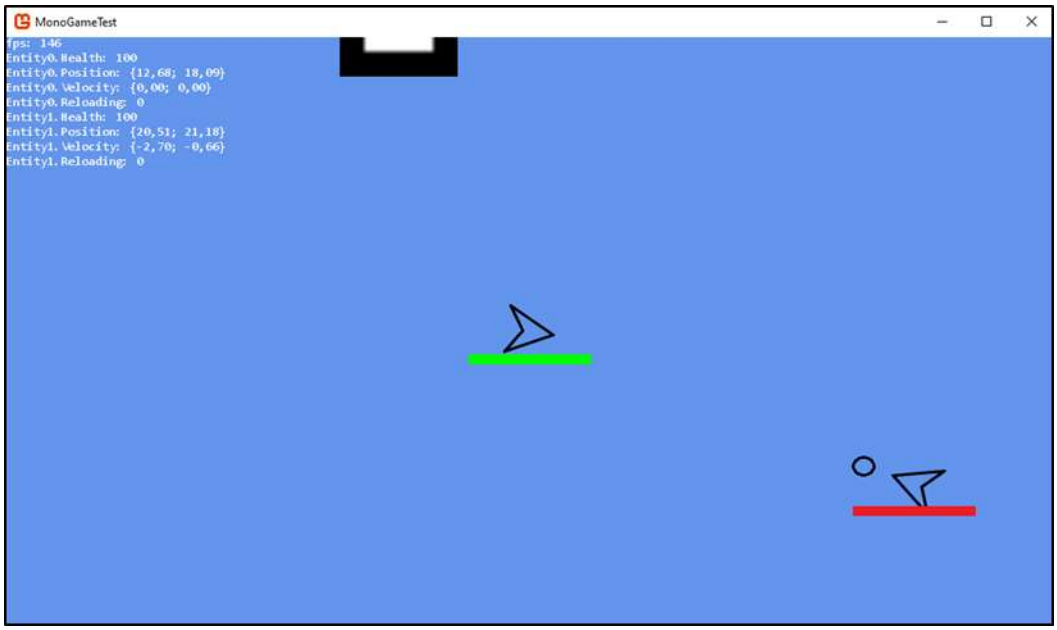


Рисунок Г.1 - Вид вікна комп’ютерної гри

Персонажі супротивники містять під собою червону шкалу, що відповідає за їхню кількість здоров’я. На даний момент існує два предмети для підбору. Перший предмет дає можливість здійснювати стрільбу повільними одиночними пострілами (квадрат з кругом всередині). Другий предмет дозволяє здійснювати автоматичну стрільбу але меншими снарядами (квадрат з трикутником всередині).